*NICE project (IST-2001-35293)*

# Natural Interactive Communication for Edutainment

# NICE Deliverable D3.6
# Multimodal Input Understanding Module for the First Prototype

*17 October 2003*

*Authors*

*Limsi-Cnrs: Jean-Claude Martin, Sarkis Abrilian, Stéphanie Buisine*

*Teliasonera: Joakim Gustafson, Johan Boye*

*NISLab: Manish Mehta, Andrea Corradini, Niels Ole Bernsen*

*Liquid Media AB: Morgan Fredriksson*

| | |
|---|---|
| **Project ref. no.** | IST-2001-35293 |
| **Project acronym** | NICE |
| **Deliverable status** | Restricted |
| **Contractual date of delivery** | 1 September 2003 |
| **Actual date of delivery** | 17 October 2003 |
| **Deliverable number** | D3.6 |
| **Deliverable title** | Multimodal input understanding module for the first prototype |
| **Nature** | Report |
| **Status & version** | Final |
| **Number of pages** | 33 |
| **WP contributing to the deliverable** | WP3 |
| **WP / Task responsible** | LIMSI-CNRS |
| **Editor** | Jean-Claude Martin |
| **Author(s)** | Jean-Claude Martin, Sarkis Abrilian, Joakim Gustafson, Johan Boye, Andrea Corradini, Morgan Fredriksson, Niels Ole Bernsen, Stéphanie Buisine, Manish Mehta |
| **EC Project Officer** | Mats Ljungqvist |
| **Keywords** | Multimodal input, fusion, gesture recognition, gesture interpretation, reference resolution |
| **Abstract (for dissemination)** | The NICE system aims at enabling a user to interact with characters, objects and environment of a game application via speech combined with 2D gestures achieved with a tactile screen or a pen. This deliverable describes the Input Fusion module which will be used in the 1st NICE prototype for merging the semantic representations generated by the Gesture Interpretation module (GI described in D3.4) and the Natural Language Understanding module (NLU described in D3.5) and for sending the fusion results to the Character and Dialogue Manager modules (D5.1a and b). |

# Table of Contents

# 1 Introduction

## 1.1 Definitions and scope

The NICE system aims at enabling a user to interact with characters, objects and environment of a game application via speech combined with 2D gestures achieved with a tactile screen or a pen. This deliverable describes the Input Fusion module which will be used in the 1st NICE prototype for merging the semantic representations generated by the Gesture Interpretation module (GI described in D3.4) and the Natural Language Understanding module (NLU described in D3.5) and for sending the fusion results to the Character and Dialogue Manager modules (D5.1a and b).

## 1.2 Structure

This deliverable follows the following structure:

- Requirement specifications for the Input Fusion module
- Updated system architecture of the first prototype
- Interactions between the Input Fusion module and the other modules
- Description of the TYCOON technology used within the Input Fusion module

# 2 Requirement specifications for the Input Fusion module

We have not found in the literature any observations of multimodal behaviour of users when interacting with 3D characters using speech and gestures achieved via a 2D media in a conversational game application. Thus, it is not possible to know in advance the exact patterns of multimodal behaviour that will be displayed by users before the testing of the 1st prototype. In this section, we will thus refer to the literature on users' multimodal behaviour (see [Martin et al. 98] for a survey), previous experimental studies on a multimodal map application in which we were involved [Kehler et al. 1998]) and to the results of LIMSI 2D Wizard of Oz (WOZ) experiment achieved at the beginning of the NICE project with a similar but simplified simulated 2D conversational game [Buisine et al. 2003].

## 2.1 Enable switching behaviours / equivalent use of different modalities

A single user might switch between different modalities when activating the same command. For example he would once use gesture-only to move an object and later use speech-only to move another object. The Input Fusion module should thus enable such an "equivalent" access to such commands via several modalities.

## 2.2 Improve monomodal processing and contextual predictions when users display specialised behaviour

Users might always (or very often) use the same modality for a given command. For example they might always use the gesture modality to open a door and seldom use speech. The models used by the speech recogniser and the gesture interpreter should include such knowledge as it can be used to improve monomodal recognition (e.g. some commands do not need to be recognised by speech or at least not very often).

## 2.3 Take into account inter-individual differences

The experiments with the early simulated 2D system [Buisine et al. 2003] showed that one might expect inter-individual differences: children gestured more than adults (Figure 1) and display exploring gestures (e.g. the pen moves around with cursor feedback but without touching the screen). Some users might be more redundant than others, while some other users might prefer the use of speech only [Kheler et al. 1998]. The behaviour model on which the Input Fusion module is to be designed should take into account these differences either within a single user model or with concurrent user models (e.g. one multimodal specification file for children and another one for adult users).

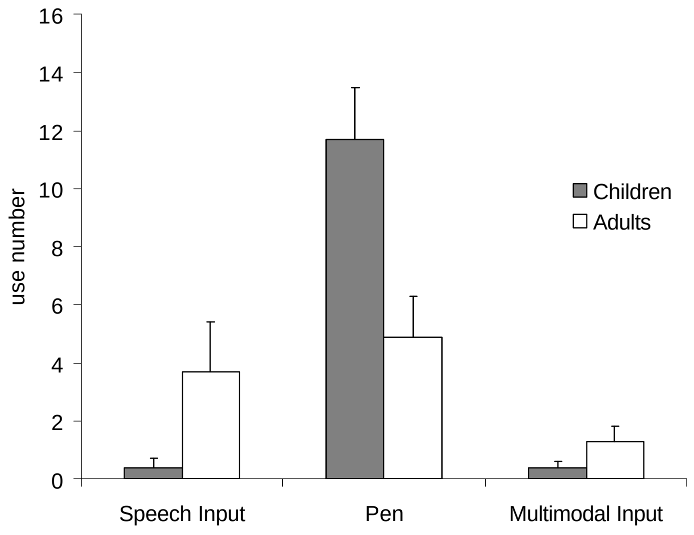


**Figure 1. Mean use number of each modality for the "get into a room" command as a function of the subjects' age [Buisine et al. 2003]**

## 2.4 Expect ambiguous behaviour in each modality / enable mutual disambiguation between modalities

In intuitive Human-Computer interfaces, the user might display ambiguous behaviour in his gestural and spoken behaviour. For example, she might not always point right in the middle of an object but instead gesture between two objects (Figure 2) or she might circle several objects with a single gesture.

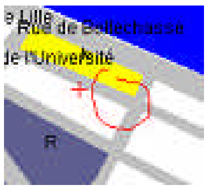


**Figure 2. Example of an ambiguous gesture in a multimodal map application. The user gestured between a museum object (yellow rectangle above) and a restaurant object (purple triangle below).**

She might also in her spoken behaviour refer to an object at different levels of precision: in a multimodal map application [Martin et al. 1998], she might refer to the "Orsay Museum" object by saying "it", "this building", "this museum", or "the Orsay museum".

She might display different multimodal patterns [Siroux et al. 1997]:

- Syntactically correct sentences requiring fusion of spoken and gestural events:
  "Is there any pictures in this book" + <gesture near the red book>
- Syntactically incorrect sentences requiring fusion of complementary spoken and gestural events:
  "Is there any pictures in" + <gesture near the red book>
- Syntactically correct sentences requiring fusion of redundant spoken and gestural events:
  "Is there any pictures in the red book" + <gesture near the red book>

One could also expect meta-communication regarding multimodal communication (e.g. "Can you open the book that I circled").

The input fusion module has to enable cooperation between modalities in order to ensure mutual disambiguation (e.g. combining "this museum" with the gesture displayed above should lead to the selection of the Orsay museum object which is above). The input fusion module should also enable developers to assign different confidence weight to different modalities to solve conflict cases and to manage misrecognised or missing events.

## 2.5 Use several criteria for driving the fusion process including temporal proximity

Temporal proximity was one of the first criteria to be used in multimodal interfaces (i.e. events occurring in the same time window should be merged). Various patterns were observed depending on the media used for gesture, the type of application or the level of detail of the temporal analysis:

- [Oviatt et al. 1997]: pen (writing, drawing) often before speech
- [Mignot and Carbonell 1996]: no obvious systematic temporal relation
- [Catinis et al. 1995]: temporal coincidence often observed

Although this is a very useful criterion for driving the fusion process, considering only this possibility might lead to problems:

- Occasionally, the user might gesture and speak for two independent tasks (e.g. moving an object with gesture while asking the removal of another object with speech [Mignot and Carbonell 1996]), or the user might anticipate the system's response and start a new command before that the previous command is executed by the system.
- The length of the temporal window might be different within different user groups, or within a single user.
- The input fusion module should take into account different processing time in different modalities (e.g. the user might speak and then gesture while the gesture would be recognised by the gesture interpreter before that the utterance is understood by the Natural Language Understanding module).
- The user may speak (e.g. "take that one"), then hesitate, and then gesture.

### 2.6 Expect conversational like multimodal dialogues

In the preliminary 2D WOZ, we observed conversational multimodal patterns such as the following one: before actually taking an object using gesture, some users would ask permission by speech before hand. Such patterns of multimodal dialogue should be used by both the input fusion module and the dialogue modules.

### 2.7 Software issues

An input fusion system has to be easily integrated within a global architecture composed of modality specific modules developed by different partners. New observed patterns of multimodal behaviours occurring during user tests should be easily added. Such combination of modalities should thus be specified in a separated text file instead of being hard-coded in the software.

# 3 Updated system architecture of the first prototype

The Input Fusion module has a central position in the architecture since it receives input from the Natural Language Understanding and Gesture Interpreter modules and send fusion results to Character and Dialogue modules.

