EURASIP Journal on Applied Signal Processing 2004:11, 1688–1707

Generic Multimedia Multimodal Agents Paradigms and Their Dynamic Reconfiguration at the Architectural Level

H. Djenidi

Département de Génie Électrique, École de Technologie Supérieure, Université du Québec, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
Email: hdjenidi@ele.etsmtl.ca

Laboratoire PRISM, Université de Versailles Saint-Quentin-en-Yvelines, 45 Avenue des États-Unis, 78035 Versailles Cedex, France

S. Benarif

Laboratoire PRISM, Université de Versailles Saint-Quentin-en-Yvelines, 45 Avenue des États-Unis, 78035 Versailles Cedex, France
Email: sab@prism.uvsq.fr

A. Ramdane-Cherif

Laboratoire PRISM, Université de Versailles Saint-Quentin-en-Yvelines, 45 Avenue des États-Unis, 78035 Versailles Cedex, France
Email: rca@prism.uvsq.fr

C. Tadj

Département de Génie Électrique, École de Technologie Supérieure, Université du Québec, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
Email: ctadj@ele.etsmtl.ca

N. Levy

Laboratoire PRISM, Université de Versailles Saint-Quentin-en-Yvelines, 45 Avenue des États-Unis, 78035 Versailles Cedex, France
Email: nlevy@prism.uvsq.fr

Received 30 June 2002; Revised 22 January 2004

The multimodal fusion for natural human-computer interaction involves complex intelligent architectures which are subject to the unexpected errors and mistakes of users. These architectures should react to events occurring simultaneously, and possibly redundantly, from different input media. In this paper, intelligent agent-based generic architectures for multimedia multimodal dialog protocols are proposed. Global agents are decomposed into their relevant components. Each element is modeled separately. The elementary models are then linked together to obtain the full architecture. The generic components of the application are then monitored by an agent-based expert system which can then perform dynamic changes in reconfiguration, adaptation, and evolution at the architectural level. For validation purposes, the proposed multiagent architectures and their dynamic reconfiguration are applied to practical examples, including a W3C application.

Keywords and phrases: multimodal multimedia, multiagent architectures, dynamic reconfiguration, Petri net modeling, W3C application.

1. INTRODUCTION

With the growth in technology, many applications supporting more transparent and flexible human-computer interactions have emerged. This has resulted in an increasing need for more powerful communication protocols, especially when several media are involved. Multimedia multimodal applications are systems combining two or more natural input modes, such as speech, touch, manual gestures, lip movements, and so forth. Thus, a comprehensive command or a metamessage is generated by the system and sent to a multimedia output device. A system-centered definition of multimodality is used in this paper. Multimodality provides two striking features which are relevant to the design of multimodal system software:

(i) the fusion of different types of data from various input devices;

(ii) the temporal constraints imposed on information processing to/from input/output devices.

Since the development of the first rudimentary but workable system, "Put-that-there" [1], which processes speech in parallel with manual pointing, other multimodal applications have been developed [2, 3, 4]. Each application is based on a dialog architecture combining modalities to match and elaborate on the relevant multimodal information. Such applications remain strictly based on previous results, however, and there is limited synergy among parallel ongoing efforts. Today, for example, there is no agreement on the generic architectures that support a dialog implementation, independently of the application type.

The main objective of this paper is twofold.

First, we propose generic architectural paradigms for analyzing and extracting the collective and recurrent properties implicitly used in such dialogs. These paradigms use the agent architecture concept to achieve their functionalities and unify them into generic structures. A software architecture-driven development process based on architectural styles consists of a requirement analysis phase, a software architecture phase, a design phase, and a maintenance and modification phase. During the software architectural phase, the system architecture is modeled. To do this, a modeling technique must be chosen, then a software architectural style must be selected and instantiated for the concrete problem to be solved. The architecture obtained is then refined either by adding details or by decomposing components or connectors (recursively, through modeling, choice of a style, instantiation, and refinement). This process should result in an architecture which is defined, abstract, and reusable. The refinement produces a concrete architecture meeting the environmental requirements, the functional and nonfunctional requirements, and all the constraints on dynamic aspects as well as on static ones.

Second, we study the ways in which agents can be introduced at the architectural level and how such agents improve some quality attributes by adapting the initial architecture.

Section 2 gives an overview and the requirements of multimedia multimodal dialog architecture (MMDA) and presents generic multiagent architectures based on the previous synthesis. Section 3 introduces the dynamic reconfiguration of the MMDA. This reconfiguration is performed by an agent-based expert system. Section 4 illustrates the proposed MMDA with a stochastic, timed, colored Petri net (CPN) example [5, 6, 7] of the classical "copy and paste" operations and illustrates in more detail the proposed generic architecture. This section also shows the suitability of CPN in comparison with another transition diagram, the augmented transition network (ATN). A second example shows the evolution of the previous MMDA when a new modality is added, and examines the component reconfiguration aspects of this addition. Section 5 presents, via a multimodal Web browser interface adapted for disabled individuals, the novelty of our approach in terms of ambient intelligence. This interface uses the fusion engine modeled with the CPN scheme.

2. GENERIC MULTIMEDIA MULTIMODAL DIALOG ARCHITECTURE

In this section, an introduction to multimedia multimodal systems provides a general survey of the topics. Then, a synthesis brings together the overview and the requirements of the MMDA. The proposed generic multiagent architectures are described in Section 2.3.

2.1. Introduction to multimedia multimodal systems

The term "multimodality" refers to the ability of a system to make use of several communication channels during user-system interactions. In multimodal systems, information like speech, pen strokes and touches, eye gaze, manual gestures, and body movements is produced from user input modes. These data are first acquired by the system, then they are *analyzed, recognized, and interpreted*. Only the resulting interpretations are memorized and/or *executed*. This ability to *interpret* by combining parallel information inputs constitutes the major distinction between multimodal and multimedia systems. Multimedia systems are able to obtain, stock, and restore different forms of data (text, images, sounds, videos, etc.) in storage/presentation devices (hard drive, CD-ROM, screen, speakers, etc.). Modality is an emerging concept combining the two concepts of media and sensory data. The phrase "sensory data" is used here in the context of the definition of perceptions: hearing, touch, sight, and so forth [8]. The set of multimedia multimodal systems constitutes a new direction for computing, provides several possible paradigms which include at least one recognition-based technology (speech, eye gaze, pen strokes and touches, etc.), and leads to applications which are more complex to manage than the conventional Windows interfaces, like icons, menus, and pointing devices.

There are two types of multimodality: input multimodality and output multimodality. The former concerns interactions initiated by the user, while the latter is employed by the system to return data and present information. The system lets the user combine multimodal inputs at his or her convenience, but decides which output modalities are better suited to the reply, depending on the contextual environment and task conditions.

The literature provides several classifications of modalities. The first type of taxonomy can be credited to Card et al. [9] and Buxton [10], who focus on physical devices and equipment. The taxonomy of Foley et al. [11] also classifies devices and equipment, but in terms of their tasks rather than their physical attributes. Frohlich [12] includes input and output interfaces in his classification, while Bernsen's [13] proposed taxonomy is exclusively dedicated to output interfaces. Coutaz and Nigay have presented, in [14], the CARE properties that characterize relations of assignment, equivalence, complementarity, and redundancy between modalities.

Table 1: Interaction systems.

Engagement	Distance	Type of system
Conversation	Small	High-level language
Conversation	Large	Low-level language
Model world	Small	Direct manipulation
Model world	Large	Low-level world

For output multimodal presentations, some systems already have their preprogrammed responses. But now, research is focusing on more intelligent interfaces which have the ability to dynamically choose the most suitable output modalities depending on the current interaction. There are two main motivations for multimedia multimodal system design.

Universal access

A major motivation for developing more flexible multimodal interfaces has been their potential to expand the accessibility of computing to more diverse and nonspecialist users. There are significant individual differences in people's ability to use, and their preferences for using, different modes of communication, and multimodal interfaces are expected to broaden the accessibility of computing to users of different ages, skill levels, and cultures, as well as to those with impaired senses or impaired motor or intellectual capacity [3].

Mobility

Another increasingly important advantage of multimodal interfaces is that they can expand the viable usage context to include, for example, natural field settings and computing while mobile [15, 16]. In particular, they permit users to switch modes as needed during the changing conditions of mobile use. Since input modes can be complementary along many dimensions, their combination within a multimodal interface provides broader utility across varied and changing usage contexts. For example, using the voice to send commands during movement through space leaves the hands free for other tasks.

2.2. Multimodal dialog architectures: overview and requirements

A basic MMDA gives the user the option of deciding which modality or combination of modalities is better suited to the particular task and environment (see examples in [15, 16]). The user can combine speech, pen strokes and touches, eye gaze, manual gestures, and body postures and movements via input devices (key pad, tactile screen, stylus, etc.) to dialog in a coordinated way with multimedia system output.

The environmental conditions could lead to more constrained architectures which have to remain adaptable during periods of continuous change caused by either an external disturbance or the user's actions. In this context, an initial framework is introduced in [17] to classify interactions which consider two dimensions ("engagement" and "distance"), and decomposes the user-system dialog into four types (Table 1).

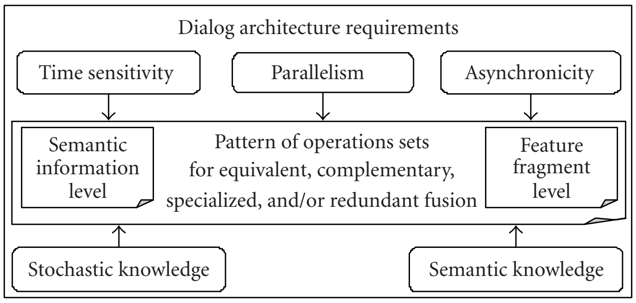

Dialog architecture requirements
Time sensitivity
Parallelism
Asynchronicity
Semantic information level
Pattern of operations sets for equivalent, complementary, specialized, and/or redundant fusion
Feature fragment level
Stochastic knowledge
Semantic knowledge

Figure 1: The main requirements for a multimodal dialog architecture (→: used by).

"Engagement" characterizes the level of involvement of the user in the system. In the "conversation" case, the user feels that an intermediary subsystem performs the task, while in the "model world" case, he can act directly on the system components. "Distance" represents the cognitive effort expended by the user.

This framework embodies the idea that two kinds of multimodal architectures are possible [18]. The first makes fusions based on signal feature recognition. The recognition steps of one modality guide and influence the other modalities in their own recognition steps [19, 20]. The second uses individual recognition systems for each modality. Such systems are associated with an extra process which performs semantic fusion of the individually recognized signal elements [1, 3, 21]. A third hybrid architecture is possible by mixing these two types: signal feature level and semantic information level.

At the core of multimodal system design is the main challenge of fusing the input modes. The input modes can be equivalent, complementary, specialized, or redundant, as described in [14]. In this context, the multimodal system designed with one of the previous architectures (features level, semantic level, or both) requires integration of the temporal information. It helps to decide whether two signal parts should belong to a multimodal fusion set or whether they should be considered as separate modal actions. Therefore, multimodal architectures are better able to avoid and recover errors which monomodal recognition systems cannot [18, 21, 22]. This property results in a more robust natural human-machine language. Another property is that the more growth there is in timed combinations of signal information or semantic multiple inputs, the more equivalent formulations of the same command are possible. For example, ["copy that there"], ["copy" (click) "there"], and ["copy that" (click)] are various ways to represent three statements of a same command (copying an object in a place) if speech and mouse-clicking are used. This redundancy also increases robustness in terms of error interpretation.

Figure 1 summarizes the main requirements and characteristics needed in multimodal dialog architectures.

As shown in this figure, five characteristics can be used in the two different levels of fusion operations, "early fusion" at the feature fragment level, and "late fusion" at the semantic level [18]. The property of asynchronicity gives the architecture the flexibility to handle multiple external events while parallel fusions are still being processed. The specialized fusion operation deals with an attribution of a modality to the same statement type. (For example, in drawing applications, speech is specialized for color statements, and pointing for basic shape statements.) The granularity of the semantic and statistical knowledge depends on the media nature of each input modality. This knowledge leads to important functionalities. It lets the system accept or reject the multi-input information for several possible fusions (selection process), and it helps the architecture choose, from among several fusions, the most suitable command to execute or the most suitable message to send to an output medium (decision process).

The property of parallelism is, obviously, inherent in applications involving multiple inputs. Taking the requirements as a whole strongly suggests the use of intelligent multiagent architectures, which are the focus of the next section.

