SEMAINE

THE SENSITIVE AGENT PROJECT

D1c
First full-scale SAL system

2007 - 2013

Date: 22 December 2009

Dissemination level: Public

ICT project contract no.	211486
Project title	**SEMAINE** **Sustained Emotionally coloured Machine-human Interaction using Nonverbal Expression**
Contractual date of delivery	*31 December 2009*
Actual date of delivery	*22 December 2009*
Deliverable number	D1c
Deliverable title	First full-scale SAL system
Type	Demonstrator
Number of pages	25
WP contributing to the deliverable	WP 1
Responsible for task	Marc Schröder (schroed@dfki.de)
Author(s)	...
EC Project Officer	Philippe Gelin

Table of Contents

1 Executive Summary

Sensitive Artificial Listeners (SAL) are virtual dialogue partners who, despite their very limited verbal understanding, intend to engage the user in a conversation by paying attention to the user's emotions and non-verbal expressions. The SAL characters have their own emotionally defined personality, and attempt to drag the user towards their dominant emotion, through a combination of verbal and non-verbal expression.

The SEMAINE system 2.0 is the first public demonstrator of the fully operational autonomous SAL system based on audiovisual analysis and synthesis. The present report is part of a group of reports describing various aspects of the SEMAINE system 2.0. The full list of reports is available from http://semaine.opendfki.de/wiki/SEMAINE-2.0.

This report provides a technical overview of the system architecture. The SEMAINE API is described as the communication middleware and component integration platform underlying the SEMAINE system, and innovations since the writeup of a pending Journal article are presented. A short overview of all system components is given, including the connection between a message flow graph of the system and the Topic names via which components communicate. We then describe the tools available for debugging the system before providing some practical information on how to install and run the system, and giving detailed information regarding the licensing conditions under which the system is available.

2 System description

2.1 The application: Sensitive Artificial Listeners

The SEMAINE 2.0 system is the first fully operational implementation of an autonomous Sensitive Artificial Listener (SAL) system, provided by the SEMAINE project consortium. The system allows users to "chat" with one of four artificial listener agents. While the agents do not understand much of what the user is saying, they do their best to keep the user talking, by inferring the user's emotion from his/her voice and facial expression, by observing the nodding and head shake behaviour, and by detecting appropriate places to give backchannel feedback or say something to keep the user talking.

In other words, the SAL agents aim to do what people do at parties, where they don't really hear what the other is saying, or what people do who want to talk to one another but do not speak the same language very well.

Each of the four characters has its own emotion-related personality:

- Poppy is cheerful, optimistic, and looks at the bright side of life;
- Spike is aggressive, and he enjoys an argument;
- Obadiah is gloomy and has a pessimistic outlook;
- Prudence is matter-of-fact and has a practical view on life.

Each character's personality is reflected in the character's face, voice, behaviour, and the things they say. Through their behaviour, they try to "drag" the user's emotional state towards their own state: Poppy tries to make the user happy, Spike aims to make the user aggressive, etc. For a detailed description of the psychological underpinnings of this, see SEMAINE report D5a.

The scientific purpose of building this system is to investigate the "soft skills" required for a natural human-machine conversation.

The system is explained by Prudence in a video on youtube at

http://www.youtube.com/watch?v=zGl41NG0fxI

A sample interaction between a user and Poppy can be seen at

http://www.youtube.com/watch?v=munqOlj3mNw

2.2 The integration framework: the SEMAINE API

The SEMAINE 2.0 system is a multimodal interactive system consisting of multiple components (see Section 2.3 below). The processing power required by some of the components is high so that the full system cannot be run on a single PC at reasonable speed. Instead, the system runs as a distributed system across several computers, programming languages, and operating systems.

The SEMAINE API was created as the communication middleware and component integration platform for the system integration in SEMAINE. It builds on a Message-Oriented Middleware as the communication layer, and adds a stack of support for system integration, including:

- monitoring of the state of the system and all components;
- simple communication in namespace-aware XML formats using standard markup languages;

- an easily extensible mechanism for communicating the system's internal state across the middleware, using a configuration file for defining how to encode the information in XML;
- callback messages in addition to data messages, to inform other components of the processing state of some identified data;
- a centralised time independent of system clock differences;
- centralised logging.

The integration framework is conceived for ease of reuse and extensibility, so that it can be used for building new emotion-oriented systems as the natural combination of existing and new components.

The SEMAINE API is described in detail in an upcoming Journal article[1]. A preprint is available at http://www.dfki.de/~schroed/publications.html#schroeder2010.

Two aspects seem worth highlighting which have been added to the framework since the Journal article was written: the extensible method for communicating state information, and the callback message framework.

2.2.1 State information

The SEMAINE system needs to maintain various kinds of state information: the context, such as which character is currently active, and whether or not there is a user present; the agent's mental state; the agent's interpretation of the user's behavioural and emotional state; and the state of the dialogue.

In a Message-Oriented Middleware (MOM) it is not straightforward to preserve a centrally held state in such a way that several components can access it, because a MOM does not provide a shared memory. One option for implementing distributed access to state information would have been a specialised information repository component; however, this would have required each component to send and receive a message every time that it wants to access a certain information.

The solution implemented in the SEMAINE API is of a different kind. A specialised class `StateReceiver` is keeping track of state-related messages; it parses incoming messages and saves the information in a local information repository. That means, every component has local access to its own copy of the latest state information. Since the local copies are updated through the same state-related messages, they are updated in synchrony. Looking up a certain piece of information is thus as easy as accessing a local variable.

An important challenge in this kind of setup is to make the encoding-decoding link between the representation of information in XML-based messages, and a unique “short name” by which components can access the information. For example, the dialogue state information that the agent does not have the turn at the moment is encoded as follows:

```
<semaine:dialog-state xmlns:semaine=”http://www.semaine-project.eu/semaineml”>
      <semaine:agent believesHasTurn=”false”/>
</semaine:dialog-state>
```

The component, on the other hand, should be able to access this information independently of the representation format, e.g. through a short name such as “agentHasTurn”.

