SEMAINE

THE SENSITIVE AGENT PROJECT

D7e

Public access to interactive SAL system

Date: 17 December 2010

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 2010*
Actual date of delivery	*17 December 2010*
Deliverable number	D7e
Deliverable title	Public access to interactive SAL system
Type	Demonstrator
Number of pages	70
WP contributing to the deliverable	WP 1, WP7
Responsible for task	Marc Schröder (schroed@dfki.de)
Author(s)	Marc Schröder, Elisabetta Bevacqua, Florian Eyben, Hatice Gunes, Mark ter Maat, Sathish Pammi, Etienne de Sevin, Michel Valstar, Martin Wöllmer
EC Project Officer	Philippe Gelin

Table of Contents

Executive Summary

This report marks the final release of the Sensitive Artificial Listener system, SEMAINE-3.1. We desribe how the system has been made available, how it can be installed, and how it can be configured.

The contents of this report are a snapshot of the live documentation on the software development website http://semaine.opendfki.de.

By providing this software package and appropriate documentation, we fulfil the key aim of the SEMAINE project to provide a reusable software platform for future research on real-time interactive Embodied Conversational Agents with emotional and non-verbal competence.

1 Introduction

This report marks the final release of the Sensitive Artificial Listener system, SEMAINE-3.1.

The project's software release practice goes beyond what had been promised in the grant agreement in two ways.

First, the original work plan previewed a release of the system to the public only at the end of the project's lifetime. Instead, the consortium has followed a continuous release pattern starting with the SEMAINE-1.0 early integration system after the first project year, and continuing throughout the project's lifetime. This way, it is ensured that the released software packages are complete and sufficiently easy to use.

Second, in the grant agreement the consortium had expressed the aim to make the majority of components available as open source software, but had anticipated that two components (speech recognition and video input components) could be made available as closed-source software under a research license only. In fact, it has been possible to release the speech recognition software as open source as well; the video input components are available as closed-source freeware, which means that use is not restricted exclusively to research purposes.

We have distributed the demonstrators via the open source hosting platform sourceforge.net for the open source part of the system (see http://sourceforge.net/projects/semaine/), and as a Windows installer package for the closed-source part (see http://semaine.opendfki.de/wiki/SEMAINE-3.1). According to sourceforge download statistics, the various versions of the package have been downloaded more than 1,000 times since February 2010.

1.1 The purpose of the present report

This report addresses the need to document the multiple aspects of the SEMAINE system that can be configured. The documentation is maintained on the wiki of the software development website http://semaine.opendfki.de, so as to be easy to find and update.

This report provides a snapshot of the documentation as it is available at the end of the project. We hope that the SEMAINE system will continue to be used and developed beyond the lifetime of the project; the infrastructure needed for this to happen is in place.

2 SEMAINE-3.1: The final Sensitive Artificial Listener system

(from http://semaine.opendfki.de/wiki/SEMAINE-3.1)

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 project has created an open-source implementation of fully autonomous SAL characters. It combines state-of-the-art emotion-oriented technology into a real-time interactive system. The SAL characters register your voice from a microphone, using a combination of speech recognition and vocal emotion recognition. They perceive your facial expressions and head movements through a video camera. They express themselves through a virtual animated face and a synthetic expressive voice. This video illustrates the concept.

2.1 New in version 3.1

This is essentially a cleanup release of the full SAL system. Its main emphasis is on improving the stability and performance, and simplifying the configuration of the system in order to make it easier to reuse the systm in whole or in part for scientific research in real-time interactive, emotionally aware Embodied Conversational Agents.

This version of the system has the following new feature:

- User presence is detected automatically, based on face and voice detection.
- Embedded middleware: The middleware ActiveMQ is now started as part of the java components; no need to start a separate ActiveMQ server anymore.

Apart from this, all components have received minor improvements under the surface.

In total, 28 issues have been fixed: http://semaine.opendfki.de/query?status=closed&group=resolution&milestone=3.1

2.2 Installation

As a pre-condition for installing and running the SEMAINE-3.1 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 the SEMAINE sourceforge project page. You can choose among two packages, each around 950 MB. (The files are so large because it includes four high-quality TTS voices as well as vocal emotion recognition models, which need a lot of space.)

- **SEMAINE-3.1-windows** includes binary versions of the full SEMAINE-3.1 system: the System manager component, the dialogue components, the speech synthesizer MARY TTS, the Greta agent components, the Opensmile speech analysis components, and the message-oriented middleware ActiveMQ.
- **SEMAINE-3.1-source** includes the source code for: the System manager component, the dialogue components, and the Opensmile speech analysis components, as well as binary versions of the speech synthesizer MARY TTS, the message-oriented middleware ActiveMQ

and all dependencies needed to compile the code. It can be used to compile the system from source under Linux, Mac and Windows.

To run the full audio-visual SEMAINE-3.1 system, the windows package is required. The system can run on a single, fast machine (tested on a laptop with a 2.53 GHz Core2Duo CPU with 4 GB RAM), or you can set up SEMAINE-3.1 as a distributed system.

The **Video analysis components** are distributed as closed-source freeware: SEMAINE Visual Components. Watch out for the Camera driver requirements if you are using a Firewire camera. If installed in the default location, the 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-3.1 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.

2.3 Running the system

2.3.1 Windows

In its simplest form, the system can be run on a single (fast) Windows 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-3.1\start.bat
```

This will start all installed components. If the system does not start correctly, double-check that you have met all the requirements.

To stop all components of the system, call

```
SEMAINE-3.1\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-3.1 as a distributed system on several computers.

2.3.2 All platforms

Whereas the video input and output components are available for Windows only, there is a configuration of the system that runs on all platforms -- the speech2speech system, with only speech input and speech output.

This system can be started as follows.

- In one shell, start the java components as `SEMAINE-3.1/bin/semaine-speech2speech.sh` (or .bat);
- In a second shell, start opensmile as `SEMAINE-3.1/bin/run_components/start_component_tum.opensmile` (linux/mac) or `SEMAINE-3.1\Opensmile\start_openSMILE.bat` (windows).

In this configuration the java components are started with a different config file, which loads a stack of audio-only output components in java.

2.4 License

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.

2.5 Developer documentation

With some effort you can build the components from source.

Detailed documentation of the SEMAINE API is available in a number of documents:

- A Journal article in Advances in Human-Machine Interaction The SEMAINE API: Towards a standards-based framework for building emotion-oriented systems;
- the deliverable report D1d Final SAL system
- the deliverable report D1c First full-scale SAL system
- section 3 of the deliverable report D1b: First integrated system
- Javadoc is available for the Java version of the SEMAINE API
- Doxygen is available for the C++ version of the SEMAINE API

2.6 Mailing list

There is a public SEMAINE-users mailing list at https://lists.sourceforge.net/lists/listinfo/semaine-users. Feel free to ask questions there.

2.7 Background Information

Detailed information on the system and the underlying software architecture can be found in the following set of public project deliverable reports:

- D1d Final SAL system
- D2b Final feature extraction
- D3c Human behaviour interpreter
- D4b Final dialogue manager
- D5b Multimodal generation component

See also the slightly older set of reports on SEMAINE-2.0, which contain complementary information.

Furthermore, information about the data collected in the project can be found in the following reports:

- D6b Statistical analysis of data from initial labelled database and recommendations for an economical coding scheme
- D7b Public access to labelled data

3 Configuring the SEMAINE system and its components

(see http://semaine.opendfki.de/wiki/ConfiguringSEMAINE for latest version)

Many aspects of the SEMAINE system can be influenced through configuration files. It suffices to change these values in the respective configuration file, and to restart the system, in order to modify the system behaviour. In particular, it is not necessary to recompile the source code for these changes to take effect.

3.1 System manager and java component runner

The java part of the SEMAINE system if highly flexibly configurable. Whereas in C++, different executables provide a hard-coded bundle of components, the presence of *reflection* in Java makes it possible to define the configuration of a given java process in a configuration file, and to load the actual components at runtime.

Therefore, a single entry point exists for all Java components in the SEMAINE API: the `eu.semaine.system.ComponentRunner`.

3.1.1 ComponentRunner

The simplest way to start the component runner is through a simple script file, such as the start scripts semaine-speech2face.sh and semaine-speech2face.bat. In addition to the memory available to the java process and the configuration settings for ActiveMQ, the ComponentRunner main method requires the user to provide as a command line parameter the configuration file to use. The syntax of that file is described in the following.

3.1.2 Java config file

The key place for the configuration of the Java subsystem is a system config file, such as SEMAINE-3.1/java/config/speech2face.config.

The java config file lists the components to be loaded, can contains any system properties that may be accessed by the java components. In particular, it points to additional config files.

The following configuration settings can be set directly in the main java configuration file.

- `semaine.components` lists the components to be loaded. It is usually a multi-line entry with one component class listed per line. The following is the entry from `speech2face.config`:

```
semaine.components = \
    |eu.semaine.components.meta.SystemManager| \
    |eu.semaine.components.dialogue.interpreters.EmotionInterpreter| \
    |eu.semaine.components.dialogue.interpreters.TurnTakingInterpreter| \
    |eu.semaine.components.dialogue.interpreters.UtteranceInterpreter| \
    |eu.semaine.components.dialogue.interpreters.NonVerbalInterpreter| \
    |eu.semaine.components.dialogue.interpreters.AgentMentalStateInterpreter| \
    |eu.semaine.components.dialogue.actionproposers.UtteranceActionProposer| \
    |eu.semaine.components.dialogue.test.TestGui| \
    |eu.semaine.components.mary.SpeechPreprocessor| \
    |eu.semaine.components.mary.SpeechBMLRealiser| \
    |eu.semaine.components.mary.QueuingSpeechPreprocessor| \
```

```
    |eu.semaine.components.mary.QueuingSpeechBMLRealiser| \
    |eu.semaine.components.control.ParticipantControl| \
    |eu.semaine.components.MessageLogComponent($semaine.messagelog.topic,
$semaine.messagelog.messageselector)| \
    |eu.semaine.components.emotion.EmotionFusion| \
    |eu.semaine.components.nonverbal.NonverbalFusion| \
    |eu.semaine.components.testing.StateLogger| \
    |eu.semaine.components.testing.AgentBehaviourObserver| \
    |eu.semaine.components.dialogue.interpreters.UserPresenceInterpreter| \
```

A minimalistic `semaine.components` entry is found in `semaine-message-logger-only.config`:

```
semaine.components = \
    |eu.semaine.components.meta.SystemManager| \
    |eu.semaine.components.MessageLogComponent($semaine.messagelog.topic,
$semaine.messagelog.messageselector)| \
```

The meaning of the lines is the following:

```
    |eu.semaine.components.meta.SystemManager| \
```

This line instantiates an object of the class `eu.semaine.components.meta.SystemManager` using the default constructor with no arguments.

The following line instantiates `eu.semaine.components.MessageLogComponent` with a constructor taking two string arguments:

```
    |eu.semaine.components.MessageLogComponent($semaine.messagelog.topic,
$semaine.messagelog.messageselector)| \
```

It would be possible to provide literal string values as parameters to the constructor inline; here, however, the values are prefixed by the special character `$`, which indicates that the actual values are to be read from property entries in the file; this way it is possible to change the system configuration by commenting / uncommenting entries in the config file.

```
# Show messages in all topics:
semaine.messagelog.topic = semaine.data.> \
        semaine.callback.>

# Show only dialog state messages:
#semaine.messagelog.topic = semaine.data.state.dialog

# Show all messages, i.e. periodic and event-based ones:
#semaine.messagelog.messageselector =
# Show only event-based messages:
semaine.messagelog.messageselector = event IS NOT NULL
```

The mechanism for instantiating components with a flexible number of string parameters in the constructor is generic. It is used here with the MessageLogComponent.

- `semaine.messagelog.topic` is a configuration setting for the message log component to indicate the list of topics or topic hierarchies for which messages should be sent to the log mechanism.

- `semaine.messagelog.messageselector` is a configuration setting for the message log component to filter messages before sending them to the log mechanism. The syntax is the JMS message selector syntax.
- `semaine.systemmanager.gui` is a boolean property which determines whether the system manager will show a system monitor GUI or not.

If the system monitor gui is shown, it will display a message flow graph. As the system becomes more complex, this graph will be increasingly difficult to read. In order to simplify the graph, the following settings can be used to hide unnecessary detail.

- `semaine.systemmanager.hide.components` provides a list of component names which should not be shown in the message flow graph. The names must match what the component's `getName()` method returns.
- `semaine.systemmanager.hide.topics` provides the same functionality for Topics: any topic listed here will not be shown in the message flow graph.
- `semaine.systemmanager.gui.topics_to_ignore_when_sorting` can be used to exclude certain topics when computing the layout of the components in the message flow graph.

The java components use jog4j for logging. log4j can be extensively configured using a specific configuration file.

- `semaine.log` provides the location of log4j configuration file to use. If `semaine.log` is not set, the file `log4j.properties` is used which is in the same directory as the main java config file. For the config files shipping with SEMAINE, this is SEMAINE-3.1/java/config/log4j.properties. The system property `log4j.logger.semaine` can be used to override the log output specification of the semaine log messages; for example,

```
java -Dlog4j.logger.semaine=DEBUG,stderr ...
```

will log semaine message of level DEBUG or higher to standard error. This is useful for temporarily starting the system with a different log setting from the default without having to change the log4.properties configuration file.

3.1.3 Embedded ActiveMQ

- `semaine.use.embedded.broker` is a boolean property which determines whether the java process will start an embedded ActiveMQ broker. If this is set to true, the ActiveMQ message-oriented middleware will be part of the java process; if it is set to false, an external ActiveMQ server must be started. See also ConfiguringActiveMQ.

3.1.4 User presence interpreter

A number of settings for the user presence interpreter component can be used to tweak how the presence or absence of a user is computed. For voice activity, face detection, and system utterances, thresholds can be set in milliseconds; only when times exceed these thresholds will the state of user presence change. The following are the values currently used in the `speech2face.config` file:

```
# For the user presence interpreter, set the thresholds (in milliseconds)
# which need to be exceeded before a certain event impacts user presence:
semaine.UserPresence.threshold.voiceAppeared = 1000
```

```
semaine.UserPresence.threshold.voiceDisappeared = 20000
semaine.UserPresence.threshold.faceAppeared = 0
semaine.UserPresence.threshold.faceDisappeared = 3000
semaine.UserPresence.threshold.systemStoppedSpeaking = 10000
semaine.UserPresence.threshold.externalUserPresence = 60000
```

These values mean that voice activity needs to go on for at least one second before a user is deemed present, whereas face presence triggers user presence immediately; if the face disappears for more than three seconds, of the voice disappears for more than 20 seconds, the user is deemed absent. User absence decisions are suspended while the system is speaking and for 10 seconds after the system has finished speaking. If an external source determines user presence (such as a gui control), this value is respected for 60 seconds.

3.1.5 Pointers to other config files

Several config file entries identify specific configuration files to be used in different contexts:

- `semaine.character-config` points to the character config file (see below)
- `semaine.stateinfo-config` points to the state information config file (see below)
- `semaine.DM-config` points to the dialogue manager config file (see below).

3.2 Character config file

The character config file (e.g., SEMAINE-3.1/java/config/character-config.xml) contains the definition of the characters' properties, including the TTS voices they can use, their emotional predispositions, and their propensity to take the turn.

In a future version this file may also refer to the facial models that should be used for the visual appearance of the character, as well as any other character properties.

3.3 State info config file

The stateinfo config file (e.g., SEMAINE-3.1/java/config/stateinfo.config) is the backbone of communicating state information between the system components. It defines the short names of any information items – anything the system knows about the current state of the user, the agent, the dialog, and the context –, and defines how they are encoded and decoded in XML for communicating state within the system. For more details, see StateInfo.

In order to use a new information item in the code, it is sufficient to add it to the stateinfo.config file and make sure all producers and consumers of this information use the revised stateinfo.config file.