2.3. *Generic multiagent architecture*

Agents are entities which can interact and collaborate dynamically and with synergy for combined modality issues. The interactions should occur between agents, and agents should also obtain information from users. An intelligent agent has three properties: it reacts in its environment at certain times (reactivity), takes the initiative (proactivity), and interacts with other intelligent agents or users (sociability) to achieve goals [23, 24, 25]. Therefore, each agent could have several input ports to receive messages and/or several output ports to send them.

The level of intelligence of each agent varies according to two major options which coexist today in the field of distributed artificial intelligence [26, 27, 28]. The first school, the cognitive school, attributes the level to the cooperation of very complex agents. This approach deals with agents with strong granularity assimilated in expert systems.

In the second school, the agents are simpler and less intelligent, but more active. This reactive school presupposes that it is not necessary that each agent be individually intelligent in order to achieve group intelligence [29]. This approach deals with a cooperative team of working agents with low granularity, which can be matched to finite automata.

Both approaches can be matched to the late and early fusions of multimedia multimodal architectures, and, obviously, there is a range of possibilities between these multiagent system (MAS) options. One can easily imagine systems based on a modular approach, putting submodules into competition, each submodule being itself a universe of overlapping components. This word is usually employed for "subagents."

Identifying the generic parts of multimodal multimedia applications and binding them into an intelligent agent architecture requires the determination of common and recurrent communication protocols and of their hierarchical and modular properties in such applications.

In most multimodal applications, speech, as the input modality, offers speed, a broad information spectrum, and relative ease of use. It leaves both the user's hands and eyes free to work on other necessary tasks which are involved, for example, in the driving or moving cases. Moreover, speech involves a generic language communication pattern between the user and the system.

This pattern is described by a grammar with production rules, able to serialize possible sequences of the vocabulary symbols produced by users. The vocabulary could be a word set, a phoneme set, or another signal fragment set, depending on the feature level of the recognition system. The goal of the recognition system is to identify signal fragments. Then, an agent organizes the fragments into a serial sequence according to his or her grammatical knowledge, and asks other agents for possible fusion at each step of the serial regrouping. The whole interaction can be synthesized into an initial generic agent architecture called the language agent (LA).

Each input modality must be associated with an LA. For basic modalities like manual pointing or mouse-clicking, the complexity of the LA is sharply reduced. The "vocabulary agent" that checks whether or not the fragment is known is, obviously, no longer necessary. The "sentence generation agent" is also reduced to a simple event thread whereon another external control agent could possibly make parallel fusions. In such a case, the external agent could handle "redundancy" and "time" information, with two corresponding components. These two components are agents which check redundancies and the time neighborhood of the fragments, respectively, during their sequential regrouping. The "serialization component" processes this regrouping. Thus, depending on the input modality type, the LA could be assimilated into an expert system or into a simple thread component.

Two or more LAs can communicate directly for early parallel fusions or, through another central agent, for late ones (Figure 2). This central agent is called a parallel control agent (PCA).

In the first case, the "grammar component" of one of the LAs must carry extra semantic knowledge for the purpose of parallel fusion. This knowledge could also be distributed between the LA's grammar components, as shown in Figure 2a. Several serializing components share their common information until one of them gives the sequential parallel fusion output. In the other case (Figure 2b), a PCA handles and centralizes the parallel fusions of different LA information. For this purpose, the PCA has two intelligent components, for redundancy and time management, respectively. These agents exchange information with other components to make the decision. Then, generated authorizations are sent to the semantic fusion component (SFCo). Based on these agreements, the SFCo carries out the steps of the semantic fusion process.

The redundancy and time management components receive the redundancy and time information via the SFCo or directly from the LA, depending on the complexity of the architecture and on designer choices.

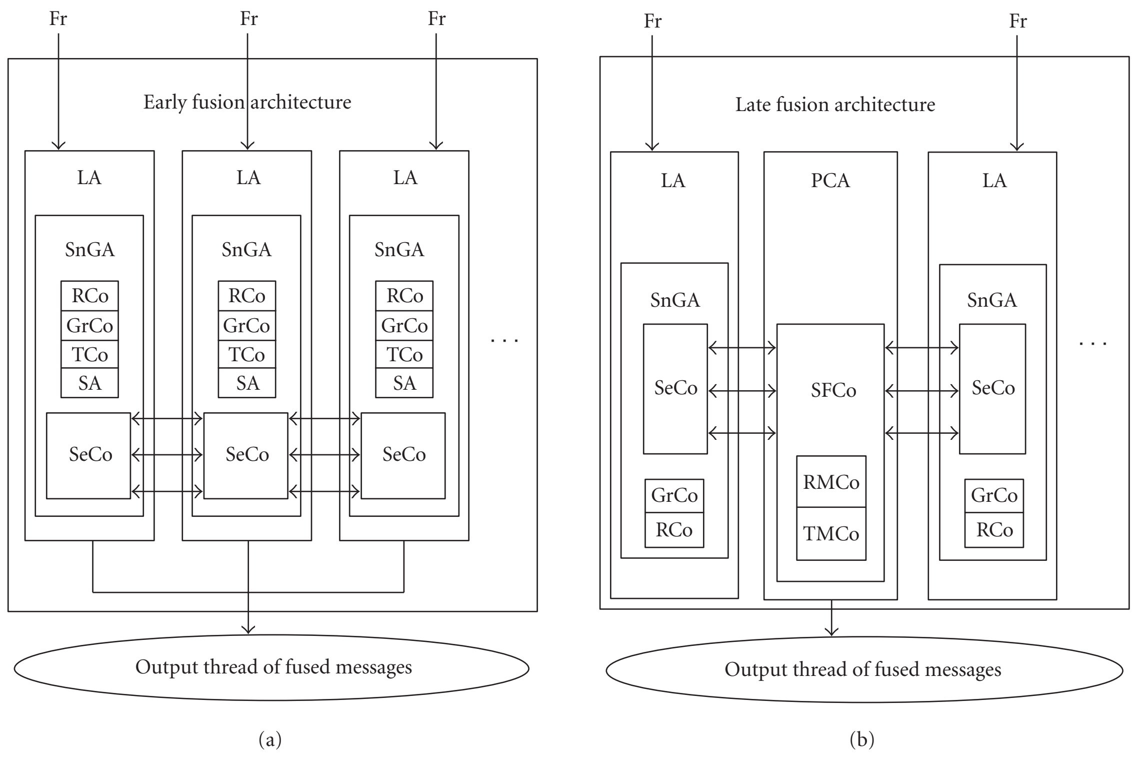


Figure 2: Principles of early and late fusion architectures (A: agent, C: control, Co: component, F: fusion, Fr: fragments of signal, G: generation, Gr: grammar, L: language, M: management, P: parallel, R: redundancy, S: semantic, Se: serialization, Sn: sentence, and T: time). More connections (arrows that indicate the data flow) could be added or removed by the agents to gather fusion information.

The paradigms proposed in this section constitute an important step in the development of multimodal user interface software. Another important phase of the software development for such applications concerns the modeling aspect. Methods like the B-method [30], ATNs [22], or timed CPN [6, 7] can be used to model the multiagent dialog architectures. Section 4 discusses the choice of CPN for modeling an MMDA.

The main drawback of these generic paradigms is that they deal with static architectures. For example, there is no real-time dynamic monitoringor reconfiguration when new media are added. In the next section, we introduce the dynamic reconfiguration of MMDA by components.

3. DYNAMIC ARCHITECTURAL RECONFIGURATION

3.1. Related work

In earlier work on the description and analysis of architectural structures, the focus has been on static architectures. Recently, the need for the specification of the dynamic aspects in addition to the static ones has increased [31, 32]. Several authors have developed approaches on dynamism in architectures, which fulfills the important need to separate dynamic reconfiguration behavior from nonreconfiguration behavior. These approaches increase the reusability of certain system components and simplify our understanding of them. In [33], the authors use an extended specification to introduce dynamism in Wright language. Taylor et al. [34] focus on the addition of a complementary language for expressing modifications and constraints in the message-based C2 architectural style. A similar approach is used in Darwin (see [35]), where a reconfiguration manager controls the required reconfiguration using a scripting language. Many other investigations have addressed the issue of dynamic reconfiguration with respect to the application requirements. For instance, Polylith (see [36]) is a distributed programming environment based on a software bus, which allows structural changes to be made on heterogeneous distributed application systems. In Polylith, the reconfiguration can only occur at special moments in the application source code. The Durra programming environment [37] supports an event-triggered reconfiguration mechanism. Its disadvantage is that the reconfiguration treatment is introduced in the source code of the application and the programmer has to consider all possible execution events, which may trigger a reconfiguration. Argus [38] is another approach based on the transactional operating system but, as a result, the application must comply with a specific programming model. This approach is not suitable for dealing with heterogeneity or interoperability. The Conic approach [39] proposes an application-independent mechanism, where reconfiguration changes affect component interactions. Each reconfiguration action can be fired if and only if components are in a

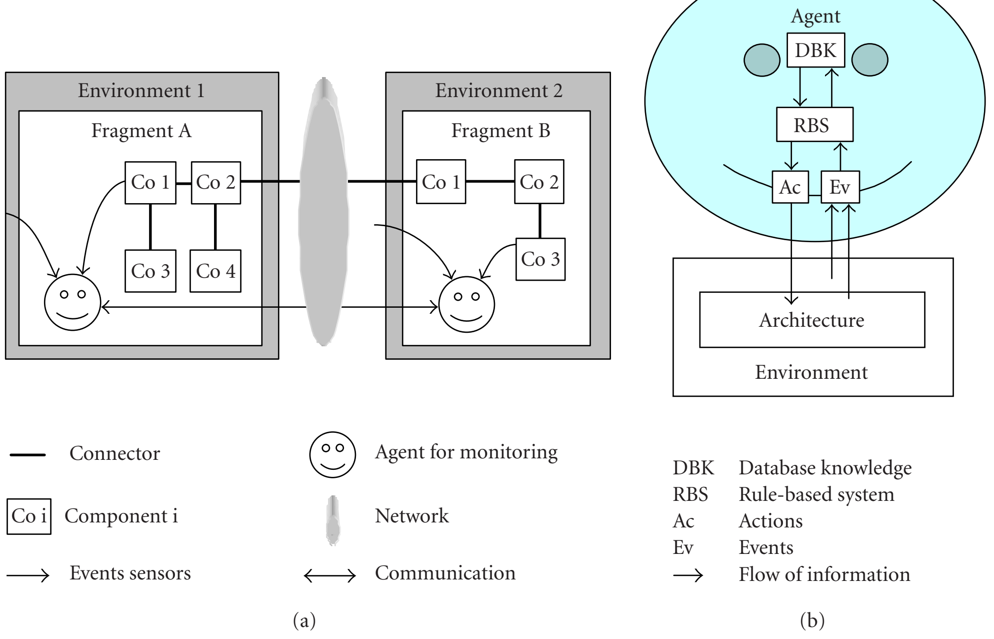


Figure 3: (a) Agent-based architecture. (b) Schematic overview of the agent.

determined state. The implementation tends to block a large part of the application, causing significant disruption. New formal languages are proposed for the specification of mobility features; a short list includes [40, 41]. In [42] in particular, a new experimental infrastructure is used to study two major issues in mobile component systems. The first issue is how to develop and provide a robust mobile component architecture, and the second issue is how to write code in these kinds of systems. This analysis makes it clear that a new architecture permitting dynamic reconfiguration, adaptation, and evolution, while ensuring the integrity of the application, is needed. In the next section, we propose such an architecture based on agent components.

3.2. Reconfiguration services

The proposed idea is to include additional special intelligent agents in the architecture [43]. The agents act autonomously to dynamically adapt the application without requiring an external intervention. Thus, the agents monitor the architecture and perform reconfiguration, evolution, and adaptation at the architectural level, as shown in Figure 3. In the world of distributed computing, the architecture is decomposed into fragments, where the fragments may also be maintained in a distributed environment. The application is then distributed over a number of locations.

We must therefore provide multiagents. Each agent monitors one or several local media and communicates with other agents over a wide-area network for global monitoring of the architecture, as shown in Figure 3. The various components Co i, of one given fragment, correspond to the components of one given LA (or PCA) in one given environment.

In the symbolic representation in Figure 3a, the environments could be different or identical. The complex agent (Figure 3b) is used to handle the reconfiguration at the architectural level. Dynamic adaptations are run-time changes which depend on the execution context. The primitive operations that should be provided by the reconfiguration service are the same in all cases: creation and removal of components, creation and removal of links, and state transfers among components. In addition, requirements are attached to the use of these primitives to perform a reconfiguration, to preserve all architecture constraints and to provide additional safety guarantees.

