An avatar–based interface for the Italian Sign Language

Vincenzo Lombardo, Cristina Battaglino, Rossana Damiano
Dipartimento di Informatica,
Università degli Studi di Torino
Torino, Italy
{vincenzo,rossana,battagl}@di.unito.it

Fabrizio Nunnari
VRMMP
Torino, Italy
fnunnari@vrmpp.it

Abstract—In this paper, we describe a virtual interpreter of the Italian sign language (Italian Sign Language, LIS), developed as part of the on-going ATLAS project, on the automatic translation from Italian to Italian Sign Language. The translation system communicates with the user through a virtual signer: the system takes as input a formal representation of a sign language sentence and produces the corresponding animation of the avatar. The architecture of the virtual signer consists of a resource planner, an executor of the planned sign animations, and an animation system.

Keywords—H.5.1 Multimedia Information Systems, H.5.2 User Interfaces, H.5.2.i Interaction styles

I. INTRODUCTION

In this paper, we describe a system that generates the real-time animations of the virtual interpreter of the Italian Sign Language (LIS, Lingua Italiana dei Segni), as part of the ATLAS (Automatic Translation into Sign LAnguageS) project for the Italian-to-LIS translation. The scenarios in which these virtual signers can be employed concern different communicative situations and supports, including web pages, mobile devices and television broadcasting [1], [2], [3], [4].

A LIS sentence consists of an ordered sequence of signs. In the LIS, like all other sign languages, signs often require the adaptation to the context of the sentence. Such phenomena include the context-dependent positioning of the gesture in the “signing space”, the increase or reduction of the “size” of a sign, or its repetition (e.g., for plural). More complex phenomena involve the movement of hands through the space from and to context-dependent positions, such as for the verbs “to go” and “to give” [5], and the use of hand-shape to indicate specific types of entities [6]. Finally, extra-linguistic context (such as pointing to physical referents) may influence the performance of signs.

Computer animation is a natural candidate for the development of virtual signers. Specifically, procedural character animation is a mandatory approach when a generic virtual agent has to react to unpredictable user input, [7], which is the case of a real-time translation from a natural spoken language to a natural signed language. Simpler systems generate concatenations of individual signs that are retrieved from a repository. For example, the BlueSign project [8] developed a system for Signed Italian (IS, Italiano Segnato), that is based on a dictionary of animated signs. More sophisticated “synthetic” approaches define how default signs (again archived in a repository) are tuned to the specific context of a sentence. This approach was pursued by [9] and [10], that are open to the real-time adaptation of a sign to the context of the sentence.

In the field of embodied conversational agents, ad-hoc languages for animation are intended to bridge the gap between the description of communicative acts and their realization through an animation engine. From the pioneering PAR language [11], a template-based representation of actions designed to program animated agents, we got to the design of markup languages for animation (starting from [12]). One of the most documented and solidly implemented is the Behavior Markup Language (BML), geared to describe the multimodal communicative behavior of an agent [13], [14]. However, BML lacks the expressive power to describe sign languages, because the gesture description is not enough detailed to account for novel gestures and not enough powerful to implement context adaptation.

The ATLAS project (see the flow in Figure 1) aims at the production of a fully-featured LIS language. The software system that generates the animation of the virtual interpreter takes as input a symbolic representation of the LIS sentence from a translation module (see [15], this workshop). The Virtual Interpreter animator module consists of an AI planner, for the allocation of the resources (hands, torso, ...) available to the signer, an executor of the motion commands generated in the translation phase, and an animation system, that composes the animated signs and the facial expressions (stored in a dictionary, the “signary”). The signary is a repository of motion-captured or hand-animated signs, which are composed and parameterized through the expressions of an animation language. The sign composition relies on blending techniques that, widely used in videogame architectures, concatenate and layer animation clips through interpolation functions in real time.

