Formal framework and necessary properties of the fusion of input modes in user interfaces


Multiple input devices are increasingly used in user interfaces to make human-computer communication more efficient and effective. Interface designers have not only to decide on which input modes should be supported, but also how to fuse them into a single representation format that can be processed by the underlying application system. Drawing appropriate decisions requires, however, a sufficient understanding of the properties of fusion itself. While others have informally characterized input fusion as a transformation between information types, the purpose of the paper is to explore fusion by means of formal process modelling. That is, fusion processes are defined in a formal framework which supports proof of the existence of necessary properties following directly from the process definitions. The presented approach can be applied to analyse and compare fusion processes in existing systems, as well as an aid for interface designers who have to verify the behaviour of their systems.

Keywords: multimodal interfaces, fusion, formal methods, ambiguity

User interfaces of many application systems have begun to include multiple devices which can be used together to input single expressions. Such interfaces (and even whole application systems) are commonly labelled multimodal, because they use different types of communication channels to acquire information.

For example, early prototype systems such as SCHOLAR (Carbonell, 1971), or NLC (Brown et al., 1979), aimed at combining natural language (NL) input from a keyboard with simple pointing gestures on the screen. A more advanced approach has been taken in the XTRA system (Wahlster, 1991), where written NL has been combined with several types of pointing gestures of different

---

*CNR, Istituto CNUCE, Via S. Maria 36, 56126 Pisa, Italy
†CNR, ITIA, Viale Lombardia 20, 20131 Milano, Italy
‡GMD, FIT, Schloß Birlinghoven, 53754 Sankt Augustin, Germany
¶ICS, PO Box 1385, 711 10 Heraklion, Crete, Greece
§DFKI, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
**CCLRC RAL, Chilton, Didcot OX11 0QX, UK
granularity, e.g. pointing with a pencil or the hand. There are also various systems which allow for speech input accompanied by pointing gestures, e.g. Bolt's put-that-there (Bolt, 1980), Cubricon (Neal and Shapiro, 1991), MATIS (Nigay et al., 1993), VoicePaint (Salber et al., 1993), Talk and Draw (Salibury et al., 1990), and VECIG (Sparrel, 1993). Also, multimodal input does not necessarily include written or spoken NL. The Gesture System (Bordegoni and Hemmje, 1993), for example, allows a user to interact with three-dimensional objects using gestures combined with the use of a force input device, namely a spaceball. Multimodal systems have been predicted to be the basis of future human-computer interfaces, and "Advances in media coordination could allow more efficient interactions between computer and user in the standard computer applications of today" (Grosz and Davis, 1994). In multimodal systems, user input can be provided in the modality most compatible with the semantics to be conveyed, e.g. spatial information by gesture, while quantified or negated statements can be made in natural language (Cohen, 1992).

When different parts of an expression are input through different modalities they must be fused into a single expression which can be processed by the underlying application system. A simple example of fusion would be the combination of the natural language utterance "Put that there", with two gestures, identifying the object denoted by "that" and the location denoted by "there", into a single interpretable command to a system. Several criteria have been proposed in the literature to characterize the fusion processes. For example, in Salber et al. (1994) two basic definitions of fusion are distinguished:

- computation of a process abstracting/concretizing a collection of information types received from distinct processes into a different information type to be transferred to another process;
- composition of multiple information types at some level of abstraction into a single information type of the same level of abstraction.

Fusion has been characterized with respect to the temporal relationship between input events which have to be fused (Contaz et al., 1993). If the modalities providing the input expressions are used in parallel, then the multimodal interaction is described as synergistic; if they are used sequentially, it is described as alternate.

Criteria like the above support the analysis and comparison of existing multimodal interfaces beyond the level of surface aspects such as the supported modalities and application domain. If, for example, classify the systems mentioned earlier with respect to the use of modalities, we find that systems which rely only on the keyboard for NL input provide alternate use of modalities, while systems with speech input usually support the synergistic use of voice and pointing devices. Other systems allow both uses of modalities. An example is MATIS which allows users to get information on flight schedules using speech, mouse and keyboard, or a combination of all these. For example, a user can input the expression: "Show me USAir flights from Pittsburgh to Boston," using speech only, or input the expression: "Show me USAir flights from Pittsburgh to this city," using speech and pointing at a label for specifying this by using the mouse.

Fonetti et al.
For defining an expression, multiple modalities available for specifying it can be combined in a sequential way. For example, the user can say: *Flights from*, then select *Pittsburgh* from a menu list, then say *to*, and then select *Boston* from a menu list. Conversely, the multiple modalities can be combined concurrently. For example, one can say: "Flights from this city to this city arriving in the afternoon," while performing mouse clicks on a menu list to denote the two cities referred to in the utterance.

Although further criteria may be introduced to get more fine-grained descriptions of fusion processes, such informal characterizations do not provide a sufficient level of precision. If multimodal interfaces are to be significant in the future, then, as the fusion process is so crucial to them, it will itself be important in many systems. When developing future multimodal interfaces, however, developers must agree upon a single clearly defined fusion process that behaves consistently across applications, and they must understand the properties of the user interface more generally that follow from the design of the fusion process itself.

In this paper, we show how fusion processes can be formally characterized in order to achieve the fourth of the general reasons for formally specifying aspects of systems:

- To formally verify an implementation by proving that it is a valid interpretation of a specification.
- To demonstrate how a valid implementation can be evaluated, in order to show that it indeed has the properties embodied in the specification.
- To gain an insight into the structure of specification and process.
- To show the properties which necessarily result from the specification. It is possible to prove the set of all processes which could be results of a formal specification, and to test which properties hold true for all of them, thereby ensuring that there can be no unexpected results from an implementation.

It is not necessary to completely understand the formal notations used in this paper to follow the argument that properties can be proved to be true of all possible implementations of a specification, even if they are not explicitly written in the specification itself. The properties of incompleteness, redundancy of information and temporal co-ordination are proved at the end of the next section following the development of formal analytic and engineering models of fusion. The final section applies this framework to an existing multimodal system — MATIS. It is hoped that the introduction to the formal notations and their use for this purpose provided in the next section, will encourage the adoption and further use of formal methods to determine the consequences of specifications of user interfaces, and help to provide a basis for determining the validity of proofs of implementation properties from specifications.

**Specifying fusion processes in a formal framework**

As a starting point for setting up a formal framework to reason about properties of fusion processes, we regard fusion as a transformation process that maps several
input units onto a single one. The formalism itself is drawn from the Z (Spivey, 1988), LOTOS (Bolognesi and Brinksma, 1987) and ACTL (De Nicola et al., 1991) notations. This introduction to the formalisms used in this paper is intended to provide a general introduction to formal notations used for aspects of user interfaces in order to gain the four benefits listed earlier.

The reason for addressing more than one formalism is that no single formal specification language can today offer sufficiently flexible expressiveness. Consequently, one is forced to use different notations to express or investigate different properties of such systems. Such different formal models have complementary roles in the process of software development.