The major problems that arise in considering the modifiability or maintainability of the architecture are

(i) evaluating the change to determine what properties are affected and what mismatches and inconsistencies may result;
(ii) managing the change to ensure protection of global properties when new components and connections are dynamically added to or deleted from the system.

3.2.1. Agent interface

The interface of each agent is defined not only as the set of actions provided, but also as the required events. For each agent, we attach the event/condition/action rules mechanism in order to react to the architecture and the architectural environment as well as to perform activities. Performing an activity means invoking one or more dynamic method modifications with suitable parameters. The agent can

(i) gather information from the architecture and the environment;

(ii) be triggered by the architecture and the environment in the form of exceptions generated in the application;
(iii) make proper decisions using a rule-based intelligent mechanism;
(iv) communicate with other agent components controlling other relevant aspects of the architecture;
(v) implement some quality aspects of a system together with other agents by systematically controlling inter-component properties such as security, reliability, and so forth;
(vi) perform some action on (and interact with) the architecture to manage the changes required by a modification.

3.2.2. *Rule-based agent*

The agent has a set of rules written in a very primitive notation at a more reasonable level of abstraction. It is useful to distinguish three categories of rules: those describing how the agent reacts to some events, those interconnecting structural dimensions, and those interconnecting functional dimensions (each dimension describes variation in one architectural characteristic or design choice). Values along a dimension correspond to alternative requirements or design choices. The agent keeps track of three different types of states: the world state, the internal state, and the database knowledge. The agent also exhibits two different types of behaviors: internal behaviors and external behaviors. The world state reflects the agent's conception of the current state of the architecture and its environment via its sensors. The world state is updated as a result of interpreted sensory information. The internal state stores the agent's internal variables. The database knowledge defines the flexible agent rules and is accessible only to internal behaviors. The internal behaviors update the agent's internal state based on its current internal state, the world state, and the database knowledge. The external behaviors of the agent refer to the world and internal states, and select the actions. The actions affect the architecture, thus altering the agent's future precepts and predicted world states. External behaviors consider only the world and internal states, without direct access to the database knowledge.

In the case of multiagents, the architecture includes a mechanism providing a basis for orchestrating coordination, which ensures correctness and consistency in the architecture at run time, and ensures that agents will have the ability to communicate, analyze, and generally reason about the modification.

The behavior of an agent is expressed in terms of rules grouped together in the behavior units. Each behavior unit is associated with a specific triggering event type. The receipt of an individual event of this type activates the behavior described in this behavior unit. The event is defined by name and by number of parameters. A rule belongs to exactly one behavior unit and a behavior unit belongs to exactly one class; therefore, the dynamic behavior of each object class modification is modeled as a collection of rules grouped together in behavior units specified for that class and triggered by specific events.

3.2.3. *Agent knowledge*

The agent may capture different kinds of knowledge to evaluate and manage the changes in the architecture. All this knowledge is part of the database knowledge. In the example of a newly added component, the introduction of this new component type is straightforward, as it can usually be wrapped by existing behaviors and new behaviors. The agent focuses only on that part of the architecture which is subject to dynamic reconfiguration.

First, the agent determines the directly related required properties P_i involving the new component, then it

(i) finds all properties P_d related to P_i and their affected design;
(ii) determines all inconsistencies needing to be revisited in the context of P_i and/or P_d properties;
(iii) determines any inconsistency in the newly added components;
(iv) produces the set of components/connectors and relevant properties requiring reevaluation.

4. EXAMPLES

The first example is a Petri net modeling of a static MMDA, including a new generic multiagent Petri-net-modeled architecture. The second shows how to dynamically reconfigure the dialog architecture when new features are added.

4.1. *Example of specification by Petri net modeling*

Small, augmented finite-state machines like ATNs have been used in the multimodal presentation system [44]. These networks easily conceptualize the communication syntax between input and/or output media streams. However, they have limitations when important constraints such as temporal information and stochastic behaviors need to be modeled in fusion protocols. Timed stochastic CPNs offer a more suitable pattern [5, 6, 7] to the design of such constraints in multimodal dialog.

For modeling purposes, each input modality is assimilated into a thread where signal fragments flow. Multimodal inputs are parallel threads corresponding to a changing environment describing different internal states of the system. MASs are also multithreaded: each agent has control of one or several threads. Intelligent agents observe the states of one or several of the threads for which they are designed. Then, the agents execute actions modifying the environment. In the following, it is assumed that the CPN design toolkit [7] and its semantics are known. While a description of CPN modeling is given in Section 4.1.2, we first briefly present, in Section 4.1.1, the augmented transition net principle and its inadequacies relative to CPN modeling.

4.1.1. *Augmented transition net modeling*

The principle of ATNs is depicted in Figure 4.

For ATN modeling purposes, a system can change its current state when actions are executed under certain conditions. Actions and conditions are associated with arcs, while

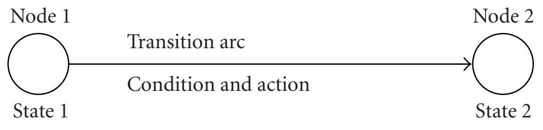


Figure 4: Principle of ATN.

nodes model states. Each node is linked to another (or to the same) node by an arc. Like CPN, ATN can be recursive. In this case, some transition arcs are traversed only if another subordinate network is also traversed until one of its end nodes is reached.

Actually, the evolution of a system depends on conditions related to changing external data which cannot be modeled by the ATN.

Achilles' heel of ATN consists in the absence of a formal associated modeling language for specifying the actions. This leads to the absence of symbols with associated values to model event attributes. In contrast, the CPN metalanguage (CPN ML) [7] is used to perform these specifications.

ATN could therefore be a good tool for modeling the dialog interactions employed in the multimodal fusion as a contextual grammatical syntax (see example in Figure 5). In this case, the management of these interactions is always externally performed by the functional kernel of the application (code in C++, etc.). Consequently, some variables lost in the code indicate the different states of the system, leading to difficulties for each new dialog modification or architectural change. The multimodal interactions need both language (speech language, hand language, written language, etc.) and action (pointing with eye gaze, touching on tactile screen, clicking, etc.) modalities in a single interface combining both anthropomorphic and physical model interactions. Because of its ML, CPN is more suitable for such modeling.

4.1.2. Colored Petri net modeling

4.1.2.1. Definition

The Petri network is a diagram flow of interconnected places or locations (represented by ellipses) and transitions (represented by boxes). A place or location represents a state and a transition represents an action. Labeled arcs connect places to transitions. The CPN is managed by a set of rules (conditions and coded expressions). The rules determine when an activity can occur and specify how its occurrence changes the state of the places by changing their colored marks (while the marks move from place to place). A dynamic paradigm like CPN includes the representation of actual data with clearly defined types and values. The presence of data is the fundamental difference between dynamic and static modeling paradigms. In CPN, each mark is a symbol which can represent all the data types generally available in a computer language: integer, real, string, Boolean, list, tuple, record, and so on. These types are called colorsets. Thus, a CPN is a graphical structure linked to computer language statements. The design CPN toolkit [7] provides this graphical software environment within a programming language (CPN ML) to design and run a CPN.

4.1.2.2. Modeling a multiagent system with CPN

In such a system, each piece of existing information is assigned to a location. These locations contain information about the system state at a given time and this information can change at any time. This MAS is called "distributed" in terms of (see [45])

(i) *functional distribution*, meaning a separation of responsibilities in which different tasks in the system are assigned to certain agents;

(ii) *spatial distribution*, meaning that the system contains multiple places or locations (which can be real or virtual).

A virtual location is an imaginary location which already contains observable information or information can be placed in it, but there is no assumption of physical information linked to it. The set of colored marks in all places (locations) before an occurrence of the CPN is equivalent to an observation sequence of an MAS. For the MMDA case, each mark is a symbol which could represent signal fragments (pronounced words, mouse clicks, hand gestures, facial attitudes, lip movements, etc.), serialized or associated fragments (comprehensive sentences or commands), or simply a variable.

A transition can model an agent which generates observable values. Multiple agents can observe a location. The observation function of an agent is simply modeled by input arc inscriptions and also by the conditions in each transition guard (symbolized by [conditions] under a transition). These functions represent *facet A* (Figure 6) of agents. Input arc inscriptions specify data which must exist for an activity to occur. When a transition is fired (an activity occurs), a mark is removed from the input places and the activity can modify the data associated with the marks (or its colors), thereby changing the state of the system (by adding a mark in at least one output place). If there are colorset modifications to perform, they are executed by a program associated with the transition (and specified by the output arc label). The program is written in CPN ML inside a dashed-line box (not connected to an arc and close to the transition concerned). The symbol **c** specifies [7] that a code is attached to the transition, as shown in Figure 7. Therefore, each agent generates data for at least one output location and observes at least one input location.

If no code is associated with the transition, output arc inscriptions specify data which will be produced if an activity occurs. The action functions of the agent are modeled by the transition activities and constitute *facet E* of the agent (Figure 6).

Hierarchy is another important property of CPN modeling. The symbol HS in a transition means [7] that this is a hierarchical substitution transition (Figure 7). It is replaced by another subordinate CPN. Therefore, the input (symbols [7] P In) and output (symbols [7] P Out) ports of the subordinate CPN also correspond to the subordinate architecture ports in the hierarchy. As shown in Figure 7, each transition and each place is identified by its name (written on it). The

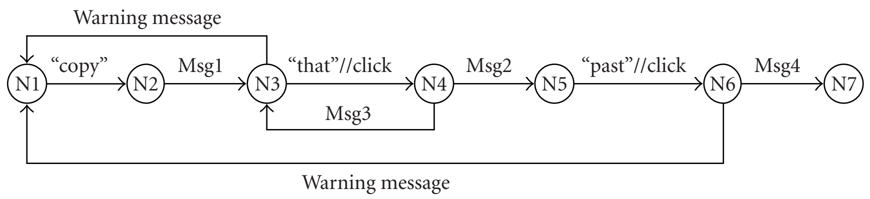


Figure 5: Example of modeling semantic speech and mouse-clicking an interaction message: ("copy" + ("that"//click) + ("paste"//click)). Symbols + and // stand for serial and concurrent messages in time. All output arcs are labeled with messages presented in output modalities, while input ones correspond to user actions. The warning message is used to inform, ask, or warn the user when he stops interacting with the system. (Msg: output message of the system, N: node representing a state of the system.)

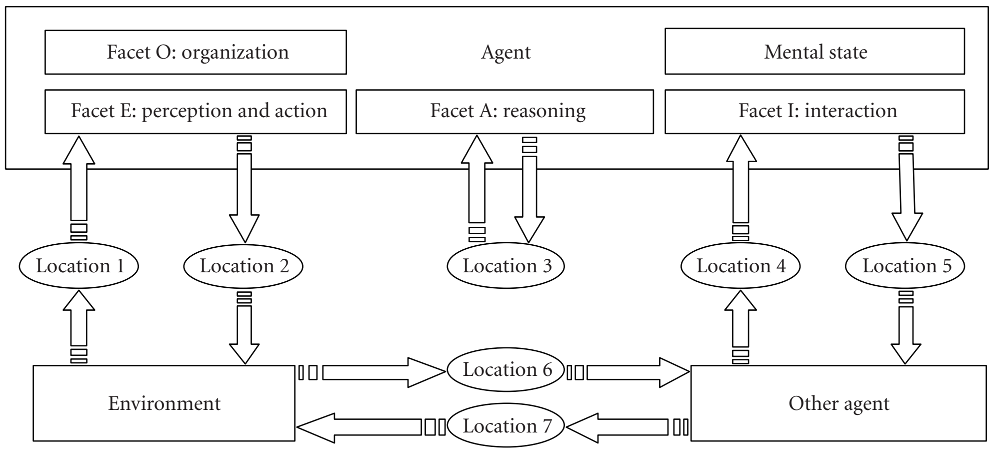


Figure 6: AEIO facets within an agent. The locations represent states, resources, or threads containing data. An output arrow from a location to an agent gives an observation of the data, while an input arrow leads to generation of data.

symbol FG in identical places indicates that the places are "global fusion" places [7]. These identical places are simply a unique resource (or location) shared over the net by a simple graphical artifact: the representation of the place and its elements is replicated with the symbol FG. All these framed symbols—P In, P Out, HS, FG, and **c**—are provided and imposed by the syntax of the visual programming toolkit of design CPN [7].

