SignSpeak Project

Scientific understanding and vision-based technological development for continuous sign language recognition and translation

Report explaining the framework and GUI design

Minor Deliverable D.6.1

Release version: V 2.0

Grant Agreement Number 231424

Small or medium-scale focused research Project (STREP)

FP7-ICT-2007-3. Cognitive Systems, Interaction, Robotics

Project start date: 1 April 2009

Project duration: 36 months

Dissemination Level		
PU	Public (can be made available outside of SignSpeak Consortium without restrictions)	X
RE	Restricted to SignSpeak Programme participants and a specified group outside of SignSpeak consortium	
IN	SignSpeak Limited (only available to a specified subset of SignSpeak programme participants)	
LI		
Distribution list (only for RE or LI documents)		

0 General Information

0.1 Document

Title	Report explaining the framework and GUI design
Type	Minor Deliverable
Ref	D.6.1
Target version	V2.0
Current issue	V2.0
Status	Draft
File	D_6_1_v2.odt
Author(s)	Pablo Alonso-Villaverde Roza / CRIC
Reviewer(s)	Gregorio Martínez / CRIC
Approver(s)	Gregorio Martínez / CRIC
Approval date	
Release date	22/09/11

0.2 History

Date	Version	Comment
03/01/01	V1.0	Released first description of D.6.1 report
22/09/11	V2.0	Released second description of D6.1 report

0.3 Document scope and structure

WP6 addresses the integration of the development made in WP3, WP4 and WP5 and a graphical user interface aimed at end users (non-technical users), who will use this interface as a tool to evaluate the translation process.

This document covers the work done and all considerations that have been taken into account for the development of these tasks.

Authors	Group
Pablo Alonso-Villaverde Roza	CRIC

Table of Contents

1 Overview

This document deals with the integration framework design of different work packages developed for SignSpeak. We will also consider the development of a user interface to evaluate the translations of SignSpeak by end users.

2 Requirements

WP6 has two main requirements:

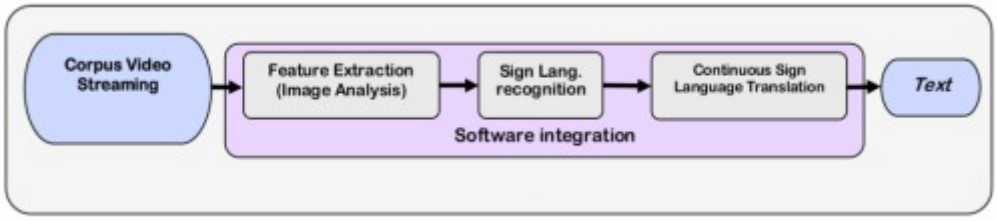
- Create an integration framework for technical users (researchers, developers) who create software modules for SignSpeak.
- Create a user interface that will allow end users to evaluate the translation process carried out by SignSpeak.

2.1 Integration framework

The integration work is of critical importance. Integration means to put each part in order to create a full system that works correctly. Heterogeneous executables with multiple parameters and different input/output formats developed by different RTDs can complicate the transmission of information among the parts and generate malfunctions.

The desired features for the integration framework are:

1. **Build a information chain**. The main aim of the integration phase is to ensure that information flows smoothly throughout a network of connected deliverables that belong to different work packages.



2. **No limitations**. The framework should not limit what can be done in each deliverable. The developer must have total freedom to perform actions as deemed necessary.
3. **Independence of input source**. It is important to make sure that the system will work with any type of input, so all the corpus must be interchangeable.
4. **Extensibility**. Extensibility is needed to deal with unexpected changes. It is also important to note that although the final prototypes are scheduled, the time to integrate this work is short and software modifications are usual.

5. **Monitoring in mind**. One of the purposes of the framework is to easily detect erroneous behaviors in different parts of the project. So an important task of the framework is the design of “monitors”. These monitors are graphic displays to check the output of different modules of the project. Breakpoints will allow developers to see the results of their calculations in the monitors. For example, a monitor can combine tracking information over video to check if the tracking system follows the hands and head of the signer accurately.

2.2 Graphic user interface for evaluation purposes

The end users that will evaluate the performance of the translations made by SignSpeak need a very simple system, without inner technical details. All they need is to check the accuracy of the translations.

The desired features are:

1. **Easy interface and simple operation**. We must avoid interfaces that require the users to learn new skills to use. Operations to check a translation must be as simple as possible.

2. **Avoid technical details**. A end user doesn't need to know details related to the techniques involved in the translation process. In most cases, that information will only make the evaluation tool seem more complicated.

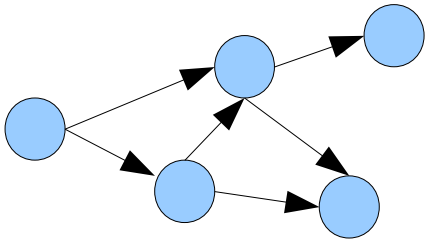
3 Design

3.1 Design of the integration framework

3.1.1 Overall design

The design of the framework is based on the data-flow model. In this approach, a graph or network structure is used to represent a full system made up by a combination of programs or processes.

In a graph structure there are nodes and connections.



Drawing 1: A graph

Nodes (blue circles in the previous drawing) represent programs. Connections (black arrows) represent dependencies between two programs and data flow, so if two nodes (programs) are connected, it means that the program at the end of the arrow needs data generated by the program at the beginning of the connection.