1 Schröder, M. (to appear). The SEMAINE API: Towards a standards-based framework for building emotion-oriented systems. To appear in: Advances in Human-Machine Interaction. [preprint]

If the link between message format and short name were hard-coded in the program code, it would not allow users to flexibly reuse the state information mechanism in novel domains and applications, which would be incompatible with the SEMAINE API's ambition to be a reusable framework. Therefore, we have developed and implemented a mechanism that allows developers to represent this relationship in a configuration file, using namespace-aware XPath expressions. XPath is a formalism for navigating through XML documents to access information. In the above example, the information whether or not the agent belives it has the turn can be accessed by going to the element "`dialog-state`" in the namespace `http://www.semaine-project.eu/semaineml`, then to the child element "`agent`" in the same namespace, and finally by accessing the value of the attribute "`believesHasTurn`". In XPath, this can be encoded as:

```
/semaine:dialog-state/semaine:user/@believesHasTurn
```

where the namespace prefix "`semaine`" is bound to the namespace `http://www.semaine-project.eu/semaineml`. By associating this XPath expression with a short name "userHasTurn", it is possible to provide the relation in a configuration file.

XPath in its general form is a very powerful framework which is intended only for accessing information in existing XML documents. It is not originally intended to be used for the generation of documents. However, by using only the subset of navigating to elements and accessing attribute values or textual content, we can reuse the XPath expressions also for the generation of XML documents and thus for encoding the information.

As a result, we have a fully configurable mechanism for encoding and decoding state information. The current content of the configuration file, `stateinfo.config`, is provided in Appendix I. It can be seen that both the relation between short names and XPath expressions and the namespace prefixes can be given in the configuration file, providing full flexibility for future extensions.

2.2.2 Callback messages

Callbacks are required to inform other components about the processing state of a certain piece of information. Specifically, the player needs to send callback messages about when it is starting and ending the playback of an utterance.

This information is used in several places in the system, in particular:

- the speech analysis components use the messages regarding start and end of audio playback to prevent voice activity detection while the agent is speaking. This is useful so that the system does not reply to itself;
- the dialogue components assign a unique identifier to an utterance that they request to be played. The player's callback messages confirm that a given utterance has been played, and thus allow for a smooth follow-on. In particular, the ActionSelection component can avoid sending additional playback requests while the system is still talking, which would interrupt the current system output.

Callback messages are sent via their own Topic hierarchy. Whereas data messages pass via Topics below semaine.data.*, callback messages are sent to Topics in the hierarchy semaine.callback.*. This way, it is possible to structurally distinguish the two types of messages very easily.

The SystemMonitor was extended to show callback Topics in the middle of the image. Callback topics are blue when inactive whereas data topics are grey. Both turn yellow when a message is sent via the respective Topic, and slowly go back to their inactive colour. Figure 1 shows a screenshot of the SystemMonitor with the full system running. It cann be seen that there have been recent callback messages in the Topics `semaine.callback.output.BAP` and `semaine.callback.output.FAP`, but nt in `semaine.callback.output.audio`. In other words, the player has recently started or stopped playing an animation without vocal output, such as a head nod.

The following is an example of a callback message sent in the Topic semaine.callback.output.audio.

```
<callback xmlns="http://www.semaine-project.eu/semaineml">
  <event data="audio" id="fml_uap_2_bml_1" time="30007" type="end"/>
</callback>
```