It is possible to describe the complete system formally in full LOTOS. However, this would require that each element in each set must be specified (at least by a function to generate it). This requirement that all data structures be declared is equivalent to an implementation language description, which is not surprising as LOTOS is a constructive logic (one where the proof mechanism results in constructing the result or process — Duke et al., 1994). In contrast, Z allows specifications to be reasoned over by merely allocating a symbol to a set, which is clearly more abstract. Unfortunately, the high level of abstraction with which describing the state of the world in terms of sets provides Z also prevents it from describing processes, so it is normally used to analyse the data structures in systems. Consequently, the process operations of fusion will be described in basic LOTOS terms of observable events and actions which characterize the behaviour of a system in time. In addition, the data structures are defined in Z where operations are defined in terms of pre- and post-conditions, but there is no ordering in time of the operations. Between them, these two languages allow the system to be specified. The consequences of the description will then be derived — the consequences of the description are all the processes which could result from it; that is the set of all implementations that could result from the specification. This derivation is performed by a software tool that operates over the LOTOS specification producing a very large set of mostly trivially differing process descriptions. Rather than check these by hand, to show that a property applies to all of them, properties will be written in ACTL, and then another software tool will be used to prove whether they apply to all the generated process descriptions. Those properties which apply to all processes can be said to be true of all possible implementations which result from the specification written in the combination of Z and basic LOTOS.

In the following section, a short introduction is given to the foundations in which the notations are grounded. The interested reader may directly address the books and papers referenced in order to gain a deeper insight into the formalism.

**Highlights of the formal notations**

*The Z notation*

The formal specification notation Z (pronounced "zed") (Spivey, 1988) is a language and a style for expressing formal specifications of computing systems. It is based on a typed set theory and first-order predicate logic. It has been
developed at the Programming Research Group at the Oxford University Computing Laboratory and elsewhere since the late 1970s. It is now used by industry as part of the software and hardware development process. It is currently undergoing BSI standardisation in the UK and has been accepted for the ISO standardisation process internationally.

In the following we introduce the $\mathcal{Z}$ concepts necessary to set up a framework to reason about fusion. In this context, we consider fusion as the composition of multiple information type at some level of abstraction into a single information type of the same level of abstraction (Salber et al., 1994).

We define the composition of multiple information types as a collection of elements from a set by addressing the $\mathcal{Z}$ concept of bag:

$$\text{bag } X \equiv X \rightarrow_p N_1$$

where bag $X$ is the set of bags of elements $X$, and $\rightarrow_p$ denotes a partial function from $X$ to the set of strictly positive integers. Bags are collections of elements of $X$ in which the number of times an element occurs is significant. As an example, given $X := \{x_1, x_2\}$, then bag $X := \{x_1 \rightarrow 2, x_2 \rightarrow 3\}$ is the bag composed of two occurrences of element $x_1$ and three occurrences of element $x_2$.

Multiplicity of elements and bag membership are specified as:

$$\forall x : X; B : \text{bag } X : \text{count } B = (\lambda x : X \bullet 0) \oplus B \land x \in B \Leftrightarrow x \in \text{dom } B$$

The above specification defines:

- count as a bijection ($\rightarrow_b$), and in as a binary relation ($\rightarrow$),
- $\oplus$ is the functional overriding so that $f \oplus g$ is defined on the union of the domains of $f$ and $g$. On the domain of $g$ it agrees with $g$, and elsewhere on its domain it agrees with $f$.
- $\lambda S \bullet E$ denotes a function which takes arguments of a type determined by $S$, and returns the result $E$.

Consequently, we say that the number of times $x$ appears in the bag $B$ is count $B x$. and the relationship $x$ in $B$ holds exactly if this number is greater than 0.

Following the previous example, we can say that count(bag $X$) $x_1 = 2$ and that $x_1$ in (bag $X$) holds.

We observe that the specification does not take into account any ordering relation amongst the elements in the set of bags. In fact, we are not interested in how a bag is composed but only in the expression of the composition. This is justified by considering that bag $X$ identifies a class of equivalence on the set of
finite sequences over $X$. This is specified as:

$$\text{seq } X == \{ f : \text{N} \to \text{fp} X | \text{dom } f = 1 \ldots \#f \}$$

where $\to_{fp}$ denotes a finite partial function, $\#$ is the number of elements of a set, and $\text{dom}$ is the domain of a relation. Then, if $s$ is a non-empty sequence, $\text{items } s$ is the bag in which each element $x$ appears exactly as often as it appears in $s$.

We can define the following relation between sequences:

$$\forall s_1, s_2 : \text{seq } X \bullet s_1 \approx s_2 \iff \text{items } s_1 = \text{items } s_2$$

that defines a partition over $\text{seq } X$ whose elements are identified by $\text{bag } X$.

From the above specification, we can easily derive the following law:

$$\forall s_1, s_2 : \text{seq } X \bullet s_1 \approx s_2 \iff (\exists f : \text{dom } s_1 \to_{fp} \text{dom } s_2 \bullet s_1 = s_2 \circ f)$$

that is, any sequence can be rewritten as any other sequence in its class of equivalence defined by $\approx$, with $\circ$ being the backward relational composition or, briefly, $(f \circ g)(x) = f(g(x))$.

Continuing with the above example, we can say that $\text{bag } X$ is the class of equivalence of all possible sequences containing two occurrences of $x_1$ and three occurrences of $x_2$ (i.e. $\langle x_2, x_1, x_2, x_2, x_1 \rangle$, $\langle x_2, x_2, x_2, x_1, x_1 \rangle$, etc.)

From the above discussion, one may realise that a guiding principle of the $\mathcal{O}$ approach to specification is the use of the ordinary structures of mathematics in the writing of software applications. The familiar language of sets and relations proves to be sufficient to describe succinctly the abstract structures needed in programming, and is already known to every mathematician and to many non-specialists.

The LOTOS notation

LOTOS, (the Language of Temporal Ordering Specification, Bolognesi and Brinksma, 1987), was originally developed by ISO/IEC for the specification of open distributed systems and in particular for those related to the Open Systems Interconnection (OSI) architecture. The language is the first and so far, only, specification technique to be standardised as an international standard.

Following the methodology firstly introduced by Milner's Calculus of Communicating Systems (CCS) (Milner, 1990), LOTOS provides a notation for
defining the temporal relation among the interactions representing the externally observable behaviour of a system.

In LOTOS a concurrent system is seen as a process able to perform internal, unobservable actions, and to interact with other processes by means of external, observable actions (interactions). Actions are atomic entities that occur at interaction points or gates, without consuming time (i.e., they occur instantaneously). Processes may interact with each other by performing the common actions defined at their gates. A process definition specifies the behaviour of a process by defining the sequences of observable actions that may occur at the process gates.

Complex behaviours are expressed by composing simpler behaviour expressions via operators such as sequentiality \((P \succ Q)\), parallelism \((P ||| Q)\), synchronization \((P\mid[G]\mid Q)\), and disabling \((P \succ\succ Q)\), where \(P\) and \(Q\) are process instances and \(G\) is a set of synchronization gates.