C++ components that use state information currently expect stateinfo.config to be in the same folder as the executable binary.

3.4 Dialog manager config file

The dialog manager config file (e.g., SEMAINE-3.1/java/config/DM.config) identifies the dialogue templates that are used to drive the verbal behaviour of the agents. By adding or removing template files from the entry “template_files”, the user can change the dialog strategies used.

For example, depending on the intended steps when changing from one character to another, exactly one of the following three config files should be included in the templates list. After finishing a dialog with one of the SAL characters:

- /eu/semaine/components/dialogue/data/templates/CharChangeModeratorEval.xml will bring up a moderator character asking evaluation questions about the user's perception of the quality of the interaction, and then introduce the next character;
- /eu/semaine/components/dialogue/data/templates/CharChangeModerator.xml will bring up the moderator character who will directly introduce the next character;
- /eu/semaine/components/dialogue/data/templates/CharChange.xml will directly change from one character to the next without showing the moderator as intermediary.

The paths shown are interpreted as classpath locations, i.e. the respective files are expected to be in one of the jar files loaded when starting the system or as substructures in a directory that is included in the classpath when starting the system.

3.5 Listener behaviour

An xml file is used to configure the listener behaviour system. It is located in SEMAINE-3.0/Greta/listener/ASconfig.xml. It contains entries such as:

```
<character name="Poppy" mimicry="0.5" backchannel="0.5" noutterance="1">
    <respondTo head="true" face="true" acoustic="true"/>
</character>
```

For each character it defines the probability of generating mimicry, response backchannel (based on the agent's mental state) and utterances.

In order to switch off mimicry, set the mimicry attribute to 0 and the backchannel to 1. Conversely, to generate only mimicry behaviour, set the mimicry attribute to 1 and the backchannel to 0. Keep them at 0.5 for a similar quantity of mimicry and reponse backchannels.

The utterance attribute allows you to block the utterances coming from the dialogue manager. It is not a mandatory tag and if it is not there it is automatically set at 1 (all sentences are let through).

The agent's responsiveness to head, face or acoustic signals is determined by the attributes of the <respondsTo> element. Setting an attribute to "true" means that the listener intent planner generates signals for a certain modality (head, face, acoustic). This tag can be used without looking into the rules.

The individual rules that trigger and determine listener behaviour are quite straightforward. They describe the signals the agent reacts to and how. They are located in SEMAINE-3.0/Greta/listener/rulesfile.xml. For example:

```
<rule name="trigger-AU12">
    <usersignals>
        <usersignal id="1" name="AU12" modality="face"/>
    </usersignals>
    <backchannels probability="1.0" priority="2">
        <mimicry probability="0.6">
            <mimicry_signal name="mouth=smile" modality="face"/>
        </mimicry>
        <response_reactive probability="0.4"/>
    </backchannels>
```

```
</rule>
```

This rule is triggered when the AU12 is detected. It can generate a signal of mimicry (that will be a smile on the face modality) or a response backchannel. The probabilities in the rules should be ignored: they are not used anymore since the action selection has been implemented.

So, in order to avoid that an agent responds to a signal, it suffices to just delete the associated rule.

3.6 Speech input component configurations

The technical details of the opensmile system are determined by a config file such as SEMAINE-3.0/Opensmile/conf/opensmileSemaine3a.conf or c++/src/tum/auxiliary/conf/opensmileSemaine3a.conf (in the source release or the SVN trunk version). The actual config file used is included in the call to Opensmile's SEMAINExtract executable, e.g. in SEMAINE-3.0/Opensmile/semaine-openSMILE-win5-run.bat (for Windows XP systems) and SEMAINE-3.0/Opensmile/semaine-openSMILE-win6-run.bat (for Windows Vista and above) or in bin/semaine-openSMILE-run.bat (in the source release or the SVN trunk version).

The top-level configuration file includes several other configuration files, which cover individual sub-tasks. Comments in the file explain which includes need to be enabled in order to run which configuration. A few most important examples are illustrated here:

- **Voice activity detector**

Two voice activity detectors exist. A simple detector which used a fixed signal energy threshold can be enabled via the configuration file opensmileSemaineVADsimple.conf. The threshold for the RMS frame energy must be adjusted in the file opensmileSemaineVADsimple.conf to your setup by changing the “threshold = xxxx” option in the section [turn:cTurnDetector]. Typical values range from 0.001 to 0.1.

The advanced, self adapting voice activity detector is configured via opensmileSemaineVAD2.conf. Details are found in the D2b report. If the agent voice from the speakers causes problems (feedback), after the system is running for a certain time, uncomment the line “alwaysRejectAgent = 1” (remove the “;”) in the section [vad:cRnnVad] in opensmileSemaineVAD2.conf.

- **Speech recognition**

By default the single stream HMM only recogniser is enabled. On faster systems one may want to try the multi-stream LSTM/HMM hybrid architecture. This can be enabled by uncommenting the line

;\{opensmileSemaineASRms.conf}

in opensmileSemaine3a.conf. Be sure to disable the single stream recogniser by commenting out the line which includes opensmileSemaineASR.conf.

To disable the speech recognition (words and non-linguistics), comment out both opensmileSemaineASR.conf and opensmileSemaineASRms.conf.

- **Emotion recognition**

The emotion recognition is split to three configuration files: feature extraction (e.g. conf_B/opensmileSemaineEmoftAc.conf for feature Set B – see D3c for details on the feature sets), dimensional emotion recognition (e.g. conf_B/opensmileSemaineEmoBling.conf for acoustic and

linguistic features or conf_B/opensmileSemaineEmoBsel.conf for acoustic features only), and detection of the user's level of interest (e.g. conf_B/opensmileSemaineIntB.conf). To disable either the dimensional affect recognition or the interest recognition, comment out the corresponding line. If both are disabled the line including the emotion feature extraction configuration should also be commented out.

Note: The feature extraction, dimensional affect recognition and interest detection configuration files from different feature sets (A, B, and C) may not be mixed.

Futher, the SEMAINExtract executable supports the command-line options -noWords, -noNonverbals, and -noInterest, to selectively disable the semaine components and their functionality as they appear in the GUI (Note, that the information is still computed and displayed in the openSMILE debug output, however it is not sent to the semaine system if the component is disabled – on the other hand, if the extraction of certain things is disabled in the opensmileSemaine3a.conf file, the the information will not be sent, even if the sender component is enabled and thus still visible).

3.7 Video input component configurations

The video input components, when installed, can be configured using C:\Program Files\iBUG\Semaine Video Components\videoConfig.cfg. Among other things, this config file determines whether a USB or Firewire camera is used.

Below all options are listed in the format of the config file, with their default settings:

```
 # -- Visualisation: ON / OFF.
visualisation=ON
```

This determines whether a visualisation of the face detection, 2D-head motion, eye detection, face normalisation, and detected facial points is shown. N.B. If you set smallvisualisation below, not all this information will be displayed because of the lack of screen real estate.

```
# -- Set cameraNr if you use OpenCV. Default is 0. Only useful if you have
multiple cameras attached to your machine
cameranr=0

# -- The directory with all video models
modeldir=C:\Program Files\iBUG\Semaine Video Components\models\
# -- Size of visualisation. ON= small, OFF = big
smallvisualisation=ON
# -- Grabbertype: set to 0 for OpenCV, 1 for cmu1394
grabbertype=0
```

N.B. You should test yourself whether your camera works with opencv. The cmu1394 driver supports most firewire cameras.

```
# -- turning on the nod-shake analysis module
nodshake=ON
# -- turning on the 5D emotion analysis module from head gestures
nodshakedimaff=ON
# -- turning on the head tilt analysis module
tilt=ON
# -- turning on the LBP-based AU detection module
laud=ON
# -- turning on the face registeration module
```

```
useFaceRegistration=ON
# -- turning on the search for profile face module
searchProfileFace=ON
# -- turning on the user presence module
useUserPresence=OFF
```

N.B. This userPresence module is superseded by a java component. It's only here for backwards compatibility.

```
# -- Parameter used by nod/shake detector (window-width)
nodshakewindowSize=20
# -- Turn facial point tracking on/off
facialPointTracking=OFF
# -- Set data directory for facial point tracker
trackdir=c:\temp
```

N.B. Unless you have an incredibly powerful machine, tracking will not work because the frame rate will be too low, and thus the appearance changes between frames too high.

3.8 MARY TTS configuration

The folder SEMAINE-3.1/MARY contains a full installation of the MARY TTS text-to-speech framework, release 4.2.0. Voices that ship with the release are the four voices for the SAL characters and a generic US English voice for the moderator character Greta.

It is possible to use the `mary-component-installer` to install and uninstall languages and voices into this instance of MARY TTS:

```
SEMAINE-3.1\MARY\bin\mary-component-installer.bat (windows)

SEMAINE-3.1/MARY/bin/mary-component-installer (linux/mac)
```

It is advisable to only install the voices that are needed, because more voices mean a higher memory footprint and longer startup times. In order to use the new voices, the java config file needs to point to a properly configured character config file. The SVN repository contains as an example a multilingual character config file, at tags/3.1.0/java/config/character-config-multilingual.xml.

The speech synthesis components are configured using a number of configuration files in SEMAINE-3.1/MARY/conf. The most important config file is marybase.config; its most important setting is “cache = false”. In order to switch on TTS caching to speed up the system, change this to “cache = true”.

4 Configuring the message-oriented middleware ActiveMQ

(see http://semaine.opendfki.de/wiki/ConfiguringActiveMQ for latest version)

Communication in the SEMAINE API passes via ActiveMQ, an open-source message-oriented middleware which implements the Java Message Service (JMS) specification.

We use the publish-subscribe model of JMS for communicating, which is based on the notion of a **Topic**: components can *publish* (send) messages to a Topic or *subscribe* to the topic to receive messages sent to that Topic. In other words, it is a flexible mechanism for n-to-m communication. To establish a communication between two components, it is sufficient for them to use the same Topic name.

ActiveMQ uses a *broker* -- a messaging server to which all components connect. In previous versions of the SEMAINE API, the activemq server had to be started separately; as of SEMAINE-3.1, there is the option of creating an **embedded** broker as part of the java process. The main java config file provides a boolean property `semaine.use.embedded.broker` through which the use of an embedded broker can be switched on or off (see ConfiguringSEMAINE#EmbeddedActiveMQ).

4.1 Connecting to an external ActiveMQ server

Let us first assume that activemq runs as a separate server. In that case, all components will connect to the activemq server via a URL. By default, the ActiveMQ server uses the "OpenWire" protocol (which for activemq is bound to the protocol prefix `tcp://` and port 61616, so that a connection to an activemq server on the local machine corresponds to the URL:

```
tcp://localhost:61616
```

This setting can be provided as a global setting to the SEMAINE API as follows:

For java:

```
java -Djms.url=tcp://localhost:61616 ...
```

For C++:

```
# for Mac / Linux:
export CMS_URL=tcp://localhost:61616

rem for Windows:
set CMS_URL=tcp://localhost:61616
```

4.2 Embedded ActiveMQ

If a SEMAINE java process is configured to run an embedded activemq server as part of the java process itself, the communication between the components in that process and the activemq server will be done in memory, which improves performance.

For all other components in the system, **nothing changes**: they will connect ot this embedded activemq broker in the same way as they connect to an external activemq broker.

4.3 Configuring ActiveMQ

All communication in the SEMAINE API passes via ActiveMQ. Since SEMAINE is a real-time system, it is essential to carefully configure ActiveMQ. For the embedded case, there is an ActiveMQ configuration file included in the java classpath: tags/3.1.0/java/src/eu/semaine/jms/activemq.xml. We also provide a reference config file to use (or start from) when running an external ActiveMQ server, in tags/3.1.0/doc/activemq.xml (tested with ActiveMQ 5.4.1 and 5.3.0).

The key problem to be aware of is producer flow control. When a producer is too fast for one of the consumers, the default behaviour of ActiveMQ is to block the producer in the send method until there is enough free space for the slow consumer to receive additional messages. In practice, that looks as if the system had locked up. To avoid this, we configure `constantPendingMessageLimitStrategy` to keep only the 1000 newest messages for each consumer and to discard the rest.

4.4 Changing the port of an embedded ActiveMQ broker

The usual way to change the port on which an ActiveMQ broker listens for connections would be to change the `transportConnector` setting in the activemq.xml config file. However, to simplify the configuration of embedded brokers, we use the following convention.

If the system property `jms.url` is present on a java process which is configured to use the embedded broker, the port given in the value of that property is used for the embedded broker to listen to connections from external clients.

For example, starting a java process with

```
java -Djms.url=tcp://localhost:61617 -Dsemaine.use.embedded.broker=true
eu.semaine.system.ComponentRunner ...
```

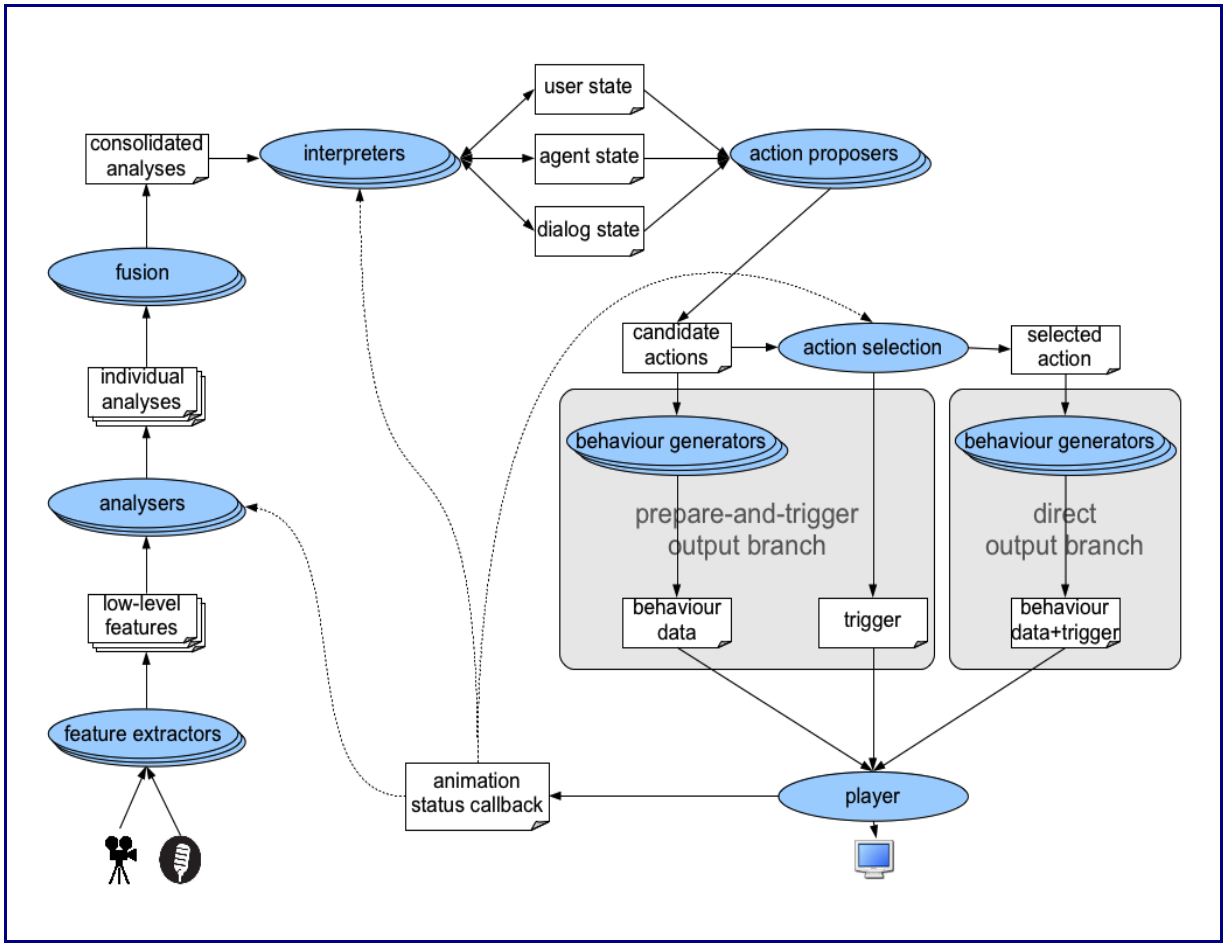
allows other components to connect to activemq via port 61617.