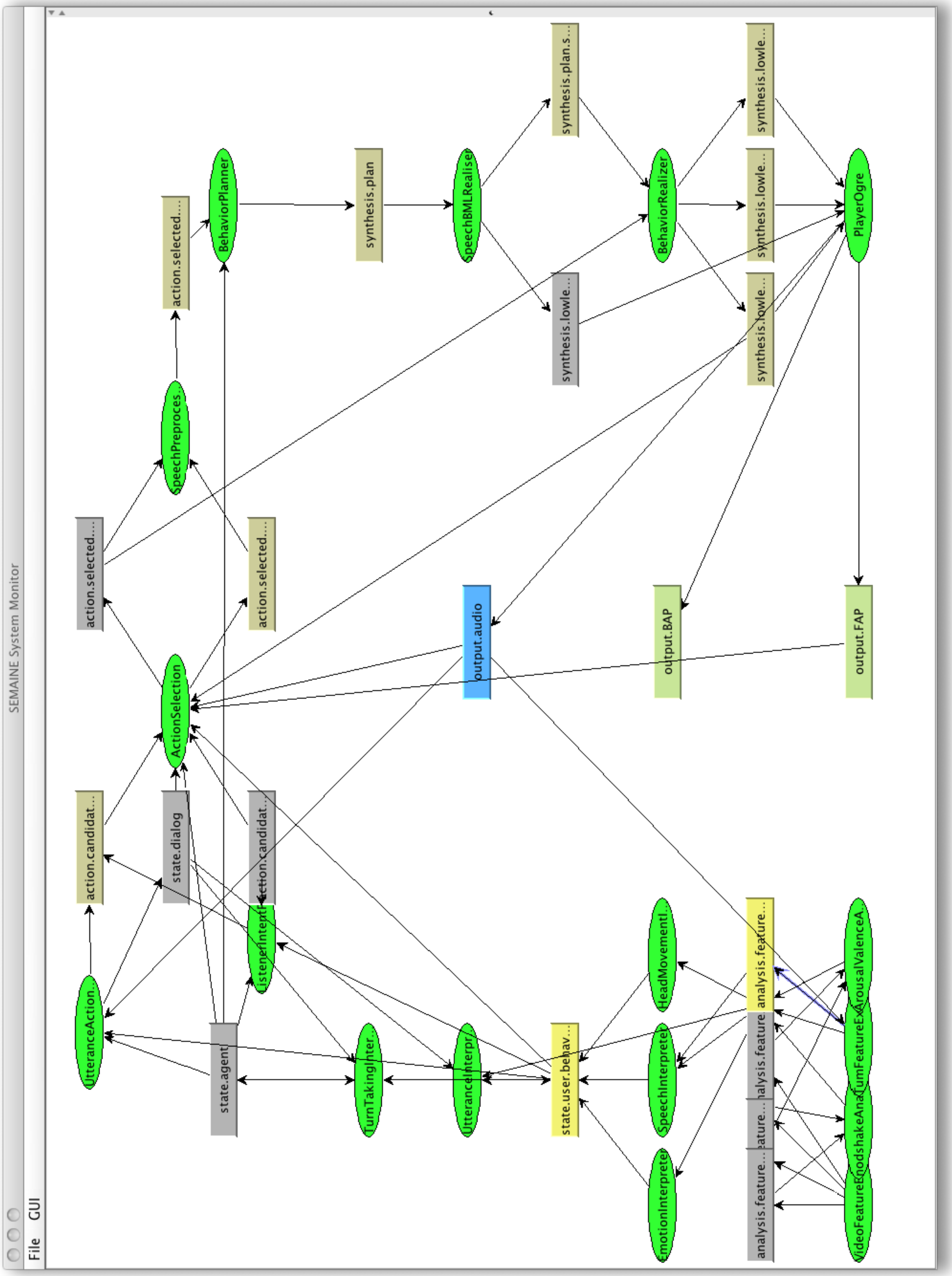


Figure 1: A screenshot of the SystemMonitor GUI showing the SEMAINE 2.0 system.

2.3 The components

The system components of the SEMAINE 2.0 system are shown in Figure 1. They are briefly characterised here; for in-depth descriptions, the reader is referred to the individual work package reports.

2.3.1 Feature extractor and analyser components

User behaviour is analysed from a camera and a microphone signal. The components TumFeatureExtractor and VideoFeatureExtractor compute features of the audio and video signals, respectively, and send them to the Topics semaine.data.analysis.features.voice and semaine.data.analysis.features.video.

We call Analyser components such components that analyse the raw input features and derive a processing result from them *without using central state information*. Analysis results often have a confidence associated to them and are encoded as EMMA messages in the SEMAINE system. Currently, there are the following analyser components in the system:

- NodShakeAnalyser determines whether the user seems to be nodding or shaking the head;
- ArousalValenceAnalyser determines from the facial points the emotion dimensions arousal and valence of the user;
- TUMFeatureExtractor determines the voice activity (speaking vs. silent) of the user, computes a number of emotion dimensions from the voice, and carries out incremental keyword spotting. TUMFeatureExtractor is observing the callback from the player on Topic semaine.callback.output.audio in order to suppress any voice activity detection messages while the agent is speaking.

All Analyser components send their output to the Topic semaine.data.user.state.emma.

2.3.2 Interpreter components

We call Interpreter components the components that produce internal state information, which corresponds conceptually to the system's “current best guess” regarding the state of the world. State information is communicated in the system as described in Section 2.2.1 and in Appendix I.

The SEMAINE 2.0 system currently includes the following interpreter components:

- An optional UserPresenceInterpreter (not shown in Figure 1) can use the information regarding voice activity and face detection to automatically determine the information whether a user is present. Is sends this information to the Topic semaine.data.state.context (not shown in Figure 1). Since the UserPresenceInterpreter is not very robust at the present moment, it is not active by default.
- HeadMovementInterpreter, SpeechInterpreter and EmotionInterpreter consolidate the evidence from the analysers regarding the current user behaviour regarding head movement, speech activity vs. silence and emotions, respectively. They send their analysis results to the Topic semaine.data.state.user.behaviour.
- UtteranceInterpreter uses the keywords spotted, the current dialog state, and the current best guess regarding the user's behaviour in order to derive a guess regarding the content of the user's utterance. It outputs its guess to the Topic semaine.data.state.user.behaviour.

- TurnTakingInterpreter uses all available “current best guess” state information, that is, it uses the user state, agent state, dialog state and context state to determine the agent's willingness to speak. It outputs that information to the Topic semaine.data.state.agent.

2.3.3 Action proposer and action selection components

Based on the state information available, action proposer components determine possible actions to carry out.

- UtteranceActionProposer observes the context, the user and agent states, and updates continuously a prioritised list of possible utterances that the agent could produce. Furthermore, UtteranceActionProposer is observing the callback messages from the player, on Topic semaine.callback.output.audio. When receiving the agent intention to start speaking, the utterance with the highest priority is sent to the Topic semaine.data.action.candidate.function. UtteranceActionProposer can also trigger a change of agent if requested by the user or decided by the agent, by sending an appropriate state change message to Topic semaine.data.state.context (not shown in Figure 1).
- ListenerIntentPlanner also observes the context, the user and agent states, and generates different kinds of listener behaviour represented in terms of the meaning of the behaviour (to agree, to show interest, etc.) or in terms of behaviour directly (e.g., to nod, to smile). Meaning-type proposed listener actions are sent to the Topic semaine.data.action.candidate.function; behaviour-type actions are sent to semaine.data.action.candidate.behaviour.

The ActionSelection component makes sure that only one action is carried out at a given time. It receives the candidate actions on the two above-mentioned Topics, and sends actions to be carried out to the Topics semaine.data.action.selected.function and semaine.data.action.selected.behaviour, respectively. Full utterances are treated with priority, i.e. they are passed on as soon as the player is free (as determined from the player callback on Topics semaine.callback.output.audio and semaine.callback.output.FAP). Listener behaviour is filtered depending on the current state information, notably the user interest as represented in the Topic semaine.data.state.user.behaviour.

2.3.4 Output generation components

Output generation components form a pipeline to make sure that synchronized vocal and visual behaviour is generated by the agent.

- SpeechPreprocessor receives the selected actions, from semaine.data.action.selected.function and semaine.data.action.selected.behaviour, computes accented syllables and prosodic phrase boundaries, and passes on the messages to the Topic semaine.data.action.selected.speechpreprocessed.
- BehaviorPlanner determines which behaviour should be executed for meaning-type actions, taking into account the personality of the agent. The identity of the current agent is obtained via Topic semaine.data.state.context (not shown in Figure 1). It sends behaviour-level actions to the Topic semaine.data.synthesis.plan.
- SpeechBMLRealiser receives data for which to generate speech through Topic semaine.data.synthesis.plan. The synthesis voice to be chosen is determined from agent identity as communicated via Topic semaine.data.state.context (not shown in Figure 1). It produces the audio data for an action in Topic semaine.data.synthesis.lowlevel.audio, and an en-

riched version of the behaviour markup including phone timings for lip synchronisation in Topic semaine.data.synthesis.plan.speechtimings.