The behaviours of processes, described in process definitions, are specified by means of the composition constructs, already described, and basic additional constructs such as the action prefix \((a; B)\), the choice \((B1 \mid\mid B2)\) and the guarding \((\langle e \rangle \rightarrow B)\). A further construct, called hiding \(hide(G\text{ in }B)\), indicates that all actions in the set \(G\) are internal non-observable actions, thus influencing how the functioning of a process may appear to the outside environment, including the user.

Communication is specified by giving a structure to actions in a similar way as in CSP (Hoare, 1985) so that \(g?x: Integer\) assigns an integer value to variable and \(g!v\) delivers the value \(v\).

Basic LOTOS does not provide the capability to express data values. This functionality is made available in Full LOTOS by loosely coupling to the basic notation the specification language for abstract data types ACTONE (Ehrig and Mahr, 1985) based on an algebraic notation.

As an example, \(P[a, b](v : Int) ::= a?x : Int; ~\langle x eq v\rangle\rightarrow stop \mid\mid \langle not(x eq u)\rangle \rightarrow b!v;\)
\(P[a, b](v + x)\) describes the behaviour of a process of state \(v\) that reads an integer value at gate \(a\) and then either stops if this value matches the process state or delivers this value at gate \(b\) and recurs with the new state value \(v + x\).

LOTOS, as it stands, supports qualitative reasoning about the temporal ordering of actions. Extensions to the notation are under consideration that allow the introduction of quantitative measurements of the time at which actions occur in systems (Bolognesi et al., 1993). In Timed-LOTOS, a process can make action transitions as in the standard notation as well as timed transitions.

The \textit{ACTL notation}\n
ACTL (De Nicola et al., 1991) is an action based branching time temporal logic. Temporal logics (de Bakker et al., 1989) have been recognised as a useful means of formally expressing properties, and hence requirements, of reactive systems. Consequently, model-checkers have been realised to verify properties, written as temporal logic formulae, on the semantic model of a system.

The model-checker of ACTL permits the verification of the validity of an ACTL formula on a Labelled Transition System (LTS). Since the semantic model of a system, specified in LOTOS, can be represented by an LTS, we are able to verify
properties of a LOTOS specification by verifying the validity of ACTL formulae, expressing these properties, on the LTS derived from the specification. This is a well established methodology in system verification, usually called model-checking.

As an introduction, even at the surface level, to temporal logic can be quite complex, we only give here the informal meaning of the few ACTL formulae used in this paper:

- $[\chi]\phi$ formulae $\phi$ holds for all states that are reached by a transition involving the $\chi$ action.
- $\langle \chi \rangle \phi$ if formula $\phi$ holds for a state, the system can perform a transition from this state involving the $\chi$ action,
- $A\phi$ formula $\phi$ holds for all paths,
- $F\phi$ eventually formula $\phi$ holds,
- $G\phi$ always formula $\phi$ holds.
- $[\phi \langle \chi \rangle U \chi']\phi'$ the system can only perform transitions involving action $\chi$ from the current state to a next state for which formula $\phi$ holds until action $\chi'$ is performed and formula $\phi'$ holds for the next state.

Analytic model of fusion

It is now possible to describe fusion within the framework already introduced. In our account of fusion, we need to deal with both information types and information values, therefore two given sets are introduced. Each names a set of values that are of interest in the specification, but whose internal structure is unimportant:

- $\text{infovalues}$ information values
- $\text{infotypes}$ information types

Firstly, the state space of fusion will be described through three schemata. The first schema characterizes the information to be fused:

\[\text{FusionStateInput}\]

\[
\begin{align*}
\text{incharacteristic} & : \text{bag infotypes} \\
\text{information} & : \text{infotypes} \rightarrow F_{1} \text{infovalues}
\end{align*}
\]

(P1) \#\text{incharacteristic} \geq 1 \lor

\[
(\#\text{incharacteristic} = 1 \land
(\exists t : \text{infotypes} \cdot (\text{count incharacteristic} t > 1))
\]

(P2) dom information = dom incharacteristic

The schema consists of one part above the central dividing line, in which the state variables are declared, and a second part below the line which defines the possible values of those variables:

- \text{incharacteristic} is a characteristic identifier of fusion; it is the bag of information types that uniquely identifies the composition of information to be fused. As
previously discussed, the concept of bag is suitable to represent the fact that the fusion of information is independent from the order in which they are entered into the system;

• *information* is a function which, when applied to certain types, gives the set of values associated with them. $F_1$ *infovalues* denotes those subsets of *infovalues* whose elements can be counted with some natural number and whose cardinality is greater than 0. In other terms, each type is associated at least with one value that is distinguished from the empty value.

The predicates of the schema, the part below the central line, give some relationships which are true in every state of the system and are maintained by every operation on it. They are referred as the *invariants* of the system, and have been numbered for ease of reference:

• From predicate (P1) we say that the cardinality of *incharacteristic* must be either greater than 1 or equal to 1 in which case there exists a unique information type in the bag and this type must occur more than once. The invariant has been introduced in order to express that a fusion has to deal with multiple information. Note that this is a free interpretation of the wording *multiple information types* given in the second definition of fusion in the introduction. In fact this definition allows the fusion of either multiple information of different types or multiple information of the same type;

• Predicate (P2) says that *information* can be validly applied only to the set of types exactly defined by *incharacteristic*.

The second schema introduces observations about the result of the fusion:

<table>
<thead>
<tr>
<th>FusionStateOutput</th>
</tr>
</thead>
<tbody>
<tr>
<td>resulttype : infothers</td>
</tr>
<tr>
<td>resultinfo : $F_1$ infovalues</td>
</tr>
</tbody>
</table>

The schema is very simple in that it just introduces two variables that can get values respectively from the set of *infothers* and of the finite subsets of *infovalues*.

• *resulttype* is a second characteristic identifier of the fusion. It defines the type of the information which is the result of the fusion;

• *resultinfo* carries the content of this information.

The third schema, which completes the description of the state space is:

<table>
<thead>
<tr>
<th>FusionStateAll</th>
</tr>
</thead>
<tbody>
<tr>
<td>FusionStateInput</td>
</tr>
<tr>
<td>transform : (infothers $\rightarrow_{F_1}$ infovalues) $\rightarrow_{F_1}$ infovalues</td>
</tr>
</tbody>
</table>

$(P1) \forall T : \text{dom transform} \cdot T = \text{information}$
The schema includes the definition of the state space given in FusionStateInput and in FusionStateResult, and adds one more observation:

- **transform** is a function which, when applied to certain functions that return values associated to types, gives functions of the same kind. The definition is very abstract and represents values from a set of the type $P \times ((P(\text{infotypes} \times (P \text{infovalues}))) \times (P \text{infovalues}))$;
- **predicate (P1)** defines an Invariant so that transform can be successfully applied only to information. Consequently, only those sets of values, for which an association exists with the types in the incharacteristic bag, can be transformed.