5 SEMAINE Component Architecture

(see http://semaine.opendfki.de/wiki/ComponentArchitecture for latest version)

This page documents how the SEMAINE components are organised into a system. Starting from a conceptual message flow graph, we explain which components exist and via which JMS Topics they communicate.

Conceptually, the message flow graph in SEMAINE looks as follows.



The following subsections describe the three main parts of the system in more detail: analysis of user behaviour, dialogue management, and generation of agent behaviour.

5.1 Analysis of user behaviour

User behaviour is first represented in terms of low-level audio and video features, then in terms of individual analysis results, and then fused together before the dialog model's "current best guess" of the user state is updated.

More details about the analysis of user behaviour can be found in SEMAINE deliverable reports D2b and D3c.

5.1.1 Feature extractors

Feature extractors are modality-specific. They produce feature vectors as key-value pairs, which are typically produced at a fixed frame rate (e.g., every 10 ms for audio features, and for every video frame for video features). The following Topics are currently used.

Topic	Description
semaine.data.analysis.features.voice	* F0frequency [0, 600] (fundamental frequency in Hz) * voiceProb [0, 1] (probability that the current frame is harmonic) * RMSenergy [0, 1] (energy of the signal frame) * LOGenergy [-100, 0] (energy of the signal frame, in dB) (more upon request...!)
semaine.data.analysis.features.video.facedetection	* xPositionTopLeft [0,xCameraResolution] (top left corner of the bounding box of the face detected) * yPositionTopLeft [0, yCameraResolution] (top left corner of the bounding box of the face detected) * width [0,xCameraResolution] (width of the bounding box of the face detected) * height [0,yCameraResolution] (height of the bounding box of the face detected) (all 0 if no face detected)
semaine.data.analysis.features.video.facialpoints	xRightOuterEyeCorner yRightOuterEyeCorner xLeftOuterEyeCorner

	yLeftOuterEyeCorner xRightInnerEyeCorner yRightInnerEyeCorner xLeftInnerEyeCorner yLeftInnerEyeCorner xRightInnerBrowCorner yRightInnerBrowCorner xLeftInnerBrowCorner yLeftInnerBrowCorner xRightOuterBrowCorner yRightOuterBrowCorner xLeftOuterBrowCorner yLeftOuterBrowCorner xRightUpperEyelid yRightUpperEyelid xLeftUpperEyelid yLeftUpperEyelid xRightLowerEyelid yRightLowerEyelid xLeftLowerEyelid yLeftLowerEyelid xRightNostril yRightNostril xLeftNostril yLeftNostril xRightMouthCorner yRightMouthCorner xLeftMouthCorner yLeftMouthCorner xUpperLip yUpperLip xLowerLip yLowerLip xChin yChin xNose yNose xRightPupil yRightPupil xLeftPupil yLeftPupil
semaine.data.analysis.features.video.2dheadmotion	* motionDirection [-π, π] (angle of the motion) * motionMagnitudeNormalised [0, large number] (pixels per frame)

	* motionX [-large number, large number] (pixels per frame)
	* motionY [-large number, large number] (pixels per frame)
semaine.data.analysis.features.video.faceappears	
semaine.data.analysis.features.video.geomfacs	
semaine.data.analysis.features.video.lbpfacs	

5.1.2 Analysers

Analysers produce three types of information: the verbal content recognised; the user's non-verbal behaviour; and the user's emotions. Analysers represent their output as EMMA messages (Johnston et al., 2009), i.e. the specific analysis output is accompanied by a time stamp and a confidence.

Topic	**Description**
semaine.data.state.user.emma.words	verbal content recognised
semaine.data.state.user.emma.nonverbal.voice	voice analysis includes: * voice activity detection * stylised pitch movements * non-verbal vocalizations
semaine.data.state.user.emma.nonverbal.face	face analysis includes: * face presence * facial expression encoded in Action Units
semaine.data.state.user.emma.nonverbal.head	head gestures such as nods, shakes etc.
semaine.data.state.user.emma.emotion.voice	emotion as recognised from the voice
semaine.data.state.user.emma.emotion.face	emotion as recognised from the face
semaine.data.state.user.emma.emotion.head	emotion as recognised from head movements

Verbal content is represented directly in EMMA. For example:

```
<emma:emma version="1.0"
  xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:sequence emma:offset-to-start=”12345” emma:duration=”110”>
    <emma:interpretation
      emma:offset-to-start="12345"
      emma:tokens="bla"
      emma:confidence="0.3"/>
    <emma:interpretation
```

```
      emma:offset-to-start="12390"
      emma:tokens="bloo"
      emma:confidence="0.4"/>
  </emma:sequence>
</emma:emma>
```

The output of the **voice activity detection** (VAD) / the speaking detector looks like this. It needs no confidence. The Speaking Analyser (part of the TumFeatureExtractor) outputs messages when the user starts or stops speaking. These messages are low-level messages, created directly from the VAD output, smoothed only over 3 frames. Thus, some thresholds must be applied in other components to reliably detect continuous segments where the user is speaking and avoid false alarms.

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">

        <semaine:speaking xmlns:semaine="http://www.semaine-
project.eu/semaineml" statusChange="start"/>

    </emma:interpretation>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/semaine:speaking/@statusChange : start, stop

Stylised pitch movements are represented as follows:

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:duration="444"
emma:confidence="0.3">

        <semaine:pitch xmlns:semaine="http://www.semaine-project.eu/semaineml"
direction="rise"/>

    </emma:interpretation>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/semaine:pitch/@direction : rise, fall, rise-fall, fall-rise, high, mid, low

User gender is encoded as shown here:

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">

      <semaine:gender name="female" xmlns:semaine="http://www.semaine-
project.eu/semaineml"/>

    </emma:interpretation>
</emma:emma>
```

Possible values of /emma:emma/emma:interpretation/semaine:gender/@name : male, female, unknown

Non-verbal user vocalizations such as laugh, sigh etc. are encoded as shown in the following example:

```
<emma:emma version="1.0"          xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">
```

```
        <semaine:vocalization xmlns:semaine="http://www.semaine-
project.eu/semaineml" name="(laughter)"/>

    </emma:interpretation>
</emma:emma>
```

Values of /emma:emma/emma:interpretation/semaine:vocalization/@name are currently "(laughter)", "(sigh)" and (breath)".

Whether there is a **face present** is encoded as follows:

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">

        <semaine:face-present xmlns:semaine="http://www.semaine-
project.eu/semaineml" statusChange="start"/>

    </emma:interpretation>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/semaine:face-present/@statusChange : start, stop

Any **action units** recognised from the user's face are encoded such that a separate confidence can be given for each action unit. Example:

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:group>
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">
        <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
            <bml:face au="1"/>
        </bml:bml>
    </emma:interpretation>
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.4">
        <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
            <bml:face au="2"/>
        </bml:bml>
    </emma:interpretation>
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.2">
        <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
            <bml:face au="4"/>
        </bml:bml>
    </emma:interpretation>
  </emma:group>
</emma:emma>
```

Head gestures such as nods or shakes are represented as follows:

```
<emma:emma version="1.0"          xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:duration="444"
emma:confidence="0.3">

      <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
          <bml:head type="NOD" start="12.345" end="12.789"/>
      </bml:bml>

    </emma:interpretation>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/bml:bml/bml:head/@type: NOD, SHAKE, TILT-LEFT, TILT-RIGHT, APPROACH, RETRACT. Left and right are defined subject centred (i.e. left is left for the user).

For all **emotion** messages, the information is encoded using the latest draft of the EmotionML standard (Schröder et al., 2010) as the payload of an EMMA container. For example:

```
<emma:emma xmlns:emma="http://www.w3.org/2003/04/emma" version="1.0">
  <emma:interpretation>
    <emo:emotion xmlns:emo="http://www.w3.org/2009/10/emotionml"
                 dimension-
set="http://www.example.com/emotion/dimension/FSRE.xml">
      <emo:dimension confidence="0.905837" name="arousal" value="0.59999996"/>
      <emo:dimension confidence="0.97505563" name="valence" value="0.42333332"/>
      <emo:dimension confidence="0.9875278" name="unpredictability"
value="0.29333335"/>
      <emo:dimension confidence="0.96318215" name="potency" value="0.31333336"/>
      <intensity confidence=”0.94343144” value=”0.04”/>
    </emo:emotion>
   </emma:interpretation>
</emma:emma>
```

5.1.3 Fusion components

All non-verbal analyses are combined by a NonverbalFusion component; all emotion analyses are combined by an EmotionFusion component, which computes the fused positions on emotion dimensions as a sum of individual positions weighted by the respective confidences.

Topic	Description
semaine.data.state.user.emma.nonverbal	consolidated non-verbal behaviour from all available modalities
semaine.data.state.user.emma.emotion	consolidated and fused emotion analysis based on information from all available modalities

5.2 Dialogue management

The dialogue management is performed by interpreters and action proposers operating on “state” information, i.e. the system's “current best guess” regarding the state of the user, the agent itself and their dialogue.

More information about the dialogue management can be found in SEMAINE deliverable report D4b.

5.2.1 Interpreters

Interpreters take both the analysis results and the existing state information into account when making various interpretations, leading to various state updates. Analyses are thresholded by confidence – only analyses with a sufficiently high confidence lead to a state update.

- EmotionInterpreter updates the user's emotion state based on the fused emotion analyses;
- NonVerbalInterpreter updates the user's nonverbal state based on the fused non-verbal analyses;

- UtteranceInterpreter updates the user state with respect to the words spoken by the user, taking the current dialogue state into account;
- TurnTakingInterpreter takes decisions on the agent's intention to take the turn, based on current user, dialogue and agent state, and updates dialog and agent state accordingly;
- AgentMentalStateInterpreter updates the agent's mental state based on user behaviour.

State information is kept in three Topics:

Topic	Description
semaine.data.state.user.behaviour	all “current best guess” information about the user
semaine.data.state.agent	the current state of the agent
semaine.data.state.dialog	all “current best guess” information regarding the state of the dialog

The state information is accessed via a short name and encoded in XML according to a stateinfo.-config file -- see StateInfo for details.

5.2.2 Action proposers

There are currently two action proposer components.

- UtteranceActionProposer is simultaneously proposer and action selection for verbal utterances. It selects suitable utterances from the available set of utterances for the current character, based on the current state information, and triggers the most suitable one when the agent has a turn-taking intention. This component manages the two output cues, see below.
- ListenerIntentPlanner proposes possible timing and meaning of reactive listener backchannels, as well as non-verbal mimicry by the agent while being a listener. It uses user and agent state information to trigger and select both reactive (meaning-based) and mimicry (behaviour-based) backchannels.

A dedicated (listener) ActionSelection component makes sure that the amount of listener actions stays moderate, and in particular holds back any backchannel intentions while the agent is currently producing a verbal utterance.

Candidate actions are produced to the following Topics.

Topic	Description
semaine.data.action.candidate.function	candidate actions described in terms of the function or meaning of what is to be expressed
semaine.data.action.candidate.behaviour	candidate actions described in terms of concrete behaviours
semaine.data.action.selected.function	selected actions described in terms of the function or meaning of what is to be expressed
semaine.data.action.selected.behaviour	selected actions described in terms of concrete behaviours

The format of candidate and selected actions is identical. Actions in Topics *.function are encoded in FML. This includes verbal utterances like the following:

```
<?xml version="1.0" encoding="UTF-8"?><fml-apml version="0.1">
```

```
   <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML" id="bml_uap_3">
      <bml:speech id="speech_uap_3" language="en-GB" text="Is that so? Tell me
about it." voice="activemary">
         <ssml:mark xmlns:ssml="http://www.w3.org/2001/10/synthesis"
name="speech_uap_3:tm1"/>Is<ssml:mark
xmlns:ssml="http://www.w3.org/2001/10/synthesis"
name="speech_uap_3:tm2"/>that<ssml:mark
xmlns:ssml="http://www.w3.org/2001/10/synthesis" name="speech_uap_3:tm3"/>so?
<ssml:mark xmlns:ssml="http://www.w3.org/2001/10/synthesis"
name="speech_uap_3:tm4"/>Tell<ssml:mark
xmlns:ssml="http://www.w3.org/2001/10/synthesis"
name="speech_uap_3:tm5"/>me<ssml:mark
xmlns:ssml="http://www.w3.org/2001/10/synthesis"
name="speech_uap_3:tm6"/>about<ssml:mark
xmlns:ssml="http://www.w3.org/2001/10/synthesis"
name="speech_uap_3:tm7"/>it.<ssml:mark
xmlns:ssml="http://www.w3.org/2001/10/synthesis" name="speech_uap_3:tm8"/>
      </bml:speech>
   </bml:bml>
   <fml:fml xmlns:fml="http://www.mindmakers.org/fml" id="fml_uap_3">
      <fml:performative end="speech_uap_3:tm4" id="tag1" importance="1"
start="speech_uap_3:tm2" type="like"/>
      <fml:emotion end="speech_uap_3:tm4" id="tag2" importance="1"
start="speech_uap_3:tm2" type="small-surprise"/>
      <fml:performative end="speech_uap_3:tm6" id="tag3" importance="1"
start="speech_uap_3:tm4" type="agree"/>
   </fml:fml>
</fml-apml>
```

Reactive backchannels are also encoded in FML:

```
<fml-apml>
  <fml xmlns="http://www.mindmakers.org/fml" id="fml1">
    <backchannel end="1.8" id="b0" importance="1.0" start="0.0"
type="understanding"/>
    <backchannel end="1.8" id="b1" importance="1.0" start="0.0"
type="disagreement"/>
    <backchannel end="1.8" id="b2" importance="1.0" start="0.0" type="belief"/>
  </fml>
</fml-apml>
```

Mimicry backchannels are encoded in BML:

```
<bml xmlns="http://www.mindmakers.org/projects/BML">
  <head end="1.8" id="s1" start="0.0" stroke="1.0">
    <description level="1" type="gretabml">
      <reference>head=head_shake</reference>
    </description>
  </head>
</bml>
```

5.3 Generation of agent behaviour

Any agent actions must be generated in terms of low-level player data before they can be rendered. In addition to the direct generation branch, the current architecture now also supports a prepare-and-trigger branch.

More information about the generation of agent can be found in SEMAINE deliverable report D5b.

5.3.1 Direct branch

In the direct branch, a selected action is converted into player data using the following intermediate steps.

- SpeechPreprocessor computes the accented syllables and any phrase boundaries as anchors to which any gestural behaviour can be attached. It reads from Topicvs semaine.data.action.selected.function and semaine.data.action.selected.behaviour, and writes its results to semaine.data.action.selected.speechpreprocessed. Whereas conceptually this processing step could work with purely symbolic specification of prosody-based anchor points, the current implementation of the behaviour planner component requires absolute timing information. For this reason the output of the SpeechPreprocessor already contains the detailed timing information.
- BehaviourPlanner determines suitable behaviour elements based on the intended function/meaning of the action. It uses character-specific behaviour lexicons to map FML to BML. It reads from semaine.data.action.selected.speechpreprocessed and writes to semaine.data.synthesis.plan.
- SpeechBMLRealiser carries out the actual speech synthesis, i.e. the generation of audio data. It reads from semaine.data.synthesis.plan; it writes a BML message including the speech timings to semaine.data.synthesis.plan.speechtimings, and the binary audio data including a file header to semaine.data.synthesis.lowlevel.audio.
- BehaviorRealizer reads from semaine.data.synthesis.plan.speechtimings and semaine.data.action.selected.behaviour, and produces the low-level video features in Topics semaine.data.synthesis.lowlevel.video.FAP and semaine.data.synthesis.lowlevel.video.BAP. In addition, it sends two types of information to Topic semaine.data.synthesis.lowlevel.command: (1) the information which modalities form part of a given animation as identified by a unique content ID; and (2) the trigger commands needed to start playing back the animation.
- PlayerOgre is the audiovisual player component. It reads the lowlevel player data from topics semaine.data.synthesis.lowlevel.audio, semaine.data.synthesis.lowlevel.video.FAP and semaine.data.synthesis.lowlevel.video.BAP. A unique content ID is used to match the various parts of a multimodal animation to be rendered. Two types of information are received via the Topic semaine.data.synthesis.lowlevel.command: the information which modalities are expected to be part of a given animation / content ID; and the trigger to start playing the animation. The Player sends callback messages to the topic semaine.callback.output.Animation, to inform about the preparation or playback state of the various animations it receives.

An example of a callback message is the following:

```
<callback xmlns="http://www.semaine-project.eu/semaineml">
  <event data="Animation" id="fml_lip_70" time="1116220" type="start"
contentType="utterance"/>
</callback>
```

Possible event types are “ready”, “start”, “end”, as well as “stopped” and “deleted”.

The contentType parameter describes the type of the animation. Possible content types are "utterance", "listener-vocalisation", "visual-only" and "content-type".

5.3.2 Prepare-and-trigger branch

The prepare-and-trigger branch replicates the processing pipeline of the direct branch, but using different Topics:

- QueuingSpeechPreprocessor reads from semaine.data.action.prepare.function and semaine.data.action.prepare.behaviour, and writes to semaine.data.action.prepare.speechpreprocessed;
- BehaviorPlannerPrep reads from semaine.data.action.prepare.speechpreprocessed and writes to semaine.data.synthesis.prepare;
- QueuingSpeechBMLRealiser reads from semaine.data.synthesis.prepare and writes to semaine.data.synthesis.prepare.speechtimings and semaine.data.synthesis.lowlevel.audio;
- BehaviorRealizerPrep reads from semaine.data.synthesis.prepare.speechtimings and semaine.data.action.prepare.behaviour and writes to semaine.data.synthesis.lowlevel.video.FAP, semaine.data.synthesis.lowlevel.video.BAP and semaine.data.synthesis.lowlevel.command.

The difference to the direct branch is at the two ends of the processing pipeline. At the input end, the UtteranceActionProposer feeds into semaine.data.prepare.function candidate utterances that the current character may perform in the near future. At the output end, the BehaviorRealizerPrep sends the content-level description to the player but it does not send the trigger commands. Instead, when the player has received all necessary parts of a given animation, it sends a “ready” callback message which is then registered by the UtteranceActionProposer. When its utterance selection algorithm determines that the selected utterance already exists in prepared form in the player, all it needs to do is send a trigger command directly from UtteranceActionPoposer to semaine.data.synthesis.lowlevel.command, which then starts the playback of the prepared animation without any further delay. If no prepared version of the selected utterance is available, e.g. because it was unexpected that this utterance was selected, or because the preparation has not completed yet, the utterance is generated using the direct branch.

The prepare-and-trigger branch is used only for full utterances. Listener actions are so short and fast to generate that they always use the direct branch.

Since both branches are technically completely independent, this architecture scales well to multiple computers: it is easy to run the direct branch on one computer and the prepare-and-trigger branch on a different computer if they jointly would over-stretch the CPU resources of a single PC.

6 Protocol for the Player in SEMAINE

(see http://semaine.opendfki.de/wiki/PlayerProtocol for latest version)

Any player component in SEMAINE must follow the following protocol so that it supports ahead-of-time preparation of possible utterances. The player must keep a collection of "Animations" which can be played by a "playCommand".

This protocol is currently implemented by two players: The audio-visual Windows native Player-Ogre using the Greta agent, and the speech-only player in Java class QueuingAudioPlayer.

6.1 Data flow

Low-level player data is sent to the player via the Topics

```
semaine.data.synthesis.lowlevel.*
```

currently

```
semaine.data.synthesis.lowlevel.audio
semaine.data.synthesis.lowlevel.video.FAP
semaine.data.synthesis.lowlevel.video.BAP
```

Incoming messages have the following properties:

- a message type specific to the payload format (currently: BytesMessage for audio, TextMessage for FAP and BAP).
- a data type (obtained by message.getDatatype()) identifying the type of message (current values are "AUDIO", "FAP" and "BAP").
- a content ID and a content creation time (obtained by message.getContentID() and message.getContentCreationTime()) which are used to assemble an Animation, to match data and command messages, and to identify the content item in callback and log messages.
- optionally, a contentType, such as "utterance" or "listener-vocalisation".

The idea is that a unit of player data (an "Animation") is assembled in the player from the individual data items that are coming in (currently, AUDIO, FAP and BAP). Certain data types are optional (currently: AUDIO). A message can either contain the complete data of the given type (currently the case for AUDIO) or it can contain a chunk of data (currently the case for FAP and BAP). A chunk contains information about its position in the Animation; it can be dynamically added even if the Animation is already playing.

6.2 Command messages

There are two types of command messages: messages with data typesdataInfoandplayCommand.

- Data info commands: For a given content ID, define the data types that must be present in the Animation: HASAUDIO, HASFAP and HASBAP. Each can be 0 for "not needed" or 1 for "needed".
- Player commands: For a given content ID, define the playback conditions. This includes the following aspects:

- STARTAT: when to start the playback of the Animation (in milliseconds from the moment when the Animation becomes ready);
- PRIORITY: the priority of the Animation in case of competing Animations;
- LIFETIME: the lifetime of the Animation, counting from the moment when the animation becomes ready. When the lifetime is exceeded and the animation has not started playing, it will be marked as "dead" and removed.

Commands are sent to topic

```
semaine.data.synthesis.lowlevel.command
```

and have the data type `dataInfo`for data info commands and `playCommand` for player start trigger commands.

For every content ID, a playCommand is required in order to play that animation. Without a matching playCommand, an animation will never be played.

A command has the following format:

- its content ID is **identical** to the content ID of the Animation for which it defines playback conditions;
- message format is **TextMessage**; the text consists of space-separated key-value pairs, one pair per line, where the keys are strings and the values are floating point numbers.

The following features are used:

- for playCommand:
- STARTAT (0 means start at the moment when all required parts are present, a positive number means milliseconds after that condition is met)
- LIFETIME (in milliseconds from the moment the animation is triggered; -1 means it will never expire)
- PRIORITY (a value between 0 and 1, where 0 is the lowest and 1 the highest possible priority)
- for dataInfo:
- HASAUDIO (a binary feature, 0 means the Animation does not have audio, 1 means the Animation has audio data)
- HASFAP (a binary feature, 0 means the Animation does not have FAP data, 1 means the Animation has FAP data)
- HASBAP (a binary feature, 0 means the Animation does not have BAP data, 1 means the Animation has BAP data)

Every player command must contain all features of its respective type.

6.3 Callback messages

Event-based callback messages are sent when certain conditions are met for a given Animation. The messages go to Topic

```
semaine.callback.output.Animation
```

and have the following format:

```
<callback xmlns="http://www.semaine-project.eu/semaineml">
   <event id="CONTENT_ID" contentType="CONTENT_TYPE" time="META_TIME"
type="EVENT_TYPE"/>
</callback>
```

where content ID and meta time are like before, and type is one of the following:

- `ready` means the Animation has received all required data, so it is ready for playing back. This event is triggered independently of the question whether a command has been received or not.
- `deleted` means the Animation was removed before it started playing, e.g. because it has exceeded its lifetime in the output queue.
- `start` means the Animation has started playing.
- `stopped` means the Animation was stopped while playing but before it was finished, e.g. because a request to change character was received.
- `end` means the Animation has finished playing.

The `contentType` attribute is present only if the incoming messages had a content type, and reproduces that content type.

6.4 Error conditions

The content ID must be unique for the lifetime of a system. This leads to the following error conditions.

It is an error condition...

- if a data chunk is received for an Animation that has already been discarded (because it finished playing, or exceeded its lifetime in the queue);
- if data is received for a data type that does not form chunks;
- if a playCommand is received for a content ID that has been started already, or that is already discarded;

An error condition should be reported as a WARN log message, and otherwise ignored.

It is **not** an error condition...

- if a second playCommand is received after an animation has become ready but before it started playing. In this case, the new priority etc. overwrites the previous values.

7 Standard and pre-standard representation formats in the SEMAINE system

(see http://semaine.opendfki.de/wiki/RepresentationFormats for latest version)

In view of future interoperability and reuse of components, the SEMAINE API aims to use standard representation formats where that seems possible and reasonable. For example, results of analysis components can be represented using EMMA (Extensible Multi-Modal Annotation), a Wold Wide Web Consortium (W3C) Recommendation. Input to a speech synthesiser can be represented using SSML (Speech Synthesis Markup Language), also a W3C Recommendation. Several other relevant representation formats are not yet standardised, but are in the process of being specified. This includes the Emotion Markup Language EmotionML, used for representing emotions and related states in a broad range of contexts, and the Behaviour Markup Language BML, which describes the behaviour to be shown by an Embodied Conversational Agent (ECA). Furthermore, a Functional Markup Language FML is under discussion, in order to represent the planned actions of an ECA on the level of functions and meanings. By implementing draft versions of these specifications, the SEMAINE API can provide hands-on input to the standardisation process, which may contribute to better standard formats.

On the other hand, it seems difficult to define a standard format for representing the concepts inherent in a given application's logic. To be generic, such an endeavour would ultimately require an ontology of the world. In the current SEMAINE system, which does not aim at any sophisticated reasoning over domain knowledge, a simple custom format named SemaineML is used to represent those pieces of information that are required in the system but which cannot be adequately represented in an existing or emerging standard format. It is conceivable that other applications built on top of the SEMAINE API may want to use a more sophisticated representation such as the Rich Description Format RDF to represent domain knowledge, in which case the API could be extended accordingly.

Whereas all of the aforementioned representation formats are based on the Extensible Markup Language XML, there are a number of data types that are naturally represented in different formats. This is particularly the case for the representations of data close to input and output components. At the input end, low-level analyses of human behaviour are often represented as feature vectors. At the output end, the input to a player component is likely to include binary audio data or player-specific rendering directives.

The following table gives an overview of the representation formats currently supported in the SEMAINE API. The row headings link to pages describing the respective representation format.

Type of data	Representation format	Standardisation status
Low-level input features	string or binary feature vectors	ad hoc
Analysis results	EMMA	W3C Recommendation
Emotions and related states	EmotionML	W3C Working Draft
Domain knowledge	SemaineML	ad hoc
Speech synthesis input	SSML	W3C Recommendation
Functional action plan	FML	very preliminary
Behavioural action plan	BML	draft specification
Low-level output data	binary audio, player commands	player-dependent

7.1 Feature vectors

Feature vectors can be represented in an ad hoc format. In text form, the feature vectors consist of straightforward key-value pairs – one feature per line, values preceding features.

```
0.000860535 rmsEnergy
12.6699 logEnergy
-2.59005e-05 rmsEnergy-De
-0.0809427 logEnergy-De
...
```

As feature vectors may be sent very frequently (e.g., every 10 ms in the SEMAINE system), compact representation is a relevant issue. For this reason, a binary representation of feature vectors is also available. In binary form, the feature names are omitted, and only feature values are being communicated. The first four bytes represent an integer containing the number of features in the vector; the remaining bytes contain the float values one after the other.

7.1.1 Details

In all Topic names, we ommit the prefix “semaine.data.”.

Type of information	Topic	Sending frequency	List of feature names + value range
frontal face detection	analysis.features.video.facedetection	cameraFrameRate	* xPositionTopLeft [0,xCameraResolution] (top left corner of the bounding box of the face detected) * yPositionTopLeft [0, yCameraResolution] (top left corner of the bounding box of the face detected) * width [0,xCameraResolution] (width of the bounding box of the face detected) * height [0,yCameraResolution] (height of the bounding box of the face detected) * (all 0 if no face detected)
head motion	analysis.features.video.2dheadmotion	cameraFrameRate	* motionDirection [-π, π] (angle of the motion) * motionMagnitudeNormalised [0, large number] (pixels per frame) * motionX [-large number, large number] (pixels per frame) * motionY [-large number, large number] (pixels per frame)
facial points	analysis.features.video.facialpoints	cameraFrameRate	xRightOuterEyeCorner yRightOuterEyeCorner xLeftOuterEyeCorner

			yLeftOuterEyeCorner xRightInnerEyeCorner yRightInnerEyeCorner xLeftInnerEyeCorner yLeftInnerEyeCorner xRightInnerBrowCorner yRightInnerBrowCorner xLeftInnerBrowCorner yLeftInnerBrowCorner xRightOuterBrowCorner yRightOuterBrowCorner xLeftOuterBrowCorner yLeftOuterBrowCorner xRightUpperEyelid yRightUpperEyelid xLeftUpperEyelid yLeftUpperEyelid xRightLowerEyelid yRightLowerEyelid xLeftLowerEyelid yLeftLowerEyelid xRightNostril yRightNostril xLeftNostril yLeftNostril xRightMouthCorner yRightMouthCorner xLeftMouthCorner yLeftMouthCorner xUpperLip yUpperLip xLowerLip yLowerLip xChin yChin xNose yNose xRightPupil yRightPupil xLeftPupil yLeftPupil
speech features	analysis.features.voice	microphoneFrameRate	* F0frequency [0,600] (fundamental frequency in Hz) * voiceProb [0,1] (probability that the current frame is harmonic) * RMSenergy [0, 1] (energy of the signal frame)

* LOGenergy [-100,0] (energy of the signal frame, in dB)
* (more upon request...!)

7.2 EMMA

The Extensible Multimodal Annotation Language EMMA, a W3C Recommendation, is “an XML markup language for containing and annotating the interpretation of user input”. As such, it is a wrapper language that can carry various kinds of payload representing the interpretation of user input. The EMMA language itself provides, as its core, the `<emma:interpretation>` element, containing all information about a single interpretation of user behaviour. Several such elements can be enclosed within an `<emma:one-of>` element in cases where more than one interpretation is present. An interpretation can have an `emma:confidence` attribute, indicating how confident the source of the annotation is that the interpretation is correct; time-related information such asemma:start,!emma:end, andemma:duration, indicating the time span for which the interpretation is provided; information about the modality upon which the interpretation is based, through the `emma:medium` and `emma:mode` attributes; and many more.

The following listing shows an example EMMA document carrying an interpretation of user behaviour represented using EmotionML. The interpretation refers to a start time. It can be seen that the EMMA wrapper elements and the EmotionML content are in different XML namespaces, so that it is unambiguously determined which element belongs to which part of the annotation.

```
<emma:emma xmlns:emma="http://www.w3.org/2003/04/emma" version="1.0">
  <emma:interpretation emma:start="123456789">
    <emotion xmlns="http://www.w3.org/2009/10/emotionml" dimension-
set="http://www.example.com/emotion/dimension/FSRE.xml">
      <dimension name="arousal" value="0.23"/>
      <dimension name="valence" value="0.62"/>
    </emotion>
  </emma:interpretation>
</emma:emma>
```

EMMA can also be used to represent Automatic Speech Recognition (ASR) output, either as the single most probable word chain or as a word lattice, using the< emma:lattice>element.

7.2.1 Details

Skeleton for all EMMA documents:

All EMMA documents MUST have a top-level element, and SHOULD have at least one element. That interpretation SHOULD a time stamp, given in its attribute "emma:offset-to-start", and MAY have a confidence, given in the attribute "emma:confidence".

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">
      my annotation
    </emma:interpretation>
</emma:emma>
```

If the document contains a single annotation, the element SHOULD be a direct child of . A sequence of interpretations can be represented using the element, as for keywords spotted. A collection of interpretations with different probabilities can be represented using the element, as e.g. for interest.