We have chosen the data-flow model for several reasons:

- The graph is a valid representation of a full system, showing the different components and their dependencies.
- It is flexible, allowing us to add or remove nodes (programs) as needed to recreate modifications on the full system being modeled; additionally we can use this model to make representations of isolated parts of the system.
- The framework is not modified by modifications on the modeled systems.

The framework will contain node definitions for each program that makes up a Work Package, and these nodes will be used as “bricks” to build a network that will represent the full translation system or a subsystem.

3.1.2 Application design

The framework application will be a desktop application that will allow the modeling of a full or partial SignSpeak system, following the concepts of data-flow explained in the previous section. It will allow settings to be modified for each deliverable and finally run the modeled system.

3.1.2.1 Graphic user interface

Most operations of the application will be used from a main window divided in three zones:

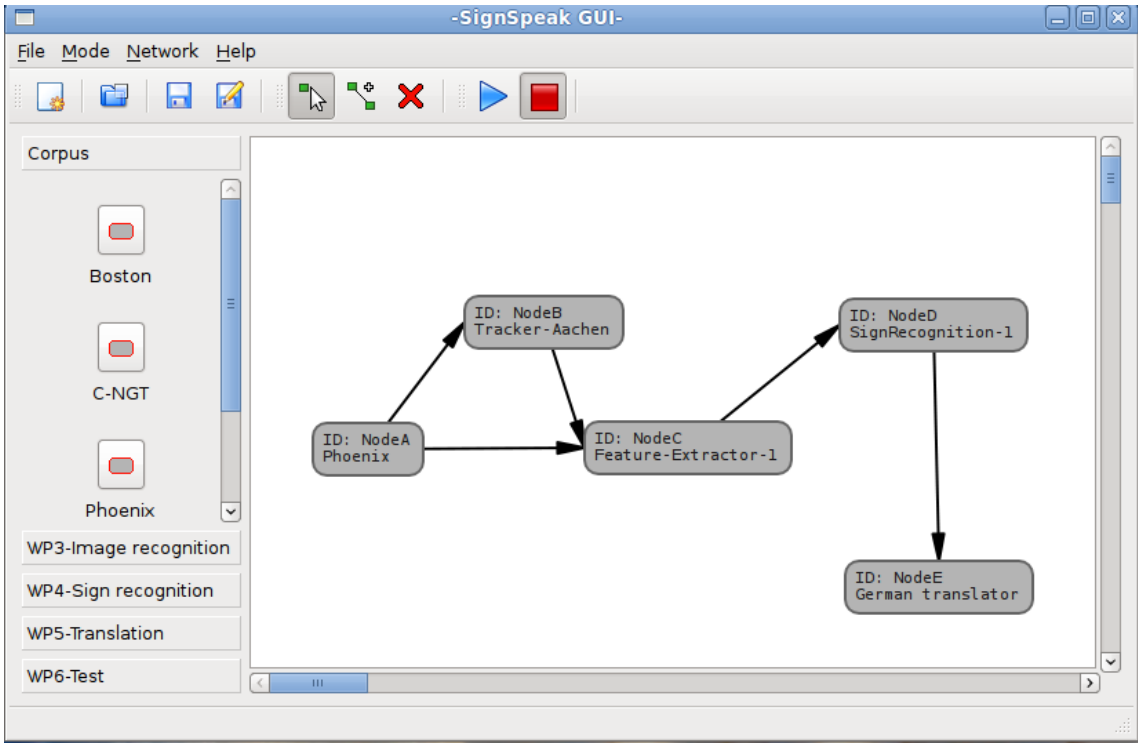
- A palette of modules on the left side to model the graph
- A drawing zone to model the graph of modules visually
- Menus and command buttons to carry out common operations, drawing functions and execute the graph in the upper zone, above the drawing zone and palette

The palette will contain modules represented by buttons or icons. These modules will be grouped according to functionality. The user can select a module with a mouse click and add it to the drawing zone with a second mouse click.