Having defined the (abstract) state space, three operations which constitute fusion can be defined. The first of these is to add new information, as described in the following schema:

<table>
<thead>
<tr>
<th>FusionAddInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>ΔFusionStateInput</td>
</tr>
<tr>
<td>inputtype? : infotypes</td>
</tr>
<tr>
<td>inputvalue? : P1 infovalues</td>
</tr>
<tr>
<td>(P1) inputtype? ∈ dom incharacteristic → information'(inputtype?) = information(inputtype?) ∪ {inputvalue?} {inputtype?} \leq information'</td>
</tr>
<tr>
<td>(P2) inputtype? ∉ dom incharacteristic ⇒ information' = information</td>
</tr>
</tbody>
</table>

The declaration ΔFusionStateAll indicates that the schema is describing a state change; it introduces a pair of variables for each variable in the state, e.g. information' and information. The primed variables are observations of the state after the change, the unprimed ones are observations of the state before the change. Each pair of variables is implicitly constrained to satisfy the invariants, so they must hold both before and after the operation.

The next declarations introduce two inputs to the operation defined as an information type and an information value (by convention, the name of inputs ends in a question mark).

Both predicates give a pre-condition and a post-condition respectively on inputtype? and on values' to be satisfied for the success of the operation. Predicate (P1) says that if the pre-condition inputtype? ∈ dom incharacteristic is satisfied then after the operation information' extends information to include {inputvalue?} and the set of information types doesn't change. Predicate (P2) says that if the pre-condition inputtype? ∉ dom incharacteristic is satisfied then the state does not change. In other terms, only values that are of a type defined by the characteristic bag of the fusion are considered by the system. This is reasonable, since we are not able to handle types of data that we do not know about.
The second operation is to fuse the information entered into the system so far, which is described by the schema:

\[ \text{FusionTransform} \]
\[ \Delta \text{FusionStateAll} \]

\[(P1) \ \text{resultinfo}' = \text{transform}(\text{information}) \]
\[(P2) \ \forall t : \text{dom incharacteristic} \bullet \exists_d \text{default} : F_1 \text{infovalues} \bullet \text{information}'(t) = \text{default} \]

This schema describes the core of the fusion in predicate (P1): that is the computation of a resultinfo' by transforming the set of information available before the operation is performed. Predicate (P2) says that after the operation the function information' returns a default value for all the types defined by the bag incharacteristic.

Since the operation has no preconditions, it can be applied in any state of the system. This property introduces a non-determinism into the system with respect to the ordering of the execution of operations. Having recognised the problem here, it will be resolved within the engineering model in the next section through a refinement of the present formalism. However, the power of \( \mathcal{Z} \) in expressing state-based specifications is complemented by its weakness in dealing with system behaviour. The exact contrary applies to process algebras where strengths in describing system behaviour are coupled with weaknesses in describing system state — hence the use of \( \mathcal{Z} \) to describe system state.

The last operation, specified with the next schema, describes the output of the fusion system.

\[ \text{FusionGiveResult} \]
\[ \Delta \text{FusionStateAll} \]
\[ \text{outvalue!} : F_1 \text{infovalues} \]
\[ \text{outtype!} : \text{infotypes} \]

\[(P1) \text{outvalue!} = \text{resultinfo} \]
\[(P2) \text{outtype!} = \text{resulttype} \]
\[(P3) \exists_d \text{default} : F_1 \text{infovalues} \bullet \text{resultinfo}' = \text{default} \]

The operation has two outputs (by convention, the name of outputs end in a exclamation mark): outvalue!, which takes value from the finite subsets of infovalues, and outtype!, which takes values from infotypes. There are no preconditions, as in the previous case, so that the operation can be applied in any state of the system. However, we don't want to output information that has not been fused. To solve this problem we will later specify two further operations

\[ \text{FusionTransform} ; \text{FusionGiveResult} \]

that can only be executed in sequence, thereby guaranteeing that only fused information can be output.
The schema $S; T$ is defined to be

\[ \exists \text{State}'' \bullet \]

\[ (\exists \text{State}' \bullet [S[\text{State}' = \theta \text{State}''])] \]

\[ (\exists \text{State} \bullet [T[\theta \text{State} = \theta \text{State}''])] \]

where the value of the expression $\theta S$ in any situation is a binding. If $p$ and $q$ are distinct identifiers, and $x$ and $y$ are objects of type $t$ and $u$ respectively, then there is a binding $z$ with components $z.p$ equal to $x$ and $z.q$ equal to $y$. In other terms, the primed components of the first operation in the sequence correspond exactly to the unprimed components of the second operation, and this state is called the hidden state. The informal semantics of the sequential composition of operations is an operation where the state before the operation is the state before the first operation, and the state after the operation is the state after the second operation in the sequence.

For this reason we have included $\Delta \text{FusionStateAll}$ in the schema declaration even if the inclusion of $\Delta \text{FusionStateOutput}$ would be sufficient for the specification of the operation by itself.

$outinfo!$ carries the value of $\text{resultinfo}$ and it is assigned a type of $\text{resulttype}$, carried by $outtype!$. After the operation the value of $\text{resultinfo}'$ is assigned to a default value. It should be noted that since $\text{resulttype}'$ never changes, it uniquely determines the type of the information given as output once it has been initialized.

Finally, we can affirm that the more abstract fusion is defined as

\[ \text{Fusion} \triangleq \text{FusionAddInfo} \lor (\text{FusionTransform}; \text{FusionGiveOutput}) \]

that is its behaviour is an interleaving of the $\text{FusionAddInfo}$ operation and of the sequence of the $\text{FusionTransform}$ and of the $\text{FusionGiveOutput}$ operations.

So far we have described in the same mathematical framework both the state space and the operations that can be performed on a fusion system. The data objects were described in terms of mathematical data types such as sets and functions. The descriptions included invariant relationships between the parts of the state that are vital to understanding the fusion. The effects of the operations are described in terms of the relationship which must hold between input and output through transformation, rather than by giving a recipe to be followed.

To complete the specification, we must indicate what state the system is in when it is first started. In defining the initial state, we implicitly define what kind of fusion we are describing by giving (more) concrete values to $\text{incharacteristic}$ and to $\text{resulttype}$.

The initialization is clearly dependent on the context in which the fusion is performed. Since we are not addressing any specific application domain, we will not define exactly what the given sets are and how they are structured. Consequently, the initialization of the state is given semiformally in the following schema.
The predicates in the above schema just specify that there must exist default values for the variables in the state of the fusion. These values are also used to reset information' and resultinfo' after they have been actually used by operations. On the contrary, the initial values assigned to incharacteristic' and to resulttype' are never updated by operations.

A semiformal definition of transform can be given to describe the class of functions addressed by the fusion.