This section introduces the system architecture and outlines the servers of the first prototype. The NICE system will involve a number of embodied conversational fairytale characters (in the first prototype there will be two scenarios with only one character in each, HC Andersen in his study or Cloddy Hans by the fairytale machine). To make the animated characters appear lifelike, they have to be autonomous, i.e. they must do things even when the user is not interacting with them. At the same time they have to be reactive and show conversational abilities when the user is interacting with them. To build a system that is both autonomous and reactive at the same time has led to the choice of the event driven, asynchronous system architecture that is shown in figure 3. There are a number of servers in the system:

- Automatic Speech Recognition
- Graphical Recognition
- Natural Language Understanding
- Graphical Interpretation

- Input Fusion
- Dialogue Management (context interpretation and response generation)
- Surface Generation (verbal and non-verbal character actions as well as graphical events)
- Speech Synthesis (speech generation as well as time calculations for animation tags)
- Animation (character animation and virtual world simulation)
- Message Dispatcher (low-level socket handling and high-level message passing rules)

The servers are shown in figure 3:

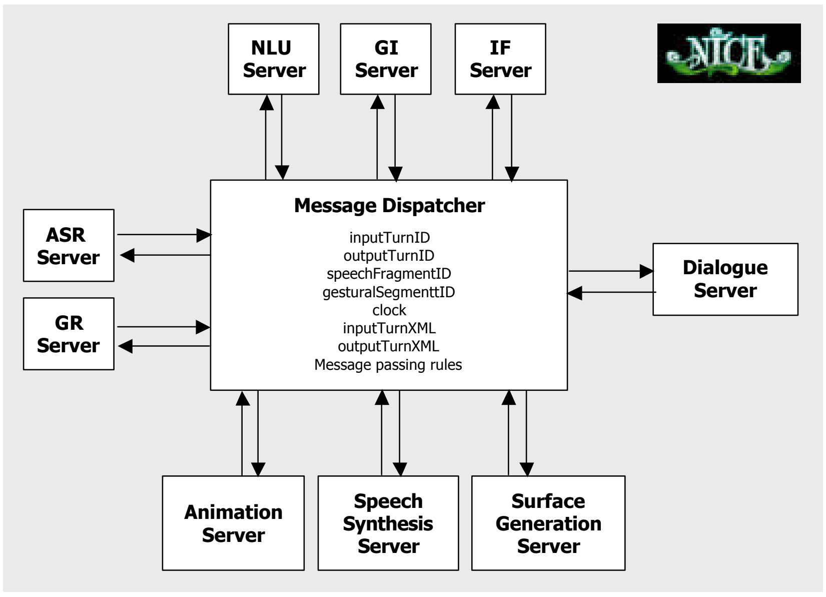


**Figure 3. The system architecture of the first prototype.**

Since the dialogue system components are written at different sites and in different programming languages we have chosen a modular architecture, where modules communicate via central Message Dispatching server, and where all messages are sent in text form over a TCP/IP socket. The Message Dispatcher has two parts:

1) A low-level Broker part that handles the hand shaking and message routing on the TCP/IP socket level.
2) A higher-level Message Dispatcher part that handles the information flow and timings in the system

Since the low-level Broker part has a very basic functionality independent of the typology and information flow in the system architecture, we have chosen to use a publicly available Broker system developed at KTH (http://www.speech.kth.se/broker). There are two advantages with this package:

1) it does not impose a specific system architecture, but simply handles the socket communication and message routing
2) there are brokerClient packages in Java/c++/tcl/perl/prolog that simplify the implementation of message handling in the dialogue component modules

An advantage of routing all messages through a central server is that it makes it possible to log all events in the system, and even replay interactions afterwards to evaluate the system behaviour. This will be a useful tool in the analysis the system performance (both as a whole and the individual system components) that will form the foundation for the iterative system development and improvements.

The high-level part of the Message Dispatcher gives the system an event-driven information flow, where the communication between the servers is coordinated by the Message Dispatcher. All messages are asynchronous, i.e. the sending module sends the message and proceeds to its next task without waiting for an answer. The Message Dispatcher is responsible for coordinating input and output events in the system, by time-stamping all messages from the various modules, as well as associating them to a certain *dialogue turn*.

There are many definitions of the term *turn*. [Allwood 1995] defines it as a speaker's right to the floor, and that this right is regulated by a number of turn management sub-functions that can be expressed verbally or non-verbally. Some of the sub-functions he lists are: means for assigning turns, means for accepting turns, means for taking the turn, means for holding the turn and means for yielding the turn. According to [Sacks et al 1974], turns are composed by smaller units called Turn Construction Units (TCUs), that end in positions where it possible but not obligatory to take the turn (Transition Relevance Places, TRPs). [Traum and Heeman 1997] use a related term that they call *utterance unit*. They list a number of definitions from the literature for utterance units: is uninterrupted speech by a single speaker, has syntactic and/or semantic completion, defines a single speech act, is an intonational phrase, are separated by pauses.

We will define the user turn as all multimodal contributions from the speaker until the system actively takes the turn. The system will decide, using an extension of the method described in [Bell et al 2001], when to take the turn by using pauses, syntax, semantics and discourse history. In the first prototype it will not use intonational cues, but it might be used in the second prototype of the system. This means that a turn, according to our definition, can contain both multiple fragments (i.e. inserted silent pauses) and multiple speech acts (e.g. "good, now put it in dangerous").

The behaviour of the Message Dispatcher is controlled by a set of simple rules, specifying how to react when receiving a message of a certain type from one the modules. Since the Message Dispatcher is connected both to the input channels and the output modalities, it can increase the system's responsiveness by giving fast but simple feedback on input events (for example by sending a request for a eyebrow raise animation to the Animation System as soon as it receives a StartOfSpeech event from the ASR Module). The Message Dispatcher can also increase the system stability using timeouts. For example, if it has sent an ASR string to the NLU and has not received a NLU event within one second, it can take certain actions. Lastly, the architecture above is very modular, in the sense new modules can be added without having to change the previous modules. For example if we would like to add a topic predictor that can use both the ASR string and the NLU analysis, neither the ASR module nor the NLU module have to be updated with information on where to send their results, as all communication goes via the Message Dispatcher.

The messages from the Dialogue Component Modules to the Message Dispatcher are shown in table 1. The resulting action carried out by the Message Dispatcher is shown in the right column.

Table 1. Some of the simple message passing rules in the Message Dispatcher.

| Sender | Message | Action |
|---|---|---|
| ASR | <SpeechDetected/> | SEND <EyebrowRaise/> to the Animation |
| ASR | <result><br>  <asrXML><br></result> | SEND <Nod/> to the Animation<br>INCREASE the speechFragmentID<br>ADD asrXML as fragment in the current inputTurnXML<br>SEND the inputTurnXML to the NLU<br>SEND listen and the ASRgrammar to the ASR |
| NLU | <result><br>  <nluXML><br></result> | UPDATE inputTurnXML with nluXML<br>IF giXML empty SEND inputTurnXML to the DM<br>ELSE SEND inputTurnXML to the IF |
| GR | <GestureDetected/> | SEND <EyebrowRaise/> to Animation |
| GR | <result><br>  <grXML><br></result> | INCREASE graphicalSegmentID<br>ADD grXML as segment in the current inputTurnXML<br>SEND the inputTurnXML to the GI<br>SEND listen to the GR |
| GI | <result><br>  <giXML><br></result> | UPDATE the inputTurnXML with giXML<br>SEND the inputTurnXML to the IF |
| IF | <result><br>  <ifXML><br></result> | UPDATE the inputTurnXML with ifXML<br>SEND the inputTurnXML to the DM |
| DM | <feedback><br>  <feedbackXML><br></feedback> | SEND the feedbackXML to the Animation |
| DM | <action><br>  <outputTurnXML><br></action> | SEND <TakeTurn/> to the Animation<br>INCREASE the inputTurnID<br>RESET the speechFragmentID<br>RESET the graphicalSegmentID<br>CLEAR the inputTurnXML<br>SEND the outputTurnXML to the Surface Generation |
| Surface Generation | <action><br>  <outputTurnXML><br></action> | UPDATE the outputTurnXML<br>SEND the outputTurnXML to the Speech Synthesizer |
| Speech Synthesizer | <action><br>  <outputTurnXML><br></action> | UPDATE the outputTurnXML<br>SEND the outputTurnXML to the Animation |
| Animation | <done><br>  <outoutTurnID> | SEND <done><outputTurnID></done> to the DM |

|  |  |  |
|---|---|---|
|  | <done> |  |

The rules in table 1 are  written in pseudo-code in order to make them readable, Furthermore, they are the subset of all rules that are needed to make it possible to implement the information flow in figure 4.

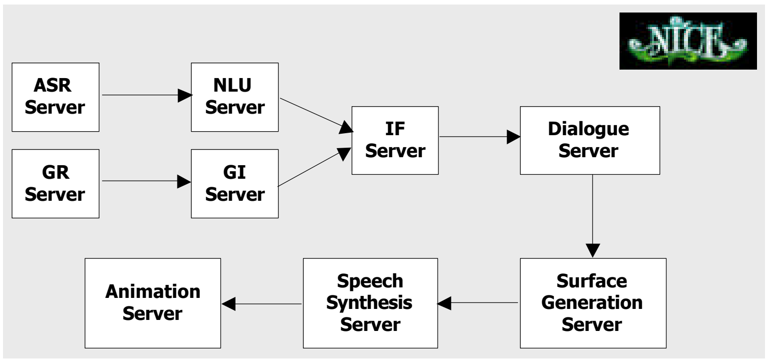


**Figure 4. The information flow in the first prototype.**

The chosen architecture makes it possible to implement complicated turn handling strategies. The Message Dispatcher associates all events from the dialogue components to a certain turn. This makes it possible to handle fragmented speech as well as speech+gesture input from the user. It also makes it possible to generate fragmented utterances from the system. The dialogue server can generate a number of fragments utterances and utterances as the result of one spoken user utterances, for example:

| User | System (Cloddy Hans) |
|---|---|
| "Take the knife" | **ASR**(StartOfSpeech) -> listening gesture<br>**ASR**(EndOfSpecch) -> thinking gesture<br>**DM**(Feedback) -> let me see...<br>**DM**(Action(clarify(knife))) -> do you want me to take this <point at knife>?<br>**Timeout**(repeat(Action(clarify(knife)))) -> do you want me to take the knife? |
| "Yes" | **DM**(Action) ->  ok I'll do that <do physical action><br>**Animation**(Action,done)) -> SEND action done to DM |

In this example the system will first give verbal and non-verbal feedback. Then it will send a clarification question, which is repeated since the user did not respond within the time specified in the Message Dispatcher's timeout rules. All analyses relevant to a turn will be saved by the Message Dispatcher this structure will be sent to the DM in the inputTurnXML-format outlined in figure 5.

```
<?xml version="1.0"?>
<NiceXML>
  <Turn>
    <ID></ID><Speaker></Speaker><Listener></Listener>
    <ASR>
      <Fragment>
        <ID></ID><Score></Score><StartTime></StartTime><EndTime></EndTime>
        <Word><ID></ID><Score></Score></Word>
        <Word><ID></ID><Score></Score></Word>
      </Fragment>
      <Fragment>
        <ID></ID><Score></Score><StartTime></StartTime><EndTime></EndTime>
        <Word><ID></ID><Score></Score></Word>
        <Word><ID></ID><Score></Score></Word>
      </Fragment>
    </ASR>
    <NLU></NLU>
    <GR>
      <Segment>
        <ID></ID><Score></Score><StartTime></StartTime><EndTime></EndTime>
        <shape></shape><begin /><end /><twoDboundingBox /><direction />
       </Segment>
       <Segment>
        <ID></ID><Score></Score><StartTime></StartTime><EndTime></EndTime>
        <shape></shape><begin /><end /><twoDboundingBox /><direction />
       </Segment>
    </GR>
    <GI></GI>
    <IF></IF>
  </Turn>
</NiceXML>
```

**Figure 5. The layout of the inputTturnXML in the first prototype.**

The approach we have chosen for turn handling builds on Telia's experiences from the AdApt project. The AdApt system had methods for handling fragmented utterances from the user [Bell et al 2001], and it could generate multimodal feedback that made it possible to generate fragmented utterances from the system [Gustafson et al 2002]. In the NICE system we are using extensions of these turn-handling methods. It is the Dialogue Manager server that decides if a turn is complete, while it is up to the Message Dispatcher to handle the actual timeouts needed to wait for more input in cases when a turn was not considered complete. To exemplify the turn-handling algorithm, let us consider a situation where Cloddy Hans stands in front of the shelf with objects. Lets consider two cases where Cloddy Hans says one of two things:

1) Which one did you mean?
2) What do you want me to do?