To summarize, modeling an MAS can be based on four dimensions (Figure 6), which are agent (A), environment (E), interaction (I), and organization (O).

(i) *Facet A* indicates all the internal reasoning functionalities of the agent.

(ii) *Facet E* gathers the functionalities related to the capacities of perception and action of the agent in the environment.

(iii) *Facet I* gathers the functionalities of interaction of the agent with the other agents (interpretation of the primitives of the communication language, management of the interaction, and the conversation protocols). The actual structure of the CPN, where each transition can model a global agent decomposed in components distributed in a subordinate CPN (within its initial values of variables and its procedures), models this facet.

(iv) *Facet O* can be the most difficult to obtain with CPN. It concerns the functions and the representations related to the capacities of structuring and managing the relations between the agents to make dynamic architectural changes.

Sequential operation is not typical of real systems. Systems performing many operations and/or dealing with many entities usually do more than one thing at a time. Activities happening at the same time are called *concurrent activities*. A system containing such activities is called a concurrent system. CPN easily models this concept of parallel processes.

In order to take time into account, CPN is timed and provides a way to represent and manipulate time by a simple methodology based on four characteristics.

(1) A mark in a place can have a number associated with it, called a *time stamp*. Such a timed mark has its *timed colorset*.

(2) The simulator contains a counter called the *clock*. The clock is just a number (integer or real number) the current value of which is the current time.

(3) A timed mark is not available for any purpose whatsoever, unless the clock time is greater than or equal to the mark's time stamp.

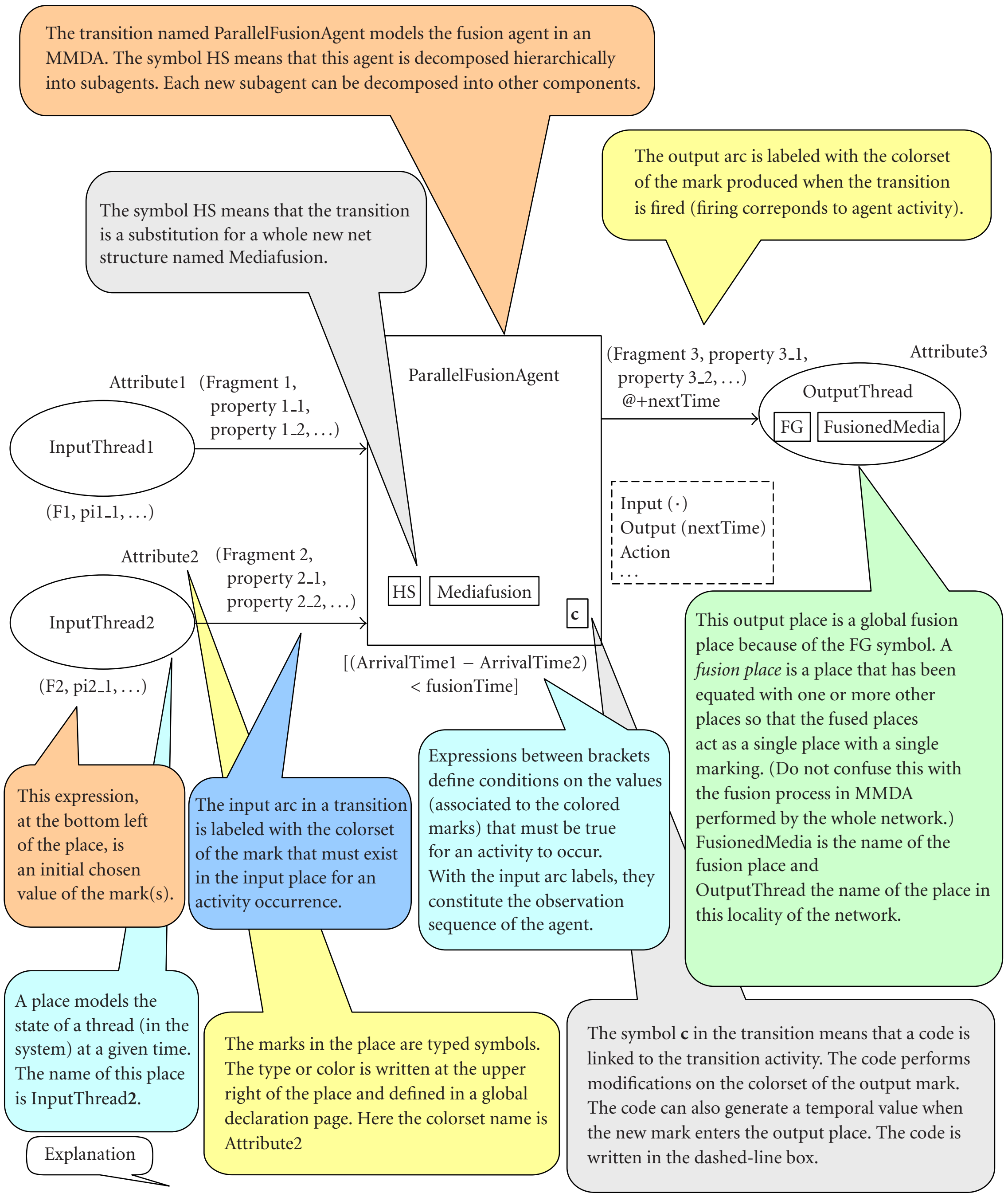

The transition named ParallelFusionAgent models the fusion agent in an MMDA. The symbol HS means that this agent is decomposed hierarchically into subagents. Each new subagent can be decomposed into other components.
The symbol HS means that the transition is a substitution for a whole new net structure named Mediafusion.
The output arc is labeled with the colorset of the mark produced when the transition is fired (firing correponds to agent activity).
Attribute1 (Fragment 1, property 1_1, property 1_2, . . .)
InputThread1
(F1, pi1_1, . . .)
ParallelFusionAgent
(Fragment 3, property 3_1, property 3_2, . . .) @+nextTime
Attribute3
OutputThread
FG
FusionedMedia
Input (·) Output (nextTime) Action . . .
Attribute2 (Fragment 2, property 2_1, property 2_2, . . .)
InputThread2
(F2, pi2_1, . . .)
HS
Mediafusion
c
[(ArrivalTime1 − ArrivalTime2) < fusionTime]
This output place is a global fusion place because of the FG symbol. A *fusion place* is a place that has been equated with one or more other places so that the fused places act as a single place with a single marking. (Do not confuse this with the fusion process in MMDA performed by the whole network.) FusionedMedia is the name of the fusion place and OutputThread the name of the place in this locality of the network.
This expression, at the bottom left of the place, is an initial chosen value of the mark(s).
The input arc in a transition is labeled with the colorset of the mark that must exist in the input place for an activity occurrence.
Expressions between brackets define conditions on the values (associated to the colored marks) that must be true for an activity to occur. With the input arc labels, they constitute the observation sequence of the agent.
A place models the state of a thread (in the system) at a given time. The name of this place is InputThread**2**.
The marks in the place are typed symbols. The type or color is written at the upper right of the place and defined in a global declaration page. Here the colorset name is Attribute2
The symbol **c** in the transition means that a code is linked to the transition activity. The code performs modifications on the colorset of the output mark. The code can also generate a temporal value when the new mark enters the output place. The code is written in the dashed-line box.
Explanation

Figure 7: CPN modeling principles of an agent in MMDA.

(4) When there are no enabled transitions (but there would be if the clock had a greater value), the simulator alters the clock incrementally by the minimum amount necessary to enable at least one transition.

These four characteristics give simulated time the dimension that has exactly the properties needed to model delayed activities. Figure 7 shows how the transition activity can generate an output-delayed mark. This mark can reach the place OutputThread only after a time (equal to nextTime). The value of nextTime is calculated by the code associated with the transition. With all these possibilities, CPN provides an extremely effective dynamic paradigm for modeling an MAS like the multimedia multimodal fusion engine.

4.1.2.3. The generic CPN-modeled MMDA chosen

The generic multiagent architecture chosen for the multimedia multimodal fusion engine within CPN modeling appears in Figure 8. It is an intermediary one between the late and early fusion architectures depicted in Figure 2. The main

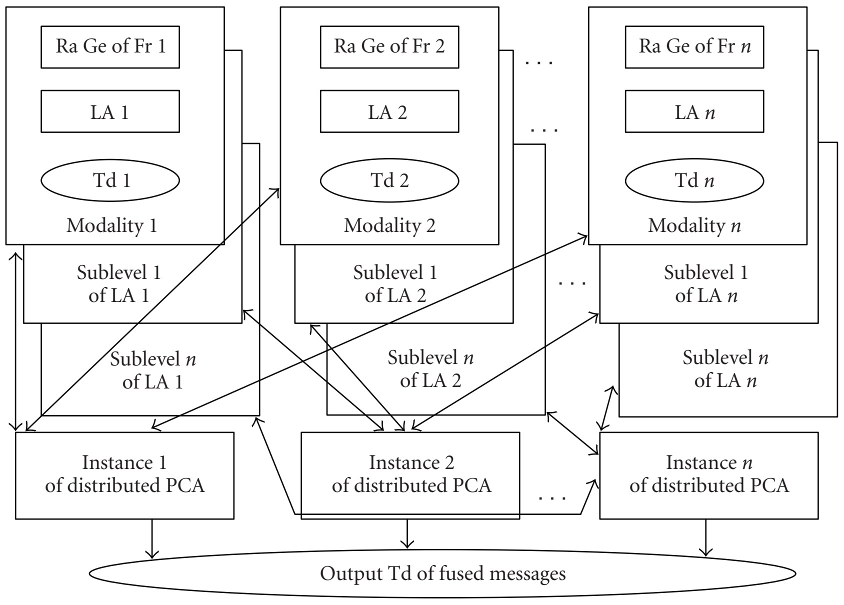


FIGURE 8: Generic multiagent scaled architecture of the fusion engine for CPN modeling purposes (A: agent, C: control, Fr: fragments of signal, Ge: generator, L: language, P: parallel, Ra: random, and Td: thread). The arrows indicate the information and data flows.

features appearing in the proposed generic CPN-modeled architecture are summarized in four points.

(i) *Distributed architecture*. CPN modeling offers the possibility of distributing PCA over the architecture, as shown in Figure 8. Each instance of the PCA has its facets of action, perception, and interaction, depending on its contextual position in the network.

(ii) *Scalable architecture*. The architecture has the ability to sustain a growing load when new modalities are added. The possibility of decomposing each LA into sublevels leads to a model which can assist in code generation in a computer language used in the final implementation of the system (hierarchy, heritage, etc.) and also gives the option of reducing the perception mechanisms of the agents and of spreading them out over the entire architecture.

(iii) *Parallel architecture*. Parallelism provides the possibility of running the application with each LA processed in a separate parallel hardware. It is also possible to easily activate or inhibit an LA (in the case of dynamic architectural reconfiguration) without perturbing the global running of the application.

(iv) *Pipelined architecture*. with several input and internal data streams and one output data stream, it becomes easy to test and follow the evolution of this multimedia multimodal architecture with a view to error avoidance. Instances of PCA can handle the diagnostics of the architecture to prevent *system-centered* errors, as shown in the next section.

4.1.2.4. Error avoidance in the proposed CPN-modeled architecture

Error avoidance can be considered from a user-centered and a system-centered point of view.

User-centered error avoidance

(i) The user will opt for the input mode that he or she considers will produce fewer errors for a particular lexical content when the user has a choice between two equivalent modalities (for example, switching from speech to pen strokes to communicate a last name).

(ii) The interactive multimodal user language is simplified. This leads to a decreasing complexity of natural language processing, thereby reducing recognition errors.

(iii) The user has a natural tendency to switch modes after a system recognition error, which will lead to fewer errors.

System-centered error avoidance

(i) Within a parallel, distributed, pipelined CPN-modeled architecture such as this, it is easy to manage errors at different sublevels of the fusion process. Each signal (or fragment of signal) thread could be checked. The error checking can be performed directly by the distributed PCA or by another agent doubling up the distributed PCA. Under contextual, temporal, syntactical, and semantic conditions, this agent purges the thread from signals (or fragments of signal) which do not correspond to a monomodal action or a multimodal command. Its complexity could be equal to or beyond the complexity of the PCA. This agent is also responsible for warning the user when a fusion or a command is aborted and/or when the system does not recognize the user's messages.