For the individual types of content / payload, we use by default the same representation as for the "current best guess" user state, unless there are reasons speaking against it.

We distinguish verbal information, emotion-related information, and non-verbal information.

7.2.2 Verbal information

Type of information	Topic
keywords spotted	state.user.emma.words

Keywords

```
<emma:emma version="1.0"
  xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:sequence emma:offset-to-start="12345" emma:duration="110">
    <emma:interpretation
      emma:offset-to-start="12345"
      emma:tokens="bla"
      emma:confidence="0.3"/>
    <emma:interpretation
      emma:offset-to-start="12390"
      emma:tokens="bloo"
      emma:confidence="0.4"/>
  </emma:sequence>
</emma:emma>
```

7.2.3 Emotion-related information

Type of information	Topic
emotion	state.user.emma.emotion.(modality)
interest	state.user.emma.emotion.(modality)

Emotion

The global user emotion is represented using the five dimensions intensity, arousal, valence, unpredictability and potency.

```
<?xml version="1.0" encoding="UTF-8"?><emma:emma
xmlns:emma="http://www.w3.org/2003/04/emma" version="1.0">
   <emma:interpretation>
      <emotion xmlns="http://www.w3.org/2009/10/emotionml" dimension-
set="http://www.example.com/emotion/dimension/FSRE.xml">
         <intensity confidence="0.30086732" value="0.4115755"/>
         <dimension confidence="0.9518124" name="arousal" value="0.1852386"/>
         <dimension confidence="0.2734806" name="valence" value="0.7791835"/>
         <dimension confidence="0.22194415" name="unpredictability"
value="0.09359175"/>
         <dimension confidence="0.2912501" name="potency" value="0.050632834"/>
      </emotion>
```

```
   </emma:interpretation>
</emma:emma>
```

Interest

User interest is represented using a custom vocabulary of interest-related category labels: *bored*, *neutral*, and *interested*. The confidence is used to indicate the extent to which each of the three categories is recognised.

```
<?xml version="1.0" encoding="UTF-8"?><emma:emma
xmlns:emma="http://www.w3.org/2003/04/emma" version="1.0">
   <emma:interpretation>
      <emotion xmlns="http://www.w3.org/2009/10/emotionml" category-
set="http://www.semaine-project.eu/emo/category/interest.xml">
         <category confidence="0.6955442" name="bored"/>
      </emotion>
      <emotion xmlns="http://www.w3.org/2009/10/emotionml" category-
set="http://www.semaine-project.eu/emo/category/interest.xml">
         <category confidence="0.24825269" name="neutral"/>
      </emotion>
      <emotion xmlns="http://www.w3.org/2009/10/emotionml" category-
set="http://www.semaine-project.eu/emo/category/interest.xml">
         <category confidence="0.6315944" name="interested"/>
      </emotion>
   </emma:interpretation>
</emma:emma>
```

7.2.4 Non-verbal information

Type of information	Topic
head movement	state.user.emma.nonverbal.head
user speaking	state.user.emma.nonverbal.voice
pitch direction	state.user.emma.nonverbal.voice
gender	state.user.emma.nonverbal.voice
nonverbal vocalizations	state.user.emma.nonverbal.voice
face presence	state.user.emma.nonverbal.face
action units	state.user.emma.nonverbal.face

Head movement

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:duration="444"
emma:confidence="0.3">

      <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
          <bml:head type="NOD" start="12.345" end="12.789"/>
      </bml:bml>

    </emma:interpretation>
</emma:emma>
```

The payload format is the same as for user state -- here it is below emma:interpretation, there it is below semaine:user-state.

Note that the proposal includes the redundant specification of time: start time is in emma:interpretation/@emma:offset-to-start (in milliseconds), and in bml:head/@start (in seconds). End time is given indirectly by emma:interpretation/@emma:duration (in milliseconds), and directly through bml:head/@end (in seconds). Experience will tell whether this double representation is useful.

Possible values for /emma:emma/emma:interpretation/bml:bml/bml:head/@type: NOD, SHAKE, TILT-LEFT, TILT-RIGHT, APPROACH, RETRACT. Left and right are defined subject centred (i.e. left is left for the user).

User speaking

The output of the voice activity detection (VAD) / the speaking detector looks like this. It needs no confidence. The Speaking Analyser (part of the TumFeatureExtractor) outputs messages when the user starts or stops speaking. These messages are low-level messages, created directly from the VAD output, smoothed only over 3 frames. Thus, some thresholds must be applied in other components to reliably detect continuous segments where the user is really speaking.

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">

        <semaine:speaking xmlns:semaine="http://www.semaine-
project.eu/semaineml" statusChange="start"/>

    </emma:interpretation>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/semaine:speaking/@statusChange : start, stop

Pitch direction

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:duration="444"
emma:confidence="0.3">

        <semaine:pitch xmlns:semaine="http://www.semaine-project.eu/semaineml"
direction="rise"/>

    </emma:interpretation>
</emma:emma>
```

The core difference with the user state is that here we have a start and a duration.

Possible values for /emma:emma/emma:interpretation/semaine:pitch/@direction : rise, fall, rise-fall, fall-rise, high, mid, low

Gender

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">
```

```
      <semaine:gender name="female" xmlns:semaine="http://www.semaine-
project.eu/semaineml"/>

    </emma:interpretation>
</emma:emma>
```

Possible values of /emma:emma/emma:interpretation/semaine:gender/@name : male, female, unknown

Nonverbal vocalisations

Any non-verbal vocalizations produced by the user.

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">

        <semaine:vocalization xmlns:semaine="http://www.semaine-
project.eu/semaineml" name="(laughter)"/>

    </emma:interpretation>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/semaine:vocalization/@name : (laughter), (sigh), (breath)

Face presence

Whether there is a face currently present.

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">

        <semaine:face-present xmlns:semaine="http://www.semaine-
project.eu/semaineml" statusChange="start"/>

    </emma:interpretation>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/semaine:face-present/@statusChange : start, stop

Action units

Any action units recognised from the user's face.

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:group>
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.3">
        <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
            <bml:face au="1"/>
        </bml:bml>
    </emma:interpretation>
```

```
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.4">
        <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
            <bml:face au="2"/>
        </bml:bml>
    </emma:interpretation>
    <emma:interpretation  emma:offset-to-start="12345" emma:confidence="0.2">
        <bml:bml xmlns:bml="http://www.mindmakers.org/projects/BML">
            <bml:face au="4"/>
        </bml:bml>
    </emma:interpretation>
  </emma:group>
</emma:emma>
```

Possible values for /emma:emma/emma:interpretation/bml:bml/bml:face/@au : a single integer number

7.3 EmotionML

The Emotion Markup Language EmotionML is, as of October 2010, a public Working Draft at the W3C.

The SEMAINE API is one of the first pieces of software to implement EmotionML. It is our intention to provide an implementation report as input to the W3C standardisation process in due course, highlighting any problems encountered with the current draft specification in the implementation.

EmotionML aims to make concepts from major emotion theories available in a broad range of technological contexts. Being informed by the affective sciences, EmotionML recognises the fact that there is no single agreed representation of affective states, nor of vocabularies to use. Therefore, an emotional state<emotion>can be characterised using four types of descriptions: `<category>`, `<dimension>`, `<appraisal>` and `<action-tendency>`. Furthermore, the vocabulary used can be identified. The EmotionML markup the example on the EMMA page uses a dimensional representation of emotions, using the dimension set “FSRE.xml”, out of which two dimensions are annotated: arousal and valence.

EmotionML is aimed at three use cases: 1. Human annotation of emotion-related data; 2. automatic emotion recognition; and 3. generation of emotional system behaviour. In order to be suitable for all three domains, EmotionML is conceived as a “plug-in” language that can be used in different contexts. In the SEMAINE API, this plug-in nature is applied with respect to recognition, centrally held information, and generation, where EmotionML is used in conjunction with different markups. EmotionML can be used for representing the user emotion currently estimated from user behaviour, as payload to an EMMA message. It is also suitable for representing the centrally held information about the user state, the system's “current best guess” of the user state independently of the analysis of current behaviour. Furthermore, the emotion to be expressed by the system can also be represented by EmotionML. In this case, it is necessary to combine EmotionML with the output languages FML, BML and SSML.

7.4 SemaineML

A number of custom representations are needed to represent the kinds of information that play a role in the SEMAINE demonstrator systems. Currently, this includes the centrally held beliefs about the user state, the agent state, and the dialogue state. Most of the information represented here is domain-specific and does not lend itself to easy generalisation or reuse. The following shows an example of a dialogue state representation, focused on the specific situation of an agent-user dialogue targeted in the SEMAINE system.

```
<dialog-state xmlns="http://www.semaine-project.eu/semaineml" version="0.0.1">
   <speaker who="agent"/>
   <listener who="user"/>
</dialog-state>
```

7.5 SSML

The Speech Synthesis Markup Language SSML is a well-established W3C Recommendation supported by a range of commercial text-to-speech (TTS) systems. It is the most established of the representation formats described in this section. The main purpose of SSML is to provide information to a TTS system on how to speak a given text. This includes the possibility to add `<emphasis>` on certain words, to provide pronunciation hints via a `<say-as>` tag, to select a `<voice>` which is to be used for speaking the text, or to request a `<break>` at a certain point in the text. Furthermore, SSML provides the possibility to set markers via the SSML `<mark>` tag. The following shows an example SSML document that could be used as input to a TTS engine. It requests a female US English voice; the word “wanted” should be emphasised, and there should be a pause after “then”.

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
    xml:lang="en-US">
  <voice gender="female">
    And then <break/> I <emphasis>wanted</emphasis> to go.
  </voice>
</speak>
```

7.6 FML

The functional markup language FML is still under discussion. Its functionality being needed nevertheless, a working language FML-APML was created as a combination of the ideas of FML with the former Affective Presentation Markup Language APML. The following shows an example FML-APML document which contains the key elements. An `<fml-apml>` document contains a `<bml>` section in which the `<speech>` content contains `<ssml:mark>` markers identifying points in time in a symbolic way. An `<fml>` section then refers to those points in time to represent the fact, in this case, that an announcement is made and that the speaker herself is being referred to between marks `s1:tm2` and `s1:tm4`. This information can be used, for example, to generate relevant gestures when producing behaviour from the functional descriptions.

The representations in the `<fml>` section are provisional and are likely to change as consensus is formed in the community.

```
<fml-apml version="0.1">
  <bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
    <speech id="s1" language="en-US" text="Hi, I'm Poppy."
```

```
        ssml:xmlns="http://www.w3.org/2001/10/synthesis">
         <ssml:mark name="s1:tm1"/>
           Hi,
           <ssml:mark name="s1:tm2"/>
           I'm
           <ssml:mark name="s1:tm3"/>
           Poppy.
           <ssml:mark name="s1:tm4"/>
    </speech>
  </bml>
  <fml xmlns="http://www.mindmakers.org/fml" id="fml1”>
    <performative id="p2" type="announce" start="s1:tm1" end="s1:tm4"/>
    <world id="w1" ref_type="person" ref_id="self" start="s1:tm2" end="s1:tm4"/>
  </fml>
</fml-apml>
```

For the conversion from FML to BML, information about pitch accents and boundaries is useful for the prediction of plausible behaviour time-aligned with the macro-structure of speech. In our current implementation, a speech preprocessor computes this information using TTS technology. The information is added to the end of the <speech> section as shown below. This is an ad hoc solution which should be reconsidered in the process of specifying FML.

```
<fml-apml version="0.1">
  <bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
    <speech id="s1" language="en_US" text="Hi, I'm Poppy."
        ssml:xmlns="http://www.w3.org/2001/10/synthesis">
         <ssml:mark name="s1:tm1"/>
           Hi,
           <ssml:mark name="s1:tm2"/>
           I'm
           <ssml:mark name="s1:tm3"/>
           Poppy.
           <ssml:mark name="s1:tm4"/>
         <pitchaccent id="xpa1" start="s1:tm1" end="s1:tm2"/>
         <pitchaccent id="xpa2" start="s1:tm3" end="s1:tm4"/>
         <boundary id="b1" time="s1:tm4"/>
    </speech>
  </bml>
  <fml xmlns="http://www.mindmakers.org/fml" id="fml1”>
    <performative id="p2" type="announce" start="s1:tm1" end="s1:tm4"/>
    <world id="w1" ref_type="person" ref_id="self" start="s1:tm2" end="s1:tm4"/>
  </fml>
</fml-apml>
```

7.7 BML

The aim of the Behaviour Markup Language BML is to represent the behaviour to be realised by an Embodied Conversational Agent. BML is at a relatively concrete level of specification, but is still in draft status. A standalone BML document is partly similar to the <bml> section of an FML-APML document; however, whereas the <bml> section of FML-APML contains only a <speech> tag, a BML document can contain elements representing expressive behaviour in the ECA at a broad range of levels, including <head>, <face>, <gaze>, <body>, <speech> and others. The following shows an example of gaze and head nod behaviour added to the example from FML.

```
<bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
```

```
  <speech id="s1" language="en_US" text="Hi, I'm Poppy."
      ssml:xmlns="http://www.w3.org/2001/10/synthesis">
    <ssml:mark name="s1:tm1"/>
    Hi,
    <ssml:mark name="s1:tm2"/>
    I'm
    <ssml:mark name="s1:tm3"/>
    Poppy.
   <ssml:mark name="s1:tm4"/>
    <pitchaccent id="xpa1" start="s1:tm1" end="s1:tm2"/>
    <pitchaccent id="xpa2" start="s1:tm3" end="s1:tm4"/>
    <boundary id="b1" time="s1:tm4"/>
  </speech>
  <gaze id="g1" start="s1:tm1" end="s1:tm4">
    ...
  </gaze>
  <head id="h1" start="s1:tm3" end="s1:tm4" type="NOD">
    ...
  </head>
</bml>
```

While creating an audio-visual rendition of the BML document, we use TTS to produce the audio and the timing information needed for lip synchronisation. Whereas BML in principle previews a `<lip>` element for representing this information, we are uncertain how to represent exact timing information with it in a way that preserves the information about syllable structure and stressed syllables. For this reason, we currently use a custom representation based on the MaryXML format from the MARY TTS system to represent the exact timing of speech sounds. The following shows the timing information for the word “Poppy”, which is a two-syllable word of which the first one is the stressed syllable.