- BehaviorRealizer generates the low-level animation data from behavioural markup. It receives this markup from Topic semaine.data.synthesis.plan.speechtimings for behaviour involving speech, and from semaine.data.action.selected.behaviour for non-vocal mimicry behaviour. Communication of the low-level animation data goes to the player through the Topics semaine.data.synthesis.lowlevel.command, semaine.data.synthesis.lowlevel.video.FAP and semaine.data.synthesis.lowlevel.video.BAP. In the absence of any messages to process, BehaviorRealizer also computes periodic eye-blink behaviour and sends it through semaine.data.synthesis.lowlevel.video.FAP and semaine.data.synthesis.lowlevel.command.
- Finally, the Player component receives the lowlevel data from the semaine.data.synthesis.lowlevel.* Topics and executes the synchronised audio-visual behaviour. It sends callback messages when starting and ending playback of audio, facial and body actions via the Topics semaine.callback.output.audio, semaine.callback.output.FAP and semaine.callback.output.BAP, respectively.

2.3.5 Other components

To give the user some control over core aspects of the interaction which are not yet determined fully reliably from the interaction itself, a ParticipantControlGUI component is provided with SEMAINE system 2.0 (). The purpose of this component is do indicate, through GUI buttons, whether the user is present or not, and who should be the current agent.

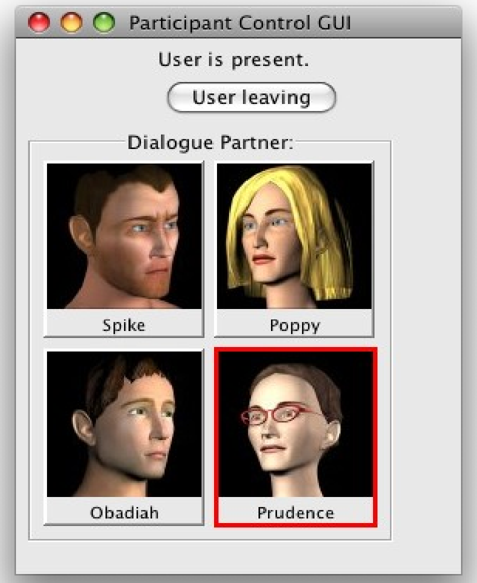


Figure 2: Participant control GUI

2.4 Debugging the system

A complex distributed system such as the SEMAINE system is not straightforward to debug. Unwanted effects can arise not only from the malfunction of individual components, but also from unexpected or otherwise suboptimal interactions between components.

The live system provides several means for following what is going on in the system.

- The centralised log functionality allows a log reader component, such as the one built into the System monitor, to follow all or individual components' log messages at a chosen log level. It is possible to get an overview of what is going on in the system by following all components simultaneously using the log level INFO. To follow a single component in detail, the log reader can be reconfigured to follow only that component with log level DEBUG.
- The system monitor GUI provides details when the user clicks on a component or a Topic. When the user clicks on a Topic, a new window opens which shows a live trace of messages sent via that Topic. A click on a component opens a new window showing generic meta information about that component, such as the Topics from which the component receives and to which it sends messages, the number of messages received so far, and the average time spent in the act() and react() methods implementing the component's functionality. When a component fails, any error output is shown in that detail view, together with the input data which caused the error.

Whereas the live debug tools are useful for some types of problems, they are not suitable for understanding complex interrelationships between components. Therefore, the SEMAINE system 2.0 comes with two off-line analysis tools.

The Dialog Manager logger (DMLogger) is keeping track of key variables regarding the user state, agent state, and agent utterances. When the system is stopped, a visualisation of that information is computed (Figure 3).