(ii) With the proposed parallel architecture, it is also possible to use two semantically rich input modes, supporting a mutual reduction in ambiguity of signals [21]. For example, in speech recognition associated with a

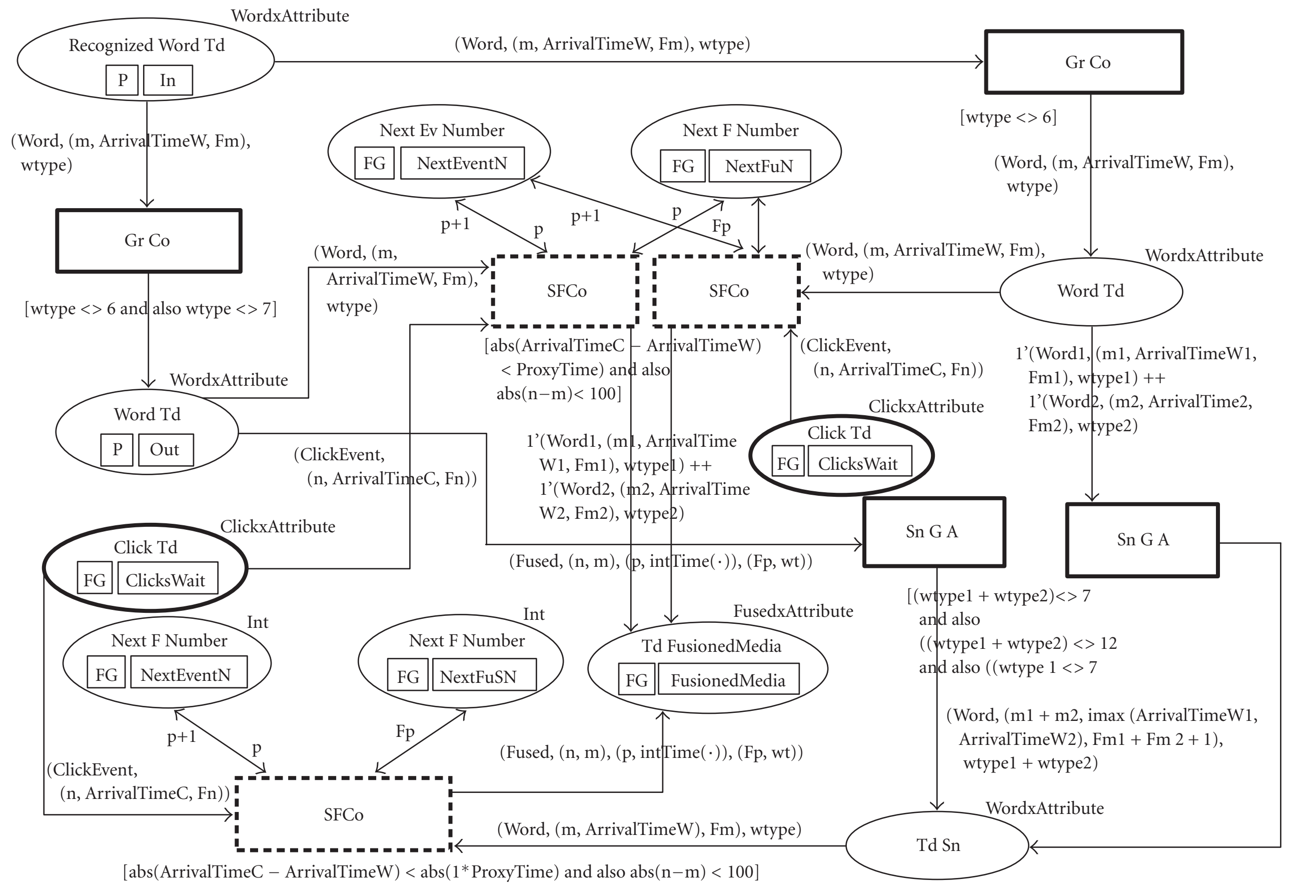

WordxAttribute
Recognized Word Td
P
In
(Word, (m, ArrivalTimeW, Fm), wtype)
Gr Co
[wtype <> 6]
(Word, (m, ArrivalTimeW, Fm), wtype)
Next Ev Number
FG
NextEventN
Next F Number
FG
NextFuN
p+1
p
p+1
p
Fp
Gr Co
(Word, (m, ArrivalTimeW, Fm), wtype)
SFCo
SFCo
(Word, (m, ArrivalTimeW, Fm), wtype)
WordxAttribute
Word Td
[wtype <> 6 and also wtype <> 7]
[abs(ArrivalTimeC − ArrivalTimeW) < ProxyTime) and also abs(n−m)< 100]
(ClickEvent, (n, ArrivalTimeC, Fn))
1'(Word1, (m1, ArrivalTimeW1, Fm1), wtype1) ++ 1'(Word2, (m2, ArrivalTime2, Fm2), wtype2)
WordxAttribute
Word Td
P
Out
ClickxAttribute
Click Td
FG
ClicksWait
(ClickEvent, (n, ArrivalTimeC, Fn))
1'(Word1, (m1, ArrivalTime W1, Fm1), wtype1) ++ 1'(Word2, (m2, ArrivalTime W2, Fm2), wtype2)
ClickxAttribute
Sn G A
Sn G A
Click Td
FG
ClicksWait
(Fused, (n, m), (p, intTime(·)), (Fp, wt))
Int
Int
FusedxAttribute
[(wtype1 + wtype2)<> 7 and also ((wtype1 + wtype2) <> 12 and also ((wtype 1 <> 7
Next F Number
FG
NextEventN
Next F Number
FG
NextFuSN
Td FusionedMedia
FG
FusionedMedia
(Word, (m1 + m2, imax (ArrivalTimeW1, ArrivalTimeW2), Fm1 + Fm 2 + 1), wtype1 + wtype2)
p+1
p
Fp
(Fused, (n, m), (p, intTime(·)), (Fp, wt))
(ClickEvent, (n, ArrivalTimeC, Fn))
WordxAttribute
SFCo
(Word, (m, ArrivalTimeW), Fm), wtype)
Td Sn
[abs(ArrivalTimeC − ArrivalTimeW) < abs(1*ProxyTime) and also abs(n−m) < 100]

Figure 9: Sublevel of the bimodal fusion dialog: the functions used, intTime(·) and imax(·), return the current time and maximum integer value, respectively (A: agent, Co: component, F: fusion, G: generation, Gr: grammar, L: language, S: semantic, Sn: sentence, and Td: thread).

pen stroke or lip movement recognition device, each input mode should have a complementary mode in the architecture. Also, each of the two complementary modes has to provide duplicated functionality in order to offer the user two equivalent ways to achieve his or her goals. If the user proceeds with the two modes in parallel, this could lead to an error avoidance of between 19% and 40% in comparison with the monomodal recognition system [21]. The performance improvement is the direct result of the reduction in ambiguity between signals that can occur in parallel threads because of the fact that each scaled LA provides a context for interpreting the other LAs during integration. The time of integration (temporal window in which the system waits for a signal equivalent to the one that has already arrived but has not been recognized with certainty) is an important criterion. The contextual information is used by the distributed PCA to confirm to the user that a command has been executed. This confirmation is sent to an output modality only if the agent needs user corroboration to make its prognostication. The system is able to decide by itself under criteria associated with scenarios (decision trees) where probability, grammar, and semantics play an important role. The structure of the proposed CPN-modeled architecture is very suitable for such system-centered error avoidance.

4.1.3. *Example of an engine fusion modeled by CPN*

A typical example of a distributed architecture for fusion, using the paradigm in Figure 8, is presented in Figures 9 and 10. The "copy and paste" fusion engine architecture chosen involves a high-level LA, for speech modality, linked, by a distributed PCA, to a rudimentary mouse-clicking LA (thread of clicks). The PCA performs the semantic fusion between speech and mouse-clicking via two levels (Figures 9 and 10). Tables 2 and 3 give the vocabulary, used by the speech LA, and the basic corresponding grammar. Each word has a label which is used in the CPN design.

In the following, a few regular symbolic expressions are used to represent semantic elements. These expressions use the arrow operator for sequential concatenation in the time domain. For the chosen example, in the semantic expression

$$(\text{word }1 \longrightarrow \text{word }2),$$

word 1 is simply followed by (or contiguous to) word 2. In Table 3, the codes (last column) are obtained simply by summing the word labels of each semantic code. The obtained codes give information used by the speech LA for serial constructions of sentences in the network.