If the user selects the knife, a graphical event will be generated and sent to the DM as something like *selection(knife)*. In the context of utterance (1) above, this graphical event will be considered a complete turn by the Dialogue Manager server. This means that it will tell the Message Dispatcher to generate a *positive feedback gesture,* while at the same time plan a response. The generated response would then likely be to let Cloddy say “Ok”, and then let him perform the physical action of picking up the knife.

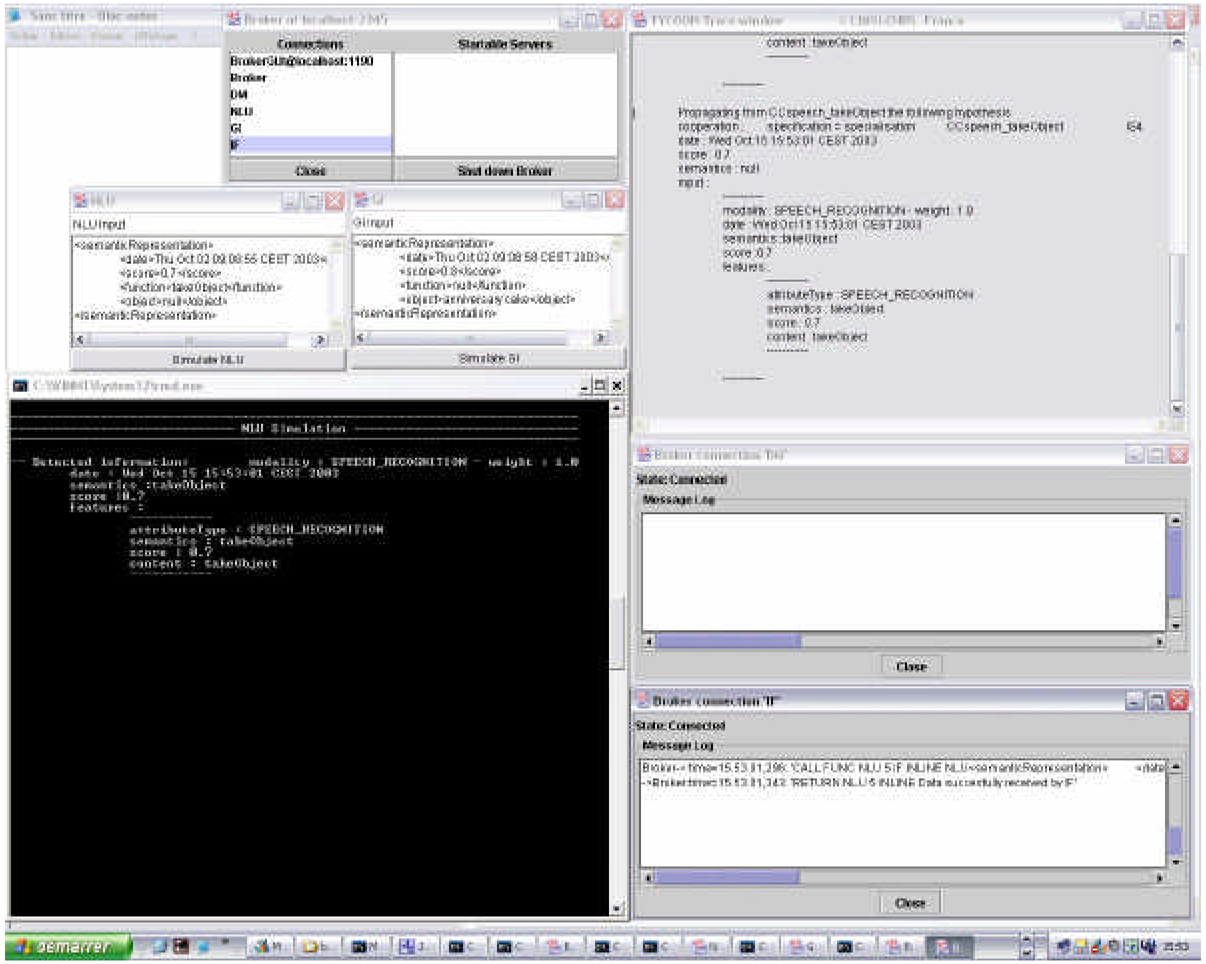
In the context of utterance (2) above, the Dialogue Manager will not consider the user contribution as a complete turn, since  the intention of the user is not clear. It will then tell the Message Dispatcher to generate a *continued attention gesture,* while at the same time produce a *request_for_clarification* response to this incomplete turn. The Message Dispatcher will then use temporal rules to decide how long to wait for more input from the user (say up to four seconds).

If no more input has arrived within four seconds, the Message Dispatcher will use the *request_for_clarification* response generated by the Dialogue Manager, and send this response to Surface Generation and Speech Synthesis. However, let us now consider the alternative case where  the user after two seconds says “put it in the ‘dangerous’ slot”. The Message Dispatcher will now add the new speech fragment to the current inputTurnXML, which now contains both a speech fragment and a graphical segment. Analysis and fusion will now generate the combined interpretation *request(user, putdown(cloddy, knife, dangerous)),* which will be sent to the Dialogue Manager server. The generated response might then be to let Cloddy say “Ok” and then let him perform the physical action of putting the knife in the ‘dangerous’ slot in the fairy-tale machine.

# 4 Interactions between the Input Fusion module and the other modules

The following figures illustrate how we have integrated the Input Fusion module within a preliminary architecture using the Broker described in the previous section.

In these testing examples, a simple XML output is provided by the IF module but can be easily extended to include further information needed by the Character Module and Dialogue Manager (see D5.1a and b) such as the confidence score in each modality or the gesture shape.



```
<semanticRepresentation>
        <date>Thu Oct 02 09:08:56 CEST 2003</date>
        <score>0.7</score>
        <function>takeObject</function>
        <object>null</object>
</semanticRepresentation>
```

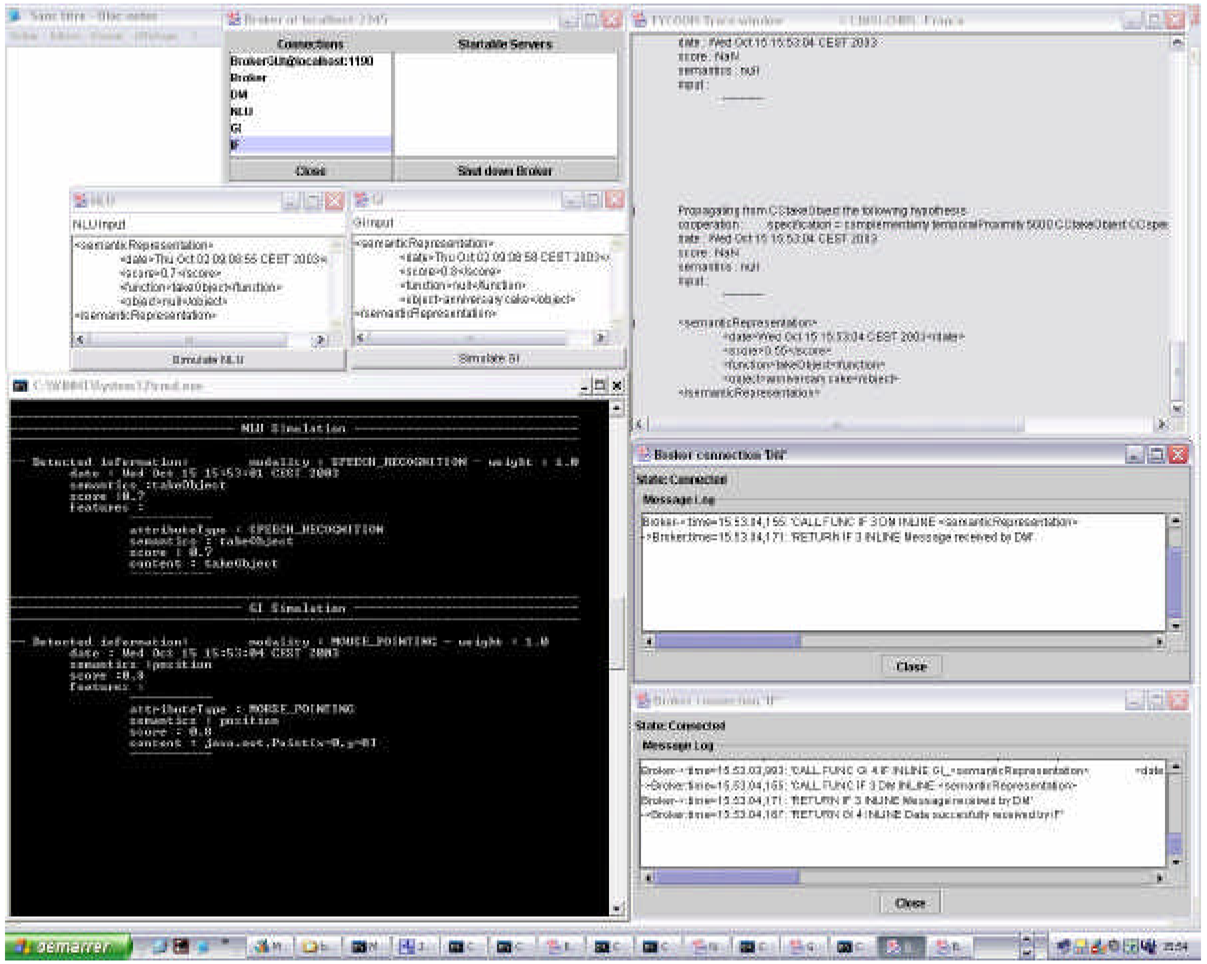
```
<semanticRepresentation>
        <date>Thu Oct 02 09:08:58 CEST 2003></date>
        <score>0.8</score>
        <function>null</function>
        <object>anniversary cake</object>
</semanticRepresentation>
```

```
------------------------------------ NLU Simulation ------------------------------------

-- Detected information:        modality : SPEECH_RECOGNITION - weight : 1.0
        date : Wed Oct 15 15:53:01 CEST 2003
        semantics :takeObject
        score :0.7
        features :
                ---------------
                attributeType : SPEECH_RECOGNITION
                semantics : takeObject
                score : 0.7
                content : takeObject
                ---------------
```

```
            content : takeObject
            --------

        --------

Propagating from CCspeech_takeObject the following hypothesis
cooperation :      specification = specialisation      CCspeech_takeObject       64
date : Wed Oct 15 15:53:01 CEST 2003
score : 0.7
semantics : null
input :
        --------
        modality : SPEECH_RECOGNITION - weight : 1.0
        date : Wed Oct 15 15:53:01 CEST 2003
        semantics :takeObject
        score :0.7
        features :
            --------
            attributeType : SPEECH_RECOGNITION
            semantics : takeObject
            score : 0.7
            content : takeObject
            --------

        --------
```

```
Broker<- time=15:53:01,296: 'CALL FUNC NLU 5 IF INLINE NLU<semanticRepresentation>      <date
->Broker time=15:53:01,343: 'RETURN NLU 5 INLINE Data successfully received by IF'
```