Let $m$ be the number of types that are in the domain of incharacteristic; such types are denoted by $t_i$, $i = 1, \ldots, m$. Let $v_{i,j}$ be the jth value of type $t_i$ input to the system, with $v_{i,0} = \text{default}$. Each value can be thought to depend from some parameter $p$, such as the time at which it has been input, so that we write $v_{i,j} = f_i(p_{t_i,j})$. Consider the set of all possible $m$-tuples $(p_{t_1,1}, \ldots, p_{t_m,m})$ whose elements are ordered according to a sequence $(t_1, \ldots, t_m)$ so that $\text{items}(t_1, \ldots, t_m) = \text{incharacteristic}$, and let us define $(p_1, \ldots, p_m) \mapsto \max(p_i) - \min(p_i)$, $i = 1, \ldots, m$ then the value returned by transform is

$$
someinfo \iff \forall (p_{t_1,1}, \ldots, p_{t_m,m}) \exists v_{t_i} = \text{default}$$

$$\exists (p_{t_1,1}, \ldots, p_{t_m,m}) \exists (p_{t_1}, \ldots, p_{t_m}) \leq g(p_{t_1,1}, \ldots, p_{t_m,m})$$

In other words, the result of the fusion is some application defined information (someinfo) in the case of incomplete input or a transformed $m$-tuple $(T(v_{t_1,1}, \ldots, v_{t_m,m}))$ so that the distance between the points at which the information is computed is minimized. The minimum itself can refer to different spaces, i.e., minimal distance in a sequence or minimal distance in time.

**Engineering model of fusion**

Starting from the analytic model of fusion developed in the previous section, we give a refinement of its behaviour by developing an engineering model, specified in the LOTOS process algebra.

In rewriting the $Z$ specification in LOTOS, we apply the following rules:

- for every non-sequential composition of $Z$ operations, we define a corresponding LOTOS process;
- for each sequence of $Z$ operations, we define a unique process where the behaviour of an operation is prefixed to the behaviour of the next operation in the sequence;
- we recover the global state space of $Z$ by the LOTOS synchronization with value passing construct.
Moreover, we introduce the following notational conventions:

- $T$ is the characteristic bag of the fusion, each type in the bag is denoted by $T_i$
- $GS$ is the set of gates at which the fusion process receives inputs from its environment. There exists a gate, denoted by $g_i$, for each of the information types so that the structure of the actions occurring at the gate is $g_i ? x : T_i$
- $o$ is the gate at which the fusion process makes available to its environment the information resulting from fusion, and $T_o$ is the type assigned to that information. The structure of the actions occurring at the gate is denoted by $o v$ and the condition $v \in T_o$ holds,
- $S$ is a list of information. Any information received at gate $g_i$ in the set $GS$ of gates is added to the list,
- $[]_i$ is defined to be $[\ldots Bex_i ::= Bex_i] \ldots [Bex_n]$
- $|[]_i$ is defined to be $[\ldots Bex_i ::= Bex_i] | \ldots | Bex_n$

Finally we define the following operations:

- $\text{add}: S \times T \rightarrow S$ that returns a list from a list and an information so that $\#S' = \#\text{add}(S, T_i)$
- $\text{transform}: S \rightarrow T_o$ that transforms a list to an output information of type $T_o$

The definition of the fusion process is consequently:

$$\text{FUSION}[GS, o](s : S, t, : T,) ::=$$

$$\text{hide } synch \text { in}$$

$$\text{FusionAddInfo}[GS, synch](\bar{S})[\bar{synch}]$$

$$\text{FusionTransformAndGiveResult}[synch, o](T_o)$$

where

- $\text{FusionAddInfo}[GS, synch](s : S) ::= [\ldots g_i ? x : T_i; \text{FusionAddInfo}[GS, synch](\text{add}(s, x)) [\ldots ; synch ! \sigma; \text{FusionAddInfo}[GS, synch](\bar{S})$]
- $\text{FusionTransformAndGiveResult}[synch, o](t_o : T_o) ::= synch ? \sigma : S; o ! \text{transform}(\sigma)$

$\text{FusionTransformAndGiveResult}[synch, o](T_o)$

where $\iota$ denotes a silent non-observable action.

![Graphical representation of the fusion process](image-url)
A graphical and more intuitive, although still formal, representation of the above process is shown in Figure 1, where processes are represented by boxes and gates by circles; a gate connected to more than one process implies a synchronization event on that gate.

The specification says that the FUSION process is defined over the set GS of gates and the o gate; its state is defined by the variable s of type S and by the variable t_0 of type T_0; its behaviour is the parallel composition with synchronization on the hidden gate synch of the FusionAddInfo and the FusionTransformAndGiveResult processes.

The FusionAddInfo process is defined over the set GS of gates and the synch gate; its state is defined by the variable s of type S and it is initialized by the FUSION process to be the value S; it receives inputs x : T_i that are added to the actual list s, or it performs an internal action 1 after which it offers to its environment the actual value of s at the gate synch. After the synchronization, the state of the process becomes the initial state.

The FusionTransformAndGiveResult is defined over the synch and o gates; its state is defined by the variable t_0 of type T_0 which is initialized to T_0 by the FUSION process; it synchronizes on the gate synch, receiving the list s and subsequently offers to its environment a value computed by transforming the received list at the gate o. Then, the process restarts its behaviour with the initial state.

With respect to the analytic model, the engineering model hardwires the types defined in the characteristic bag of the fusion by means of the set GS of gates and ensures that the fusion is correctly applied to the actual set of input information by synchronization. Moreover it relaxes the effect of the V operator in

\[ \text{Fusion} \triangleq \text{FusionAddInfo} \lor (\text{FusionTransform}; \text{FusionGiveOutput}) \]

by allowing input and output operations to be freely interleaved without danger for the safety of the system.

However, this engineering model is, as was the analytic one, non-deterministic in that the FusionAddInfo process decides autonomously whether to receive a new input or to synchronize with the FusionTransformAndGiveResult process in order to produce a fused piece of information. The non-determinism is required for its effect on the ordering in which input is received, but the fusion definition also requires control to be asserted over when information is output. Therefore the specification is refined by substituting the internal action \( \tau \) with the observable action trigger that is available to the environment for synchronization so that an external process can constrain its occurrence following a predefined behaviour. The refined definition of the FUSION process is:

\[
\begin{align*}
\text{FUSION}|GS, trigger, o|(s : S, t_o : T_o) := \\
\text{hide synch in} \\
\text{FusionAddInfo}|GS, trigger, synch|(S) \text{|| synch || FusionTransformAndGiveResult|(synch, o|(T_o))}
\end{align*}
\]

where

\[
\begin{align*}
\text{FusionAddInfo}|GS, trigger, synch|(s : S) := \\
[|g? x : T_i; FusionAddInfo|GS, trigger, synch|(add(s, x)) \\
[|trigger; synch \mid s; FusionAddInfo|GS, trigger, synch|(S)
\end{align*}
\]
Figure 2. Fusion process including trigger available to external processes

\[
\text{FusionTransformAndGiveResult}(\text{synch}, o)(t_o : T_o) := \\
\text{synch} ? s : S; o! \text{transform}(s); \text{FusionTransformAndGiveResult}(\text{synch}, o)(T_o)
\]

The process is still non-deterministic, but the non-determinism can be resolved by defining constraints on the process behaviour by imposing a synchronization on the trigger gate, as will be shown in the following sections.