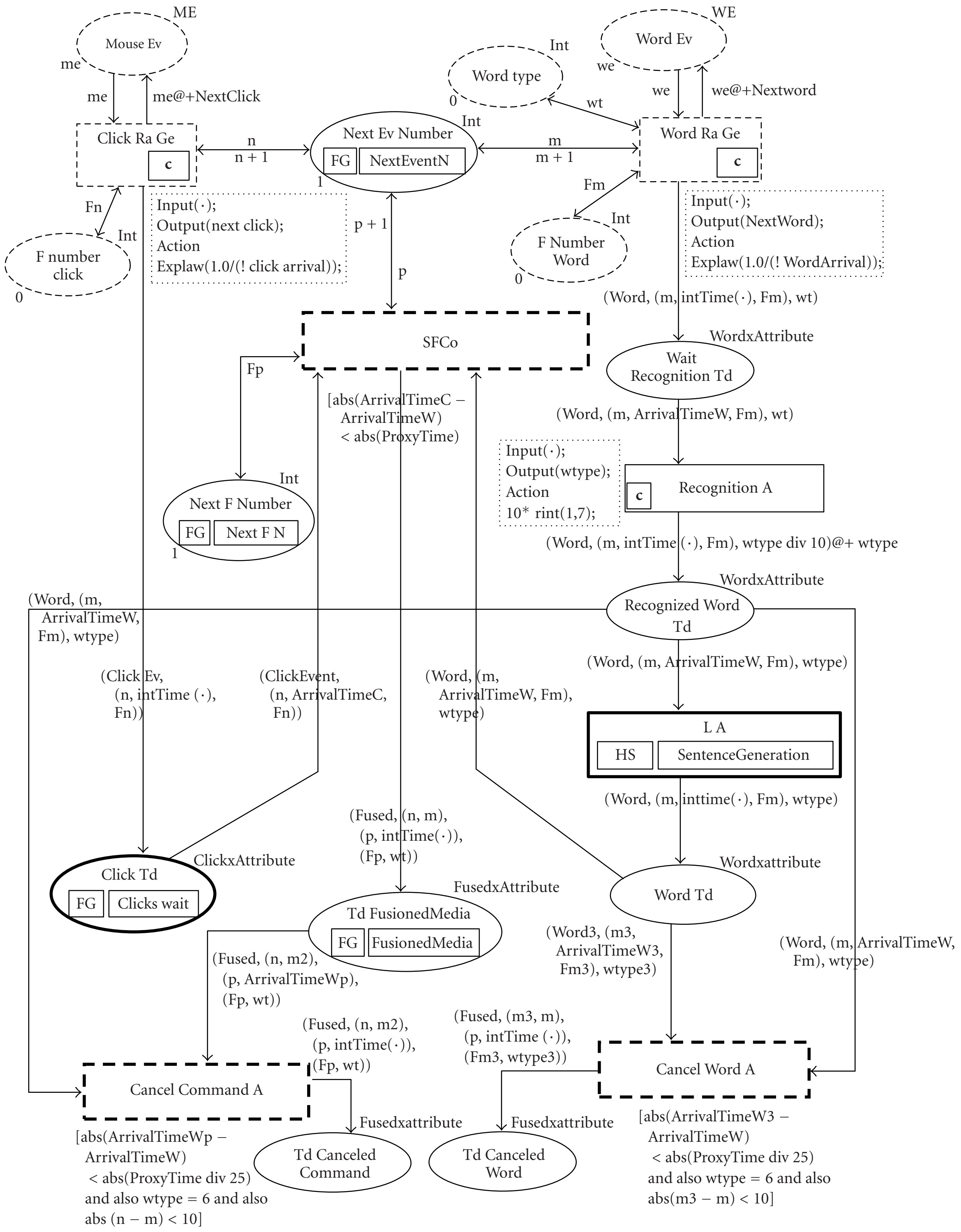

ME
Mouse Ev
me
me
me@+NextClick
Click Ra Ge
c
n
n + 1
Next Ev Number
Int
FG
NextEventN
1
m
m + 1
Word Ev
WE
we
we
we@+Nextword
Word type
Int
0
wt
Word Ra Ge
c
Fm
F Number Word
Int
0
Fn
F number click
Int
0
Input(·);
Output(next click);
Action
Explaw(1.0/(! click arrival));
p + 1
p
Input(·);
Output(NextWord);
Action
Explaw(1.0/(! WordArrival));
(Word, (m, intTime(·), Fm), wt)
SFCo
Fp
WordxAttribute
Wait Recognition Td
[abs(ArrivalTimeC − ArrivalTimeW) < abs(ProxyTime)
(Word, (m, ArrivalTimeW, Fm), wt)
Input(·);
Output(wtype);
Action
10* rint(1,7);
Recognition A
c
Next F Number
Int
FG
Next F N
1
(Word, (m, intTime(·), Fm), wtype div 10)@+ wtype
WordxAttribute
Recognized Word Td
(Word, (m, ArrivalTimeW, Fm), wtype)
(Word, (m, ArrivalTimeW, Fm), wtype)
(Click Ev, (n, intTime (·), Fn))
(ClickEvent, (n, ArrivalTimeC, Fn))
(Word, (m, ArrivalTimeW, Fm), wtype)
L A
HS
SentenceGeneration
(Word, (m, inttime(·), Fm), wtype)
(Fused, (n, m), (p, intTime(·)), (Fp, wt))
ClickxAttribute
Click Td
FG
Clicks wait
Wordxattribute
Word Td
FusedxAttribute
Td FusionedMedia
FG
FusionedMedia
(Fused, (n, m2), (p, ArrivalTimeWp), (Fp, wt))
(Word3, (m3, ArrivalTimeW3, Fm3), wtype3)
(Word, (m, ArrivalTimeW, Fm), wtype)
(Fused, (n, m2), (p, intTime(·)), (Fp, wt))
(Fused, (m3, m), (p, intTime (·)), (Fm3, wtype3))
Cancel Command A
Cancel Word A
Fusedxattribute
Fusedxattribute
Td Canceled Command
Td Canceled Word
[abs(ArrivalTimeWp − ArrivalTimeW) < abs(ProxyTime div 25) and also wtype = 6 and also abs (n − m) < 10]
[abs(ArrivalTimeW3 − ArrivalTimeW) < abs(ProxyTime div 25) and also wtype = 6 and also abs(m3 − m) < 10]

FIGURE 10: Bimodal fusion dialog. The functions intTime(·), rint(·), and Explaw(·) return current time, discrete uniform, and exponential distribution, respectively (A: agent, Co: component, Ev: event, F: fusion, Ge: generator, L: language, Ra: random, S: semantic, and Td: thread).

Table 2: Vocabulary.

Word	Word label	Word	Word label
Open	1	Paste	5
Close	2	Cancel	6
Delete	3	That	7
Copy	4		

The word "cancel" is a command which automatically cancels the last action among the authorized sentences. Therefore, if the user says one of the words labeled in the set $\{1, 2, 3, 4, 5\}$ just after "cancel," the time proximity between the two words is the decision criterion for suppressing the second word or taking it as a next command. For the proposed architecture, both scenarios are processed. The multimodal dialog gives, for each sentence, a set of possible redundant fusions. The symbol // models these concurrent associations in regular expressions.

For example, depending upon temporal information, the first command given in Table 3 is an element of the following semantic fusion set:

$$\{(\text{click} \longrightarrow \text{open} \longrightarrow \text{that}); (\text{open} \longrightarrow \text{click});$$
$$(\text{click} \longrightarrow \text{open}); (\text{click//open}); ((\text{click//open}) \longrightarrow \text{that});$$
$$(\text{click//}(\text{open} \longrightarrow \text{that}))\}.$$

This semantic set includes the grammatical sentences corresponding to the command "open object." Words, temporally isolated and labeled in the set $\{1, 2, 3, 4, 7\}$, are not considered by the PCA. The remaining fusion entities, like ((close → open)//click), (click//(delete → open)), and so forth, or isolated clicks, are also ignored by the system. Thus, some errors made by the user are avoided by the model. The whole sets constitute the semantic knowledge.

The associated CPN in Figures 9 and 10 uses two random generators to design the arrival time of the input media events (mouse clicks and words). The random (Ra) generators (Ge) are drawn at the top of Figure 9 with dashed nonbold lines, and both are modeled with the transitions named "Ra Ge Click" and "Ra Ge Word." The interarrival time between two pronounced words, as well as the time between two consecutive "clicks," is exponentially distributed. Events (like words and clicks) are generated or arrive at two different threads (the places "Click Td" and "Word Td"). The time between two click (resp., word) arrivals has a mean = ClickArrival (resp., = WordArrival). The interarrival time between two click (resp., word) events has an exponential distribution with parameter $\mathbf{r}$ = 1/ClickArrival (resp., 1/WordArrival).

The interarrival time follows an exponential law for the words and also for the clicks. If the time proximity between a word event and a click event is below the variable ProxyTime and if these two events verify the grammatical and semantic conditions (given between brackets under the semantic fusion component modeled by the transition named "SFCo"), then these two events are fused into one command.

The transitions drawn with bold dashed lines model the PCA components distributed over the network. Transitions, with bold lines, model the speech LA components in Figures 9 and 10. The mouse click LA is reduced to a simple thread (Td), "Click Td", where symbols flow. The transition "Recognition A" (Figure 9) assigns a random label "wtype" to each word present in the place "Recognized Word Td." This random assignment does not model a real flowing speech because the automatic modeling of user speech is outside the scope of this paper. However, it is sufficient to model times of recognition.

One of the main focuses in this paper is how to use timed semantic knowledge to achieve a multimodal fusion. Therefore, the network in Figure 9 describes interactions at a sublevel of the network in Figure 10. More precisely, Figure 9 models the interactions of the speech LA (Figure 10) grammatical components (Gr Co) and the sentence generation agent (Sn G A in Figure 9). It also models different instances of the SFCo of the PCA distributed on the hierarchical (here, two-level) network architecture.

4.1.4. Simulation results

Figures 11a and 11b show the simulation results for WordArrival = ClickArrival = 5000 milliseconds and ProxyTime = 10000 milliseconds.

Figure 11a presents the number of fusions achieved in the time period (or the number of marks in the "FusionedMedia" place of the CPN). In the same way, a command can be canceled if the user says the word "cancel" just after a command has been carried out (the proximity time between the two events, the command and the word "cancel," is chosen below (ProxyTime/25)). Figure 11b shows the resulting canceled commands in the time period (or the number of marks arrived at in the place "Canceled Command"). Figures 11a and 11b are obtained after simulation of the network (Figures 9 and 10).

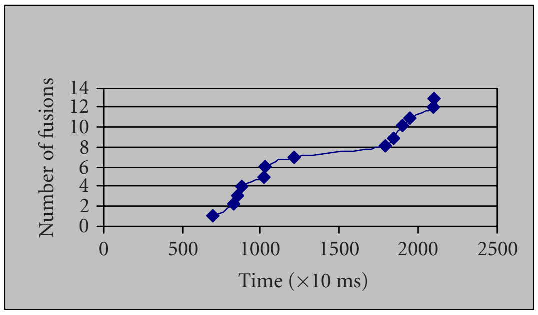
The results in Figure 11 quantify perceivable behavior of the architecture for random arrival time of inputs. This behavior depends on a temporal proximity criterion. These results could vary according to the value of the "ProxyTime." Adjustment of this value should take into account the mean temporal behavior of users. This is done by a pertinent fine tuning of the random generators with the function Explaw$(\cdot)$. It should also consider processing time, which is modeled by the value "wtype" returned by the transition program "Recognition System." The example of this section shows that the fusion engine works and performs semantic fusion (by combining the results of commands to derive new results) as well as syntactic ones (by combining data to obtain a complete command).

4.2. Dynamic reconfiguration example

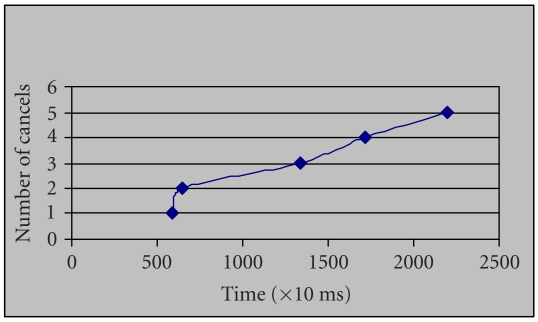
In this section, we describe, in brief, an application which is used to add new media to an MMDA while the application is running. The system is briefly presented here. In the initial architecture (Figure 12a), for simplicity, each con j (connection j between two components via a connector) corresponds to the representation in Figure 12b. The initial system

Table 3: Grammar of authorized sentences (the last two columns are codes used in the CPN).

Set of sentences	Command meaning	Set of corresponding semantic codes	Set of corresponding codes
{(open → that); (open)}	Open object	{(1 → 7); (1)}	{(8); (1)}
{(close → that); (close)}	Close object	{(2 → 7); (2)}	{(9); (2)}
{(delete → that); (delete)}	Delete object	{(3 → 7); (3)}	{(10); (3)}
{(paste)}	Paste last copied object	{(5)}	{(5)}
{(copy → that); (copy)}	Copy object	{(4 → 7); (4)}	{(11); (4)}
{(cancel)}	Cancel last command	{(6)}	{(6)}



(a)



(b)

Figure 11: (a) Achieved semantic fusions. (b) Canceled command.

is composed of two media (see Section 4.1). The fusion is performed by a PCA. According to the application requirements, efficiency in time behavior is more important than the other quality attributes. In order to improve this quality attribute, the agents must perform the reconfiguration atomically and gradually. The adaptation must be conducted in a safe manner to ensure the integrity of the global architecture during running time. On reception of the event (new modality used), the agent will state the following strategy, which consists of applying some rule operations. The architectural reconfiguration agent

(i) adds a new data collector and PCA components (for fusion purposes) to the application;
(ii) creates a new modality database;
(iii) makes decisions on deleting each old connection con j by testing whether or not it is passive (there are no transit data between the two components related to this connection).

If a connection is passive, the agent deletes it and activates the new one just created then transfers the state of the corresponding connector to the new connector.

The agent does this in real time until the desired new application (Figure 12c) replaces the initial one. The new modality is a media device called the eye-gaze response interface computer aid. It is specially adapted with imaging hardware and software. The user interface accepts input directly from the human eye. Menu options are displayed at different positions on the computer monitor.

By simply looking at a given position, the corresponding menu option is invoked. In this way, a disabled user can interact with the computer, run communications and other applications software, and manage peripheral devices. An agent will be associated with this device to control its hardware (device management agent (DMA)). Other agents manage software configuration and also implement the eye-gaze position detection algorithm.

5. THE NOVELTY OF OUR APPROACH

The novelty of our approach is demonstrated by the proposed multiagent paradigms of the generic CPN-modeled MMDA, and also with the dynamic reconfiguration of the MMDA at the architectural level. To support the novel aspect of the approach, this section describes the three main characteristics of the proposed architecture through a multimodal interface software application called Interact Software 1.0 (IAS). IAS is dedicated to use by disabled individuals (particularly those who are paralyzed, like hemiplegics, quadriplegics, etc.), and is based on a fusion engine architecture modeled with a design CPN, as shown in the previous sections. The developed application offers a Web browser interface running on the MS Windows 9x/ME/NT/2000 XP OS platform for portability reasons and using multithreading and other advantages of this 32-bit environment. Eye tracker equipment is used to detect gaze position on a screen monitor [46]. Calibration of the eye-gaze material for each user is necessary to achieve accurate gaze position tracking. The