**Figure 6: Screendump of the test of the Input Fusion module with the KTH broker. Two simple clients have been developed for typing in XML messages that may come out of the Natural Language Understanding and the Gesture Interpreter modules (NLU and GI in the two small windows above left). When the button "simulate NLU" is pressed, an XML message simulating the recognition of the spoken command "takeObject" is sent to the IF module via the broker (see "Tycoon trace Window" above).**



```
BrokerGUI@localhost:1190
Broker
DM
NLU
GI
IF
```

NLU Input

```
<semanticRepresentation>
        <date>Thu Oct 02 09:08:56 CEST 2003</date>
        <score>0.7</score>
        <function>takeObject</function>
        <object>null</object>
</semanticRepresentation>
```

GI Input

```
<semanticRepresentation>
        <date>Thu Oct 02 09:08:58 CEST 2003</date>
        <score>0.8</score>
        <function>null</function>
        <object>anniversary cake</object>
</semanticRepresentation>
```

```
------------------------------------ NLU Simulation ------------------------------------

-- Detected information:        modality : SPEECH_RECOGNITION - weight : 1.0
      date : Wed Oct 15 15:53:01 CEST 2003
      semantics :takeObject
      score :0.7
      Features :
                ----------
                attributeType : SPEECH_RECOGNITION
                semantics : takeObject
                score : 0.7
                content : takeObject
                ----------

------------------------------------ GI Simulation ------------------------------------

-- Detected information:        modality : MOUSE_POINTING - weight : 1.0
      date : Wed Oct 15 15:53:04 CEST 2003
      semantics :position
      score :0.8
      Features :
                ----------
                attributeType : MOUSE_POINTING
                semantics : position
                score : 0.8
                content : java.awt.Point[x=0,y=8]
                ----------
```

```
date : Wed Oct 15 15:53:04 CEST 2003
score : NaN
semantics : null
input :
        ------

Propagating from CCtakeObject the following hypothesis
cooperation:      specification = complementary temporalProximity 5000 CCtakeObject CCspe
date : Wed Oct 15 15:53:04 CEST 2003
score : NaN
semantics : null
input :
        ------

<semanticRepresentation>
        <date>Wed Oct 15 15:53:04 CEST 2003</date>
        <score>0.55</score>
        <function>takeObject</function>
        <object>anniversary cake</object>
</semanticRepresentation>
```

Broker connection 'DM'

State: Connected

Message Log

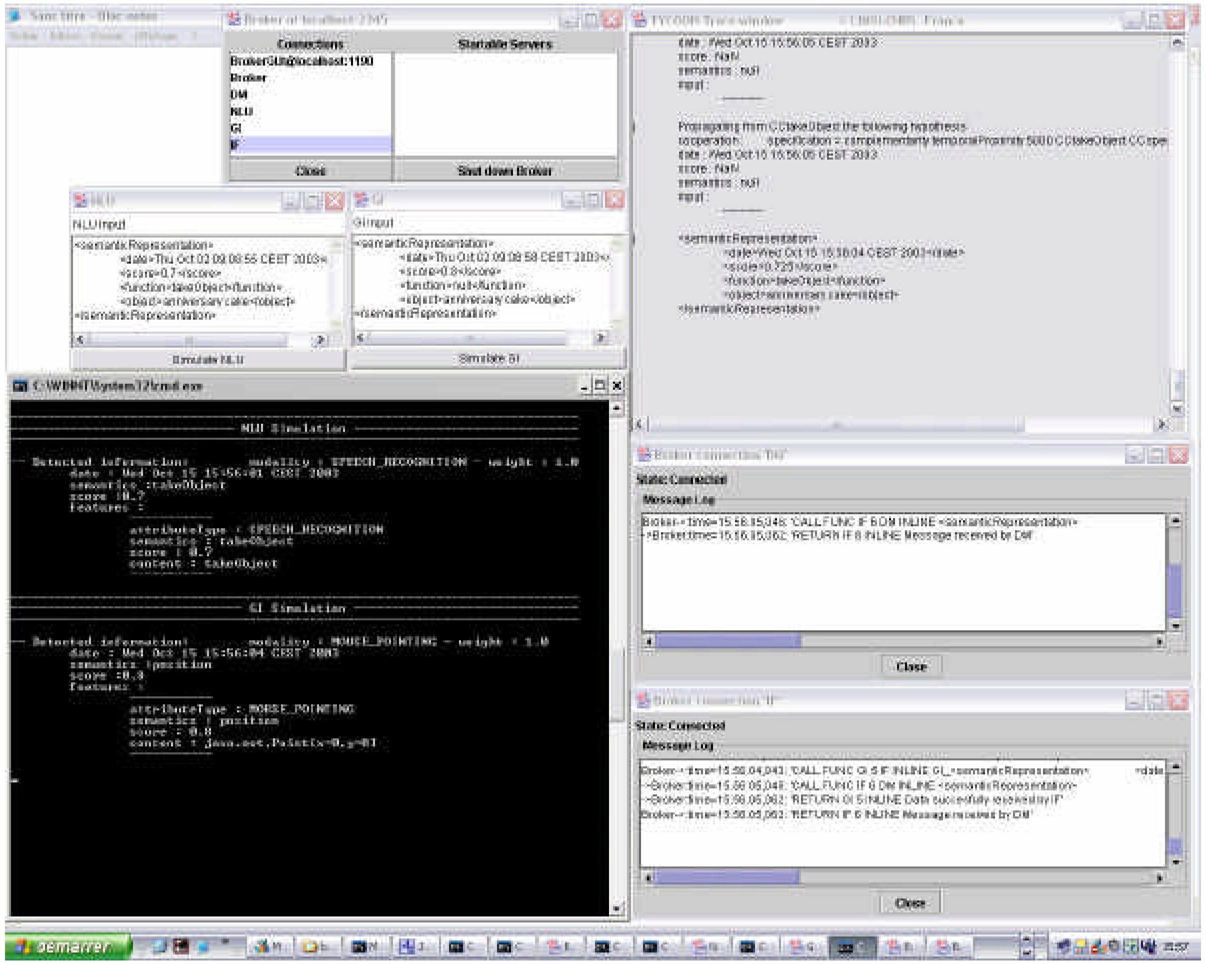
```
Broker->time=15.53.04,155: 'CALL FUNC IF 3 DM INLINE <semanticRepresentation>
->Broker:time=15.53.04,171: 'RETURN IF 3 INLINE Message received by DM'
```

State: Connected

Message Log

```
Broker->time=15.53.03,993: 'CALL FUNC GI 4 IF INLINE GI_<semanticRepresentation>        <date
->Broker:time=15.53.04,155: 'CALL FUNC IF 3 DM INLINE <semanticRepresentation>
Broker->time=15.53.04,171: 'RETURN IF 3 INLINE Message received by DM'
->Broker:time=15.53.04,187: 'RETURN GI 4 INLINE Data successfully received by IF'
```

**Figure 7: Following figure 6, the GI module simulates the recognition of a gesture around the object "anniversary cake" and sends the corresponding XML message to IF. This event is merged with the previous NLU input and a XML message is produced by the IF (Trace Window above right) and sent to the simple client simulating the Dialog Manager (middle right). The score of each modality can be combined with criteria such as temporal proximity or number of modalities providing information to get a multimodal recognition score (0.55 in this example).**



```
<semanticRepresentation>
        <date>Thu Oct 02 09:08:56 CEST 2003</date>
        <score>0.7</score>
        <function>takeObject</function>
        <object>anniversary cake</object>
</semanticRepresentation>
```

```
<semanticRepresentation>
        <date>Thu Oct 02 09:08:58 CEST 2003</date>
        <score>0.8</score>
        <function>null</function>
        <object>anniversary cake</object>
</semanticRepresentation>
```

```
------------------------------ NLU Simulation ------------------------------

-- Detected information:        modality : SPEECH_RECOGNITION - weight : 1.0
        date : Wed Oct 15 15:56:01 CEST 2003
        semantics :takeObject
        score :0.7
        features :
                ---------------
                attributeType : SPEECH_RECOGNITION
                semantics : takeObject
                score : 0.7
                content : takeObject
                ---------------

------------------------------ GI Simulation ------------------------------

-- Detected information:        modality : MOUSE_POINTING - weight : 1.0
        date : Wed Oct 15 15:56:04 CEST 2003
        semantics :position
        score :0.8
        features :
                ---------------
                attributeType : MOUSE_POINTING
                semantics : position
                score : 0.8
                content : java.awt.Point[x=0,y=0]
                ---------------
```

```
date : Wed Oct 15 15:56:05 CEST 2003
score : NaN
semantics : null
input :
        -------

Propagating from CCtakeObject the following hypothesis
tcoperation :      specification = complementarity temporalProximity 5000 CCtakeObject CCspe
date : Wed Oct 15 15:56:05 CEST 2003
score : NaN
semantics : null
input :
        -------

<semanticRepresentation>
        <date>Wed Oct 15 15:56:04 CEST 2003</date>
        <score>0.725</score>
        <function>takeObject</function>
        <object>anniversary cake</object>
</semanticRepresentation>
```

```
Broker->time=15:56:05,046: 'CALL FUNC IF 6 DM INLINE <semanticRepresentation>
->Broker:time=15:56:05,062: 'RETURN IF 6 INLINE Message received by DM'
```

```
Broker->time=15:56:04,043: 'CALL FUNC GI 5 IF INLINE GI_<semanticRepresentation>        <date
->Broker:time=15:56:05,046: 'CALL FUNC IF 6 DM INLINE <semanticRepresentation>
->Broker:time=15:56:05,062: 'RETURN GI 5 INLINE Data successfully received by IF'
Broker->time=15:56:05,062: 'RETURN IF 6 INLINE Message received by DM'
```