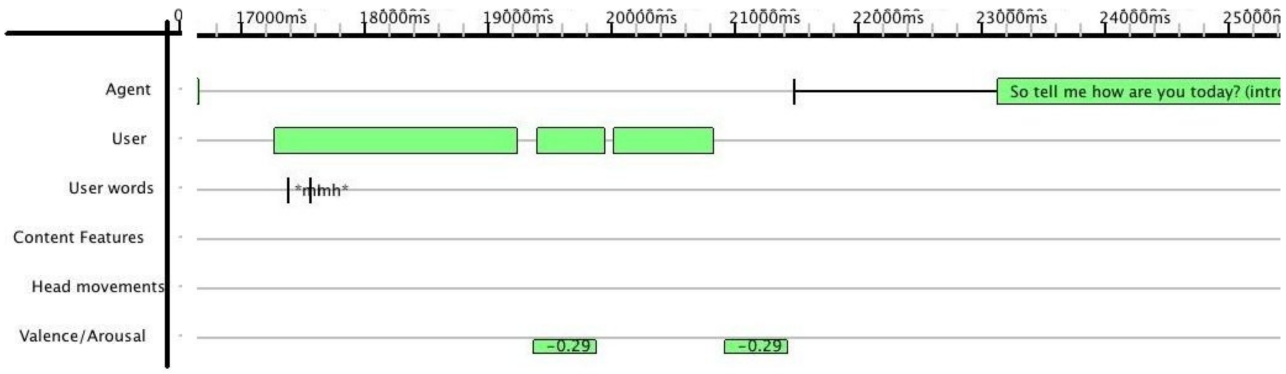
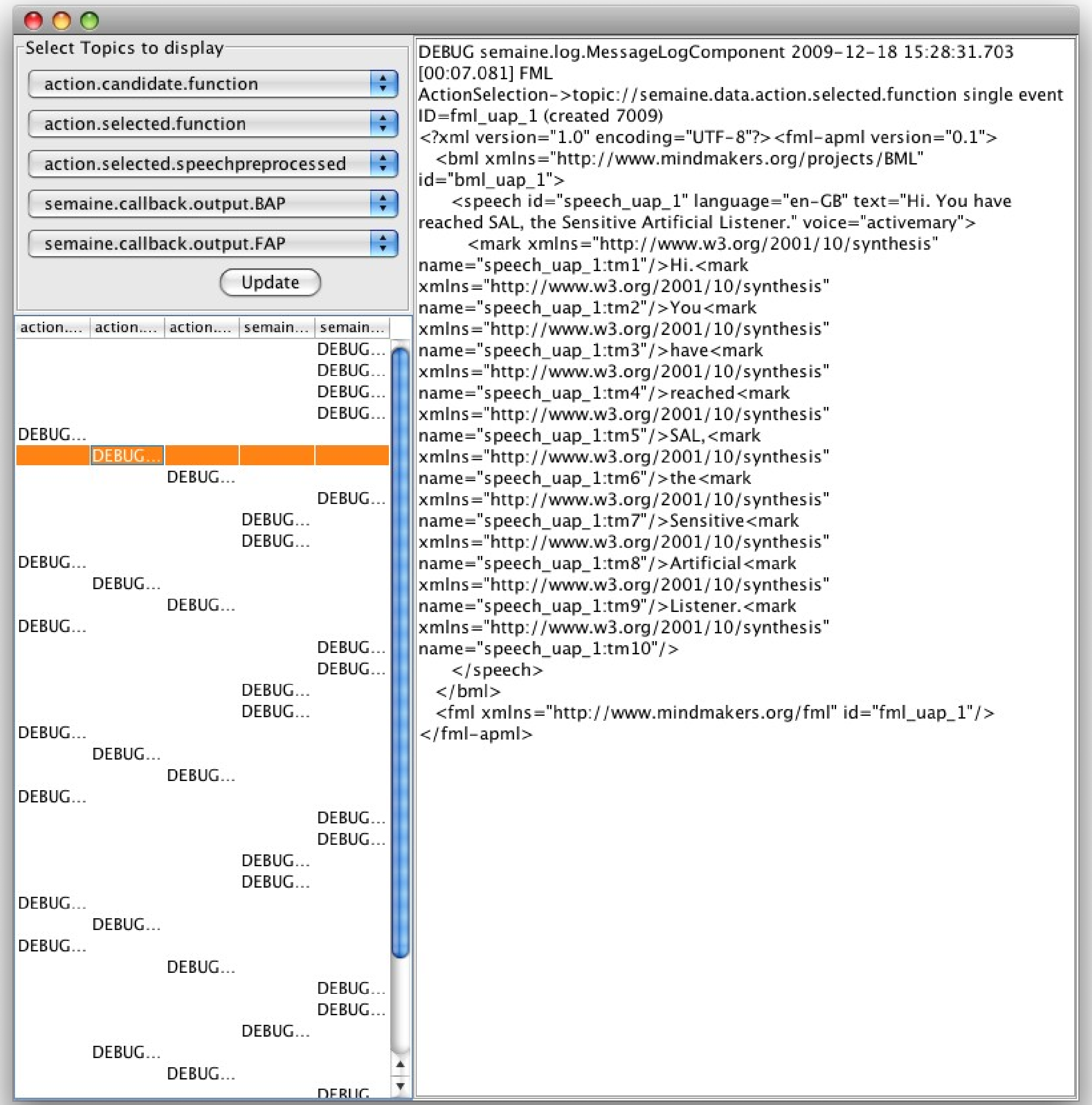


Figure 3: Excerpt from Dialog manager log visualisation for off-line analysis of the interaction.

A second tool for investigating in detail a configurable subset of the messages sent via the system is the MessageLogInspectorGUI (Figure 4). While the system is running, a command line tool is used to write a detailed log of all messages sent in the system into a file. After the system run is finished, the Java tool eu.semaine.util.MessageLogInspectorGUI is called and loads this file. The user can select up to five Topics (upper left corner of Figure 4). The messages in these Topics are shown in temporal order in a timeline from top to bottom (lower left corner of Figure 4). When a message is selected, its content is shown in a detail view (right side of Figure 4). This allows the user to flexibly follow the details of a sequence of messages and thereby to understand the exact order in which messages were sent and how a given piece of information was processed throughout the system.



action.candidate.function
action.selected.function
action.selected.speechpreprocessed
semaine.callback.output.BAP
semaine.callback.output.FAP

action....	action....	action....	semain...	semain...
				DEBUG...
				DEBUG...
				DEBUG...
				DEBUG...
DEBUG...				
	DEBUG...			
		DEBUG...		
				DEBUG...
			DEBUG...	
			DEBUG...	
DEBUG...				
	DEBUG...			
		DEBUG...		
DEBUG...				
				DEBUG...
				DEBUG...
			DEBUG...	
			DEBUG...	
DEBUG...				
	DEBUG...			
		DEBUG...		
DEBUG...				
				DEBUG...
				DEBUG...
			DEBUG...	
			DEBUG...	
DEBUG...				
	DEBUG...			
DEBUG...				
		DEBUG...		
				DEBUG...
				DEBUG...
			DEBUG...	
	DEBUG...			
		DEBUG...		
				DEBUG

```
DEBUG semaine.log.MessageLogComponent 2009-12-18 15:28:31.703
[00:07.081] FML
ActionSelection->topic://semaine.data.action.selected.function single event
ID=fml_uap_1 (created 7009)
<?xml version="1.0" encoding="UTF-8"?><fml-apml version="0.1">
  <bml xmlns="http://www.mindmakers.org/projects/BML"
id="bml_uap_1">
    <speech id="speech_uap_1" language="en-GB" text="Hi. You have
reached SAL, the Sensitive Artificial Listener." voice="activemary">
      <mark xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm1"/>Hi.<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm2"/>You<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm3"/>have<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm4"/>reached<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm5"/>SAL,<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm6"/>the<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm7"/>Sensitive<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm8"/>Artificial<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm9"/>Listener.<mark
xmlns="http://www.w3.org/2001/10/synthesis"
name="speech_uap_1:tm10"/>
    </speech>
  </bml>
  <fml xmlns="http://www.mindmakers.org/fml" id="fml_uap_1"/>
</fml-apml>
```

Figure 4: The MessageLogInspectorGUI shows messages in up to five Topics in a timeline.

