GAIA: A Software Platform for the Integration of Speech Translation Technologies

Javier Pérez and Antonio Bonafonte

Department of Signal Theory and Communication
TALP Research Center
Technical University of Catalonia (UPC), Barcelona, Spain
javier.perez@tsc.upc.es, antonio.bonafonte@upc.es

Abstract

We present here an open-source software platform for the integration of speech translation components. This tool is useful to integrate into a common framework different automatic speech recognition, spoken language translation and text-to-speech synthesis solutions. The Gaia platform is also useful in the task of obtaining the text and speech corpora needed when performing speech translation. The platform follows a modular distributed approach, with a specifically designed extensible network protocol handling the communication with the different servers. This facilitates the integration of existing solutions into the architecture. The software is provided together with completely functional audio (telephone or microphone) and text clients. Remote configuration and monitoring is possible with standard Java compliant browsers. Gaia has been successfully used in the European LC-STAR and the national ALIADO projects.

2. Architecture

The platform structure follows a modular design, with different parts running in separate machines. The speech processing tasks (recognition, translation and synthesis) are implemented in independent servers, communicating with the platform kernel via standard network sockets. Distribution is mandatory since each task can be computationally demanding. Moreover, it allows the different partners to implement their particular solutions in different platforms if desired, achieving a high grade of flexibility. Two different paradigms could be followed. If designed as a single entity, with the different client interfaces, the recognition, translation and synthesis engines, and the platform kernel would be objects or components of a single software program. In this case, several distributed computing models exist that permit components to be spread across multiple computers: CORBA (Common Object Request Broker Architecture)[8] or OOA (Open Agent Architecture) [9] among others. The later is the paradigm followed in the related CHIL [5] project, aimed at the exploration and creation of environments in which computers are used in human to human communication. On the other hand, if instead of as components each part is conceived as an entity in itself, with the ability or requirement to run independently of the platform, a client/server approach could be followed. Hence, the different parts of the platform are implemented as independent programs, using standard sockets and a predefined network protocol to communicate with the different modules of the system. This is the approach followed in the Galaxy [10] project at MIT or in the concluded national Spanish project BASURDE [11] among others. The development of the Gaia platform follows this last paradigm, since our interest is to integrate existing solutions into a common framework. Figure 1 shows the different modules of the platform and its distribution across a network environment.

2.1. Kernel

The platform logic is implemented in the Gaia kernel. It handles the communication with the platform clients, has knowledge of where the different servers (ASR, SLT and TTS) reside, and establishes the appropriate connections based on the language and configuration chosen by the user. Figure 2 shows how the kernel is built by different interfaces that communicate with the external entities, controlled by a core module. When a new user connects to the platform, the core performs the call configuration according to the preferences and/or characteristics. The protocol allows the user to select its own language and, optionally, the language of the destination user. Furthermore, depending on the client interface, audio or text, the speech recognizer and synthesizer will be used or not, respectively (fig. 3).

Once the configuration is finished, and all the necessary in-
interfaces created and connected to the appropriate servers, the call starts. In order not to overload the core unnecessarily, the interfaces of the kernel are connected directly to each other through internal memory buffers. In the complete configuration (assuming an audio user), the user interface receives the audio data from the client, and writes it to its output buffer. The ASR interface reads it, sends it to the ASR server, receives the recognized text and writes into its output buffer. The next interface, reads and sends the text data to the SLT, writing to its output buffer the received translated text. The TTS interface then sends it to the synthesizer server, and writes to the SLT, writing to its output buffer the received translated text.

The kernel implements an exception handling mechanism, in order to allow the interfaces to notify the core that a special situation has occurred. A non-exhaustive list of events include: end of turn detected by the speech recognizer, end of turn selected by the user (if a telephone client, by pressing the '*' key), network errors of any kind while communicating with the different servers.

2.2. Servers

Speech to speech translation is a combination of three different technologies: Automatic Speech Recognition (ASR), Spoken Language Translation (SLT) and Text To Speech synthesis (TTS). As seen in a previous section, Gaia follows the client/server approach to account for the distribution of tasks. We have implemented a network protocol to communicate with the different servers involved in the process. Due to space restrictions, we will only present here the generalities of the protocol, and the different issues that are accounted for. For a detailed description of the protocol and other aspects of the software implementation, the reader is referred to the technical documentation available in [12]. A common structure is shared by the different structures, incorporating commands to start, pause, resume and stop the operation of the server. An abort possibility is available to immediately stop the server in case some exceptional situation occurs (network failure, user disconnection, etc.).

2.2.1. Automatic Speech Recognition

Gaia can be configured to use several independent recognizers, and the protocol contemplates the possibility of a different grammar for each session; in its current implementation, the grammar used by the server can not be changed on a per turn basis. The connection to the server is not closed and reopened again each time there is a change of turn. Instead, it remains open for the whole session, making it possible to use different adaptive techniques in recognition. For instance, the models could be adapted to the speaker or the different background noise conditions.