```
<bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
  <speech id="s1" language="en_US" text="Hi, I'm Poppy."
      ssml:xmlns="http://www.w3.org/2001/10/synthesis"
      mary:xmlns="http://mary.dfki.de/2002/MaryXML">
    ...
    <ssml:mark name="s1:tm3"/>
    Poppy.
    <mary:syllable stress="1">
      <mary:ph d="0.092" end="1.011" p="p"/>
      <mary:ph d="0.112" end="1.123" p="A"/>
      <mary:ph d="0.093" end="1.216" p="p"/>
    </mary:syllable>
    <mary:syllable>
      <mary:ph d="0.141" end="1.357" p="i"/>
    </mary:syllable>
    <ssml:mark name="s1:tm4"/>
   ...
</bml>
```

The custom format we use for representing timing information for lip synchronisation clearly deserves to be revised towards a general BML syntax, as BML evolves.

7.8 Player data

Player data is currently treated as unparsed data. Audio data is binary, whereas player directives are considered to be plain text. This works well with the current MPEG-4 player we use but may need to generalised as other players are integrated into the system.

8 State information defined in the SEMAINE system

(see http://semaine.opendfki.de/wiki/StateInfo for latest version)

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`.

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 is at trunk/java/config/stateinfo.config. 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.

The following tables aim to give a complete account of the meaning of state information currently defined.

These are the "current best guess" information we have on anything we know anything about. **Short name** is what can be used in `StateReceiver.getCurrentBestGuess()` to obtain the latest value of that variable. **XPath** is the XPath expression used for extracting the information from messages. **Possible values** is a data type and, as applicable, a value range or a list of values – e.g., "float: [0,1[" or "string: a lot, a little, not really" or "boolean". **Definition** is the meaning of the information, in human-readable terms.

The following namespace prefixes are used in the XPath expressions.

Prefix	Namespace URI
semaine	http://www.semaine-project.eu/semaineml
bml	http://www.mindmakers.org/projects/BML
emotion	http://www.w3.org/2009/10/emotionml

8.1 User State

Topic: state.user.behaviour

Short name	XPath	Possible values	Definition
headGesture	/semaine:user-state/bml:bml/bml:head/@type	string: NOD, SHAKE, TILT-LEFT, TILT-RIGHT, APPROACH, RETRACT	User is currently performing the given head gesture (note that TILT-LEFT and TILT-RIGHT is left and right from the user's perspective.
headGestureStarted	/semaine:user-state/bml:bml/bml:head/@start	float: [0, very large number]	time when the user started with the gesture, since the system last got ready, in seconds
headGestureStopped	/semaine:user-state/bml:bml/bml:head/@end	float: [0, very large number]	time when the user stopped the gesture, since the system last got ready, in

			seconds
facialExpression	/semaine:user-state/bml:bml/bml:face/@shape	string: SMILE, FROWN, RAISED_EYEBROWS (placeholder for a better solution!!)	simplified categorisation of facial expressions
facialActionUnits	/semaine:user-state/bml:bml/bml:face/@au	string: space-separated list of action unit identifiers	action units detected in the face
facialExpressionStarted	/semaine:user-state/bml:bml/bml:face[@shape]/@start	float: [0, very large number]	time when the user started with the gesture, since the system last got ready, in seconds
facialExpressionStopped	/semaine:user-state/bml:bml/bml:face[@shape]/@end	float: [0, very large number]	time when the user stopped the gesture, since the system last got ready, in seconds
facialActionUnitsStarted	/semaine:user-state/bml:bml/bml:face[@au]/@start	float: [0, very large number]	time when the user started with the gesture, since the system last got ready, in seconds
facialActionUnitsStopped	/semaine:user-state/bml:bml/bml:face[@au]/@end	float: [0, very large number]	time when the user stopped the gesture, since the system last got ready, in seconds
pitchDirection	/semaine:user-state/semaine:pitch/@direction	string: rise, fall, rise-fall, fall-rise, high, mid, low	abstraction of the pitch contour, if the pitch values exceed a certain threshold (tbd)
speaking	/semaine:user-state/semaine:speaking/@status	boolean: true, false	whether the user is currently speaking
vocalization	/semaine:user-state/semaine:vocalization/@name	string: (laughter), (sigh)	a non-verbal vocalisation produced by the user
facePresent	/semaine:user-state/semaine:face-present/@status	boolean: true, false	whether a face is currently present

interest	/semaine:user-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/interest.xml']/emotion:dimension[@name='interest']/@value	float: [0, 1]	the degree to which the user seems to be bored (<0.5), or interested (>0.5)
valence	/semaine:user-state/emotion:emotion[@dimension-set='http://www.example.com/emotion/dimension/FSRE.xml']/emotion:dimension[@name='valence']/@value	float: [0, 1]	negative (<0.5) or positive (>0.5)
arousal	/semaine:user-state/emotion:emotion[@dimension-set='http://www.example.com/emotion/dimension/FSRE.xml']/emotion:dimension[@name='arousal']/@value	float: [0, 1]	below average arousal (<0.5) or higher than average arousal (>0.5)
potency	/semaine:user-state/emotion:emotion[@dimension-set='http://www.example.com/emotion/dimension/FSRE.xml']/emotion:dimension[@name='potency']/@value	float: [0, 1]	out of control (<0.5) or in control (>0.5)
unpredictability	/semaine:user-state/emotion:emotion[@dimension-set='http://www.example.com/emotion/dimension/FSRE.xml']/emotion:dimension[@name='unpredictability']/@value	float: [0, 1]	predictable situation (<0.5) or unpredictable situation (>0.5)
intensity	/semaine:user-state/emotion:emotion/emotion:intensity/@value	float: [0, 1]	how intense the emotion is, globally
emotion-quadrant	/semaine:user-state/emotion:emotion[@category-set='http://www.semaine-project.eu/emo/category/four-quadrants.xml']/emotion:category/@name	string: positiveActive, negativeActive, positivePassive, negativePassive, neutral	
userUtterance	/semaine:user-state/semaine:userutterance/@utterance	string	A list (seperated with spaces) of all the detected words

userUtteranceStartTime	/semaine:user-state/semaine:userutterance/@starttime	float: [0, inf]	The starting time of the detected words
userUtteranceFeatures	/semaine:user-state/semaine:userutterance/@features	string	A list (separated with spaces) of all the detected linguistic features that are extracted from the detected words
gender	/semaine:user-state/semaine:gender/@name	string: male, female, unknown	The user's gender.

8.2 Agent State

Topic: state.agent

Short name	XPath	Possible values	Definition
needToSpeak	Suggestion: /semaine:agent-state/semaine:needtospeak/@value	int: [-100, 100]	A value that describes how eager the agent wants to speak. Values below 0 indicate the agent wants to be silent, values above 0 indicate that the agent wants to speak. A higher or a lower value increases the strength of this.
turnTakingIntention	Suggestion: /semaine:agent-state/semaine:turntakingintention/@value	String: startSpeaking, stopSpeaking	Describes the turn taking intention of the agent, if the agent wants to start speaking or if it wants to stop.
agentUtterance	/semaine:agent-state/semaine:agentutterance/@utterance	String	The text currently spoken by the agent, be it as a speaker or as a listener. empty or empty string if agent is currently not speaking.
agentUtteranceStartTime	/semaine:agent-state/semaine:agentutterance/@starttime	long	the system time at which the agentUtterance started.

agentHead	/semaine:agent-state/semaine:agent-head/@behaviour		The behaviour currently shown by the agent, be it as a speaker or as a listener. empty or empty string if agent is currently not showing any behaviour on this modality.
agentHeadStartTime	/semaine:agent-state/semaine:agent-head/@starttime	long	the system time at which the behaviour started.
agentFace	/semaine:agent-state/semaine:agent-face/@behaviour		The behaviour currently shown by the agent, be it as a speaker or as a listener. empty or empty string if agent is currently not showing any behaviour on this modality.
agentFaceStartTime	/semaine:agent-state/semaine:agent-face/@starttime	long	the system time at which the behaviour started.
agentGaze	/semaine:agent-state/semaine:agent-gaze/@behaviour		The behaviour currently shown by the agent, be it as a speaker or as a listener. empty or empty string if agent is currently not showing any behaviour on this modality.
agentGazeStartTime	/semaine:agent-state/semaine:agent-	long	the system time at which the behaviour

	gaze/@starttime		started.
agentGesture	/semaine:agent-state/semaine:agent-gesture/@behaviour	The behaviour currently shown by the agent, be it as a speaker or as a listener. empty or empty string if agent is currently not showing any behaviour on this modality.	
agentGestureStartTime	/semaine:agent-state/semaine:agent-gesture/@starttime	long	the system time at which the behaviour started.
agentTorso	/semaine:agent-state/semaine:agent-torso/@behaviour	The behaviour currently shown by the agent, be it as a speaker or as a listener. empty or empty string if agent is currently not showing any behaviour on this modality.	
agentTorsoStartTime	/semaine:agent-state/semaine:agent-torso/@starttime	long	the system time at which the behaviour started.
agreement	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/communicative-functions.xml']/emotion:dimension[@name='agreement']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) disagreement-agreement
acceptance	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/communicative-functions.xml']/emotion:dimension	float: [0, 1]	bipolar scale (neutral point: 0.5) refusal-acceptance

	[@name='acceptance']/@value		
belief	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/communicative-functions.xml']/emotion:dimension[@name='belief']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) disbelief-belief
liking	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/communicative-functions.xml']/emotion:dimension[@name='liking']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) disliking-liking
understanding	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/communicative-functions.xml']/emotion:dimension[@name='understanding']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) no-understanding - understanding
interest	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/communicative-functions.xml']/emotion:dimension[@name='interest']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) no-interest - interest
anger	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='anger']/@value	float: [0, 1]	unipolar scale (0 = not present, 1 = intensely present)
sadness	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='sadness']/@value	float: [0, 1]	unipolar scale (0 = not present, 1 = intensely present)
amusement	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='amusement']/@value	float: [0, 1]	unipolar scale (0 = not present, 1 = intensely present)

happiness	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='happiness']/@value	float: [0, 1]	unipolar scale (0 = not present, 1 = intensely present)
contempt	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='contempt']/@value	float: [0, 1]	unipolar scale (0 = not present, 1 = intensely present)
anticipation	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='anticipation']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) low-to-high anticipation
solidarity	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='solidarity']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) low-to-high solidarity
antagonism	/semaine:agent-state/emotion:emotion[@dimension-set='http://www.semaine-project.eu/emo/dimension/listener-meanings.xml']/emotion:dimension[@name='antagonism']/@value	float: [0, 1]	bipolar scale (neutral point: 0.5) low-to-high antagonism

8.3 Dialog State

Topic: state.dialog

Short name	XPath	Possible values	Definition
userTurnState	/semaine:dialog-state/semaine:user/@believesHasTurn	boolean: true, false	
agentTurnState	/semaine:dialog-state/semaine:agent/@believesHasTurn	String: speaking, listening, expectingAnswer	
convState	/semaine:dialog-state/semaine:agent/@convState	String: listening, asking	

8.4 Context State

Topic: state.context

Short name	XPath	Possible values	Definition

userPresent	/semaine:situational-context/semaine:user/@status	string: present, absent	whether there is currently a user ready to interact with the system
character	/semaine:situational-context/semaine:character/@name	string: Poppy, Prudence, Obadiah, Spike, Moderator	the name of the character who is currently the active character in the system.
nextCharacter	/semaine:situational-context/semaine:character/@next	string: Poppy, Prudence, Obadiah, Spike	only used with character='Moderator' -- the name of the next character that the moderator should introduce
dialogContext	/semaine:situational-context/semaine:dialog-context/@name	string: AnnounceNextCharacter, Introduction, Questions	only used with character='Moderator' -- the type of dialog the moderator should perform

9 Building emotion-oriented systems with the SEMAINE API

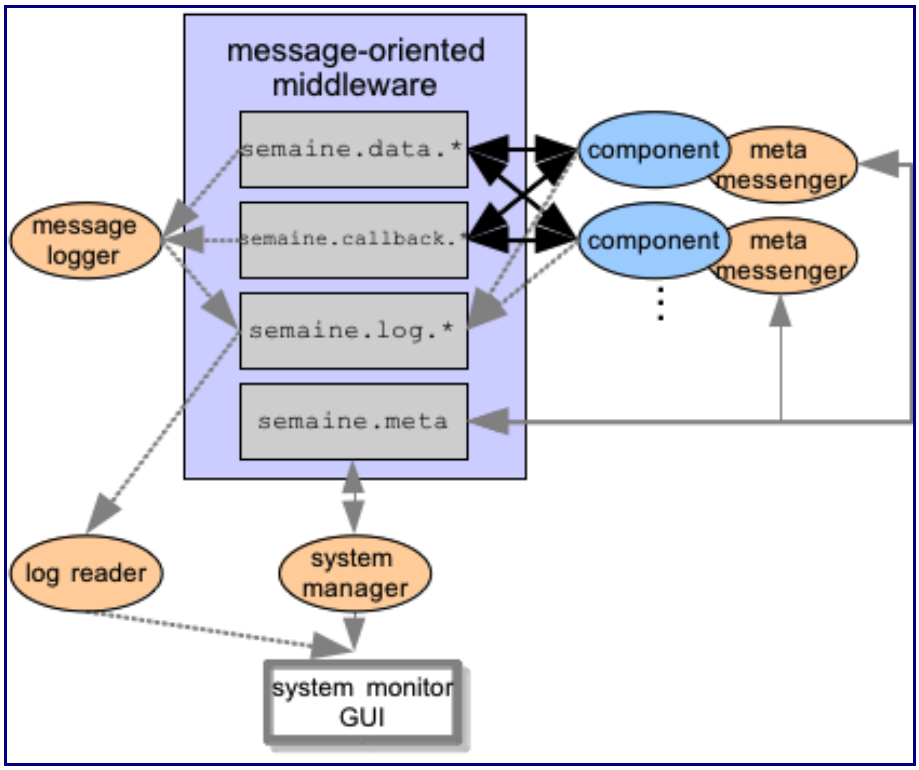
(see http://semaine.opendfki.de/wiki/BuildingSystems for latest version)

The SEMAINE API is the component integration middleware created for the SEMAINE project, serving as the integration layer for system components in SEMAINE.

The SEMAINE API uses a message-oriented middleware (MOM) for all communication in the system. As a result, all communication is asynchronous, which decouples the various parts of the system. The actual processing is done in "components", which communicate with one another over "Topics" below the named Topic hierarchysemaine.data.*. Each component has its own "meta messenger", which interfaces between the component and a central system manager. When a component is started, its meta messenger registers with the system manager over a special meta communication channel, the Topicsemaine.meta. At registration time, the meta messenger describes the component in terms of the data Topics that it sends data to and that it receives data from; if the component is an input or output component (in the sense of the user interface), that status is communicated as well. The system manager is keeping track of the components that have been registered, and checks at regular intervals whether all components are still alive by sending a "ping". In reply to such a ping, each meta messenger confirms the respective component's status and sends debug information such as the average time spent processing incoming requests. The system manager keeps track of the information about registered components, and sends global meta messages informing all components that the overall system is ready or, if a component has an error or is stalled, that the system is not ready. Also, the system manager resets a global timer to zero when the system becomes ready. All components use this global time via their meta messenger, and thus can meaningfully communicate about timing of user and system events even across different computers with potentially unsynchronised hardware clocks.

A centralised logging functionality uses the Topics below `semaine.log.*`. By convention, messages are sent to `semaine.log.<component>.<severity>`, e.g. the Topic `semaine.log.UtteranceInterpreter.debug` would be used for debug messages of component UtteranceInterpreter. The severities used are "debug", "info", "warn" and "error". Through this design, it is possible for a log reader to subscribe, e.g., to all types of messages from one component, or to all messages from all components that have at least severity "info", etc. Furthermore, a configurable message logger can optionally be used to log certain messages in order to follow and trace them. Notably, it is possible to read log messages in one central place, independently of the computer, operating system or programming language used by any given component.

The following figure illustrates this system architecture. Components communicate with each other via Topics in the `semaine.data` hierarchy (indicated by black arrows). Meta information is passed between each component's meta messenger and the system manager via the `semaine.meta` Topic (grey arrows). Optionally, components can write log messages, and a message logger can log the content messages being sent; a configurable log reader can receive and display a configurable subset of the log messages (dashed grey arrows).



Optionally, a system monitor GUI visualises the information collected by the system manager as a message flow graph. Input components are placed at the bottom left, output components at the bottom right, and the other components are sorted to the extent possible based on the data input/output relationships, along a half-circle from left to right. Component B comes later in the graph than component A if A's output is an input to B or if there is a sequence of components that can process A's output into B's input. This criterion is overly simplistic for complex architectures, especially with circular message flows, but is sufficient for simple quasi-linear message flow graphs. If a new component is added, the organisation of the flow graph is recomputed. This way, it is possible to visualise message flows without having to pre-specify the layout.

The following pages present three emotion-oriented example systems, in order to corroborate the claim that the SEMAINE API is easy to use for building new emotion-oriented systems out of new and/or existing components. Source code is provided in order to allow the reader to follow in detail the steps needed for using the SEMAINE API. The code is written in Java, and can be obtained from the SEMAINE sourceforge page [57]. The SEMAINE API parts of the code would look very similar in C++.

The source code for these examples is included in the SVN repository and in the source release package of SEMAINE-3.0.

9.1 Hello world

The “Hello” example realises a simple text-based interactive system. The user types arbitrary text; an analyser component spots keywords, and deduces an affective state from them; and a rendering component outputs an emoticon corresponding to this text. Despite its simplicity, the example is instructive because it displays the main elements of an emotion-oriented system.

The input component simply reads one line of text at a time, and sends it on. It has an input device (line 4) and a Sender writing TEXT data to the Topic `semaine.data.hello.text` (line 3). In

its constructor, the component registers itself as an input component (l. 7), and registers its sender (l. 8). Its `act()` method, which is automatically called every 100 ms while the system is running, checks for new input (l. 12), reads it (l. 13), and sends it to the Topic (l. 14).