3 Installation

As a pre-condition for installing and running the SEMAINE-2.0 system, make sure that you have suitable hardware and that you have installed the Required Software.

The open source parts of the system can be downloaded from http://sourceforge.net/projects/semaine/. It comes in three parts.

- SEMAINE-2.0-windows (~100 MB) includes binary versions of the Greta agent components, the Opensmile speech analysis components, and the message-oriented middleware ActiveMQ.
- SEMAINE-2.0-java (~650 MB) includes the System manager component, the dialogue components, and the speech synthesizer MARY TTS. The file is so large because the speech synthesizer comes with four high-quality TTS voices which need a lot of space.

To run the SEMAINE-2.0 system, both the windows and the java package are required. They can run together on a fast machine (tested on a laptop with a 2.53 GHz Core2Duo CPU with 4 GB RAM), or you can set up SEMAINE-2.0 as a distributed system.

- SEMAINE-2.0-source is an optional package which can be used to compile the system for Linux or Mac OS X, or to rebuild the windows components from source. Information on how to build the system from source is available on the SEMAINE wiki at http://semaine.opendfki.de/wiki/SEMAINE-2.0-Compile.

The Video analysis components are distributed as closed-source freeware from http://www.doc.ic.ac.uk/~maja/ (search for "SEMAINE Visual Components" on that page). Watch out for the Camera driver requirements if you are using a Firewire camera. If installed in the default location, the SEMAINE-2.0-windows start.bat script will notice that the video analysis components are installed and will try to run them. Since they are computationally heavy, you may need an additional computer to run them.

The SEMAINE-2.0 system will work without the video analysis components, but will then not be able to pick up the same amount of information from the user.

3.1 Hardware required to run SEMAINE-2.0

Computer

At least one fast computer running Windows Vista. Core2Duo with 2.5 GHz or faster and 4 GB RAM is recommended.

To run the video analysis components, a second computer with similar specs.

The system can run as a distributed system by running the various components on different machines. In that case, the CPU and memory requirements are lower.

Camera

You require a Firewire or USB camera with OpenCV-2.0 compatibility.

Microphone

A headset is needed for reasonable speech analysis quality. The headset should be configured as the default audio input device. Since the software is adaptive, reasonable results are expected with a broad range of headsets.

Tested setup

We have run the system with the following setup:

- Windows Vista SP1 on a Dell XPS 1730 Laptop with Core2Duo 2.53 GHz and 4 GB RAM, running Greta and SEMAINE video components
    - A Stingray F-046B Firewire camera connected to the Winodws machine;
- Ubuntu Linux 9.10 on a Lenovo Thinkpad T60 with 2 GB RAM, running ActiveMQ, java components, and opensmile
    - a Plantronics noise-cancelling headset connected to the Linux machine.
- (optionally, to reduce the load on the Linux machine, one of ActiveMQ, java components and opensmile has been run on a Mac Book Pro with 2.33 GHz Core2Duo and 2 GB RAM)

3.2 SEMAINE-2.0 Dependencies: Required software

The following software is required to run the SEMAINE-2.0 system. (additional software is required to compile SEMAINE-2.0 components).

Java 1.6

The SEMAINE-2.0-java components require Java 1.6. For just running the system, the Java Runtime Environment (JRE) is sufficient; you can obtain it from http://www.java.com/getjava

ActiveMQ

Communication among all SEMAINE-2.0 components passes via the Java Messaging Service (JMS) server ActiveMQ. SEMAINE-2.0-windows includes ActiveMQ. You only need to install ActiveMQ separately if you want to run activemq on a linux or MacOS X machine. Binary distributions can be found at http://activemq.apache.org/download.html . All versions 5.x seem to work fine.

After installation, ActiveMQ can be started by running, on Linux/Mac OS X:

```
apache-activemq-5.x.y/bin/activemq
```

On Windows, ActiveMQ is automatically started from the start.bat script.

Windows Vista

The SEMAINE-2.0-windows components are distributed as binaries for Windows Vista. You need to have the following Software installed to **run** the windows components:

- DirectX 9c or newer

Firewire camera

In order to use the SEMAINE Video components with a Firewire camera (such as the Stingray F-046B) compatible with Windows Vista and OpenCV 2.0, the camera must be recognized in the Device Manager as 1394 Digital Camera. This can be achieved by downloading and installing the CMU 1394 Digital Camera Driver (http://www.cs.cmu.edu/~iwan/1394/). If after the installation the camera is not recognized as CMU 1394 Digital Camera, proceed as follows:

- The IEEE 1394 Bus host controller named RICOH OHCI Compliant IEEE 1394 host controller should be uninstalled;
- 'Scan for hardware changes' should be done;
- In the imaging devices list, there should be now a 1394 Digital Camera. The driver for this camera should be updated by choosing the CMU 1394 Digital Camera Driver as the location for this update.

4 Running the SEMAINE system

In its simplest form, the system can be run on a single (fast) Windows Vista machine by installing all system components on the same computer as described above. The system is then run by starting the following batch file:

```
SEMAINE-2.0\start.bat
```

This will start ActiveMQ, wait until it is started, and then start all other installed components. If the system does not start correctly, double-check that you have unpacked both the windows and the java components in the same folder, and that you have met all the requirements.

To stop all components of the system, call

```
SEMAINE-2.0\stop.bat
```

The system with windows and java open source components runs OK on a Core2Duo with 2.53 GHz and 4 GB RAM. When the video analysis components are added, the system is running but very slow. Therefore, it is recommended to run SEMAINE-2.0 as a distributed system on several computers. Appendix II describes two distributed scenarios and provides the necessary start scripts.

5 License and availability

The SEMAINE API for Java and C++, the SEMAINE dialogue components (in Java), and the speech synthesizer MARY TTS are distributed under the GNU Lesser General Public License (LGPL), version 3. The speech synthesis voices for the SAL agents are distributed under the Creative Commons ShareAlike - No Derivatives license.

The 3D agent animation software Greta and the speech analysis software Opensmile are distributed under the GNU General Public License (GPL).

The separately installable SEMAINE Video components for camera image analysis come as a freeware binary.