**Figure 8: A example similar to the previous one except that in this simulation the user said "take the anniversary cake" specifying both the function and an object in speech. In this case, the spoken and gestural input are also merged but the final multimodal recognition score is higher (0.725) than in the previous example since more information was provided by the speech modality.**

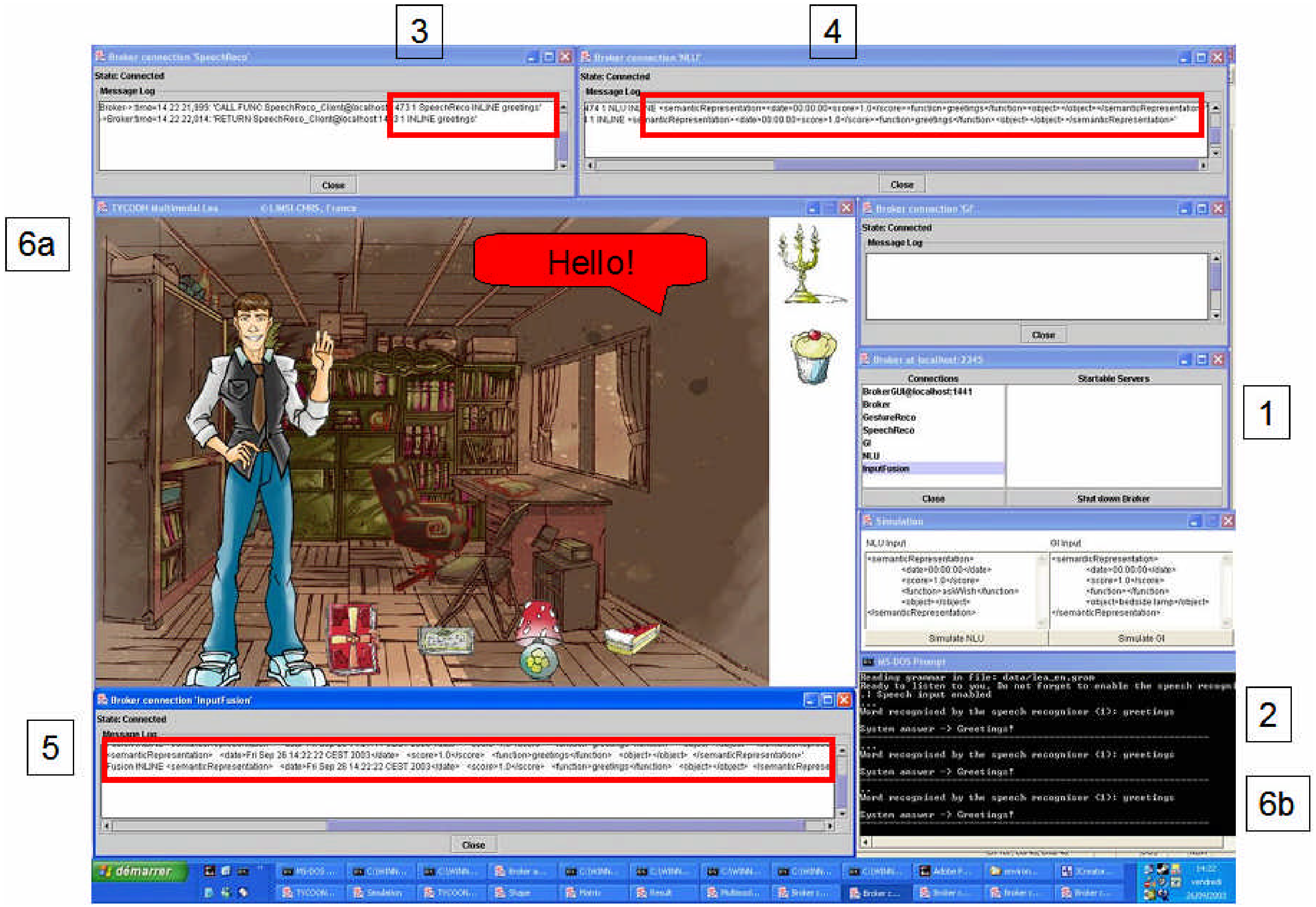
We have also tested the IF with the 2D testing environment we had developed for the 2D Wizard of Oz experiment. This 2D testing environment was designed for the preliminary collection of user's behaviour via a Wizard of Oz which was carried out early in the project before the NICE 3D environments were designed and implemented (D3.3 April 2003). In the forthcoming 1st NICE prototype, the GI and IF modules will be connected with the 3D environments (the Hans Christian Andersen study and the fairy tale world).

For individual testing purposes, we have used the IBM ViaVoice development kit[1] for speech recognition with which had used for fast specification of grammar and connection with Java software using Java Speech API.  The testing grammar was defined out of data produced by the experimental Woz studies with the same 2D application. The French spoken corpus contained 1250 multimodal or spoken utterances. The grammar contains 120 rules which were translated from a summary of the corpus. The grammar is provided in appendix. The IBM ViaVoice speech synthesis was also used for the 2D test. Since we use the Broker architecture and XML messages, the IBM ViaVoice client used for fast prototyping purposes can be easily replaced with another client using SpeechPearl recognition engine (see D3.1 Trained Acoustic Models for Swedish Recogniser).

In the screendump displayed in figure 9 one can see the different steps of processing.

- 1 : the SR, NLU, GR, GI, IF and 2D Lea test environment modules connect to the Broker
- 2 :  the user said "Hello" which was associated with the vocal command "greetings" in the IBM ViaVoice grammar
- 3 : the SpeechReco module sends to the Broker the recognised command : greetings ; this message is forwarded to the NLU module
- 4 : a semantic representation in XML is sent by the NLU to the IF module via the broker
- 5 : the IF module forward (since no gestures) to the 2D Lea environment application
- 6 : the 2D Lea environment application activates the character's response (spoken and displayed utterance : "Hello")

[1] IBM ViaVoice developer's corner:
http://www-3.ibm.com/software/voice/viavoice/dev/index.shtml?Open&ca=daw-prod-cantts



Broker connection 'SpeechReco'
State: Connected
Message Log

```
Broker->time=14.22.21,895: 'CALL FUNC SpeechReco_Client@localhost 1473 1 SpeechReco INLINE greetings'
<-Broker time=14.22.22,014: 'RETURN SpeechReco_Client@localhost 1473 1 INLINE greetings'
```

Broker connection 'NLU'
State: Connected
Message Log

```
474 1 NLU INLINE <semanticRepresentation><date>00:00:00<score>1.0</score><function>greetings</function><object></object></semanticRepresentation>'
4 1 INLINE <semanticRepresentation><date>00:00:00<score>1.0</score><function>greetings</function><object></object></semanticRepresentation>'
```

Broker connection 'GI'
State: Connected
Message Log

Broker at localhost:2345

Connections
BrokerGUI@localhost:1441
Broker
GestureReco
SpeechReco
GI
NLU
InputFusion

Startable Servers

Simulation

NLU Input

```
<semanticRepresentation>
        <date>00:00:00</date>
        <score>1.0</score>
        <function>askWish</function>
        <object></object>
</semanticRepresentation>
```

GI Input

```
<semanticRepresentation>
        <date>00:00:00</date>
        <score>1.0</score>
        <function></function>
        <object>bedside lamp</object>
</semanticRepresentation>
```

MS-DOS Prompt

```
Reading grammar in file: data/lea_en.gram
Ready to listen to you. Do not forget to enable the speech recogn
.! Speech input enabled
...
Word recognised by the speech recogniser (1): greetings

System answer -> Greetings!
------------------------------------------------------------
...
Word recognised by the speech recogniser (1): greetings

System answer -> Greetings!
------------------------------------------------------------
..
Word recognised by the speech recogniser (1): greetings

System answer -> Greetings!
------------------------------------------------------------
```

Broker connection 'InputFusion'
State: Connected
Message Log

```
<semanticRepresentation>   <date>Fri Sep 26 14:22:22 CEST 2003</date>   <score>1.0</score>   <function>greetings</function>   <object></object>   </semanticRepresentation>'
Fusion INLINE <semanticRepresentation>   <date>Fri Sep 26 14:22:22 CEST 2003</date>   <score>1.0</score>   <function>greetings</function>   <object></object>   </semanticReprese
```

**Figure 9. Screendump of a test of the IF module within the 2D environment. Other screendumps with gesture and fusion examples are provided in appendix.**

# 5 The TYCOON technology used within the IF module

We describe in this section the TYCOON technology that was used within the IF module. This technology includes a specification language and a software architecture for the recognition of multimodal commands.

The specification files used for the 2D tests are provided in appendix (the speech grammar, the specification of 2D "referenceable" objects, the specification of expected cooperations between modalities).

In this section, we describe each of the components of a multimodal application in the Tycoon context:

- a monomodal application defined by a set of command templates and a set of objects (e.g. a map with a classical graphical user interface),
- a set of referenceable graphical objects,
- a module managing each modality,
- a multimodal recognition engine which makes use of a multimodal specification file containing the expected cooperations between modalities in order to merge events detected by each monomodal module and execute application commands.

## 5.1 Monomodal application

Tycoon has been initially designed for command language applications. The application interface is represented by a list of commands. Each command is represented by a template made of a command name and a list of couples binding values to parameters:

```
    CommandName [ Parameter [Value]* ]*
```

Such a model has also been used in other multimodal systems such as [Teil and Bellik 2000, Faure and Julia 1994]. A command is *recognized* when the system has recognized the command but still needs to solve references in order to bind values to the parameters. After this reference resolution process is over, the command is said to be *interpreted*. The values attached to the parameters can either be constants or referenceable objects.