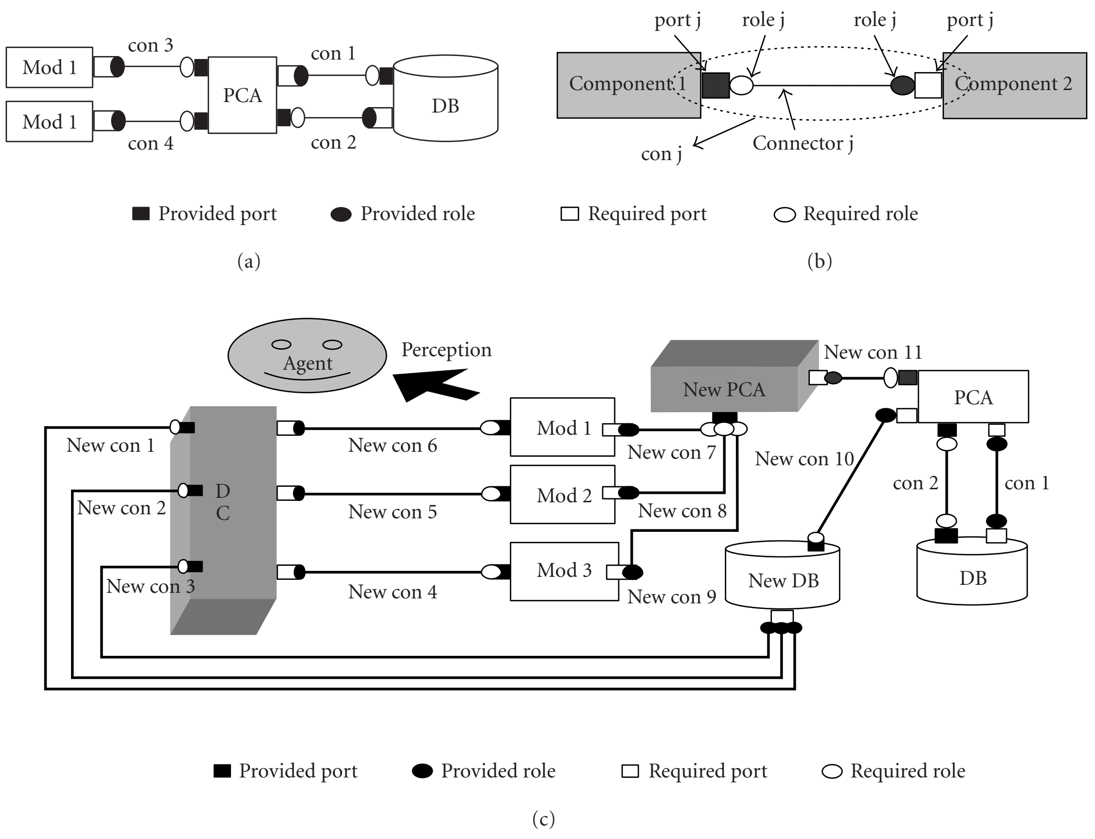


FIGURE 12: (a) The initial architecture. (b) Connection j (con j). (c) The final desired architecture. (Mod: modality, Cl: collector, D: data, DB: database, P: parallel, C: control, A: agent, and con: connection.)

calibration needs to be performed only once, and is saved in a database for each user. The calibration data are therefore set for individual users who are identified by the IAS via a password protocol. The eye-gaze position information on the screen is used in real time to move the mouse pointer. The software used for voice recognition and vocal synthesis is Microsoft Speech API: SDK4.0. A set of vocal commands (a word or a word combination) is predefined for this purpose. In IAS developed with C++, the input modalities are voice recognition, eye-gaze position detection, a mouse, and/or a tactile screen. The keyboard can also be used, and a virtual vocal keyboard and a virtual vocal mouse-clicking device are also available. The output modalities are voice synthesis and monitor-screen display. It should be noted that the screen device is involved in both input and output modes.

5.1. *Flexibility of the architecture*

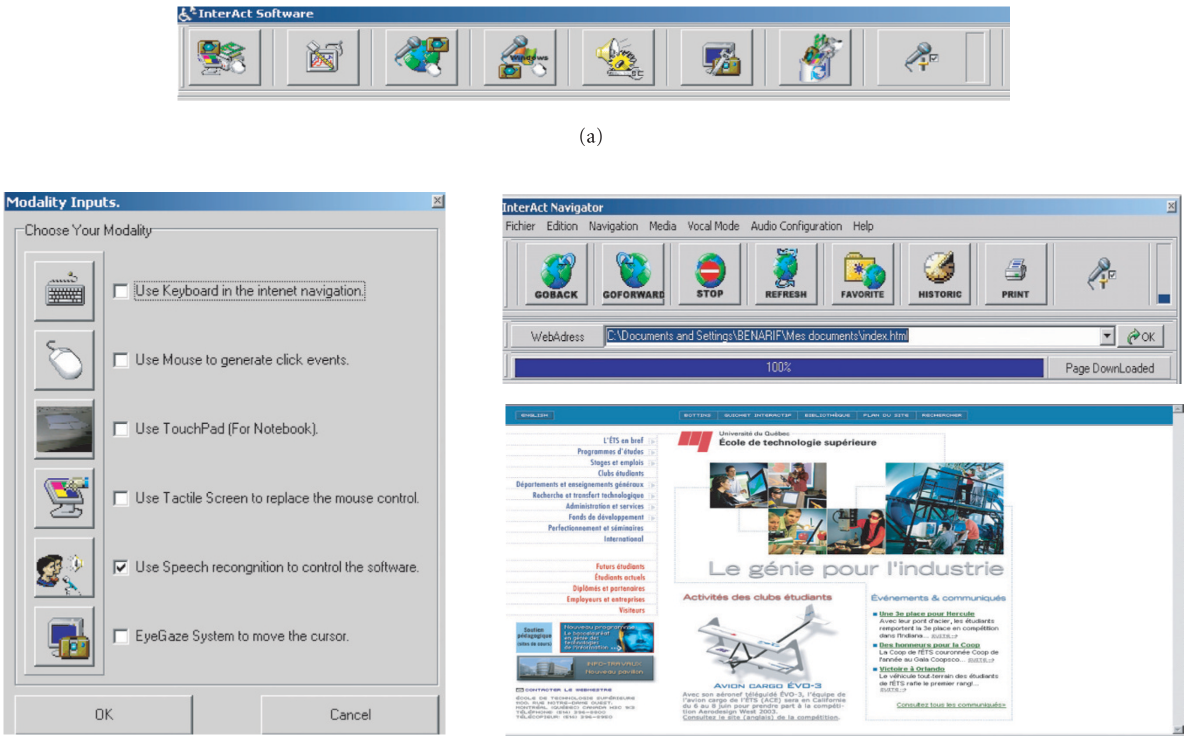
Several characteristics confer flexibility onto the architecture. Each LA has an "interpreter" component which has two functions:

(i) interpretation of the signals coming from the input devices;
(ii) transformation of these signals into events which can be understood by the PCA.

With each specialized DMA, there is an interpreter component, depending on the nature of the mode. This property gives the architecture its generic characteristics in terms of flexibility: the user can change, add, or abandon the modalities while the application is running. This is possible because each new modality is automatically adapted to the fusion engine. Switching between modalities in the architecture is easy to do in run-time mode. IAS also gives the user the ability to configure the fusion time, and in fact this is done automatically, either by default after the interface has been used or directly via the interface.

5.2. *Dynamic aspects*

The vision of ubiquitous computing, which was introduced in the early '90s [47], is now giving rise to extensive research [48, 49, 50, 51, 52]. While ubiquitous computing could have been considered to be quite futuristic in those days, the combined effect of advances in the areas of hardware and network technologies and of the universal use of Internet-based technologies and wireless phones makes it almost a reality. The vision is now termed "ambient intelligence" or "pervasive computing" to emphasize that it does not rely solely on its ubiquitous nature (i.e., the useful, pleasant, and unobtrusive presence of computing devices everywhere), but also on ubiquitous networking (i.e., access to networks and computing facilities everywhere) and on intelligent, "aware"



(a)

☐ Use Keyboard in the internet navigation.

☐ Use Mouse to generate click events.

☐ Use TouchPad (For Notebook).

☐ Use Tactile Screen to replace the mouse control.

☑ Use Speech recognition to control the software.

☐ EyeGaze System to move the cursor.

WebAdress: C:\Documents and Settings\BENARIF\Mes documents\index.html

(b) (c)

FIGURE 13: (a) The intuitive menu of Interact Software. (b) Input modality interface in Interact Software. (c) The Interact navigator: a user-friendly Web navigator with functionalities adapted to predefined eye gaze and/or voice commands.

interfaces (i.e., the system is perceived as intelligent by people who naturally interact with a system which automatically adapts to their preferences). For example, the IAS menu groups together all the modes (eye gaze, voice, etc.) to access the tools of the software environment. The IAS can thus be said to constitute an intelligent and aware interface.

As shown in Figure 13a (from left to right), the user can select and activate the following tools:

(1) initialize IAS;
(2) load Interact real-time fusion engine analysis and parameter interface;
(3) load Web Interact navigator;
(4) work in Windows OS;
(5) configure audio device;
(6) calibrate eye gaze;
(7) set time-fusion parameter;
(8) switch on vocal inactivity gauge.

The vocal inactivity gauge in particular constitutes an example of ambient intelligence. The last button on this menu bar contains a visual gauge indicating the vocal inactivity of the user. If there is no input event at all (total inactivity of the user in a given period of time), a multimodal warning message is presented, and, as a result, a screen saver can be executed. The interface is thus attentive to the activity as well as to the passivity of the user.

When the IAS is reduced to a tray icon system, IAS is still running and the user can use his or her gaze and voice to operate Windows OS, and so open and use the classic Windows application. (Open Media Player and Windows Explorer, move files, use the virtual keyboard to edit text, etc.) Of course, an application like Windows Explorer does not work efficiently with multimodal inputs like eye-gaze and voice detection. As a remedy for its inadequacies, the Web navigator of IAS (Figure 13c), with its set of vocal commands and large icons (adapted to eye-gaze detection), offers a better environment. With all the possibilities it offers, the IAS Web navigator provides the W3C application with its ubiquitous computing aspect.

Enabling the ambient intelligence vision means providing consumers with universal and immediate access to available content and services, together with ways of effectively exploiting them. This is also true of the proposed interface, where the user switches from one modality to another in run time. In terms of the software system's development aspect, this means that the actual implementation of our ambient intelligence application as requested by a user can only be resolved at run time according to the user's specific situation.

This is possible while the architecture for MMDA and its associated core agent-based middleware are attentive to each dynamic composition according to the task environment. We can confirm that the proposed multiagent architecture corroborates W3C MMI activities.

5.3. *Reliability of the architecture*

The quality attributes of the software elegantly manage errors made by the user, offering a user-friendly help environment throughout the output modalities (vocal synthesis, hints, and pop-up messages displayed on-screen). When the software is executed for the first time, a pop-up window appears. Eye-gaze and vocal input and output modalities are activated by default. The user is then able to choose other available modalities or deactivate any of them (Figure 13b). After 10 seconds of user inactivity, a vocal signal within a displayed password window message initiates the user's identification protocol. This identification lets the system load the calibration settings of the eye-gaze tracker system. If there is no reaction from the user, IAS vocally confirms that the eye-gaze and speech modalities are activated with the default calibration parameters, and a pop-up help window displays the available help command.

The user can also say "help" anytime and a contextual help message (related to the mouse cursor position) is displayed.

The IAS then continues with the remaining human-machine communication protocol processes. The application never breaks down if a word spoken by the user or a command is not recognized. An alternative message can always be displayed or synthesized vocally to avoid system jamming. In terms of error avoidance, the whole application, within its dynamic architecture, constitutes a robust interface.

6. CONCLUSION

In this paper, new agent-based architectural paradigms for multimedia multimodal fusion purposes are proposed. These paradigms lead to new generic structures unifying applications based on multimedia multimodal dialog. They also offer developers a framework specifying the various functionalities used in multimodal software implementation. In the first phase, the main common requirements and constraints needed by multimodal dialogs are set down. Then the interaction types related to early and late fusions are presented. After identifying the recurrent generic characteristics shared by all modalities, each input medium is associated with a specific language agent (LA). The LAs are interconnected directly or through a distributed parallel control agent (PCA) to perform the dialog. The architecture of the PCA is decomposed into intelligent components to make up a modular structure. These components manage temporal, redundant, and grammatical conflicts. The proposed architectures are modeled with timed, colored Petri networks and support both parallel and serial fusions. These architectures do not change dynamically. In order to introduce real-time changes to the previous MMDAs, a new agent-based architecture with the ability to react to events and perform architectural reconfiguration autonomously is proposed. The efficiency of the new architecture is detailed through several application examples, including a multimodal W3C interface for disabled individuals.

ACKNOWLEDGMENTS

The authors would like to thank the Commission Permanente de Coopération Franco-Québécoise 2003–2004, which is in turn supported by Québec's Ministry for International Relations and France's Ministry for Foreign Affairs (General Consulate of France in Québec). This project was also financially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

[1] R. A. Bolt, "Put-that-there: Voice and gesture at the graphics interface," *ACM Computer Graphics*, vol. 14, no. 3, pp. 262–270, 1980.

[2] J. L. Crowley and F. Bérard, "Multi-modal tracking of faces for video communications," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '97)*, pp. 640–645, IEEE Computer Society, San Juan, Puerto Rico, June 1997.