Appendix I: The current state information configuration file

The following is the current configuration file encoding the relation between state information message formats and short names as used by the system components to access the information. The types of information that are available and the way they are encoded can easily be changed by modifying this configuration file.

```
# This is the config file for user state info messages in the SEMAINE project.
# It provides for each piece of information that can be represented a simple XPath
expression
# describing how to encode this information in state info messages.

######################################################################

[UserState]

[namespace prefixes]
semaine http://www.semaine-project.eu/semaineml
bml     http://www.mindmakers.org/projects/BML
emotion http://www.w3.org/2005/Incubator/emotion


[short names]
headGesture   /semaine:user-state/bml:bml/bml:head/@type
headGestureStarted  /semaine:user-state/bml:bml/bml:head/@start
headGestureStopped  /semaine:user-state/bml:bml/bml:head/@end
facialExpression    /semaine:user-state/bml:bml/bml:face/@shape
facialActionUnits   /semaine:user-state/bml:bml/bml:face/@au
facialExpressionStarted   /semaine:user-state/bml:bml/bml:face[@shape]/@start
facialExpressionStopped   /semaine:user-state/bml:bml/bml:face[@shape]/@end
facialActionUnitsStarted  /semaine:user-state/bml:bml/bml:face[@au]/@start
facialActionUnitsStopped  /semaine:user-state/bml:bml/bml:face[@au]/@end
pitchDirection      /semaine:user-state/semaine:pitch/@direction
speaking     /semaine:user-state/semaine:speaking/@status
interest     /semaine:user-
state/emotion:emotion/emotion:dimensions[@set='interestDimension']/emotion:interest/@valu
e
valence /semaine:user-
state/emotion:emotion/emotion:dimensions[@set='genericEmotionDimensions']/emotion:valence
/@value
arousal /semaine:user-
state/emotion:emotion/emotion:dimensions[@set='genericEmotionDimensions']/emotion:arousal
/@value
potency /semaine:user-
state/emotion:emotion/emotion:dimensions[@set='genericEmotionDimensions']/emotion:potency
/@value
emotion-quadrant /semaine:user-
state/emotion:emotion/emotion:category[@set='fourQuadrants']/@name
userUtterance       /semaine:user-state/semaine:userutterance/@utterance
userUtteranceStartTime     /semaine:user-state/semaine:userutterance/@starttime
userUtteranceFeatures      /semaine:user-state/semaine:userutterance/@features
gender       /semaine:user-state/semaine:gender/@name

######################################################################

[AgentState]

[namespace prefixes]
semaine http://www.semaine-project.eu/semaineml
emotion http://www.w3.org/2005/Incubator/emotion

[short names]
needToSpeak    /semaine:agent-state/semaine:needtospeak/@value
```

```
turnTakingIntention        /semaine:agent-state/semaine:turntakingintention/@value
listenerAgreement    /semaine:agent-state/semaine:listener-agreement/@value
listenerDisagreement       /semaine:agent-state/semaine:listener-disagreement/@value
listenerAcceptance   /semaine:agent-state/semaine:listener-acceptance/@value
listenerRefusal      /semaine:agent-state/semaine:listener-refusal/@value
listenerBelief       /semaine:agent-state/semaine:listener-belief/@value
listenerDisbelief    /semaine:agent-state/semaine:listener-disbelief/@value
listenerLiking       /semaine:agent-state/semaine:listener-liking/@value
listenerDisliking    /semaine:agent-state/semaine:listener-disliking/@value
listenerUnderstanding      /semaine:agent-state/semaine:listener-understanding/@value
listenerNoUnderstanding    /semaine:agent-state/semaine:listener-nounderstanding/@value
listenerInterest     /semaine:agent-state/semaine:listener-interest/@value
listenerNoInterest   /semaine:agent-state/semaine:listener-no-interest/@value
emotion-quadrant /semaine:agent-
state/emotion:emotion/emotion:category[@set='fourQuadrants']/@name
interest      /semaine:agent-
state/emotion:emotion/emotion:dimensions[@set='interestDimension']/emotion:interest/@valu
e

#####################################################################

[DialogState]
[namespace prefixes]
semaine http://www.semaine-project.eu/semaineml

[short names]
userTurnState        /semaine:dialog-state/semaine:user/@believesHasTurn
agentTurnState      /semaine:dialog-state/semaine:agent/@believesHasTurn
convState    /semaine:dialog-state/semaine:agent/@convState

#####################################################################

[ContextState]
[namespace prefixes]
semaine http://www.semaine-project.eu/semaineml

[short names]
userPresent   /semaine:situational-context/semaine:user/@status
character    /semaine:situational-context/semaine:character/@name

#####################################################################

[SystemState]
[namespace prefixes]
semaine http://www.semaine-project.eu/semaineml

[short names]
cameraPresent        /semaine:setup/semaine:camera/@status
cameraXResolution   /semaine:setup/semaine:camera/@xres
cameraYResolution   /semaine:setup/semaine:camera/@yres
cameraFrameRate     /semaine:setup/semaine:camera/@framerate
cameraNumChannels   /semaine:setup/semaine:camera/@numChannels
microphonePresent   /semaine:setup/semaine:microphone/@status
microphoneFrameRate       /semaine:setup/semaine:microphone/@framerate
ecaPresent   /semaine:setup/semaine:eca/@status
```

Appendix II: Running SEMAINE-2.0 as a distributed system

SEMAINE-2.0 is fundamentally distributed. All communication runs via the Message-oriented Middleware ActiveMQ. This allows users to run the SEMAINE system in a distributed fashion, across different computers and optionally across different operating systems, thus matching CPU and memory requirements of the components with available compute power.

The SEMAINE-2.0-windows package comes with a number of configuration examples to show how to easily achieve a distributed system. It is assumed that low-level issues of setting up a network and identifying the IP address of a given machine are solved -- there is abundant information on these issues available on the internet.

In essence, seting up the distributed system amounts to the following tasks:

- determine the IP address of the machine that should run ActiveMQ;
- adapting the configuration variable CMS_URL on all machines to point to the right ActiveMQ instance;
- starting the right components on each machine.

The key idea here is to create a custom start script for each machine, starting from SEMAINE-2.0\start.bat. All machines can use the common stop.bat.