```
 1 public class HelloInput extends Component {
 2
 3   private Sender textSender = new Sender("semaine.data.hello.text", "TEXT",
getName());;
 4   private BufferedReader inputReader = new BufferedReader(new
InputStreamReader(System.in));
 5
 6   public HelloInput() throws JMSException {
 7     super("HelloInput", true/*is input*/, false);
 8     senders.add(textSender);
 9   }
10
11   @Override protected void act() throws IOException, JMSException {
12     if (inputReader.ready()) {
13       String line = inputReader.readLine();
14       textSender.sendTextMessage(line, meta.getTime());
15     }
16   }
17 }
```

As a simplistic central processing component, the HelloAnalyser makes emotional judgements about the input. It registers a Receiver (l. 7) for the Topic that HelloInput writes to, and sets up (l. 3) and registers (l. 8) an XML Sender producing data of type EmotionML. Whenever a message is received, the method `react()` is called (l. 11). It receives (l. 13) and analyses (l. 14-17) the input text, and computes values for the emotion dimensions arousal and valence from the text. Finally, it creates an EmotionML document (l. 18) and sends it (l. 19).

```
 1 public class HelloAnalyser extends Component {
 2
 3   private XMLSender emotionSender = new
XMLSender("semaine.data.hello.emotion", "EmotionML", getName());
 4
 5   public HelloAnalyser() throws JMSException {
 6     super("HelloAnalyser");
 7     receivers.add(new Receiver("semaine.data.hello.text"));
 8     senders.add(emotionSender);
 9   }
10
11   @Override protected void react(SEMAINEMessage m) throws JMSException {
12     int arousalValue = 0, valenceValue = 0;
13     String input = m.getText();
14     if (input.contains("very")) arousalValue = 1;
15     else if (input.contains("a bit")) arousalValue = -1;
16     if (input.contains("happy")) valenceValue = 1;
17     else if (input.contains("sad")) valenceValue = -1;
18     Document emotionML = createEmotionML(arousalValue, valenceValue);
19     emotionSender.sendXML(emotionML, meta.getTime());
20   }
21
22   private Document createEmotionML(int arousalValue, int valenceValue) {
23     Document emotionML = XMLTool.newDocument(EmotionML.ROOT_ELEMENT,
EmotionML.namespaceURI);
24     Element emotion =
```

```
XMLTool.appendChildElement(emotionML.getDocumentElement(), EmotionML.E_EMOTION);
25     Element dimensions = XMLTool.appendChildElement(emotion,
EmotionML.E_DIMENSIONS);
26     dimensions.setAttribute(EmotionML.A_SET, "arousalValence");
27     Element arousal = XMLTool.appendChildElement(dimensions,
EmotionML.E_AROUSAL);
28     arousal.setAttribute(EmotionML.A_VALUE, String.valueOf(arousalValue));
29     Element valence = XMLTool.appendChildElement(dimensions,
EmotionML.E_VALENCE);
30     valence.setAttribute(EmotionML.A_VALUE, String.valueOf(valenceValue));
31     return emotionML;
32   }
33 }
```

As the SEMAINE API does not yet provide built-in support for standalone EmotionML documents, the component uses a generic XMLSender (l. 3) and uses the XMLTool to build up the EmotionML document (l. 23-30).

		Valence		
		-	0	+
Arousal	+	8-(	8-	|8-)
	0	:-(	:-	|:-)
	-	*-(	*-	|*-)

The output of the Hello system should be an emoticon representing an area in the arousal-valence plane as shown in the Table above. The EmoticonOutput component registers an XML Receiver (l. 5) to the Topic that the HelloAnalyser sends to. Whenever a message is received, the `react()` method is called (l. 8), which analyses the XML document in terms of EmotionML markup (l. 10-12), and extracts the arousal and valence values (l. 14-15). The emotion display is rendered as a function of these values (l. 17-19).

```
 1 public class EmoticonOutput extends Component {
 2
 3   public EmoticonOutput() throws JMSException {
 4     super("EmoticonOutput", false, true /*is output*/);
 5     receivers.add(new XMLReceiver("semaine.data.hello.emotion"));
 6   }
 7
 8   @Override protected void react(SEMAINEMessage m) throws
MessageFormatException {
 9     SEMAINEXMLMessage xm = (SEMAINEXMLMessage) m;
10     Element dimensions = (Element) xm.getDocument().getElementsByTagNameNS(
                                EmotionML.namespaceURI,
EmotionML.E_DIMENSIONS).item(0);
11     Element arousal = XMLTool.needChildElementByTagNameNS(dimensions,
EmotionML.E_AROUSAL,
                                EmotionML.namespaceURI);
12     Element valence = XMLTool.needChildElementByTagNameNS(dimensions,
EmotionML.E_VALENCE,
                                EmotionML.namespaceURI);
13
14     float a = Float.parseFloat(arousal.getAttribute(EmotionML.A_VALUE));
15     float v = Float.parseFloat(valence.getAttribute(EmotionML.A_VALUE));
16
17     String eyes  = a > 0.3 ? "8"/*active*/   : a < -0.3 ? "*"/*passive*/  :
":"/*neutral*/;
```

```
18      String mouth = v > 0.3 ? ")"/*positive*/ : v < -0.3 ? "("/*negative*/ :
"|"/*neutral*/;
19      System.out.println(eyes+"-"+mouth);
20    }
21 }
```

In order to build a system from the components, a configuration file is created. It includes the SystemManager component as well as the three newly created components. Furthermore, it requests a visible system manager GUI providing a message flow graph.

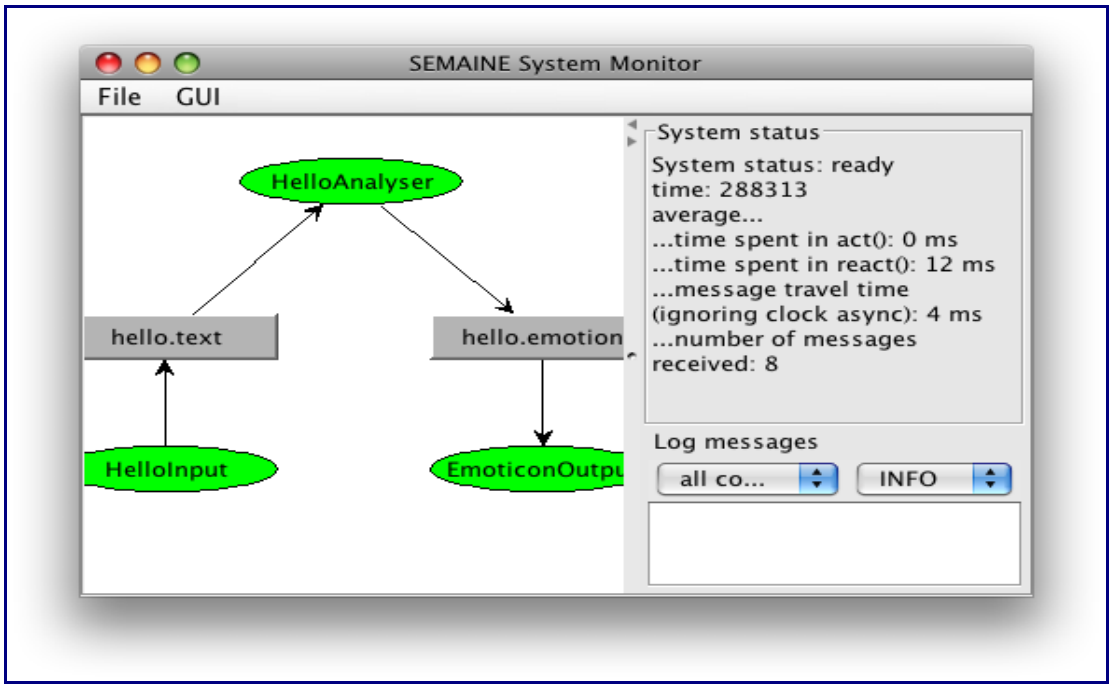
```
semaine.components = \
    |eu.semaine.components.meta.SystemManager| \
    |eu.semaine.examples.hello.HelloInput| \
    |eu.semaine.examples.hello.HelloAnalyser| \
    |eu.semaine.examples.hello.EmoticonOutput|

semaine.systemmanager.gui = true
```

The system is started in the same way as all Java-based SEMAINE API systems:

```
activemq; java eu.semaine.system.ComponentRunner example-hello.-
config
```

The following figure shows a screenshot of the resulting message flow graph. As the communication passes via the middleware ActiveMQ, the system would behave in the exact same way if the four components were started as separate processes, on different machines, or if some of them were written in C++ rather than Java.



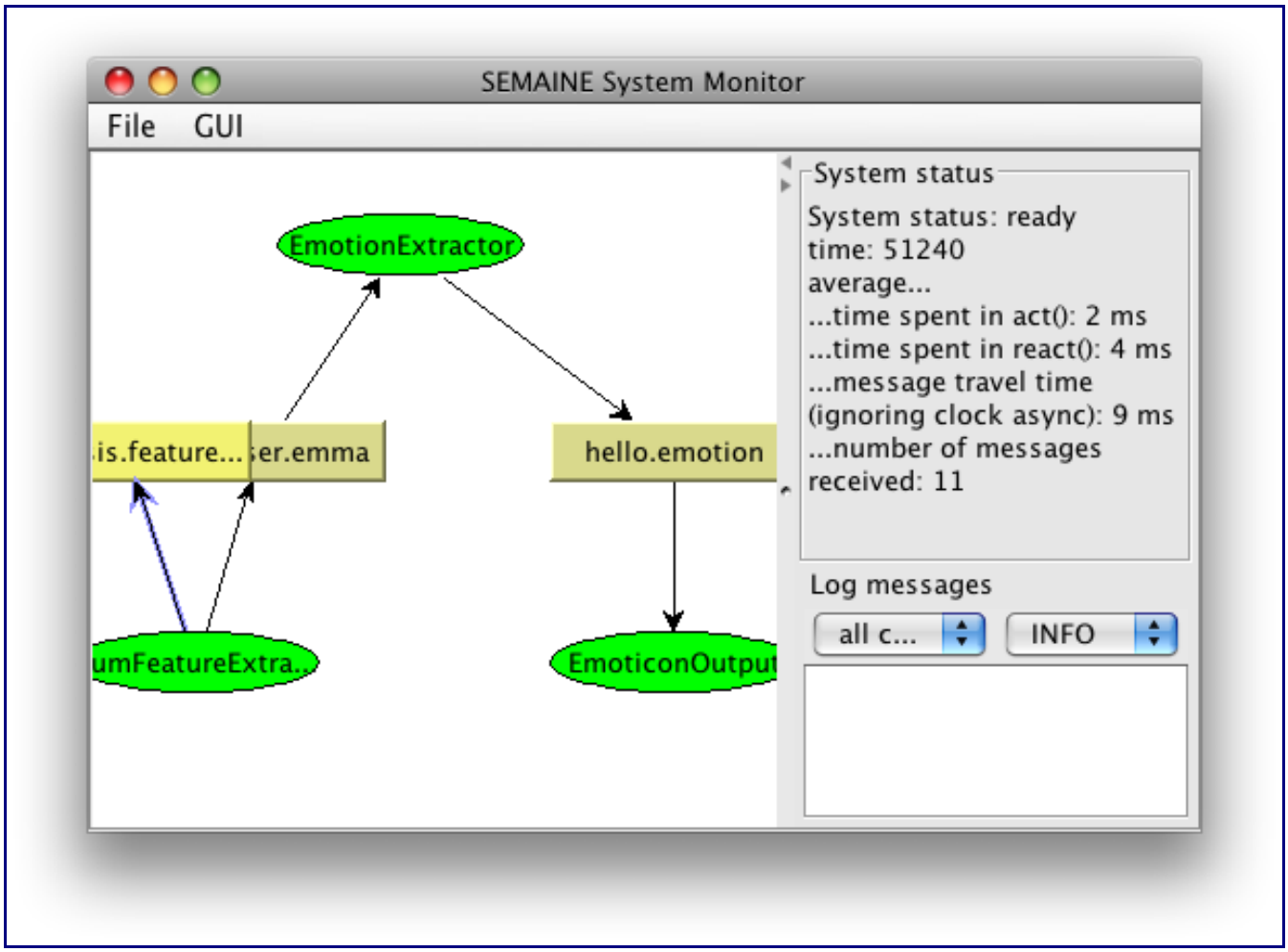
9.2 Emotion mirror

The Emotion mirror is a variant of the Hello system. Instead of analysing text and deducing emotions from keywords, it uses the openSMILE speech feature extraction and emotion detection for interpreting the user's emotion. The output is rendered using the same EmoticonOutput component from the Hello system.

Only one new component is needed to build this system. EmotionExtractor has an emotion Sender (l. 2 and l. 7) just like the HelloAnalyser had, but uses an EMMA Receiver (l. 6) to read from the topic that the Emotion detection component from the SEMAINE system publishes to (see ComponentArchitecture). Upon reception of an EMMA message, the method `react()` is called (l. 10). As the only receiver registered by the component is an EMMA receiver, the message can be directly cast into an EMMA message (l. 11) which allows for comfortable access to the document structure to extract emotion markup (l. 12-13). Where emotion markup is present, it is inserted into a standalone EmotionML document (l. 16-18) and sent to the output Topic (l. 19).

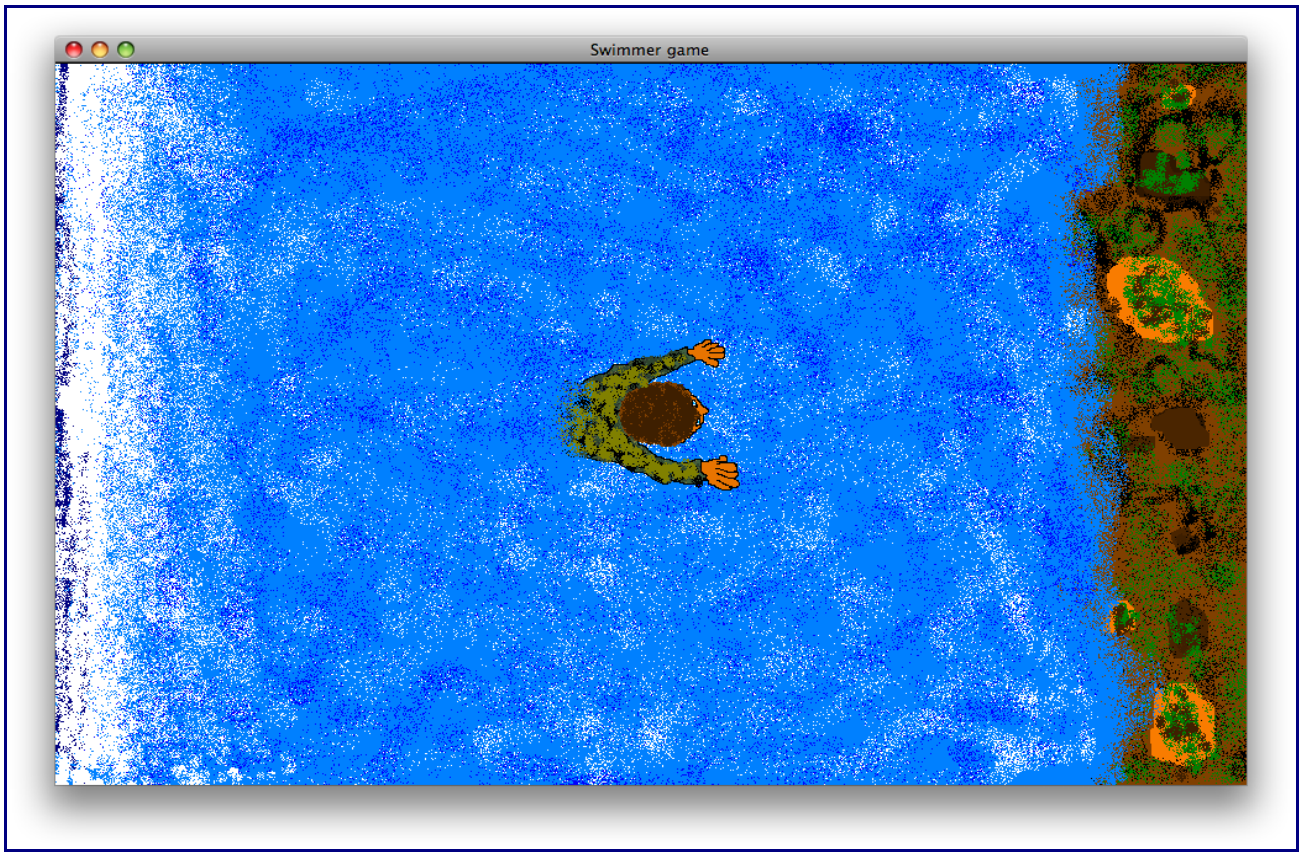
```
 1 public class EmotionExtractor extends Component {
 2   private XMLSender emotionSender = new
XMLSender("semaine.data.hello.emotion", "EmotionML", getName());
 3
 4   public EmotionExtractor() throws JMSException {
 5     super("EmotionExtractor");
 6     receivers.add(new EmmaReceiver("semaine.data.state.user.emma"));
 7     senders.add(emotionSender);
 8   }
 9
10   @Override protected void react(SEMAINEMessage m) throws JMSException {
11     SEMAINEEmmaMessage emmaMessage = (SEMAINEEmmaMessage) m;
12     Element interpretation = emmaMessage.getTopLevelInterpretation();
13     List<Element> emotionElements =
emmaMessage.getEmotionElements(interpretation);
14     if (emotionElements.size() > 0) {
15       Element emotion = emotionElements.get(0);
16       Document emotionML = XMLTool.newDocument(EmotionML.ROOT_ELEMENT,
EmotionML.namespaceURI);
17       emotionML.adoptNode(emotion);
18       emotionML.getDocumentElement().appendChild(emotion);
19       emotionSender.sendXML(emotionML, meta.getTime());
20     }
21   }
22 }
```

The config file contains only the components SystemManager, EmotionExtractor and EmoticonOutput. As the SMILE component is written in C++, it needs to be started as a separate process as documented in the SEMAINE wiki documentation SEMAINE-3.0. The resulting message flow graph is shown in the following figure.



9.3 A game driven by emotional speech: The swimmer's game

The third example system is a simple game application in which the user must use emotional speech to win the game. The game scenario is as follows. A swimmer is being pulled backwards by the stream towards a waterfall. The user can help the swimmer to move forward towards the river bank by cheering him up through high-arousal speech. Low arousal, on the other hand, discourages the swimmer and drives him more quickly to the waterfall.



The system requires the openSMILE components as in the Emotion mirror system; a component computing the swimmer's position as time passes, and considering the user's input; and a rendering component for the user interface. Furthermore, we will illustrate the use of TTS output in the SEMAINE API by implementing a commentator providing input to the speech synthesis component of the SEMAINE system.

The PositionComputer combines a `react()` and an `act()` method. Messages are received via an EMMA receiver and lead to a change in the internal parameter `position` (l. 22). The `act()` method implements the backward drift (l. 29) and sends regular position updates (l. 30) as a plain-text message.