## 5.2 Referenceable objects

One feature of multimodality is the possibility to refer to graphical objects either through speech, gesture or a combination of both. In this perspective, we have introduced the notion of *referenceable object.* A *referenceable object* embeds an object of the graphical application with knowledge on how to refer to this object (with linguistic or non-linguistic means). In a classical map application, such objects are hotels, restaurants or streets that the user is able to refer to with commands. In Tycoon, the reference resolution process is based on the computation of *salience values* (already used by [Huls et al. 1995] in the domain of multimodal systems). The salience value of a referenceable object in a modality gives an idea of how much this object is explicitly referred to in this modality. It can take any value between 0 and 1. A global multimodal salience value is computed across several modalities to find the best candidate for reference resolution. Two referenceable objects may have the same salience in one modality (e.g., graphics). Yet, this ambiguity might disappear when considering the salience of these objects in another modality (e.g., gesture). Redundancy is involved when multimodal salience is very high, whereas complementarity is involved when multimodal salience is very low.

A referenceable object is thus a graphical object (from the monomodal application) which can be referred to by one or several modalities such as speech and gesture thanks to the definition of *properties* and *salience* values. A property is a pair (label, value). The label is an attribute of the graphical objects of the application. In a map application, the main graphical objects are buildings and streets. The properties of such objects are "name", "type", "address", "location". The advantage of this representation of properties is that references to graphical objects become possible through their name ("Orsay museum"), their type ("this museum"), their address ("the museum which is in rue de la Légion d'Honneur"), or a combination ("this museum" + gesture). We have included in the notion of referencable object a set of salience values for each modality. The salience value of a referenceable object in a modality gives an idea of how much this object is explicitly referred to in this modality. This salience value is computed thanks to modality specific rules. For instance, the recognition of the word "hotel" by the speech recogniser increases the salience of all hotels located on the map. Table 2 provides informal definitions of such rules for updating the salience of referencable objects when events are detected. These rules do not depend on the application but are intended to be generic. A global multimodal salience is computed for each object as a weighted sum of the salience of this object in each modality. This multimodal salience is used to find the best candidate for reference resolution. In case of ambiguity, two referencable objects may have the same salience in the same modality. Yet, this ambiguity might be removed when considering the salience of these objects on other modalities.

Would several objects have exactly the same multimodal salience value, in the current version, one object is selected randomly. We can also add dialogue or another modality which was not considered for the computation of the multimodal salience value such as the graphical context (e.g. a big object in the middle of the screen would be selected).

Tables 3 and 4 provide two examples of salience computation. In Table 3 the Orsay Museum was fully specified without any ambiguity via speech. Thus, the salience of this museum in the speech modality is 1. Since no gesture was detected, the salience of the Orsay Museum in the gesture modality is 0. Since this museum is between the centre of the map and the border, the salience in the graphics modality is 0.6. A multimodal salience value is computed as a weighted sum of the three values. The result (0.178) is higher than the salience of any other object. In Table 4, the user said “show me some information about this museum”. Thus the salience value of the Orsay Museum in the speech modality is lower (0.5 for every museum of the map). But an arrow gesture was also detected near the Orsay Museum. The gesture salience value is computed for each object as a function of the distance between the object and the focus point of the gesture. By combining these salience values, the multimodal value of the Orsay Museum is higher than the salience of any other object and information about this museum is finally displayed.

**Table 2**
**Informal definition of rules for updating the salience of objects when an event is detected**

| | |
|---|---|
| **Spoken event** | If the recognised sentence contains the unique name of an object (e.g. "the Orsay museum"), set the salience in the speech modality of this object to the score provided by the speech recogniser.<br>If the recognised sentence contains the value of a property of an object (e.g. "the museum"), increase the salience in the speech modality of all referencable object having the same property value (e.g. all the museums) taking into account the score provided by the speech recogniser. |
| **Gesture event** | Set the salience in the gesture modality as a function of the distance between the location of the object and the focus point of the recognised gesture. |
| **Graphics** | Set the salience in the graphics modality as a function of the distance between the location of the object and the centre of the screen. |
| **History** | After the recognition of a command, the salience of objects referred to in this command is decreased by a forgetting factor. |

**Table 3**
**Salience values of the "Orsay museum" object in the case the user said "Give me some information about the Orsay museum" and did not gesture**

| Speech Salience | Gesture Salience | Graphics Salience | Multimodal salience |
|---|---|---|---|
| 1 | 0 | 0.624 | 0.178 |

**Table 4**
**Salience values of the "orsay museum" object in the case the user said "Show me some information about this museum" and did a circling gesture**

| Speech Salience | Gesture Salience | Graphics Salience | Multimodal salience |
|---|---|---|---|
| 0.5 | 0.994 | 0.479 | 0.257 |

## 5.3 Multimodal Recognition Engine

The specification file describing the expected cooperations between modalities is parsed by a multimodal recognition module which builds a hybrid symbolic-connectionist network inspired from Guided Propagation Networks which enable coincidence and sequence detection thanks to propagation of internal signals between connected processing units [Béroule 1989]. We have adapted such networks by 1) dedicating some nodes to the processing of each type of cooperation between modalities, 2) by introducing the propagation of symbolic structures for the management of hypothesis regarding interpretation of detected events.

When the systems starts, the specification file described above is parsed and a network is built. The network is composed of two types of nodes. Information nodes are associated to events detected on each modality. Cooperation nodes are associated to each expected cooperation between modalities. During execution, the following algorithm is used by the multimodal recognition engine:

```
Parse the multimodal specification file
Create the multimodal cooperation network
Create the set of referenceable objects
Init the monomodal application
While the user does not ask for exit
        If  an event is detected on a modality
            Update the salience of all referenceable objects according to this event
            Create an information object associated to this event
            Put it in the output of the information node managing this event
        For each cooperation node N in the network
            Compute a boolean toBeActivated as a function of (type of cooperation, output of the input
            nodes to node N)
            If toBeActivated is true
                Build a hypothesis object, compute its score and put it in the output of node N
                If N is a terminal node (associated to an application command)
                    Solve references if needed
                    Execute the command
```

The activity level of a node at the end of a multimodal command pathway corresponds to the way an occurrence of this command matches its internal representation. This "matching score" accounts for the degree of distortions undergone by the reference multimodal command, including noisy, missing or inverse components. Initially applied to robust parsing this feature has been adapted to multimodality. This quantified matching score results of three properties of GPN:

- the amplitude of the signal emitted by a speech detector is proportional to the recognition score provided by the speech recogniser,
- a multimodal unit can be activated even if some expected events are missing (in this case, the amplitude of the signal emitted by this variable is lower than the maximum),
- the higher the temporal distortion between two events, the weaker their summation (or note of temporal proximity), because of the decreasing shape of the signals.

## 5.4 Specification of cooperations between modalities

A file describing the expected cooperations between the modalities is associated to each multi-modal application. We have defined a command language for specifying such cooperations. This language is based on the Tycoon typology of types of cooperation (equivalence, specialisation, complementarity, redundancy). The value of temporal parameters is also included in the specification (i.e. coincidence duration, sequence delays).

## 6 References

Allwood, J. "Reasons for management in dialog" in Beun, R.J., Baker, M. and Reiner, M. (eds.) Dialogue and Instruction. Springer-Verlag.pp 241-50, 1995.

Bell, L, Boye, J, and Gustafson, J. "Real-time Handling of Fragmented Utterances", Proceedings of NAACL 2001.

Béroule, D. “Management of time distorsions through rough coincidence detection”. Proc. EuroSpeech'89, 1989, pp. 454-457.

Buisine, S., Martin, J.-C. (2003). “Experimental Evaluation of Bi-directional Multimodal Interaction with Conversational Agents”. Proceedings of the the Ninth IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'2003), September 1-5, 2003 - Zürich, Switzerland. http://www.interact2003.org/

Catinis, L. and Caelen, J. Analyse du comportement multimodal de l’usager humain dans une tache de dessin. Actes des 7èmes Journees sur l’Ingénierie de l’Interaction Homme Machine (IHM’95). In French. 1995.

Corradini, A., Mehta, M., Bernsen, N.O., Martin, J.-C., Abrilian, S. “Multimodal input fusion in human-computer interaction - On the Example of the NICE Project”. Proceedings of the conference held in in Yerevan, Armenia. NATO-ASI series, Kluwer, 2003.

Faure, C. and Julia, L. “An Agent-Based Architecture for a Multimodal Interface”. Proc. AAAI Intelligent Multi-Media Multi-Modal Systems Symposium, Stanford University. 1994.

Gustafson, J, Bell, L, Boye, J, Edlund, J and Wiren, M. "Constraint Manipulation And Visualization In A Multimodal Dialogue System", Proceedings of the ISCA Workshop Multi-Modal Dialogue in Mobile Environments Kloster Irsee, Germany. 2002.

Huls, C., Claassen, W. and Bos, E. “Automatic Referent Resolution of Deictic and Anaphoric Expressions”. Computational Linguistics, vol. 21 (1), pp. 59-79. 1995.

Kehler, A., Martin, J.C., Cheyer, A., Julia, L., Hobbs, J. & Bear, J. “On Representing Salience and Reference in Multimodal Human-Computer Interaction” Proceedings of the AAAI'98 workshop on Representations for Multi-modal Human-Computer Interaction. July 26-27, 1998, Madison, Wisconsin. USA.

Martin, J.C., Julia, L. & Cheyer, A. “A Theoretical Framework for Multimodal User Studies”. Proceedings of the Second International Conference on Cooperative Multimodal Communication, Theory and Applications (CMC'98), 28-30 January 1998, Tilburg, The Netherlands

Mignot, C. & Carbonell, N. "Commandes orales et gestuelles: une étude empirique." Techniques et Sciences Informatiques 15(10): 1399-1428. 1996.

Oviatt, S., De Angeli, A. & Kuhn, K. Integration and synchronization of input modes during multimodal human-computer interaction. Human Factors in Computing Systems (CHI'97), New York, ACM Press. 1997.

Sacks, H., Schegloff, E. & Jefferson G. “A simplest systematics for the organisation of turn-taking for conversation”. Language 50, pp. 696–735. 1974.

Siroux, J., Guyomard, M., Multon, F. & Remondeau, C.” Multimodal references in GEORAL TACTILE”. Workshop "Referring phenomena in a multimedia context and their computational treatment" held in cunjunction with ACL/EACL'97, Madrid, Spain. 1997.