The change in the specification is highlighted in the graphical representation in Figure 2 by the addition of the trigger gate.

**Necessary properties of fusion**

In the following we will examine a number of properties of the LOTOS specification, expressed by using the ACTL (De Nicola et al., 1991) notation, a branching time temporal logic based on actions that allows for qualitative reasoning about systems. These properties follow for any implementation of the specification so far described even though they were not explicitly stated in it. The consequences of these emergent properties of the specification should be considered by anybody designing a multimodal architecture (e.g. Binot et al., 1990), since they must be accounted for in the dialogue layer of the architecture after the fusion of input information has taken place.

**Delivery of fused information**

The first property we express is that after a trigger event has occurred, a corresponding action occurring at gate o and delivering fused information will always be generated and we write

\[
[\text{trigger}]\text{AF}[o]\text{true}
\]

This property guarantees that the FUSION process will always deliver a result.

**Incompleteness of information**

A further property is particularly relevant as it addresses an issue of interest in this context; it is written

\[
\text{AG}(\text{trigger})\text{true}
\]
The formula says that it is always possible for a trigger event to occur even if no input has occurred. This property is necessary to ensure that any process behaviour, constraining the FUSION process to deliver fused information at a certain moment, can be safely applied (i.e. avoiding deadlocks). The consequences of this property are that the FUSION process must be able to deal with possibly incomplete data in order to always deliver valid results. The minimal heuristic we can develop for this purpose, is to address the classical solution to this problem by distinguishing a specific symbol \( \perp \) amongst the set of values bound to a data type, that represents the default value for this data type. Since an actual instance of the FUSION process is dependent on the problem domain space, nothing more can be said other than that we have recognized the existence of such a problem.

**Redundancy of information**

The last property we consider is written

\[ \forall i: \#T \cdot (g_i)\text{true} \]

where \( \#T \) is the number of information types that can be input, and consequently of the input gates. The formula says that after any information has been input, other information of the same type can be input again. The consequences induced by this property are that in fusion one has to deal with redundancy of data because many tokens of the same type can in general be introduced into the system before fusion actually takes place, and consequently some data may be redundant.

Redundancy is itself not a problematic property of the fusion process, because when duplicated symbols are detected a simple deletion rule can be applied to produce in a single symbol (e.g., \( A \land A \Rightarrow A \)). Equally this phenomenon is clearly observed in users of multimodal systems. For example, the user of a travel reservation system when selecting a journey destination from a menu by issuing the commands verbally, will continue to use the verbal modality in addition to the selection, by selecting the town name and saying it at the same time.

However, users are imperfect and make slips and errors. The introduction of redundancy with an error by a user in one of two instances, results in a contradiction in input between two channels. This possibility of contradiction would not exist without the redundancy. Without redundancy, an error by the user in a symbol which results in valid but incorrect message would not be detected. That is to say, without redundancy at the level of the message from the user to the system, contradictions between a user's intention and his or her action would not be detected, because the action in itself would be valid. Redundancy in the message allows the contradiction existing between a user's intention and the resulting action, to be extended to two linked user actions in different modes. Therefore the introduction of redundancy at the message level allows the introduction in the system of a mechanism to repair temporarily inconsistent knowledge i.e. a mechanism to choose between two inconsistent messages.

**Temporal co-ordination of fusion**

The truth of the ACTL formulae, given in the previous section, raises a non-trivial issue that can be expressed as an answer to the question: *When does the trigger firing occur within the FUSION process?*
A possible way to proceed is to categorize a number of basic modes of fusion and to define a composition operator that builds more complex modes starting from the basic set.

Starting from the generic FUSION process, we add constraints to the specification by defining further processes that are composed in parallel with synchronization gates. In this way the behaviour of FUSION must agree with the behaviour of the processes in the parallel composition. To explain this fact, let's give some examples of basic modes of fusion. Figure 3 clarifies the resulting basic composition of processes.

The first basic mode, called pivot mode, is defined by the following property

\[ \exists i : \#T \cdot \mu x [x:T] (\text{pivot}) \rightarrow (\text{trigger}) \text{true} \]

saying that after information of a distinguished type (the pivot) has been input in the next state of the system a trigger event will occur that will direct the delivery of fused information.
A corresponding process definition is

\[
P[gi, trigger] ::= gi; trigger; P[gi, trigger]
\]

that when instantiated in the expression

\[
FUSION[GS, trigger, o][gi, trigger] P[gi, trigger]
\]

directs the FUSION process to deliver information on action \( o \) transform(s) after the action \( gi?x : Ti \) has occurred and internal synchronization on gate \( synch \) has taken place.

The second basic mode, called complete mode, is defined by the following property:

\[
\forall i : \#T \cdot A[true \{\sim trigger\} U{gi}true] \land [gi]A[true \{\sim gi\} U{trigger}true]
\]

Here no \( trigger \) events can occur until the set of information to be fused is complete and information of this type cannot be input twice between the occurrence of two \( trigger \) events. Clearly, this operating mode avoids both incompleteness and redundancy, but it might be too restrictive for most applications. A corresponding process definition is:

\[
COMPLETE[GS, trigger] ::= (|||GS; exit) \gg trigger; COMPLETE[GS, trigger]
\]

an instance of which is used to constrain the behaviour of the fusion as in:

\[
FUSION[GS, trigger, o][GS, trigger] COMPLETE[GS, trigger]
\]

Up to this point, we have been reasoning qualitatively about different aspects of fusion. The fusion process itself has been expressed abstractly from modalities. In contrast, a number of systems considered in the introduction to this paper deal with the synergistic use of different modalities together. Those systems heavily rely on models of time that cannot be expressed by only capturing the temporarily ordering of events in which a process can engage.

Whenever users effectively co-ordinate multiple modalities, such as speech and gestures, the most appropriate solution is to explicitly introduce a model of time such as the one proposed in the time enhancement of LOTOS (Bolognesi et al., 1993). In this case, fusion cannot be directed by a pivot, and the use of the complete mode would break the requirement for users to be free to choose their preferred modality.

As in the case of pivot and complete modes, a further process is defined that describes the basic timed mode once it is composed in parallel with the FUSION process. With this mode we are able to provide temporal windows to enter information into the system and we write

\[
\forall t : \#I \cdot A[true \{\sim trigger\} U{true} (age(gi) = wait_time)]
\]
to say that no trigger events can occur until the interval of time in which it is
allowed to input information is not expired. Clearly, this operating mode may
introduce both incompleteness and redundancy that the system must be prepared
to handle properly. A corresponding process definition is

\[
\text{TIMED}[\text{trigger}](\text{wait-time} : \text{Time}) ::= \langle \text{wait-time} \rangle ; \text{trigger}
\]

that when instantiated in the expression

\[
\text{FUSION}[\text{GS}, \text{trigger}, o][\text{trigger}][\text{TIMED}[\text{trigger}](\text{wait-time})
\]

directs the FUSION process to deliver information on action \( o \) ! transform(s) after a
period of time equal to the value of wait_time.