[3] Y. Bellik and D. Burger, "Multimodal interfaces: new solutions to the problem of computer accessibility for the blind," in *Proc. ACM Conference on Human Factors in Computing Systems (CHI '94)*, pp. 267–268, Boston, Mass, USA, April 1994.

[4] D. R. McGee, P. R. Cohen, and L. Wu, "Something from nothing: Augmenting a paper-based work practice with multimodal interaction," in *Proc. Designing Augmented Reality Environments Conference (DARE '00)*, pp. 71–80, ACM Press, Elsinore, Denmark, April 2000.

[5] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Volume 1, Basic Concepts*, Monographs in Theoretical Computer Science. Springer-Verlag, New York, NY, USA, 2nd corrected edition, 1997.

[6] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Volume 2, Analysis Methods*, Monographs in Theoretical Computer Science. Springer-Verlag, New York, NY, USA, 2nd corrected printing, 1997.

[7] K. Jensen, S. Christensen, P. Huber, and H. Holla, *Design/CPN Reference Manual*, Department of Computer Science, University of Aarhus, Aarhus, Denmark, 1995.

[8] F. Azémard, *Des références dans le dialogue homme machine multimodal une approche adaptée du formalisme des graphes conceptuels*, Ph.D. thesis, University Paul Sabatier, Toulouse, France, February 1995.

[9] S. Card, J. Mackinlay, and G. Robertson, "The design space of input devices," in *Proc. ACM Conference on Human Factors in Computing Systems (CHI '90)*, pp. 117–124, Seattle, Wash, USA, April 1990.

[10] W. Buxton, "Lexical and pragmatic considerations of input structures," *Computer Graphics*, vol. 17, no. 1, pp. 31–37, 1983.

[11] J. Foley, V. Wallace, and P. Chan, "The human factors of computer graphics interaction techniques," *IEEE Computer Graphics and Applications*, vol. 4, no. 11, pp. 13–48, 1984.

[12] D. Frohlich, "The design space of interfaces, multimedia systems, interaction and applications," in *Proc. 1st Eurographics Workshop*, pp. 53–69, Stockholm, Sweden, April 1991.

[13] N. Bernsen, *Taxonomy of HCI Systems: State of the Art*, ESPRIT BR GRACE, Deliverable 2.1, 1993.

[14] J. Coutaz and L. Nigay, "Les propriétés CARE dans les interfaces multimodales," in *Sixièmes Journées sur l'Ingénierie des Interfaces Homme-Machine (IHM '94)*, pp. 7–14, Lille, France, December 1994.

[15] S. L. Oviatt, "Multimodal signal processing in naturalistic noisy environments," in *Proc. 6th International Conference on Spoken Language Processing (ICSLP '00)*, B. Yuan, T. Huang,

and X. Tang, Eds., vol. 2, pp. 696–699, Chinese Friendship Publishers, Beijing, China, October 2000.

[16] S. L. Oviatt, "Multimodal system processing in mobile environments," in *Proc. 13th Annual ACM Symposium on User Interface Software Technology (UIST '00)*, vol. 2, pp. 21–30, ACM Press, San Diego, Calif, USA, November 2000.

[17] E. L. Hutchins, J. D. Hollan, and D. A. Norman, "Direct manipulation interfaces," in *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper, Eds., pp. 87–124, Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1986.

[18] S. L. Oviatt, P. R. Cohen, L. Wu, et al., "Designing the user interface for multimodal speech and pen-based gesture applications: state-of-the-art systems and future research directions," *Human Computer Interaction*, vol. 15, no. 4, pp. 263–322, 2000.

[19] C. Bregler, H. Hild, S. Manke, and A. Waibel, "Improving connected letter recognition by lipreading," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP '93)*, vol. 1, pp. 557–560, IEEE Press, Minneapolis, Minn, USA, April 1993.

[20] Projet AMIBE, "Rapport d'activité, GDR |*n*°9, GDR-PRC Communication Homme-Machine," Tech. Rep., CNRS, MESR, pp. 59–70, 1994 (French).

[21] S. L. Oviatt, "Mutual disambiguation of recognition errors in a multimodal architecture," in *Proc. ACM Conference on Human Factors in Computing Systems (CHI '99)*, pp. 576–583, ACM Press, Pittsburgh, Pa, USA, May 1999.

[22] Y. Bellik, *Interfaces multimodales: concepts modèles architectures*, Ph.D. thesis, University of Paris XI, Paris, France, 1995.

[23] N. R. Jennings and M. J. Wooldridge, "Applications of intelligent agents," in *Agent Technology: Foundations, Applications, and Markets*, N. R. Jennings and M. Wooldridge, Eds., pp. 3–28, Springer-Verlag, New York, NY, USA, 1998.

[24] G. Weiss, *Multiagent Systems*, MIT Press, Cambridge, Mass, USA, 1999.

[25] S. D. Bird, "Toward a taxonomy of multi-agent systems," *International Journal of Man-Machine Studies*, vol. 39, no. 4, pp. 689–704, 1993.

[26] A. H. Bond and L. Gasser, *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, San Mateo, Calif, USA, 1988.

[27] T. Ishida, *Real-Time Search for Learning Autonomous Agents*, Kluwer Academic, Norwell, Mass, USA, 1997.

[28] H. J. Muller, "Negotiation principles," in *Foundations of Distributed Artificial Intelligence*, G. M. P. O'Hare and N. R. Jennings, Eds., pp. 211–230, John Wiley & Sons, New York, NY, USA, 1996.

[29] P. R. Cohen, H. R. Levesque, and I. Smith, "On team formation," in *Contemporary Action Theory*, J. Hintikka and R. Tuomela, Eds., Synthese, Kluwer Academic, Dordrecht, Netherland, 1997.

[30] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, Cambridge, UK, 1996.

[31] J. Kramer and J. Magee, "The evolving philosophers problem: dynamic change management," *IEEE Transactions on Software Engineering*, vol. 16, no. 11, pp. 1293–1306, 1990.

[32] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based runtime software evolution," in *Proc. 20th International Conference on Software Engineering (ICSE '98)*, pp. 177–186, Kyoto, Japan, April 1998.

[33] R. J. Allen, R. Douence, and D. Garlan, "Specifying dynamism in software architectures," in *Proc. Workshop on Foundations of Component-Based Software Engineering*, Zurich, Switzerland, September 1997.

[34] R. N. Taylor, N. Medvidovic, K. M. Anderson, et al., "A component- and message-based architectural style for GUI software," *IEEE Transactions on Software Engineering*, vol. 22, no. 6, pp. 390–406, 1996.

[35] J. Kramer and J. Magee, "Self organizing software architectures," in *Joint Proceedings of the 2nd International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96)*, pp. 35–38, ACM Press, San Francisco, Calif, USA, October 1996.

[36] J. M. Purtilo, "The POLYLITH software bus," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 1, pp. 151–174, 1994.

[37] M. Barbacci, C. Weinstock, D. Doubleday, M. Gardner, and R. Lichota, "Durra: a structure description language for developing distributed applications," *Software Engineering Journal*, vol. 8, no. 2, pp. 83–94, 1993.

[38] T. Bloom and M. Day, "Reconfiguration and module replacement in Argus: theory and practice," *Software Engineering Journal*, vol. 8, no. 2, pp. 102–108, 1993.

[39] J. Magee and J. Kramer, "Dynamic structure in software architectures," in *Proc. 4th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE '96)*, pp. 3–14, ACM Press, San Francisco, Calif, USA, October 1996.

[40] R. De Nicola, G. Ferrari, and R. Pugliese, "KLAIM: a kernel language for agents interaction and mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 315–330, 1998.

[41] P. Ciancarini and C. Mascolo, "Software architecture and mobility," in *Proc. 3rd Annual International Workshop on Software Architecture (ISAW '98)*, pp. 21–24, ACM Press, Orlando, Fla, USA, November 1998.

[42] W. Van Belle and J. Fabry, "Experience in mobile computing/The Cborg mobile multi-agent system," in *Technology of Objet-Oriented Languages and Systems, Tools 38*, pp. 7–18, Zurich, Switzerland, March 2001.

[43] A. Ramdane-Cherif and N. Levy, "An approach for dynamic reconfigurable software architectures," in *Proc. 6th World Conference on Integrated Design and Process Technology (IDPT '02)*, Pasadena, Calif, USA, June 2002.

[44] S.-C. Chen and R. L. Kashyap, "Temporal and spatial semantic models for multimedia presentations," in *Proc. International Symposium on Multimedia Information Processing (ISMIP '97)*, pp. 441–446, Taipei, Taiwan, December 1997.

[45] P. Tabeling, "Multi-level modeling of concurrent and distributed systems," in *Proc. International Conference on Software Engineering Research and Practice (SERP '02)*, Las Vegas, Nev, USA, June 2002.

[46] User's Guide, *The eyegaze development system for windows NT/2000*, LC Technologies, May 2003. http://www.eyegaze.com.

[47] M. Weiser, "Some computer science problems in ubiquitous computing," *Communications of the ACM*, vol. 36, no. 7, pp. 74–83, 1993.

[48] M. L. Dertouzos, "The future of computing," *Scientific American*, vol. 281, no. 2, pp. 52–55, 1999.

[49] M. Esler, J. Hightower, T. Anderson, and G. Borriello, "Next century challenges: data-centric networking for invisible computing," in *Proc. 5th ACM Conference on Mobile Computing and Networking (MOBICOM '99)*, pp. 256–262, Seattle, Wash, USA, August 1999.

[50] IST Advisory Group (ISTAG), "Scenarios for ambient intelligence in 2010," Tech. Rep., Information Society Technologies Advisory Group, 2001, http://www.cordis.lu/ist/istag.htm.

[51] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: toward distraction-free pervasive computing," *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 22–31, 2002.

[52] D. Milojicic, A. Messer, P. Bernadat, et al., "ψ-pervasive services infrastructure," in *Proc. 2nd International Workshop on Technologies for E-Services (TES '01)*, vol. 2193 of *Lecture Notes in Computer Science*, pp. 187–200, Rome, Italy, September 2001.

H. Djenidi received his B.S. degree from the University of Science and Technology Houari Boumediene, Algiers, Algeria, in electrical engineering and his M.S. degree in metrology and systems from Conservatoire National des Arts et Métiers (CNAM), Paris, France, respectively, in 1992 and 1994. From 1995 to 2002, he has been a Research Assistant for Canal Plus Inc. via CNAM and for Aircraft Bombardier Inc. via École de Technologie Supérieure (ETS), University of Quebec at Montreal, Canada. He was also an Electronic and Software Engineering Teacher at the Universities of Versailles and Créteil, France, and ETS, Canada. Since 2003, he has been studying for his Ph.D. at PRISM Laboratory, University of Versailles Saint-Quentin-en-Yvelines, France, and ETS, Canada. His investigations and field interests include multimodal interactions, multiagent architectures, and software specifications. Hicham Djenidi is a Student Member of IEEE.

S. Benarif received his B.S. degree from the University of Science and Technology Houari Boumediene, Algiers, Algeria, in electrical engineering and his M.S. degree from the University of Versailles Saint-Quentin-en-Yvelines, France, respectively, in 2000 and 2002. Since 2003, he has been studying for his Ph.D. at PRISM Laboratory, University of Versailles, France. His investigations and field interests include dynamic architecture, architectural quality attributes, architectural styles, and design patterns.

A. Ramdane-Cherif received his Ph.D. degree from Pierre and Marie Curie University of Paris in 1998 in neural networks and AI optimization for robotic applications. Since 2000, he has been an Associate Professor at PRISM Laboratory, University of Versailles Saint-Quentin-en-Yvelines, France. His main current research interests include software architecture and formal specification, dynamic architecture, architectural quality attributes, architectural styles, and design patterns.

C. Tadj is a Professor at École de Technologie Supérieure (ETS), University of Quebec, Montreal, Canada. He received a Ph.D. degree in signal and image processing from ENST Paris in 1995. He is a Member of the Laboratory of Integration of Technologies of Information (LITI) at ETS. His main research interests include automatic recognition of speech and voice mark, word spotting, HMI, and multimodal and neuronal systems.

N. Levy is a Professor at the University of Versailles Saint-Quentin-en-Yvelines, France. She has a Doctorate in Mathematics from the University of Nancy (1984). She directs an engineering school, the ISTY, and is responsible for the SFAL team (Formal Specification and Software Architecture) of PRISM, the laboratory of the university associated with the CNRS. Her main research interests include formal and semiformal development methods, style and architectural models formalization, and quality attributes of software architectures and distributed systems.