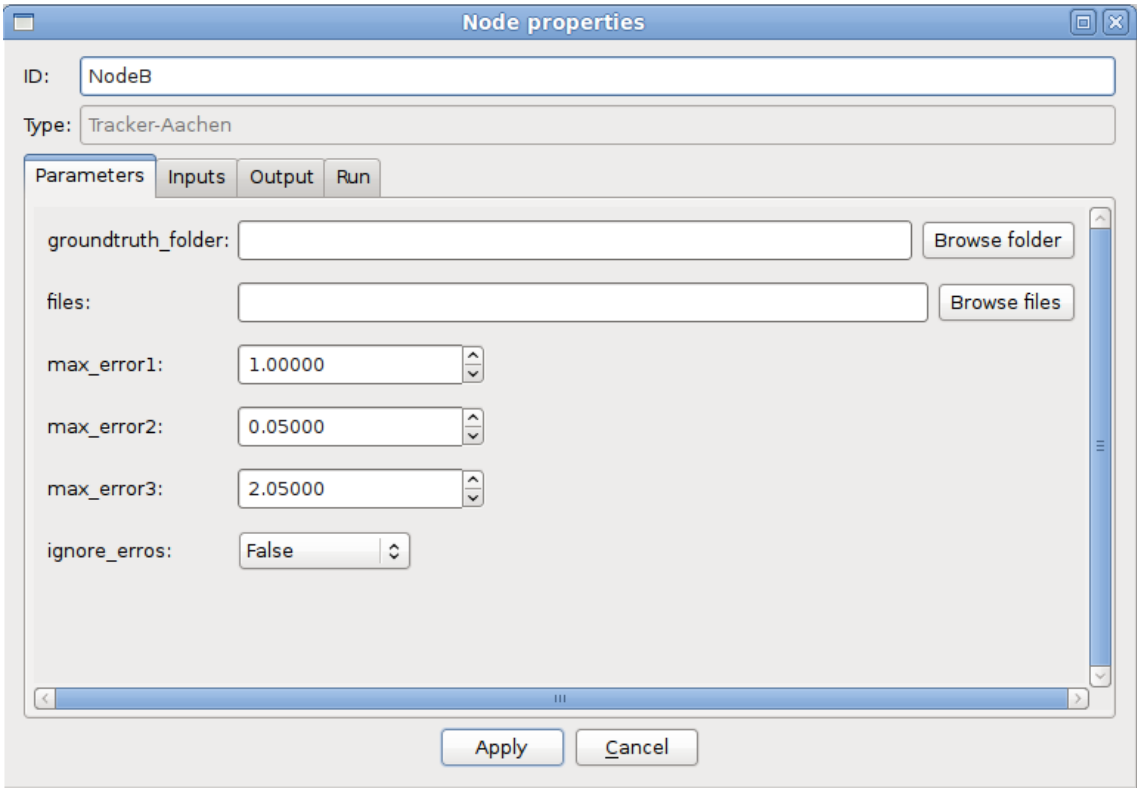
In the drawing zone, the user can model the graph structure using the modules of the palette as components.

Additional secondary windows are used for different purposes:

- Validate user operations
- Edit properties of the nodes in the graph, etc.



Drawing 2: Main window



Drawing 3: Node properties

3.1.2.2 Graph model

As mentioned in previous sections, the framework will allow technical users to create a representation of the system by means of a graph or network model. Using a graph model provides some important benefits:

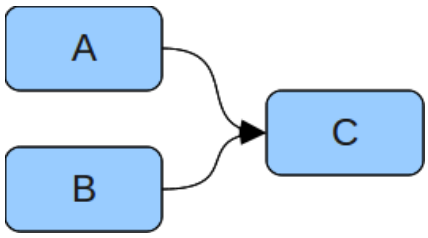
3.1.2.2.1 A fast overview of the system

This model will give us information about the modules involved in the system (represented by the nodes of the graph) and their dependencies (connections of the graph) in an easy visual way.

3.1.2.2.2 Easy orderings of modules

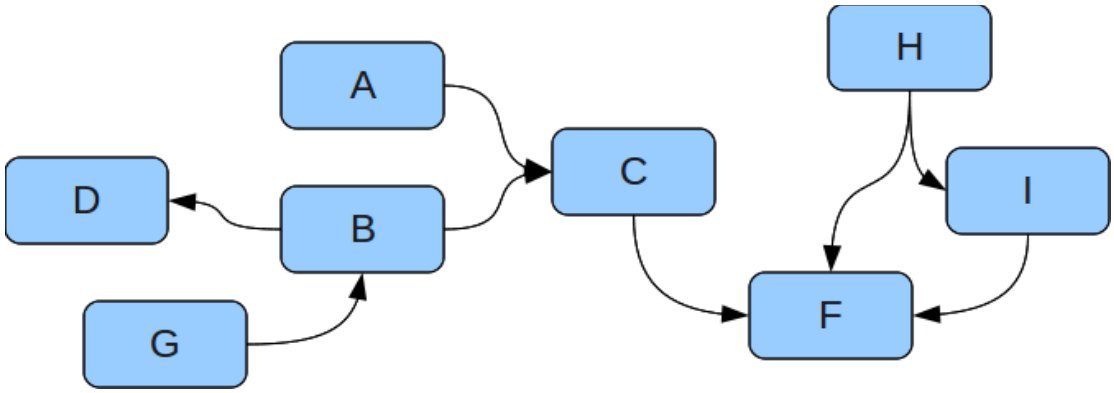
Using a graph allows us to deduce a linear ordering of the connected nodes. This is specially useful to generate an execution ordering of the modules included in the graph using the information added by connections.

In simple graphs, is easy to detect correct execution orderings, for instance in the following graph, it is clear that module A and module B must be executed before module C.



Drawing 4: Simple graph

When graphs are more complicated it is difficult to guess a correct execution order, but there are graph-specific algorithms that will give us correct orderings.

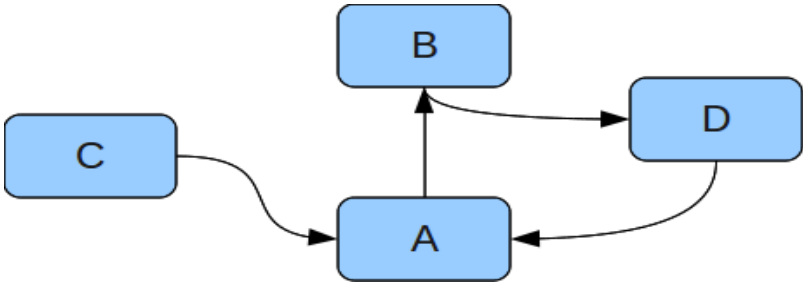


Drawing 5: Complex graph

The algorithm used to deduce the orderings is known as “Topological ordering traversal”.