The three modes of fusion can be combined to form complex modes allowing to
incrementally define the behavior of complex systems starting from more
elementary and manageable building blocks.

The next section shows such a development process as applied to an example
derived from MATIS (Nigay et al., 1993).

Applying the framework to MATIS

MATIS has been developed at University of Grenoble to experiment on
multimodal interfaces. It allows an end-user to obtain information about flight
schedules from a database that responds to queries built by using speech, mouse
and keyboard, or a combination of them. The system has already been the subject
of considerable study (Coutaz et al., 1993; Duke and Harrison, 1993; Paterno and
Mezzanotte, 1994).

A full analysis of MATIS is beyond the scope of this paper due to its
inherent complexity. Here we address only the issue related to fusion when
using both a speech recognition system and a pointing device synergistically
to generate input information. Moreover we abstract from the current
implementation of MATIS and confine our analysis to the building of a
query from two slots respectively containing information on the departure and
on the arrival city of flights. This simplification does not influence the results
of the analysis, which can easily be extended to the case of more complex
queries.

Requirements of fusion in MATIS

The user may request information both by speaking (i.e., give me all the flights from
London to LameziaTerme) and by pointing to select a city name from a menu (i.e.
LameziaTerme). The system uses the user input to fill the from and to slots that will
form a query to the database.

The system is initially modeled by two processes: the first one (QUERY) is
responsible for composing the query and submitting it to the database, while the
second one (INPUT_SLOT) is responsible for filling the individual query items.
The state space of the system is defined by:

\[ \text{departure, destination, city} \]

query ::= a-slot \times d-slot  
the query

a-slot ::= destination \times a-city  
the destination slot

d-slot ::= departure \times d-city  
the departure slot

slot ::= a-slot \cup d-slot  
the generic slot

a-city ::= city  
the destination city

d-city ::= city  
the departure city

The QUERY process
The QUERY process is modelled as a fusion process operating in complete mode. This guarantees that only complete queries are submitted to the database. In fact, we observe that the complete mode, as defined above, only checks for the occurrence of an event at the input gates. Because we must also verify that the input tokens are valid ones, we refine the COMPLETE process as follows:

\[
\text{COMPLETE}[a\text{-slot-in}, d\text{-slot-in}, trigger](a : a\text{-slot}, d : d\text{-slot}) : =
\]

\[
a\text{-slot-in}? a : a\text{-slot}; \text{CONDITION}[a\text{-slot-in}, d\text{-slot-in}, trigger](a, d)
\]

\[
d\text{-slot-in}? d : d\text{-slot}; \text{CONDITION}[u\text{-slot-in}, d\text{-slot-in}, trigger](a, d)
\]

where

\[
\text{CONDITION}[a\text{-slot-in}, d\text{-slot-in}, trigger](a, d) : =
\]

\[
\left[ \begin{array}{l}
\text{not}(a \downarrow a\text{-slot}) \text{ and not}(d \downarrow d\text{-slot}) \\
\text{trigger}; \text{COMPLETE}[a\text{-slot-in}, d\text{-slot-in}, trigger](a, d)
\end{array} \right]
\]

The CONDITION process enforces a trigger firing only when the query has been filled with a complete set of valid values. Both textual and graphical representation of the QUERY process are given below and in Figure 4.

\[
\text{QUERY}[a\text{-slot-in}, d\text{-slot-in}, bd] : =
\]

\[
\text{FUSION}[a\text{-slot-in}, d\text{-slot-in}, trigger, db](\downarrow\text{query}, \downarrow a\text{-slot} \downarrow d\text{-slot})
\]

\[
\left[ [a\text{-slot-in}, d\text{-slot-in}, trigger]\right]
\]

\[
\text{COMPLETE}[a\text{-slot-in}, d\text{-slot-in}, trigger](\downarrow a\text{-slot} \downarrow d\text{-slot})
\]

Figure 4. MATIS query process modelled as a fusion process operating in complete mode
The process receives departure and destination city names respectively at the gates d_slot_in and a_slot_in. Following the cited example, an event may occur at the gate db (representing an interaction with the data base) only after both from \times London and to \times LameziaTerme are received.

The INPUT SLOT process
Both speech and pointing input modalities could in principle be used separately in filling the slots. Their synergistic use is justified by the case of a user who chooses to start an input operation by speaking, but may not know how to pronounce a city name. In this case, he or she can simply refer to the city as 'this' and supply its name by pointing to it. The INPUT SLOT process is consequently modelled as a fusion process. In this case, however, the complete mode of operation is not appropriate because it would always force users to enter information by using both available modalities, so in this instance, either the pivot or the timed modes are more suitable.

- In the case of timed mode, the definition of the INPUT SLOT process is straightforward (see also Figure 5):

\[
\text{INPUT SLOT}[\text{gesture, speech, a_slot_in}] := \\
\quad \text{FUSION}[\text{gesture, speech, trigger, a_slot_in, d_slot_in}](\downarrow \text{city}, \downarrow \text{a_slot}, \downarrow \text{d_slot}) \\
\quad \text{TIMED}[\text{trigger}](\text{wait_time})
\]

The process receives input sentences and city names through the speech and gesture gates. It delivers slots of departure and destination information to the QUERY process through the gates d_slot_in and a_slot_in upon time expiration.

In this case the user is free to enter information into the system from any of the available modalities by using them either in sequence or in synergistic mode. This information will be properly fused and progressed into the system at any wait_time interval expiration.

It is clear that the above specification has the properties discussed above as necessary properties of fusion and, consequently, we must be prepared to deal with incompleteness and redundancy of information.

- In the case of pivot mode, a decision should be made to give priority to one modality with respect to another. The user perceives that the system would

![Diagram](image)

Figure 5. MATIS Input Slot process modelled as a timed-mode fusion process

Faconti et al.
give freedom in the use of modalities as long as both are used. In practice, he or she is forced to use the relevant modality to trigger the fusion while introducing nothing but redundancy by using both of them.

When priority is given to speech, users can enter both departure and destination information in one chunk of information while pointing at city names in a corresponding order. As an example, one may say give me all the flights from London to LameziaTerme or, alternatively, give me all the flights to LameziaTerme from London. As soon as the phrase is recognized, the fusion process is triggered and the two slots of the query will be filled. The query, being complete, will be delivered to the database. The use of gesture is immaterial unless the sentence is not recognised in which case the fusion process will use redundant information to resolve it.