Teil, D. and Bellik, Y. “Multimodal Interaction Interface Using Voice and Gesture”. The Structure of Multimodal Dialog II, M. M. Taylor, F. Néel, and D. G. Bouwhuis, Eds. 2000.

Traum, D and Heeman, P. "Utterance Units in Spoken Dialogue," Dialogue Processing in Spoken Language Systems, Lecture Notes in Artificial Intelligence, E. Maier, M. Mast, and S. LuperFoy (editors), Springer-Verlag, 1997.

# 7 Appendix

## 7.1 ViaVoice grammar file used for the 2D test

```
grammar lea;
public <greetings> = good morning {greetings}
                   | hi you {greetings}
                   | hi sir {greetings}
                   | hello you {greetings}
                   | hello sir {greetings}
                   | greetings {greetings}
                   | hello {greetings}
                   | hi {greetings}      ;

public <bye> = see you soon {bye}
                   | see you later {bye}
                   | good bye {bye}
                   | may i leave {bye}
                   | i want to leave {bye}
                   | i am going {bye}
                   | i leave {bye}
                   | i go out {bye}
                   | i want to quit {bye}
                   | i quit {bye}
                   | i would like to go out {bye}
                   | could i go out {bye}
                   | bye see you {bye}
                   | bye {bye}
                   | leave {bye}
                   | ciao {bye}
                   | see you {bye}      ;

public <giveObject> = put {giveObject}
                   | take it {giveObject}
                   | take i give it to you {giveObject}
                   | go ahead take it {giveObject}
                   | you can take it {giveObject}
                   | i give it to you {giveObject}
                   | give {giveObject}
                   | you want <determine> <object> {giveKnownObject}
                   | give <determine> <object> {giveKnownObject}
                   | i bring you <determine> <object> {giveKnownObject}
                   | i bring <determine> <object> {giveKnownObject}
                   | i come to give you <determine> <object> {giveKnownObject}
                   | i come to bring you <determine> <object> {giveKnownObject}
                   | put <determine> <object> {giveKnownObject}
                   | put <object> {giveKnownObject}
                   | take <determine> <object> {giveKnownObject}
                   | give you <determine> <object> {giveKnownObject}
                   | here is <determine> <object> {giveKnownObject}
                   | give <determine> <object> {giveKnownObject} ;

public <takeObject> = give me the object {takeObject}
                   | i take the object {takeObject}
                   | again {takeObject}
                   | can i take {takeObject}
                   | can i have {takeObject}
                   | can i take from you {takeObject}
```

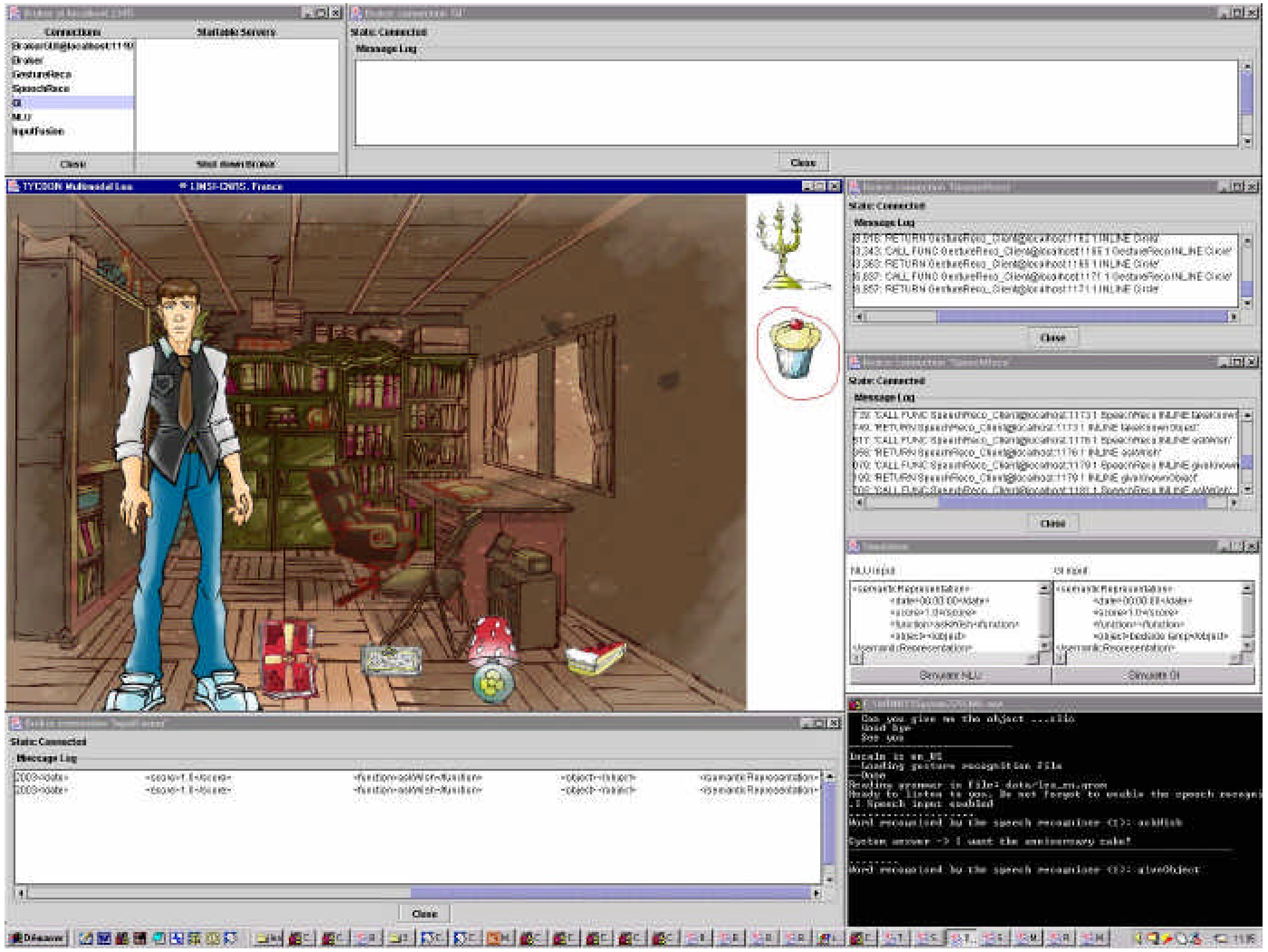
```
                        | i take it {takeObject}
                        | can you give me the object {takeObject}
                        | can you give me {takeObject}
                        | could i have {takeObject}
                        | do you want to give me {takeObject}
                        | give it to me {takeObject}
                        | give it {takeObject}
                        | i would like to take {takeObject}
                        | i serve myself {takeObject}
                        | give me {takeObject}
                        | take {takeObject}
                        | <object> {takeKnownObject}
                        | give me <determine> <object> {takeKnownObject}
                        | can i take from you <determine> <object> {takeKnownObject}
                        | can i have <determine> <object> {takeKnownObject}
                        | can i take <determine> <object> {takeKnownObject}
                        | i take from you <determine> <object> {takeKnownObject}
                        | i would like to take <determine> <object> {takeKnownObject}
                        | i would like <determine> <object> {takeKnownObject}
                        | can you give me <determine> <object> {takeKnownObject}
                        | <determine> <object> {takeKnownObject}
                        | i can take <determine> <object> {takeKnownObject}
                        | i take <determine> <object> {takeKnownObject}
                        | i want <determine> <object> {takeKnownObject}
                        | i come to take <determine> <object> {takeKnownObject}
                        | i would like to have <determine> <object> {takeKnownObject}
                        | could i have <determine> <object> {takeKnownObject}
                        | take <determine> <object> {takeKnownObject}
                        | take <object> {takeKnownObject}
                        | do you want to give me <determine> <object> {takeKnownObject}
                        | give me <object> {takeKnownObject}
                        | i would like to take <determine> <object> {takeKnownObject}      ;

public <askWish> = what do you need {askWish}
                        | what do you want {askWish}
                        | do you need anything {askWish}
                        | do you need an object {askWish}
                        | have you a question {askWish}
                        | what {askWish}
                        | ask {askWish}
                        | say what you want {askWish}
                        | can i help you {askWish}
                        | do you need {askWish}
                        | do you want something {askWish}
                        | do you need something {askWish}
                        | i can help you {askWish}
                        | which {askWish}
                        | what else do you want {askWish}
                        | what would you want {askWish}
                        | what do you want me to do {askWish}
                        | serve you {askWish}        ;

<determine> = this  | that | the ;

<object> = book     | story book
                        | kitchen book
                        | lamp
                        | bedside lamp
                        | candlestick lamp
                        | cake
                        | anniversary cake
                        | vanilla cake       ;
```

## 7.2 TYCOON file specifying cooperation between modalities for the 2D test

```
#-----------------------------------------------------------------------------
# MODALITIES
modality   MOUSE_POINTING          1
modality   SPEECH_RECOGNITION  1

#-----------------------------------------------------------------------------
# EVENTS
input       IS1         SPEECH_RECOGNITION greetings
input       IS2         SPEECH_RECOGNITION bye
input       IS3         SPEECH_RECOGNITION giveObject
input       IS4         SPEECH_RECOGNITION takeObject
input       IS5         SPEECH_RECOGNITION askWish
input       IS6         SPEECH_RECOGNITION giveKnownObject
input       IS7         SPEECH_RECOGNITION takeKnownObject

input   IG1             MOUSE_POINTING  position

#-----------------------------------------------------------------------------
# COMMANDS

#-- Cooperations used by each command

#- greetings command
specialisation        CCgreetings          IS1
endHypothesis         CCgreetings          greetings

#- bye command
specialisation        CCbye    IS2
endHypothesis         CCbye    bye

#- askWish command
specialisation        CCaskWish            IS5
endHypothesis         CCaskWish            askWish

specialisation        CCmouse_position IG1
semantics             CCmouse_position position

#- giveObject command
specialisation        CCspeech_giveObject                IS3
complementarity       temporalProximity 5000      CCgiveObject       CCspeech_giveObject          CCmouse_position
endHypothesis         CCgiveObject        giveObject

#- takeObject command
specialisation        CCspeech_takeObject                IS4
complementarity       temporalProximity 5000      CCtakeObject       CCspeech_takeObject          CCmouse_position
endHypothesis         CCtakeObject        takeObject

#- targetObject command
#- endHypothesis    CCmouse_position targetObject

#- giveKnownObject command
specialisation        CCgiveKnownObject          IS6
endHypothesis         CCgiveKnownObject          giveKnownObject

#- takeKnownObject command
specialisation        CCtakeKnownObject          IS7
endHypothesis         CCtakeKnownObject          takeKnownObject
```