3.1.2.2.3 Check for design mistakes

This algorithm will also detect some common mistakes like cyclic dependencies. Taking a look at the next graph we can see a cyclic dependency to be avoided, because three programs (A, B and D) are interdependent, and that is not possible because one cannot have an execution order with interdependent programs.



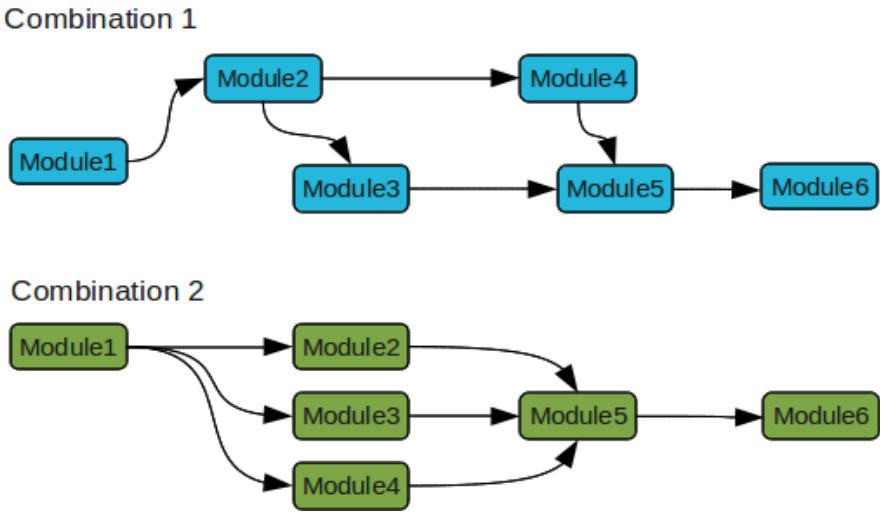
Drawing 6: Graph with a cycle

3.1.2.2.4 Control over the execution process

Given a correct linear ordering of modules, we can execute the modules one after another to carry out a full execution of the system and we can use that list to control the execution and set breakpoints whenever necessary, to analyze results of a specific module before executing the next module on the execution list.

3.1.2.2.5 Flexible design and faster modifications

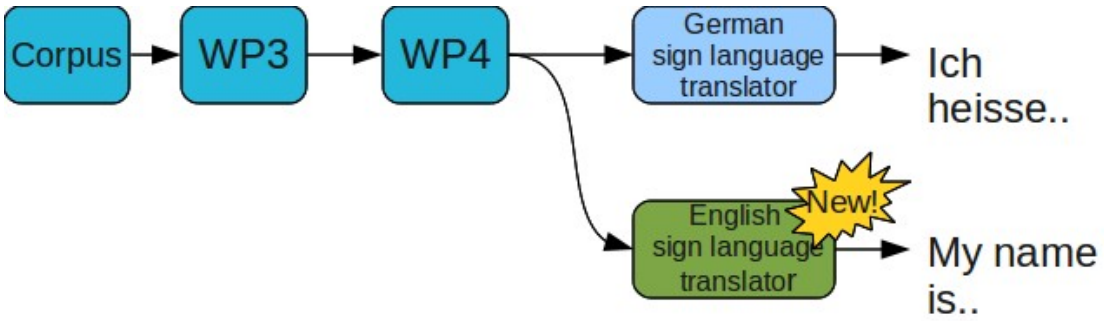
It will allow us to try and experiment different combinations of modules. For example we could try to compare two combinations of modules to check which one generates better results just reordering and reconnecting the nodes.



Drawing 7: Different combinations of modules

3.1.2.2.6 Easy to expand

It is easy to add new modules as they are created by developers. To add a new software module we must only create a new node definition so the application knows about the existence of the new module. That process only needs to be done once by the user.



Drawing 8: Adding a new module to a graph

New modules are not limited to modules directly related to the translation process. It is possible to add any kind of necessary software, for example programs to combine information, data format converters, etc.

3.1.2.2.7 Graph model implementation

A minimal implementation of a graph model for our needs must include:

- A graph data structure with data types to represent generic nodes, connections and adjacency information among nodes
- Graphic classes to draw the graphs on the application window
- An implementation of a topological ordering traversal for graphs, in order to:
  - Get execution orderings of the modules
  - Check for cyclic dependencies among modules
- Load and save functions for the graphs
- The graph nodes must be as generic as possible, i.e. without knowledge of the details relative to modules or corpora

3.1.2.3 Wrapping modules and corpora

The modules and corpora, represented in a graph model by nodes, are probably the most variable objects in the framework. Corpora receive updated annotations, software modules are released, updated or need different parameters, etc.

The graph should be isolated from these changes, so wrapping the modules and corpora with a data type that manages all the internal details is necessary.

Wrapping is a methodology to encapsulate a foreign object and make it pluggable into another system more easily.

3.1.2.3.1 Modules

Among the modules we must wrap two types of software modules:

- Executables: any conventional program written in any programming language. Executables can be called directly, by means of scripts, or as subprocess of other executables.
- Matlab programs: There are some deliverables developed using Matlab. These programs must be executed by means of the Matlab development tools.

In order to call different modules, the framework will use an intermediate shell script that will call the desired module by means of a valid command line using the corresponding parameters. The parameters needed to call a module can be set from the framework, editing the properties of the node that represents that module in the graph model.

3.1.2.3.2 Corpora

Corpora is a set of data. Each corpus can be defined by different parameters like folders that store videos, images, groundtruth information, etc, so they can be considered as software modules without executables, although with a variable number of parameters.