Along with the recognized text, it is often interesting in recognizing speech to have more detailed information regarding the confidence of the guessed text and the position of the word inside the speech stream. A special structure is provided to obtain information about when the recognized text started and finished, and what was the cost or confidence of the recognition. The protocol incorporates several commands to set some timing parameters related to the proper ASR operation: maximum allowed initial silence, maximum allowed end silence, and a maximum utterance duration. Different situations require the speech to be sampled with a particular frequency, hence it is desirable that the protocol allows for several speech stream parameters to be changed. Among others, the sampling frequency and compression scheme (A-law, μ-law and linear PCM) can take different values at each turn.

2.2.2. Spoken Language Translation

Two different approaches to spoken language translation are contemplated by the Gaia platform. First, the platform implements the translation step in an engine (i.e. server) independent from that of the ASR. The current SLT server API has been kept quite simple, since the translation track of the LC-STAR project is under active development and no requirements other than to send and receive the original and translated text, respectively, have been set. Needless to say, the common set of commands shared by all the different platform interfaces is available (start, pause, resume and abort operation).

On the other hand, one of the implemented prototypes integrates the translation engine in the ASR. In this case, no specific SLT server is required, since the text received from the recognition server is already translated into the target language. As figure 1 shows, in case of integrating recognition and translation, an ASR server must be defined for each pair of languages. This is not necessary if using independent engines, since no knowledge about the target language is required in the recognition step.
2.2.3. Text-To-Speech Synthesis

We are currently using our own UPC synthesis engine and the Festival synthesis system. The existing implementation of the synthesizer interface only incorporates commands to configure the audio stream. Similar to the recognition interface, the compression schema (A-law, μ-law and linear PCM) and the sampling frequency can be set on a per turn basis. The common approach of using a standarized mark-up language [13] can be used to pass any other option to the recognition server. Among others, standard marks allow the recognition client to select which speaker to use for synthesizing, the rate of the generated speech and whether we want an special pronunciation mode to be used (e.g. spelling an acronym or saying a date or proper name). If required, the platform can be easily modified to include any other specific option the synthesis server may require.

2.3. Clients

Different ways of accessing the platform are contemplated, falling into two broad categories, depending on whether they use an audio or a text interface. The former has more interesting capabilities since makes use of the speech processing engines (ASR and TTS), while the later has more theoretical interest, since its main use is to debug the translation modules. Some examples of possible configurations will be presented in section 3. Both telephone (via a Dialogic card connected to a computer) or a combination of microphone and speakers are accounted for in the audio interface. We will distinguish here between the user, the person using the platform, and the client, the program used by the user to access it.

Configuration of the call scenario involves setting an appropriate origin-destination language pair, connecting to and configuring the servers that will be used during the call (see section 2.2), and start the required audio or text client interfaces. The platform expects each client to send a numerical code, each digit being the value of an option. The origin client transparently indicates its interface, but user interaction is required in order to set the current language and, optionally, the destination language, and the type of destination to be contacted (telephone or machine connected to the network). Currently, both the telephone and text clients provided with Gaia ask the user to enter this code directly, using the phone keypad or the computer keyboard. This requires the users to know about the numerical values used by the platform, but on the other hand, the time elapsed in the scenario configuration is very small. Pre-setting the destination language by the origin user allows for a specific pair of languages to be tested. Otherwise, the configuration stage of the end client will include a section to configure the language.

Users communicate by turns controlled by the Gaia kernel, one of them sending the audio or text data to the platform (active user), and the other one receiving the corresponding processed result from the platform (passive user). Only one of the users can be active at a time. Currently, both the clients and the ASR may indicate the kernel that a change of turn must occur. Conveniently choosing the time parameters of the speech recognizer, the turns may be defined either by setting a maximum duration, or by using a certain amount of silence to detect the final pause. The clients may also be allowed to signal the change of turn by reacting to certain keys (e.g. the current telephone client signals an end of turn event each time the talking user presses the asterisk key of the keypad).

3. Usage scenarios

Gaia has been developed with flexibility as one of the main goals. Hence, multiple configurations of the platform can be selected at runtime, depending on the desired scenario. The complete configuration would be with two different clients, and an interface for the ASR, the SLT and the TTS for each direction of the call flow. But multiple variations are allowed, where some of the interfaces may or may not be present. Imagine a scenario where two users want to use the platform to communicate to each other in a different language using the telephone. This requires the complete version of the platform to be active, since both clients have an audio interface, thus requiring the recognition and synthesis stages to be performed, and the text translation engine is necessary due to the different languages involved. For each client, the speech recognizer is used to obtain the transcription of what the user said, the translation would be obtained using a standalone text translation server, and then this text would be sent to the speech synthesizer in order to obtain the audio data for the destination user.