When priority is given to gesture, the user must point at the departure and destination cities in a predefined order or alternatively use the possibly redundant information from speech to solve the ambiguity. In the first case the formal definition of the bag of information, described in the analytic model of fusion, is broken while the second one introduces an unnecessary complexity into the system. In other words, if the user points to London and subsequently to LameziaTerme the input may be interpreted either as from London to LameziaTerme or an explicit sentence is spoken saying it. The conclusion is that, the user of the pivot mode of fusion, when relevance is given to gesture, is not compliant to our formal framework. For this reason, this mode of operation does not apply to our example and it will not be considered further.

Figure 6 shows the composition of processes in the resulting system.

Handling redundancy
The obvious motivation for having redundancy in the system, is that the user is free to enter a city name by pointing while speaking. The effect of this freedom is
that a city name entered by pointing may not match the spoken name of that city leading to an inconsistent state of the system.

Several solutions to this problem can be explored:

- The implementators of MATIS have adopted a solution that leads to results that might be not desired by the user. In fact, MATIS resolves the inconsistency outside of the fusion process by considering the actual state of the query. We explain this with two examples.
  - Let the two slots composing a query be empty, and let the user point to London while saying from LameziaTerme, then the system fills the Departure slot with the value of LameziaTerme and the Destination slot with the value of London.
  - Let the Destination slot be filled with the value London, and the user repeats the previous input, then the system fills the Departure slot with the value of LameziaTerme and presents a new query where both slots are filled with the value of London.

Obviously, the results given by the system in such cases will appear to the user as non-deterministic answers. Furthermore, this approach has the disadvantage of depending on the semantics of the application.

- The simplest approach is to provide the distinguished value from the type Slot as the result of the fusion and to notify the user of the inconsistency of input data. In this case, we will define

  \[ \text{transform}(\text{information}) = \bot \iff \text{last} (\text{Slot}) \neq \text{CityName} \land \text{Slot} \neq \bot \land \text{CityName} \neq \bot \]

- A further solution is to give higher priority to one of the processes providing the fusion process with information.

  We define:

  \[ \text{transform}(\text{information}) = \text{Slot} \leftrightarrow \text{Slot} \neq \bot \]
  \[ \text{transform}(\text{information}) = \text{first}(\text{Slot}) \times \text{CityName} \leftrightarrow \text{last} (\text{slot}) = \bot \land \text{CityName} \neq \bot \]
  \[ \text{transform}(\text{information}) = \bot \leftrightarrow \text{Slot} = \bot \land \text{CityName} = \bot \]

  in the case of speech priority, and

  \[ \text{transform}(\text{information}) = \text{Slot} \leftrightarrow \text{Slot} \neq \bot \land \text{CityName} = \bot \]
  \[ \text{transform}(\text{information}) = \text{first}(\text{Slot}) \times \text{CityName} = \text{CityName} \neq \bot \]
  \[ \text{transform}(\text{information}) = \bot \leftrightarrow \text{Slot} = \bot \land \text{CityName} = \bot \]

  in the case of pointing priority.

- More complex solutions can be explored, which will not be addressed here. As an example, one can think of having a knowledge base system embedded
within the transform function that could find solutions by applying inference rules.

Handling incompleteness
The specification previously developed is already able to deal with incomplete data. The minimal heuristic provided when discussing incompleteness above is powerful enough to provide a solution to this problem.

Enhanced temporal coordination of fusion in MATIS
The use of timed mode for the INPUT SLOT process only solves a part of the problem. What is readily needed is to relate the user input from both the considered modalities and not just a 'time-out' mechanism.

We observe that the user might be requested to synchronize the user's actions so that a pointing event occurs at the same time as the related word is spoken. However, further formalism of this suggestion requires a definition of users' perception of simultaneity. Our formalism cannot be used to derive any user perception of time or any property related to the usability of specific devices, which would follow from a psychological model of users. Nevertheless, we can introduce an imprecision in the specification by allowing input from both devices to occur only on precisely defined temporal windows. A further constraint is consequently added to the specification, by refining the TIMED process as in the following:

\[
\text{TIMED}[\text{gesture, speech, trigger}](\text{wait_time} : \text{Time}) ::= \\
\text{time gesture, speech in} \\
\begin{align*}
\{(\text{gesture}(0, \text{wait_time}) @ p ; \text{stop} || \text{speech}(0, \text{wait_time}) @ s; \text{stop}) \\
> &\text{wait_time}; \text{age to wait time}(p, s, \text{wait_time}); \text{trigger}; \\
\text{TIMED}[\text{gesture, speech, trigger}](\text{wait_time})
\end{align*}
\]

The above process declares gesture and speech as timed events so that they are aged. They can occur once at any time and in any order as soon as the process is activated. After a time-span equal to wait_time the events are disabled until a new activation of the process. The final composition of processes describing the architecture of the system is presented in Figure 7.

The above specification is the best we can do to reason about systems when unknown quantities and unpredictability are an intrinsic part of the problem to be described. A number of issues, which remain unresolved, do not depend on the specific style of specification adopted. In particular, no claim is made about the perception that users have of the existence of temporal windows. Consequently, we cannot predict how usable such system will be.

A further non-trivial issue is the exact definition of

\[
\text{age to wait time} : a @ x \times b @ y \rightarrow \text{Time}
\]

computing the value of system time-outs for the fusion process from age of input events. It is clear that only an approximate solution can be found here because we
Figure 7. Model of MATIS system including speech and gesture as timed events

are dealing with a large number of different metrics of time when considering the user's mental model of time, the physical performance of the user either as an absolute value or in specific conditions, the timing of input devices, the timing of the physical computer, etc.

Conclusions

A formal approach to describe multimodal interactive systems offers us the possibility of gaining insight into the problem of fusion and of finding solutions by exploiting different viable scenarios whose potential capabilities are well understood. Following this perspective, we have formally specified fusion in its most general form as a many-to-one transformation process. The specification was then used to derive necessary properties of fusion. In particular, we showed that redundancy and incompleteness are unavoidable properties of any fusion process, which the architecture of any multimodal system must resolve after fusion has taken place.

However, there is an inherent difficulty in applying classical formal description techniques of software systems to describe the interaction between users and systems. The reason is obvious: whereas computer systems are finite and deterministic, users are unpredictable, and discussing interactive behaviour is dealing with an unknown quantity. Consequently, while formal system modelling is able to provide a description of the structures and properties that relate to design, it cannot produce an explanation of why a particular option would be most appropriate. Nevertheless, a formal modelling approach helps to clarify design issues and provides insight into potential problems.
Acknowledgments

The work reported in this paper was partially funded by the European Union with grants through Human Capital and Mobility Program, Contract CHRX-CT93-0085, ESPRIT Project Contract MIPS 6542 and ESPRIT Basic Research Action, Contract AMODEUS 7040.

References

de Bakker, de Roever and Rozemberg (eds) (1989) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (Lecture Notes in Computer Science 354) Springer Verlag


Interacting with Computers vol 8 no 2 (1996)


Received January 1995; accepted April 1996