This paper is organized as follows. We describe the architecture of the Virtual Interpreter (Sect. II), then the

1ATLAS is a three–year project (2009 2012) funded by Regione Piemonte of Italy within the Programme Converging Technologies.
The interpretation process of the ATLAS project. It consists of two steps. First (left), the Italian sentence is translated into an intermediate, symbolic, LIS representation, named AEWLIS. Then (right), the Virtual Interpreter animates the AEWLIS sign sequence.

Animation System and the implementation (Sect. III, IV), and provide a detailed example (Sect. V). Discussion and Conclusions end the paper.

II. THE ATLAS VIRTUAL INTERPRETER

The animation module of the ATLAS project takes as input a symbolic form of the sign language (provided by the linguistic modules) and generates the real-time animation of the virtual interpreter. The linguistic input to the animation system consists of an AEWLIS structure [16], [15], i.e., a sequence of lemmata, accompanied by a description of the meaning of each lemma, its syntactic number and the link to the corresponding sign. The semantic and pragmatic information includes the thematic roles played by the lemma (such as agent/patient, initial/final location, etc.) in the situation described by the sentence.

The signing space in front of the interpreter is structured with locations where the items mentioned in the sentence are positioned and kept for further references in the sentence. Signs are described by a default animation (stored in a repository, the “signary”), that has to be parameterized with respect to the context given by the specific sentence. Each sign may encompass both manual and non-manual components. Each sign is defined by a set of features, such as a default position (in most cases, the center of the signing space) and the relocation possibility of the sign depending on the context (e.g., the sign “mom”, with the fist touching the chin, cannot be relocated); it also exposes a set of parameters that are potentially set (e.g., speed of the sign, width of the gesture, iteration).

The Animator architecture consists of three main components (Figure 1): the Planner, the Executor and the Animation engine. From a conceptual point of view, the knowledge about actions encoded in the Planner represents the decision making capabilities of the system, necessary to generate the interpreter’s linguistic behavior; the Executor can be equated to the “actor” who plays the interpreter, since it knows how to enact the actions contained in the plan; finally, the Animation engine abstracts from these notions to implement the actual manipulation of the animated objects that realize the agent embodiment (the perceivable “body” of the interpreter) together with the environment in which it is situated.

Each component of the architecture is related to a specific knowledge base (the cylinders in Fig. 1). For the planner, the knowledge base consists of the library of action operators it employs to devise linguistic plans. Given a plan, the Executor retrieves the sign definitions from the catalogue of signs, and applies the corresponding mapping rule to translate it into commands for the Animation engine (animation mapping rules). A repository stores the animation data for the retrieval and the reuse in the generation of signs.

Given the sequence of signs, the Animator plans the use of the signing resources, namely hands, facial expression, torso, head, ..., and the organization of the signing space in order to accomplish the animation of the given sign sequence.

The input to the planner consists of a goal (the sign sequence to be animated), an initial world state consisting of the initial resource allocation (e.g., are the hands of the interpreter free, what is their current position, ...) and the virtual interpreter settings (e.g., what is the dominant hand). The planner relies on the formalism of Hierarchical Task Networks (HTN) [20]: when invoked, it matches the tasks to be achieved onto a high-level method and starts refining it into simpler tasks, discarding alternatives that do not fit the given world state. The refinement ends when all high-
level tasks have been expanded into one or more sequences of elementary, not further decomposable tasks. The obtained sequence of operators is the plan that describes the agent’s behavior.

The planning library encodes the communicative knowledge and schedules the expressive resources of the virtual interpreter. For example, it contains specific operators for posting the referents of locative verbs onto the appropriate locations. Basically, the high-level task decomposes into the task of assigning the signs to the interpreter’s hands (not shown) and finding the location for each sign (Localize in the figure) - if different from the default position and provided that the sign can be relocated (Find-position), then performing the sign (Make-sign). Some relations (such as locative relations) may require that a special sign is generated if the sign that represent one of the entities involved in it cannot be relocated (not shown). Also, the planner must solve the initial and final location of the signs that have parameterized trajectory, such as movement verbs (Sign-relation).