A slightly different scenario would occur if the translation engine and the speech recognizer were integrated. In this case, the output text obtained from the recognizer could be directly sent to the synthesizer module, since it would already be translated, and there would be no need to use a different SLT. A direct connection between the ASR interface and the TTS interface would be set by the kernel, and no SLT interface would be created.

It is also possible for each user to have a different interface to access the platform (e.g. one using a telephone client, and the other a text client). It is clear that in one direction the ASR module would not be necessary, whereas in the other the synthesis part can obviously be skipped. The client interface in the kernel assigned to the text client would send and receive data directly from the translation interfaces, while the interface connected to the audio client would send data to the recognizer, and receive its input from the TTS interface.

So far, we have assumed that the platform was used by more than one user, but there are several scenarios contemplated where it functions in single-user mode. If we were to test different solutions for the SLT module, instead of having two users sending feedback to each other externally to the platform, it would be more useful for the user performing the tests to receive the feedback directly from the platform. In this case, the user would be active and passive simultaneously, since once the turn is over, it would receive the response of the platform to its input (translated text in case of text interface, synthesized audio otherwise), and then immediately the turn would be on again.

Gaia is also an appropriate tool to record controlled dialogues. In this case, one of the clients would be the user whose prompts are to be recorded, and the other client would be a program reading recorded prompts (i.e. audio files) and sending them to the platform. This offers many interesting possibilities. For instance, we could have a recording scenario where the user repeats exactly what is received from the platform. Another useful option would be to implement a question/answering system, where the questions and answers are recorded to create a standard mark-up language [13] can be used to pass any other option to the recognition server. Among others, standard marks allow the recognition client to select which speaker to use for synthesizing, the rate of the generated speech and whether we want an special pronunciation mode to be used (e.g. spelling an acronym or saying a date or proper name). If required, the platform can be easily modified to include any other specific option the synthesis server may require.

Configuration of the call scenario involves setting an appropriate origin-destination language pair, connecting to and configuring the servers that will be used during the call (see section 2.2), and start the required audio or text client interfaces. The platform expects each client to send a numerical code, each digit being the value of an option. The origin client transparently indicates its interface, but user interaction is required in order to set the current language and, optionally, the destination language, and the type of destination to be contacted (telephone or machine connected to the network). Currently, both the telephone and text clients provided with Gaia ask the user to enter this code directly, using the phone keypad or the computer keyboard. This requires the users to know about the numerical values used by the platform, but on the other hand, the time elapsed in the scenario configuration is very small. Pre-setting the destination language by the origin user allows for a specific pair of languages to be tested. Otherwise, the configuration stage of the end client will include a section to configure the language.

Users communicate by turns controlled by the Gaia kernel, one of them sending the audio or text data to the platform (active user), and the other one receiving the corresponding processed result from the platform (passive user). Only one of the users can be active at a time. Currently, both the clients and the ASR may indicate the kernel that a change of turn must occur. Conveniently choosing the time parameters of the speech recognizer, the turns may be defined either by setting a maximum duration, or by using a certain amount of silence to detect the final pause. The clients may also be allowed to signal the change of turn by reacting to certain keys (e.g. the current telephone client signals an end of turn event each time the talking user presses the asterisk key of the keypad).

3. Usage scenarios

Gaia has been developed with flexibility as one of the main goals. Hence, multiple configurations of the platform can be selected at runtime, depending on the desired scenario. The complete configuration would be with two different clients, and an interface for the ASR, the SLT and the TTS for each direction of the call flow. But multiple variations are allowed, where some of the interfaces may or may not be present. Imagine a scenario where two users want to use the platform to communicate to each other in a different language using the telephone. This requires the complete version of the platform to be active, since both clients have an audio interface, thus requiring the recognition and synthesis stages to be performed, and the text translation engine is necessary due to the different languages involved. For each client, the speech recognizer is used to obtain the transcription of what the user said, the translation would be obtained using a standalone text translation server, and then this text would be sent to the speech synthesizer in order to obtain the audio data for the destination user.

A slightly different scenario would occur if the translation engine and the speech recognizer were integrated. In this case, the output text obtained from the recognizer could be directly sent to the synthesizer module, since it would already be translated, and there would be no need to use a different SLT. A direct connection between the ASR interface and the TTS interface would be set by the kernel, and no SLT interface would be created.

It is also possible for each user to have a different interface to access the platform (e.g. one using a telephone client, and the other a text client). It is clear that in one direction the ASR module would not be necessary, whereas in the other the synthesis part can obviously be skipped. The client interface in the kernel assigned to the text client would send and receive data directly from the translation interfaces, while the interface connected to the audio client would send data to the recognizer, and receive its input from the TTS interface.