## 7.3 Object file used by the IF module in the 2D test

```
  <?xml version="1.0" encoding="utf-8" ?>
- <listObjects>
  - <object>
      <fileName>story_book.gif</fileName>
      <name>story book</name>
      <size>big</size>
      <shape>square</shape>
      <color>red</color>
      <characteristic>heavy</characteristic>
      <x>300</x>
      <y>530</y>
    </object>
  - <object>
      <fileName>kitchen_book.gif</fileName>
      <name>kitchen book</name>
      <size>small</size>
      <shape>rectangular</shape>
      <color>gray</color>
      <characteristic>light</characteristic>
      <x>430</x>
      <y>530</y>
    </object>
  - <object>
      <fileName>bedside_lamp.gif</fileName>
      <name>bedside lamp</name>
      <size>small</size>
      <shape>round</shape>
      <color>red</color>
      <characteristic>broken</characteristic>
      <x>560</x>
      <y>530</y>
    </object>
  - <object>
      <fileName>anniversary_cake.gif</fileName>
      <name>anniversary cake</name>
      <size>medium</size>
      <shape>triangular</shape>
      <color>yellow</color>
      <characteristic>cold</characteristic>
      <x>690</x>
      <y>530</y>
    </object>
  - <object>
      <fileName>candlestick_lamp.gif</fileName>
      <name>candlestick lamp</name>
      <size>medium</size>
      <shape>fork</shape>
      <color>silver</color>
      <characteristic>shining</characteristic>
      <x>320</x>
      <y>185</y>
    </object>
  - <object>
      <fileName>vanilla_cake.gif</fileName>
      <name>vanilla cake</name>
      <size>small</size>
      <shape>round</shape>
      <color>yellow</color>
      <characteristic>hot</characteristic>
      <x>310</x>
      <y>500</y>
    </object>
</listObjects>
```



```
BrokerGUI@localhost:1110
Broker
GestureReco
SpeechReco
GI
NLU
InputFusion
```

```
3.516: RETURN GestureReco_Client@localhost:1162 1 INLINE Circle
3.343: CALL FUNC GestureReco_Client@localhost:1165 1 GestureReco INLINE Circle
3.363: RETURN GestureReco_Client@localhost:1165 1 INLINE Circle
6.837: CALL FUNC GestureReco_Client@localhost:1171 1 GestureReco INLINE Circle
6.857: RETURN GestureReco_Client@localhost:1171 1 INLINE Circle
```

```
7.29: CALL FUNC SpeechReco_Client@localhost:1173 1 SpeechReco INLINE tpeakown
7.40: RETURN SpeechReco_Client@localhost:1173 1 INLINE tpeakownObject
8.11: CALL FUNC SpeechReco_Client@localhost:1176 1 SpeechReco INLINE askWish
9.58: RETURN SpeechReco_Client@localhost:1176 1 INLINE askWish
0.70: CALL FUNC SpeechReco_Client@localhost:1179 1 SpeechReco INLINE giveKnown
1.00: RETURN SpeechReco_Client@localhost:1179 1 INLINE giveKnownObject
1.06: CALL FUNC SpeechReco_Client@localhost:1181 1 SpeechReco INLINE askWish
```

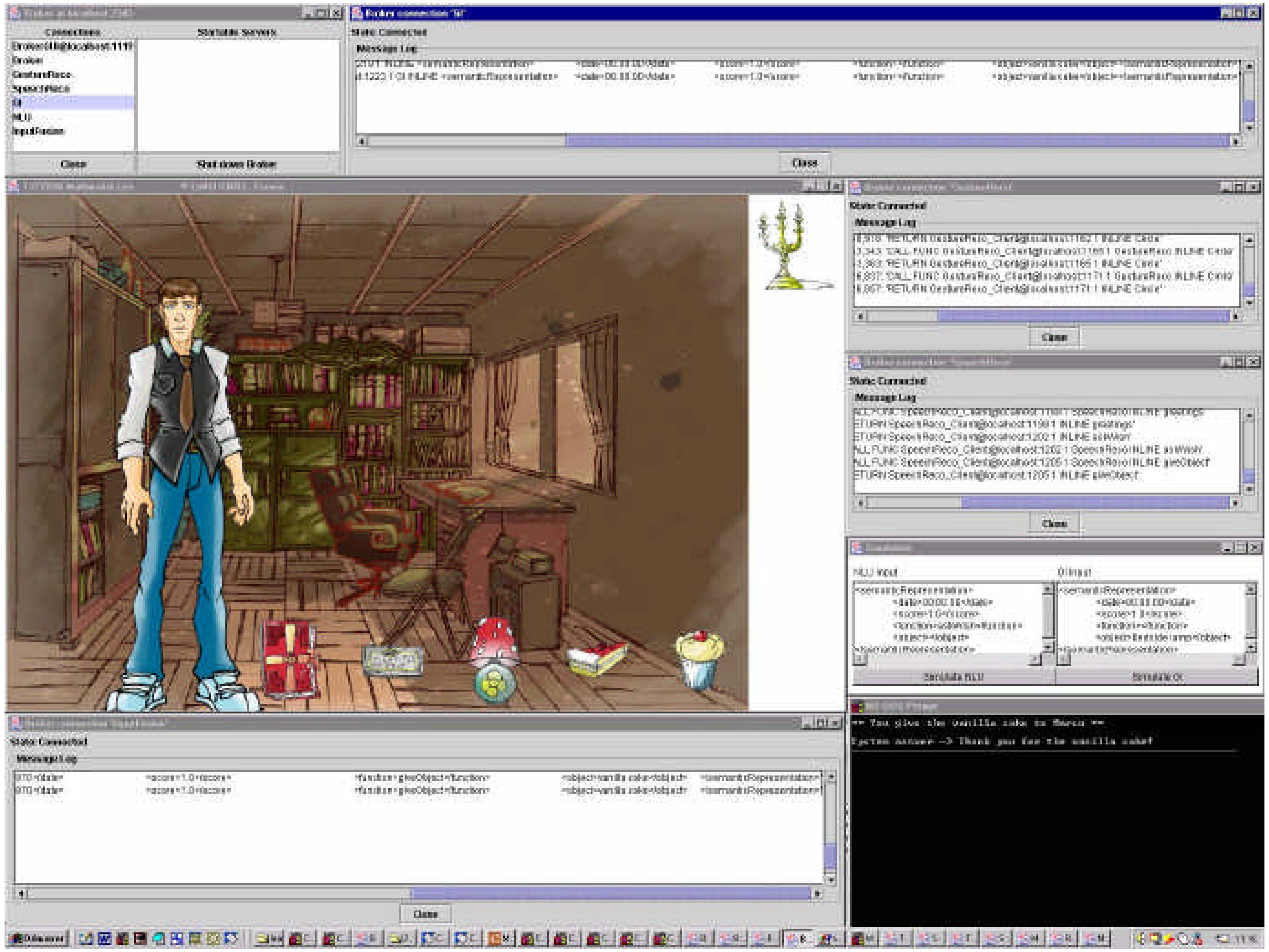
```
<semanticRepresentation>
        <date>00:00:00</date>
        <score>1.0</score>
        <function>askWish</function>
        <object></object>
</semanticRepresentation>
```

```
<semanticRepresentation>
        <date>00:00:11</date>
        <score>1.0</score>
        <function></function>
        <object>bedside lamp</object>
</semanticRepresentation>
```

```
   Can you give me the object ....ilio
   Good bye
   See you
--------------------------------
Locale is en_US
---Loading gesture recognition file
---Done
Reading grammar in file: data/lea_mi.gram
Ready to listen to you. Do not forget to enable the speech recogni
.1 Speech input enabled
---------------------
Word recognized by the speech recognizer (1): askWish

System answer -> I want the anniversary cake!
------------------------------------------------------------------------
--------
Word recognized by the speech recognizer (1): giveObject
```

```
2003</date>        <score>1.0</score>        <function>askWish</function>        <object></object>        </semanticRepresentation>
2003</date>        <score>1.0</score>        <function>askWish</function>        <object></object>        </semanticRepresentation>
```

Figure 10: A screendump with gesture in the 2D test environment.



```
Broker connection "GI"
State: Connected
Message Log
2101 INLINE <semanticRepresentation>    <date>00.00.00</date>    <score>1.0</score>    <function></function>    <object>vanilla cake</object></semanticRepresentation>
d:1223 1 GI INLINE <semanticRepresentation>    <date>00.00.00</date>    <score>1.0</score>    <function></function>    <object>vanilla cake</object></semanticRepresentation>
```

```
State: Connected
Message Log
8,316: RETURN GestureReco_Client@localhost:1162 1 INLINE Circle'
3,343: CALL FUNC GestureReco_Client@localhost:1165 1 GestureReco INLINE Circle'
3,363: RETURN GestureReco_Client@localhost:1165 1 INLINE Circle'
6,837: CALL FUNC GestureReco_Client@localhost:1171 1 GestureReco INLINE Circle'
6,867: RETURN GestureReco_Client@localhost:1171 1 INLINE Circle'
```

```
State: Connected
Message Log
ALL FUNC SpeechReco_Client@localhost:1198 1 SpeechReco INLINE greetings'
ETURN SpeechReco_Client@localhost:1198 1 INLINE greetings'
ETURN SpeechReco_Client@localhost:1202 1 INLINE askWho'
ALL FUNC SpeechReco_Client@localhost:1202 1 SpeechReco INLINE askWho'
ALL FUNC SpeechReco_Client@localhost:1205 1 SpeechReco INLINE giveObject'
ETURN SpeechReco_Client@localhost:1205 1 INLINE giveObject'
```

```
NLU Input
<semanticRepresentation>
        <date>00.00.00</date>
        <score>1.0</score>
        <function>askWhat</function>
        <object></object>
</semanticRepresentation>

GI Input
<semanticRepresentation>
        <date>00.00.00</date>
        <score>1.0</score>
        <function></function>
        <object>bedside lamp</object>
</semanticRepresentation>
```

```
State: Connected
Message Log
970</date>    <score>1.0</score>    <function>giveObject</function>    <object>vanilla cake</object>    </semanticRepresentation>
970</date>    <score>1.0</score>    <function>giveObject</function>    <object>vanilla cake</object>    </semanticRepresentation>
```

```
** You give the vanilla cake to Marco **
System answer -> Thank you for the vanilla cake!
```

**Figure 11: A screendump with fusion in the 2D test environment.**