When the Executor receives a plan from the Planner, it consults the sign repository (the “signary”) to replace each action operator with the corresponding sign. Then, for each sign, it applies the matching animation mapping rule to obtain the animation language expression that specifies how the animation of the sign is realized by using the animation data in the repository or by procedural functions.

The animation language contains a set of primitive animation functions, such as procedural motion paths, blending and layering. These primitives can be embedded into control structures that support the parallel and sequential use of the animation primitives, and the combination of both [17]. So, interpreting a mapping rule involves running the procedural animation routines that execute the animation functions contained in the definition. These routines, run by the Animation Engine, create the objects that, together with the existing data, generate the final animation.

III. THE ANIMATION SYSTEM

The Animation System is implemented as part of the Enthusiasm project (see below), where animations are associated to AnimatedObjects (e.g., in the ATLAS project, the virtual signer Donna). The Animation System is based on the concept of AnimationTrack. An AnimationTrack is characterized by an elapsed time, a state and a blend operation. Each AnimatedObject contains a reference to an Animation Track. At each update cycle, the AnimatedObject prepares an instance of a Pose object, reset at the “T-stance” position. A Pose is a container for all the degrees of freedom (DOFs) available for animation. During the update, the pose is passed to the Animation Tracks, that transforms it into the new Pose to be applied to the AnimatedObject.

An Animation Track is characterized by a “weight”, initially set to 0 by default. During its lifetime an AnimationTrack starts in a BLEND_IN state, during which the weight ramps up at 1. The weight stays at 1 during the BLEND_NONE state and ramps down again during the BLEND_OUT state. After reaching 0, the track jumps to the DEAD state. The way an Animation Track transforms an input Pose might depend on its current weight, in a way specified by concrete subclasses. Finally, an AnimationTrack declares on which DOFs it operates: they can be skeletal bones, blend shapes or a mixture of them.

The ClipTrack is a concrete implementation of an AnimationTrack, which realizes an animation by “playing back” a stored animation file. This is a so-called "data-driven" approach to animation. The ParallelTrack is the animation mixing system, and supports both sequential blending and layering of multiple animations. It is a container of several AnimationTracks, that can be dynamically removed and added to the system. It is characterized by a priority system, so that each added track is associated to one priority (see Figure 2).

The ParallelTrack processes animations in priority order, from the lowest to the highest. Within a priority all the tracks are updated, one-by-one (in no predictable order). At each update, the Animation Track returns a Pose that is mixed with the current pose according to the track weight and the mix operation specified for the priority (ADD, MUL, OVR). The ADD operation is meant to add the modification of the DOFs of the current animation to the current pose. The MUL operation works similarly to an image layering system, where the weight acts as an alpha channel: with weight at 0, the track has no effect, with weight at 1 the track overrides the current pose. The in/decrease of the track weight, varying along time, realizes smooth transitions between tracks. The OVR operation corresponds to a MUL operation with a weight fixed at 1, though computationally optimized. Finally, when a track reaches the DEAD state, it is automatically removed from the layer. Within a single priority, the behavior of the ParallelTrack with a MUL operation behaves for the skeletal animation exactly like the Ogre engine, while with the ADD operation behaves like Ogre for blend shape animations. More ParallelTracks can
be nested to build a hierarchical set of mixing tracks.

The Animation system core API is common to most digital media players, loading media objects to be played. It can be in one of three states: Stopped, Paused, Playing. The animation system is setup by imposing a SpeedModulated-Track to the AnimatedObject of the virtual signer Donna. This track wraps a ParallelTrack acting as a Mixer for the ATLAS Sign language needs. The speed modulated track is used to control the overall speed of signing, through a speed multiplication factor. The latter points to a parallel track (as mixer for all animations, including a default idle animation, the basic sign and its modifiers. Some modifiers are realized through the use of layered AnimationTracks, each one associated to a different priority. Modifiers realized through the use of layering are: Facial expressions, Head nods, Gaze. Other modifiers are realized by transforming (on the fly) the signs and producing new, temporary Animations from which a new ClipAnimation is created. Modifiers realized through transformation are: Speed modification and Sign Relocation.