3.1.2.3.3 Module Parameters

Most software packages accept parameters, and the SignSpeak deliverables are no exception. In the framework application we can set values to the parameters of each module. If we edit the properties of any node of a graph, we will be able to set these values and use them later when the module is executed.

The framework support different types of parameters adapted to different data. There are parameters specialized in numeric values, lists of files, folders, text strings, boolean values, etc.

When a module is defined in the framework we also define its parameters such as:

- **Common parameters**: These parameters are used internally by the module to modify its internal settings, activate options, etc.

- **Output parameters**: These parameters can be used by the module as a common parameter, but they can also be sent to any node connected to the current module. A typical case is using an output parameter to indicate where the results of the program are to be stored after its execution. This parameter can be sent as input of another module so that module knows where to look for those results.

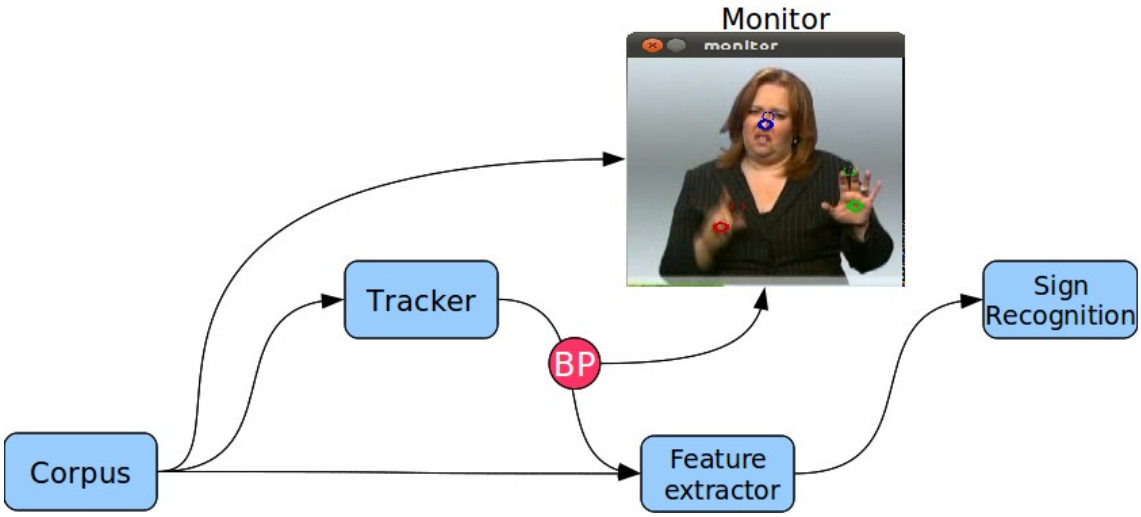
- **Input parameters**: These parameters can be used as common parameters too, but they can receive values from output parameters of other modules.

In Drawing 3: Node properties, we can see how we can set values to parameters when editing the properties of a node that represents a module.

3.1.2.4 Monitors and breakpoints

One of the purposes of the framework is to easily detect erroneous behaviors in different parts of the project. So an important work in the framework applications is the design of these monitors such as graphic displays to check how the hand/head tracker is working.

It is also important to allow the user to add node breakpoints to see the values in the monitors. Breakpoints (BPs) can be set by the user at points of the graph where s/he wants to check the results of previous operations.



Drawing 9: A monitor showing tracking information

Monitors can be part of the graph, represented as a node like any other software module. They can be programmed independently of the framework, so it is possible to develop new monitors when needed without modifying the framework.

3.1.2.5 Standard Outputs

One problem of the integration work is the dependence of data formats between different interconnected modules. When a module generates a result, it must be defined in a format that can be used by other modules that need this information. This data interchange in made at file level so standard output formats for results are necessary.

To solve this problem, a good option is to define some fixed data structures where the results of the computations made by the modules can be accommodated.

Of course, an alternative solution would be create new modules specialized in data conversions, but if the rest of modules keep generating results without following certain conventions, new data converters must be created. This alternative is not a clean solution.

Using standard formats will favor:

- Independence between modules
- Modules that are easily interchangeable
- The development of monitors because they do not need to support different data formats

3.2 Graphic user interface for evaluation purposes

Perhaps the most familiar interface for people to use nowadays are web pages.

- They don't require the user to learn new skills. Most people can use them.
- Allows simple usage. Most operations are performed with single mouse clicks.

For these reasons, the chosen system to create the graphic interface for evaluation purposes will be a web page.

The functions to be carried out by this page are:

- Select a corpus
- Select a video in the corpus by means of different selection rules (name of signer, a gloss in the video)
- Once the video is selected, choose between watching the translation or watch the translation with gloss information.

The process is quite linear, and easy to repeat.

As a result of these ideas, we have created a web proposal in collaboration with EUD (European Union of the Deaf) to be used as an evaluation tool. Contents of the pages are provisional and subject to change and/or revisions if necessary. These are only provisional designs.

3.2.1 Feedback system

The users of these web pages will be able to give feedback after watching any video. Once the video playing has finished, a new feedback form will appear on the screen requesting the user for any information he would like to submit about the translation or the web system. This information will be useful to correct mistakes and improve the accuracy and performance of the system.