```
 1 public class PositionComputer extends Component {
 2   private Sender positionSender =
         new Sender("semaine.data.swimmer.position", "TEXT", getName());
 3   private float position = 50;
 4
 5   public PositionComputer() throws JMSException {
 6     super("PositionComputer");
 7     receivers.add(new EmmaReceiver("semaine.data.state.user.emma"));
 8     senders.add(positionSender);
 9   }
10
11   @Override protected void react(SEMAINEMessage m) throws
MessageFormatException {
12     SEMAINEEmmaMessage emmaMessage = (SEMAINEEmmaMessage) m;
13     Element interpretation = emmaMessage.getTopLevelInterpretation();
14     List<Element> emotionElements =
emmaMessage.getEmotionElements(interpretation);
15
```

```
16     for (Element emotion : emotionElements) {
17       Element dimensions = XMLTool.getChildElementByTagNameNS(emotion,
EmotionML.E_DIMENSIONS,
                                    EmotionML.namespaceURI);
18       if (dimensions != null) {
19         Element arousal = XMLTool.needChildElementByTagNameNS(dimensions,
EmotionML.E_AROUSAL,
                                    EmotionML.namespaceURI);
20         float arousalValue =
Float.parseFloat(arousal.getAttribute(EmotionML.A_VALUE));
21         // Arousal influences the swimmer's position:
22         position += 10*arousalValue;
23       }
24     }
25   }
26
27   @Override protected void act() throws JMSException {
28     // The river slowly pulls back the swimmer:
29     position -= 0.1;
30     positionSender.sendTextMessage(String.valueOf(position), meta.getTime());
31   }
32 }
```

The SwimmerDisplay implements the user interface shown above. Its messaging part consist of a simple text-based Receiver (l. 5) and an interpretation of the text messages as single float values (l. 10).

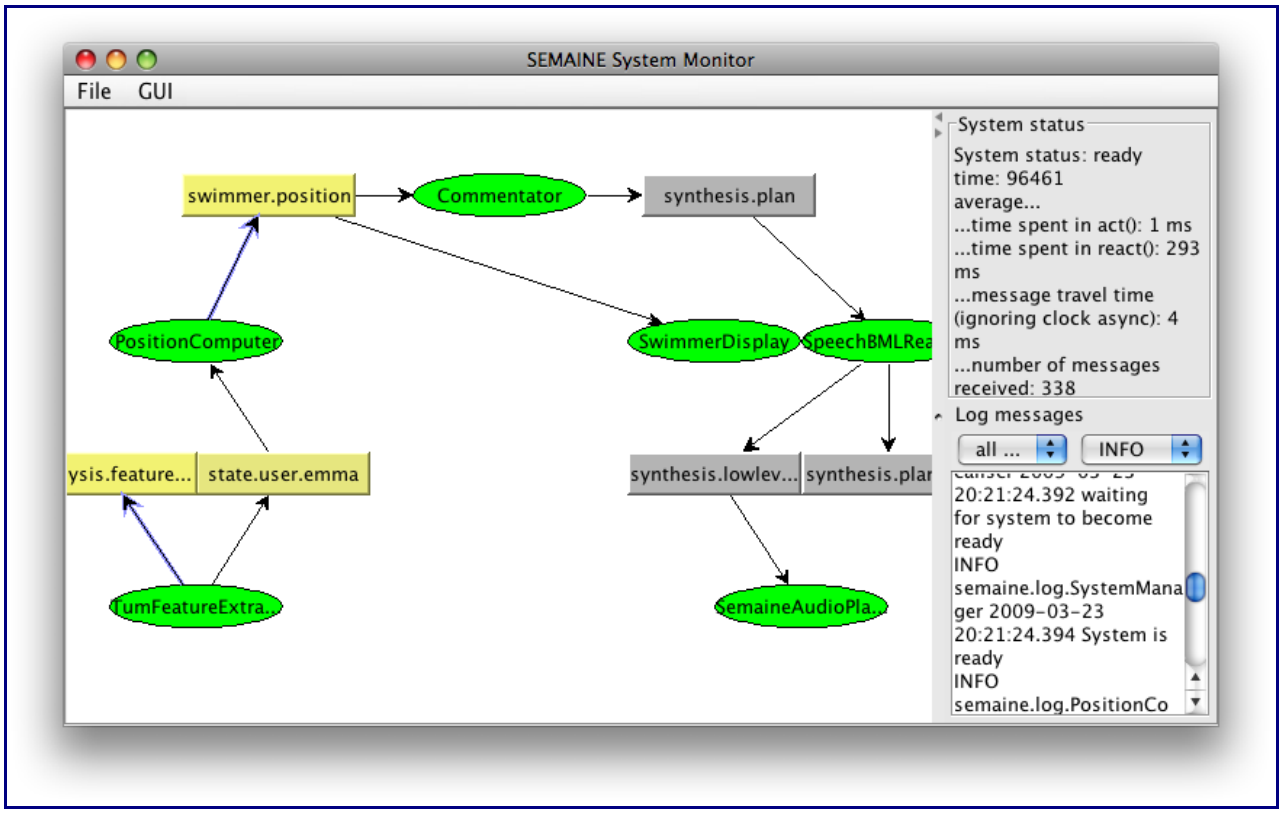
```
 1 public class SwimmerDisplay extends Component {
 2
 3   public SwimmerDisplay() throws JMSException {
 4     super("SwimmerDisplay", false, true/*is output*/);
 5     receivers.add(new Receiver("semaine.data.swimmer.position"));
 6     setupGUI();
 7   }
 8
 9   @Override protected void react(SEMAINEMessage m) throws JMSException {
10     float percent = Float.parseFloat(m.getText());
11     updateSwimmerPosition(percent);
12     String message = percent <= 0 ? "You lost!" : percent >= 100 ? "You
won!!!" : null;
13     if (message != null) {
         ...
       }
     }
     ...
   }
```

Due to the separation of position computer and swimmer display, it is now very simple to add a Commentator component that generates comments using synthetic speech, as a function of the current position of the swimmer. It subscribes to the same Topic as the SwimmerDisplay (l. 7), and sends BML output (l. 2) to the Topic serving as input to the speech synthesis component of the SEMAINE system. Speech output is produced when the game starts (l. 18-20) and when the position meets certain criteria (l. 13-14). Generation of speech output consists in the creation of a simple BML document with a `<speech>` tag enclosing the text to be spoken (l. 25-28), and sending that document (l. 29).

```
 1 public class Commentator extends Component {
```

```
 2   private BMLSender bmlSender = new BMLSender("semaine.data.synthesis.plan",
getName());
 3   private boolean started = false;
 4
 5   public Commentator() throws JMSException {
 6     super("Commentator");
 7     receivers.add(new Receiver("semaine.data.swimmer.position"));
 8     senders.add(bmlSender);
 9   }
10
11   @Override protected void react(SEMAINEMessage m) throws JMSException {
12     float percent = Float.valueOf(m.getText());
13     if (percent < 30 /*danger*/) say("Your swimmer needs help!");
14     else if (percent > 70 /*nearly there*/) say("Just a little more.");
15   }
16
17   @Override protected void act() throws JMSException {
18     if (!started) {
19       started = true;
20       say("The swimmer needs your support to reach the river bank. Cheer him
up!");
21     }
22   }
23
24   private void say(String text) throws JMSException {
25     Document bml = XMLTool.newDocument(BML.ROOT_TAGNAME, BML.namespaceURI);
26     Element speech = XMLTool.appendChildElement(bml.getDocumentElement(),
BML.E_SPEECH);
27     speech.setAttribute("language", "en-US");
28     speech.setTextContent(text);
29     bmlSender.sendXML(bml, meta.getTime());
30   }
31 }
```

The complete system consists of the Java components SystemManager, PositionComputer, SwimmerDisplay, Commentator, SpeechBMLRealiser and SemaineAudioPlayer, as well as the external C++ component openSMILE. The resulting message flow graph is shown in the following figure.


SEMAINE System Monitor
File GUI
swimmer.position
Commentator
synthesis.plan
PositionComputer
SwimmerDisplay
SpeechBMLRea
ysis.feature...
state.user.emma
synthesis.lowlev...
synthesis.plar
TumFeatureExtra..
emaineAudioPla..
System status
System status: ready
time: 96461
average...
...time spent in act(): 1 ms
...time spent in react(): 293 ms
...message travel time (ignoring clock async): 4 ms
...number of messages received: 338
Log messages
all ...
INFO
20:21:24.392 waiting for system to become ready
INFO
semaine.log.SystemManager 2009-03-23 20:21:24.394 System is ready
INFO
semaine.log.PositionCo

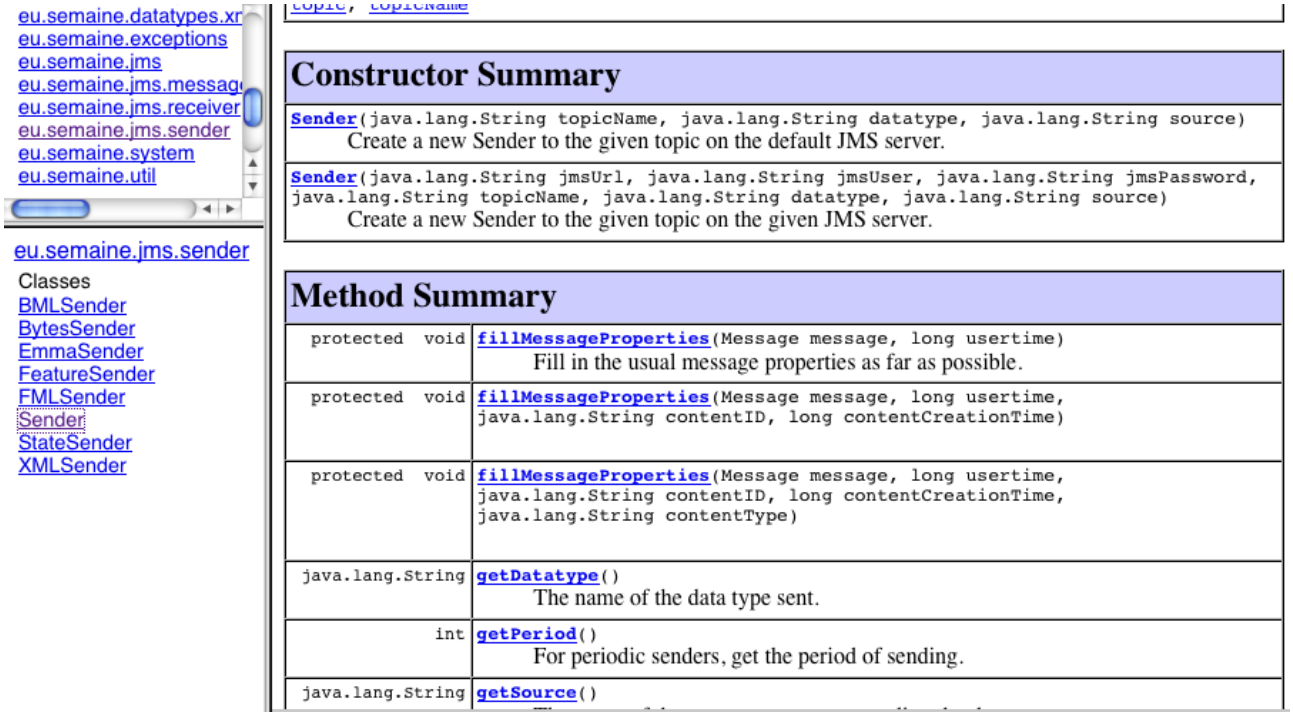
10 API documentation

A documented Application Programming Interface is the basis for software reuse. SEMAINE-3.1 comes with online documentation of both the Java and the C++ version of the API.

Both APIs are very much aligned to be as similar as possible, but of course there are some differences in practical syntax, so it is important to provide appropriate documentation for both.

10.1 Java API: Javadoc

The Java API documentation uses the de-facto standard javadoc, and can be found at http://semaine.sourceforge.net/SEMAINE-3.1/javadoc. The following screenshot illustrates the basic idea.



10.2 C++ API: Doxygen

The C++ API documentation uses the documentation tool Doxygen, and can be found at http://semaine.sourceforge.net/SEMAINE-3.1/doxygen/. The following screenshot illustrates the basic idea.