The “sign” channel beats the time of the playback. For each sign in the sentence, the player performs the sign, applying the needed modifiers. Given a sign, its animation is either retrieved from the Signary (e.g., the sign “temperature”) or built on-the-fly through dedicated procedures (e.g.: the sign “to give”). It is then transformed according to the AEWLIS directive (e.g., relocated somewhere). The resulting Animation is used to build the ClipTrack for the “Sign” channel. The track is blended with the current interpreter pose. For each track that pivots a modifier, the current weight fades to the appropriate value; if the modifier value is unspecified, the track is blended out (weight goes to zero).

When a sign has been performed, the next one is “open blended”, that is the blending of the two signs has no restrictions on the time duration (can last as much as necessary) and the preparatory part of a sign starts after the complete release of the previous sign. A sign is performed when its corresponding track in the Sign channel is “starving”; which means that is PAUSED on the BLEND_NONE state.

When playing a single AEWLIS sentence, the sign sequence is played at “Natural Speed”, i.e. at the speed signs were authored. The transition between consecutive signs is performed with a fixed blending time of 500ms, whose value has been determined through a few observations of human signers.

The realization of an animated sequence of signs requires a sophisticated authoring, that takes into account the capacities of the real-time procedures of animation and the amount of work for producing the pre-recorded signs, archived in the Signary. The latter are to be blended in sequences or layered in parallel. We identify the actual signs through the analysis of a LIS dictionary, also taking into account a few sentences in which the signs are contextualized. Then, we analyze what is the composition of the signs, in order to implement the corresponding animations. In some cases it is possible to identify sub parts of the signs, so some animations can be re-used in several cases. This analysis leads to the design of animation mapping rules, that are part of the animation language.

Then we proceed in producing the animations, through automatic and manual techniques. The automatic techniques that we use in the ATLAS project are the procedural motion paths over hands and arms, the blending of animations in succession on a skeleton, the layering of blending shapes for the facial expressions and over some skeleton portions. In the cases where the automatic techniques cannot provide a solution, we produce the complete sign animations manually, and only sequential blending is used for putting the signs in the sentential context.

The actual production of animations (see two examples in Figure 3) is entirely manually (by expert animators). Motion Capture techniques with human interpreters [18], [19], though very effective in time and cost in the first experiments, revealed a number of imperfections in the grabbing with electro-magnetic gloves and was also onerous in terms of cleaning the spurious animation data as well as exporting for the real-time engine.

IV. SYSTEM IMPLEMENTATION

In the current implementation of the ATLAS system, we use the JSHOP2 planning system [20]. Declaring the knowledge in the JSHOP system consists of decomposing complex behaviors, or methods, into simpler and simpler ones, until the level of basic behaviors, or operators, is reached. The operators provide the visible acts executed by the animated agent. The set of all actions, methods and operators, constitutes the plan library.

The Executor is a C++ library, which means that it has to be loaded and initialized by some startup code. At
startup, the Executor initializes the Tarta4D system (see below), opening a new window or using a window already instantiated by the startup code (e.g., on the set-top box, a window floating in front of the video playback). An existing scene is loaded, which sets up a scene with an instance of the character Donna, a background panel, and some lights. Also, all the needed animations (stored as .animations files) are loaded. To test the functionalities of the player a basic command-line based interface is provided. To facilitate an integration with other test applications, the application can be launched in “network” mode, in which case a listening socket is opened, so that the commands can be received through UDP packets.