3.2.2 Home page

The home page will contain:

- A links bar near the top of the window to access to the home page, description of the Phoenix corpus, translation operations and help
- A short description of the project and a video signing the same description
- A link to the official website (www.signspeak.eu)
- A set of buttons to change the language of the web page

3.2.3 Corpus description

This page is accessed by means of the link labeled “Phoenix(DGS)”. It will contain:

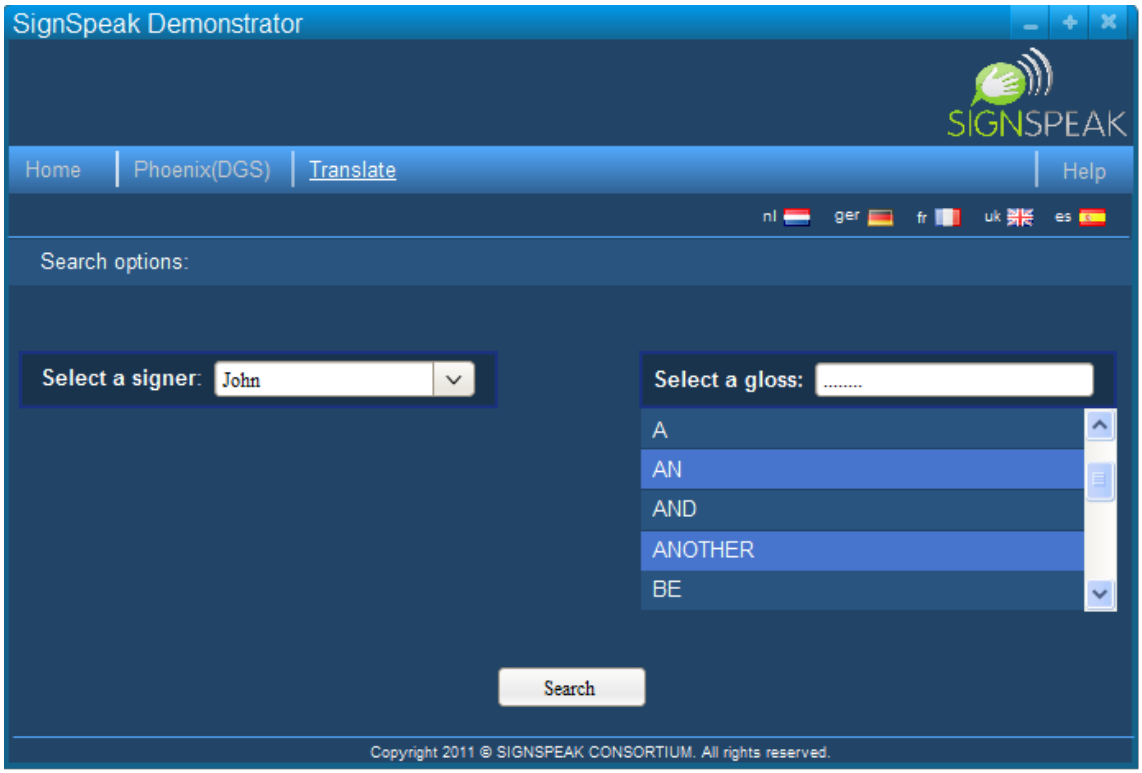
- A links bar near the top of the window to access to the home page, description of the Phoenix corpus, translation operations and help
- A text describing the corpus and a video signing the same text
- A link to the official website (www.signspeak.eu)
- A set of buttons to change the language of the web page

3.2.4 Search criteria

This is the first page shown when the “Translate” link is clicked.

It will allow the user to define search rules based on the name of the signer and a gloss.

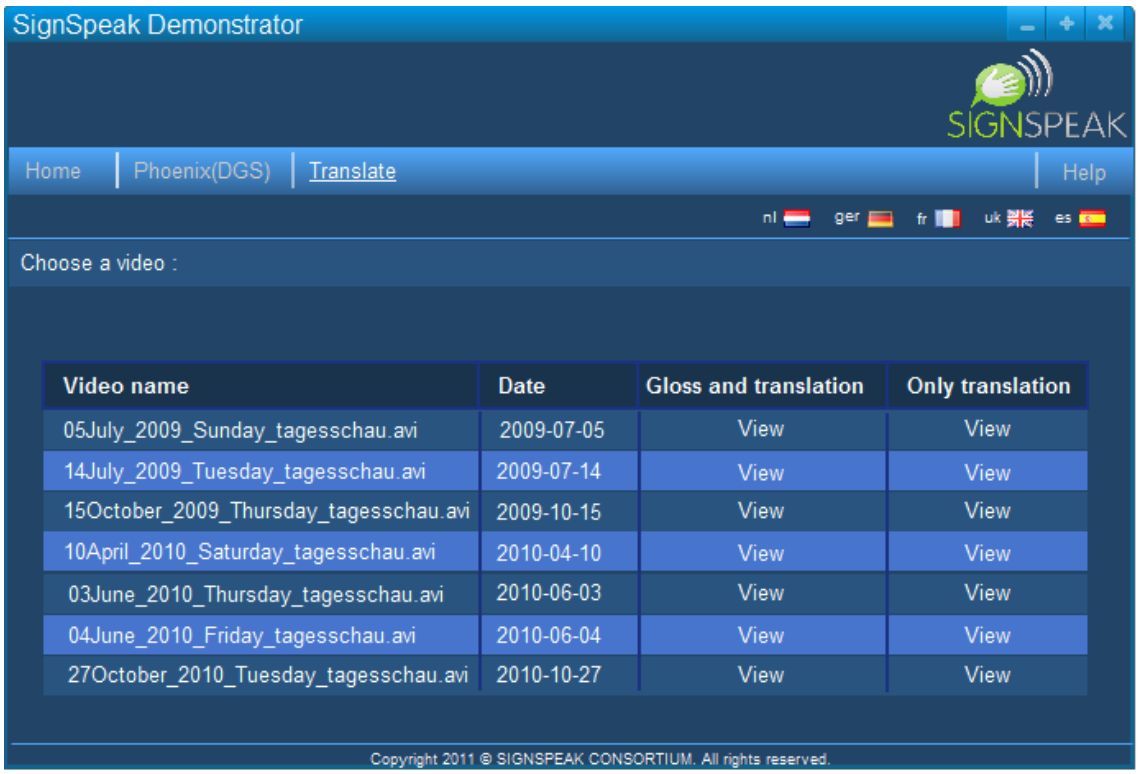
If the “search” button at the bottom of the page is clicked, a new page will be appear on the screen showing a list of videos that follow our search rules.