In SEMAINE-2.0\config-examples, example start scripts are provided for the following scenarios.

Scenario 1: Two windows machines

In this scenario we assume two machines:

Machine 1 is a Windows machine. It should run ActiveMQ, the java components, and the Greta components. Its IP address is 192.168.1.1

Machine 2 is also a Windows machine. It should run Opensmile and the Video analysis components.

Make sure the respective components are available in the respective machines.

- copy start-machine1.bat into the SEMAINE-2.0 folder on machine 1.
- copy start-machine2.bat into the SEMAINE-2.0 folder on machine 2.
- Now, first run start-machine1 on machine 1, then start-machine2 on machine 2.

start-machine1.bat:

```
:: Which ActiveMQ to connect to:
set CMS_URL=tcp://192.168.1.1:61616

:: Locations of installed software:
:: %~dp0 is expanded pathname of the current script
set SEMAINEDIR=%~dp0%
set ACTIVEMQDIR="%SEMAINEDIR%apache-activemq-5.3.0"
set JAVADIR="%SEMAINEDIR%java"
set GRETADIR="%SEMAINEDIR%Greta"

:: ACTIVEMQ: start and wait for it to be started
if exist %ACTIVEMQDIR% (
  echo.Trying to start ActiveMQ from %ACTIVEMQDIR%...
  set SUNJMX=-Dcom.sun.management.jmxremote.port=1099
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
  start "activemq" /MIN /D %ACTIVEMQDIR%\bin activemq-admin.bat start
  :loop
  %GRETADIR%\utils\WAIT.exe 1
  for /f "delims=" %%a in ('netstat -an ^| findstr "61616"') do @set
ACTIVEMQ_READY=%%a
  if "%ACTIVEMQ_READY%"=="" (
    echo.Waiting for ActiveMQ...
    goto loop
  ) else (
    echo.ActiveMQ seems to be ready.
  )
)

:: Java components
if exist %JAVADIR% (
  echo.Starting Java components
  start "speech2face" /MIN /D "%SEMAINEDIR%" bin\semaine-speech2face.bat
)

:: GRETA components
if exist %GRETADIR% (
  echo.Starting Greta components
  start /MIN /D %GRETADIR% Active_ActionSelection.exe
  start /MIN /D %GRETADIR% Active_BehaviorPlanner.exe
  start /MIN /D %GRETADIR% Active_BehaviorRealizer.exe
  start /MIN /D %GRETADIR% Active_ListenerIntentPlanner.exe
  start /D %GRETADIR% Active_Player.exe
)
```

start-machine2.bat:

```
:: Which ActiveMQ to connect to:
set CMS_URL=tcp://192.168.1.1:61616

:: Locations of installed software:
:: %~dp0 is expanded pathname of the current script
set SEMAINEDIR=%~dp0%
set OPENSMILEDIR="%SEMAINEDIR%Opensmile"
set IMPERIALDIR="C:\Program Files\HCI^2 group, Imperial College London\Semaine
Video Components"


:: Start audio components if installed
if exist %OPENSMILEDIR%\SEMAINExtract.exe (
  echo.Starting Opensmile audio input components
  start /MIN /D %OPENSMILEDIR% SEMAINExtract.exe -C opensmileSemaine.conf
)

:: Start video components if installed
if exist %IMPERIALDIR%\VideoFeatureExtractorConsole.exe (
  echo.Starting video input components
  start /MIN /D %IMPERIALDIR% VideoFeatureExtractorConsole.exe
)
```

Scenario 2: One Linux/Mac and two Windows machines

In this scenario we assume three machines:

Machine 1 is a Linux or Mac OS X machine. It should run ActiveMQ, Opensmile, and the Java components. Its IP address is 192.168.1.10.

Machine 2 is a Windows machine runing the Greta components.

Machine 3 is a Windows machine. It should run the Video analysis components.

Make sure the respective components are available in the respective machines.

- copy start-machine1.sh into the SEMAINE-2.0 folder on machine 1.
- copy start-machine2.bat into the SEMAINE-2.0 folder on machine 2.
- copy start-machine3.bat into the SEMAINE-2.0 folder on machine 3.
- Now, first run start-machine1 on machine 1, then start-machine2 on machine 2 and start-machine3 on machine 3.

start-machine1.sh:

```
#!/bin/sh
# This assumes that activemq is on the PATH; adapt call to activemq if
necessary.

SEMAINEDIR=`dirname "$0"`
SEMAINEDIR=`(cd $SEMAINEDIR ; pwd)`

# Which ActiveMQ to connect to:
export CMS_URL=tcp://192.168.1.10:61616

activemq &

$SEMAINEDIR/bin/semaine-speech2face.sh &

$SEMAINEDIR/bin/run_components/start_component_tum.opensmile &
```

start-machine2.bat:

```
:: Which ActiveMQ to connect to:
set CMS_URL=tcp://192.168.1.10:61616

:: Locations of installed software:
:: %~dp0 is expanded pathname of the current script
set SEMAINEDIR=%~dp0%
set GRETADIR="%SEMAINEDIR%Greta"

:: GRETA components
if exist %GRETADIR% (
  echo.Starting Greta components
  start /MIN /D %GRETADIR% Active_ActionSelection.exe
  start /MIN /D %GRETADIR% Active_BehaviorPlanner.exe
  start /MIN /D %GRETADIR% Active_BehaviorRealizer.exe
  start /MIN /D %GRETADIR% Active_ListenerIntentPlanner.exe
  start /D %GRETADIR% Active_Player.exe
)
```

start-machine3.bat:

```
:: Which ActiveMQ to connect to:
set CMS_URL=tcp://192.168.1.10:61616

:: Locations of installed software:
:: %~dp0 is expanded pathname of the current script
set IMPERIALDIR="C:\Program Files\HCI^2 group, Imperial College London\Semaine
Video Components"

:: Start video components if installed
if exist %IMPERIALDIR%\VideoFeatureExtractorConsole.exe (
  echo.Starting video input components
  start /MIN /D %IMPERIALDIR% VideoFeatureExtractorConsole.exe
)
```