The low-level animation engine is part of the Enthusiasm project, an open source platform that supports the authoring of 3D real-time interactive virtual environments. The Enthusiasm system has a GUI for scene authoring that supports the editing of the animation timeline and the management of lights and cameras. Enthusiam relies on the Tarta4D rendering library, a multi-platform 3D engine that offers high-level functionalities for building applications based on real-time 3D technologies. As for graphics, Tarta4D supports the import of 3D objects authored with the most popular 3D authoring tools, real-time 3D rendering (on top of DirectX or OpenGL), 3D objects animation and automated animation blending.

V. Example

For an example related to the application domain of the project, consider the sentence: “Cloudy at north-east. During the evening cloudiness increases at north-west”. This sentence is characterized by some peculiarities. First, the north-east location is bound to a fixed location in the signing space because the interpreter is before a whether map. Second, there is a semantic location involving the clouds and the north–east location, so the sign representing the cloud must be performed in the same location as the north–east location. Finally, the increase of the cloudiness represents a process, so the corresponding sign is given by the shifting (the proper increase) of the sign represented the cloud, repeated to obtain the plural form (clouds). This shift is targeted to the north-west location, whose position is also constrained to the map. Notice that this sentence is part of a corpus of LIS sentences collected by the ATLAS project, so the original sequence of sign has been employed to verify the interpreter’s performance.

The planning task, represented in Figure 4, is generated from three different sources of knowledge: the AEW LIS representation of the sentence, the information about signs extracted from the signary and the interpreter’s configuration (dominant hand), plus some information about the context.

Given the task definition above, the plan for signing the sentence is generated by refining a high–level task that consists of signing the entire sign sequence. The planning process results in the following sign sequence:

north-east;
zone (relocated top-left);
cloud (relocated top-left);
instead;
evening;
cloud (repeating and shifting from top-left to top-right);
more (relocated top-right);
zone (relocated top-right)

The animation language allows us to define parametric macro functions to cope with the specific cases of this example. `relocatedAnimation` executes an animation clip in a different position than the default one, while `repeatingShiftingAnimation` shifts the interpreter’s hands while they execute a certain gesture.

So the system produces the following sequence:

```plaintext
simpleAnimation('north-east')
relocatedAnimation('zone', north-east-location)
relocatedAnimation('cloud', north-east-location)
simpleAnimation('instead')
simpleAnimation('evening')
repeatingShiftingAnimation('cloud', 3, north-east-location, north-west-location)
relocatedAnimation('more', north-west-location)
relocatedAnimation('zone', north-west-location)
```

Figure 4. The world state description provided as input to the planner. Comments are marked by the semicolon.

3http://enthusiasm.sourceforge.net/
VI. Conclusions

We have presented the Animator component of the Virtual Interpreter of the Italian Sign Language, implemented in the context of the ATLAS project. We have described the three components of the system, namely the Planner, the Executor and the Animation system. The project is on–going, and lacks a thorough testing and evaluation yet.

For what concerns the animation language, we made some comparisons with the well–known BML. We identify two meaningful differences between our animation language and the BML realizer EMBR [14]. First, it requires the specification of many temporal constraints. This can be acceptable for its use within the BML architecture, where such constraints are meant to be generated by an automatic resolutor, but it can be a hard job for a human editor. Differently, here we specify animation “speeds”, for which it is easier to identify default values (e.g., walk speed), and which are at run–time converted into durations according to actual path lengths. Second, EMBR lacks the syntax to expose parameters for newly defined animations. Other signing animation systems are realized in terms of planning (see, e.g., [6]), though not in the context of a real-time performance.

We only have preliminary evaluations of the system, since this is an ongoing project, but reports from professional interpreters are very promising. The approach pursued by the system of issuing a “pure” sign language (instead of a word-to-word translation, i.e., “signed Italian”) seems to be well received by the deaf people, though it is sometimes considered a bit clumsy. We are preparing a corpus of translated sentences and a questionnaire for understanding the reception of the virtual interpreter within LIS speakers.

References