3.2.5 Select a video

This page shows a list of the videos that follow our search criteria.

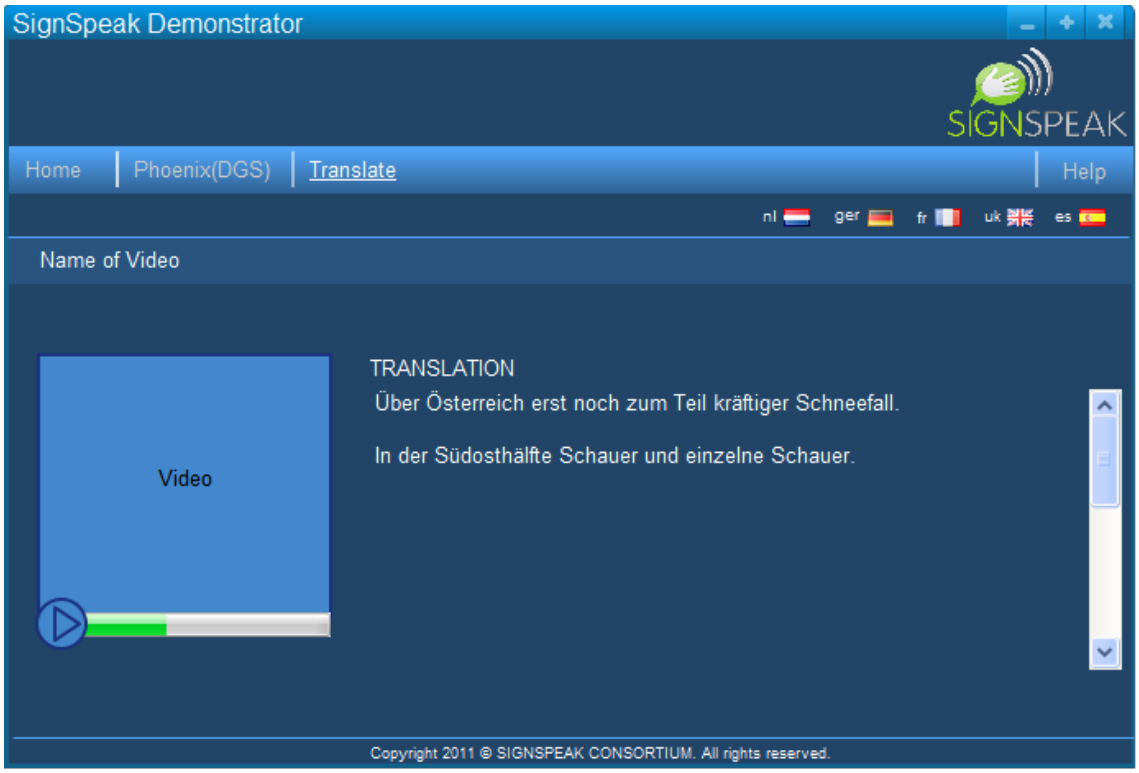
The list will contain the name and data of the video, and next to each video there are two links labeled “View”. One will take us to a page to watch the translation of the video, the other one will take us to a page to watch the translation and glosses of the video.



3.2.6 Watch translation

In this page the user will be able to read the translation of a video and watch the video while it's played next to the translation.