So far, we have assumed that the platform was used by more than one user, but there are several scenarios contemplated where it functions in single-user mode. If we were to test different solutions for the SLT module, instead of having two users sending feedback to each other externally to the platform, it would be more useful for the user performing the tests to receive the feedback directly from the platform. In this case, the user would be active and passive simultaneously, since once the turn is over, it would receive the response of the platform to its input (translated text in case of text interface, synthesized audio otherwise), and then immediately the turn would be on again.

Gaia is also an appropriate tool to record controlled dialogues. In this case, one of the clients would be the user whose prompts are to be recorded, and the other client would be a program reading recorded prompts (i.e. audio files) and sending them to the platform. This offers many interesting possibilities. For instance, we could have a recording scenario where the user repeats exactly what is received from the platform. Another useful option would be to implement a question/answering system, where the questions are stored on disk, and the client answers recorded by the platform. This general scenario is related to the functionality of the Ada [14] tool developed at our university. The Gaia platform has already been successfully used during the development of different stages in the LC-STAR project. An earlier version was used to obtain the speech corpora belonging to the travel domain needed for the project. In this case, both clients used a telephone interface, and no recognition, translation or synthesis was required, only the recorded audio files.

There are some occasions where no output to the user is needed. In this case, the result of the user connection would be the different files generated by Gaia during the call. The platform always saves the output of the different modules involved in the call (recognized text, translated text, original speech and synthesized speech).
4. Remote monitoring and configuration

A powerful aspect of Gaia is that no physical access to the machine where the kernel is running is necessary to change or incorporate new configurations. A text file specifying the different parameters is used for each configuration. The client can select a particular one using a numerical code. Each digit gives a different value to the following fields: user language (English, Catalan or Spanish in the current implementation), destination type (whether we are connected to a telephone number or a machine connected to the network), destination user language (either set it to a certain one, or allow the destination user to select it) and configuration number (directly related to the name of the configuration file we want to use).

The file provides the platform with the IP addresses and ports where the involved servers are running. In case the of ASR and SLT, a different address for each pair of languages (e.g. English–Spanish or Catalan–English) may be defined. On the other hand, this obviously does not apply to the TTS servers, since only an address for each language is needed. Nevertheless, all addresses must be explicitly included, although they need not be necessarily different (e.g. the address of the SLT performing English to Spanish translation may be the same as the one in charge of the English to Catalan pair, but they both need to be written in the configuration file). An automatic method to generate the files is provided via a web interface. An standard HTML form can be used to enter the data in an easy and clean way, since parsing errors are drastically reduced.

There is no need of physical access to the platform in order to easily visualize the results of the different modules: speech recognition, text translation and speech synthesizer. A visualization applet is included, implemented in Java and hence being accessible from any standard Internet browser. This applet presents a graphical description of the system, the output of the different interfaces and highlights the active module (recognition, translation or synthesis).

5. Conclusions

We have presented in this paper a software environment that can be useful when doing research in spoken language technologies. The principal role of the Gaia platform is to connect different solutions for each of the technologies involved (automatic speech recognition, spoken language translation and text-to-speech synthesis). Considering the high computational requirements of each individual technology and the flexibility required when multiple partners collaborate providing a particular server Gaia (with the possibility of each of them working in a different platform), it is mandatory to minimize the processor load of the platform. Furthermore, there is need of distributed computing, since we do not desire to overload a single processor or machine. After careful review of several related projects, a client/server approach has been adopted and an extensible network protocol specifically designed. This facilitates the integration of existing independent servers into the Gaia framework. The existing UPC servers for ASR, SLT and TTS have been adapted and integrated into the platform structure. We have also implemented an intermediate layer in order to use the Festival Speech Synthesis System [15] with the Gaia platform.

Several demonstrations of speech translation have been performed, using either the integrated recognition and translation engine, or the separated SLT solution. The statistical translation engine developed at RTWH [16] has been slightly modified to handle the communication using the network protocol defined in Gaia, and integrated into the platform structure. As mentioned earlier in this paper, the Spanish project ALIADO [1] has also adopted it as the demonstration platform. The speech corpus needed for the LC-STAR [7] project has been obtained using this platform.

In order to guarantee portability among platforms, standard C++ has been used to program the platform. The remote monitoring and configuration applets are programmed in Java and can be used from several Java-compatible browsers (e.g. Mozilla, Galeon or Opera). The software is provided together with full sources and technical documentation, with special focus on the network protocol and the different interfaces needed by the platform. Developers will find it easy to adapt any related server to Gaia, requiring minimum changes to the code in only few cases.

6. Acknowledgements

This work has been partially sponsored by the European Union under grant IST-2001-32216 (LC-STAR [7] project) and the Spanish Government under grant TIC2002-04447-C02 (ALIADO project [1]).

7. References