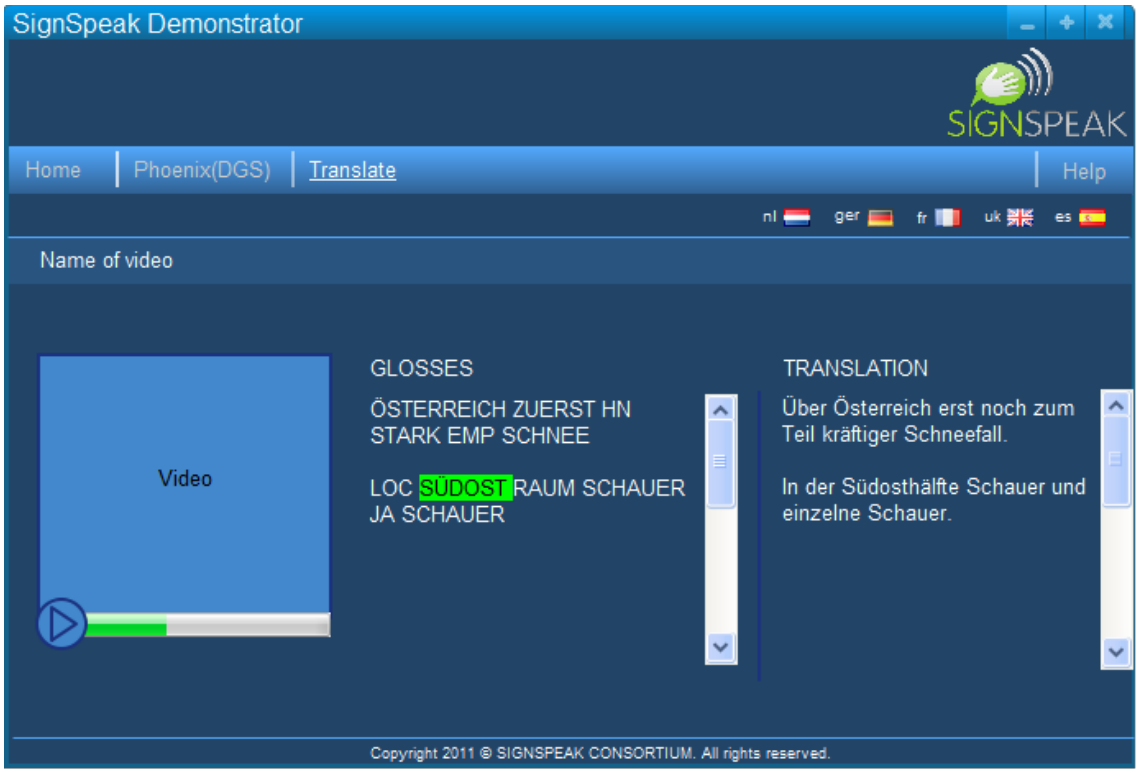
The translation will be previously generated as a result of processing the video with SignSpeak.



3.2.7 Watch translation and gloss information

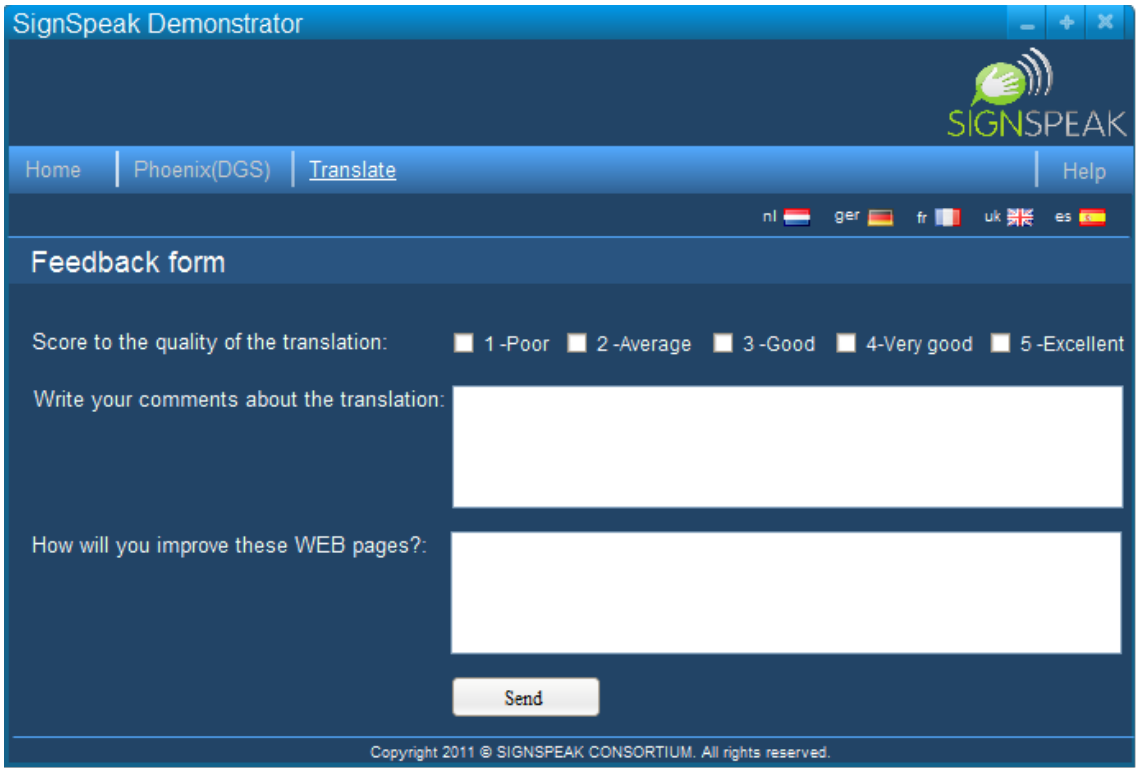
In this page the user will be able to see both the translation and the glosses of the video while the video is played.

The translation will be previously generated as a result of processing the video with SignSpeak.The glosses will be highlighted as the signer in the video signs them.



3.2.8 Feedback form

This is a very simple form page to collect information from the users of the web pages. They can submit information about the translations or the use of the web pages.



4 Conclusions

After the study of the requirements and the analysis of the most important components we have defined a formal framework that covers the main needs of the project.

The design of the framework is widely extensible thanks to the use of data-flow paradigm, so once the framework is developed we will be able to integrate all the modules easily. In addition, the framework decouples the different parts using standard outputs and making the framework more changeable.

Finally, we have presented a user interface for evaluation purposes which is easy to use for non-technical users.