FP7 - Information and Communication Technologies

Collaborative Project

Dicta-Sign

Sign Language Recognition, Generation and Modeling with application in Deaf Communication

Grant Agreement Number 231135

Deliverable D5.4: Annotation Tools Update

Leading Partner: Université Paul Sabatier (UPS)

Version 0.4

February 7, 2012

Contents

Introduction

Linguists of sign languages need annotations of sign language video corpora. Video corpora annotations are mainly done manually on Annotation Tools (ATs). This work is really time-consuming and frequently very repetitive. Furthermore, for some measures, humans are not reliable, for example, to estimate the gaze direction. It is the reason why automation, even partial, is really beneficial. Such automation technologies are often at the forefront of research. In these cases, the knowledge (algorithms, code and even binaries) behind may be confidential. Besides, almost all these modules are very power consuming. These are two reasons to enable material separation of such modules from ATs.

A solution had been proposed in Dictasign's Deliverable 5.2. It is a Distributed System Architecture based on what have been called Automatic Annotator Assistants (A3).

This solution introduces four agents:

- Annotation Tool (AT): It is the piece of software on which the user is working.
- Automatic Annotator Assistant (A3): It is the module providing the service of processing annotations.
- A3 Supervisor (A3S): Acts like a service directory.
- Video File Server (VFS): Hosts videos of corpuses and manage accesses.

Since its presentation in D5.2, the specification of the system and its integration have evolved. This Deliverable 5.4 presents these evolutions in five parts. The first part shows evolutions in a very general point of view on the global architecture of the system. The second part gives a report on all A3 developed in the project Dictasign. The third part provide a full specification of VFS and gives information about the reference implementation. The fourth part presents evolutions of types and the Common Type Library, two features introduced in Deleverable 5.3. The fifth part presents the progress done in the integration of A3 architecture in the AT AnColin.

1 General overview

In this part, we describe the distributed system architecture to integrate automatic processing tools within existing annotation software. This architecture aims to enable the cooperation among a heterogeneous set of modules. These modules are developed in many different programming languages, for several operating systems and software platforms. The system comprises four parts : the Annotation Tool (AT), the Automatic Annotation Assistants (A3), the A3 Supervisor (A3S) and a Video File Server (VFS). The AT is the software which interacts with the annotator and allows to annotate video corpora manually. The A3 is the software which does automatic annotation of corpora and acts as a server. The A3S is an A3 directory. The VFS is the place where all the video corpuses are stored for easy sharing.

In D5.2, the global working process exposed was :

- At A3 launch,
  1. the A3 gives to the A3S the list of all the function it provides.
- At user event “Get the list of available functions”,
  1. the AT gets the lists of all functions declared to the A3S.
- At user event “Apply the function F”,
  1. the AT prepares the parameters for the A3.
  2. the AT orders the annotation to the concerned A3.
  3. the A3 gets the video.
  4. the A3 executes F.
  5. the AT gets the results back.

In this first specification, the focus was on the logical part, the general principles of this distributed system. The implementation of this system revealed many concrete requirements and problems. Most of them can be merged in one theme : security.

The security of sensitive code had been thought early and was one subject of the D5.2 (cf. Introduction above). It is data security which is discussed now. Indeed, it appeared that annotations and videos are sensitive data as well as A3s’ code. For example, videos are commonly under restrictive image rights. The security of data is designed over three axis: authenticity, authorization and confidentiality.

- The authenticity is the area which aims to enable agents identify one another and to ensure that there is no impersonation. The solution choosed for authenticity is TLS. It is based on the fact that each agent owns an identity certificate.

These individual certificates are shown as keys in figure 1.1. When starting a communication, agents shake hands and send one another their certificate.

- The authorization is the area which covers rights. It is the fact for each agent to choose knowingly to serve or not another agent. As it hosts videos, it is the VFS which centralizes the authorization power about videos, it manages trust in AT users and A3s. This is symbolized by certificate collections under the VFS in figure 1.1. Depending on their implementation A3S and A3s can also control authorization, for example, to save power resource.

- The confidentiality is the fact to secure communications against listening. This is resolved with TLS which cypher all communications. This appears on figure 1.1 as padlocks.

These changes on communications do not impact on the general sequence of operations. It stays the same as in the version summarized above. See figure 1.2 for a more concise presentation.

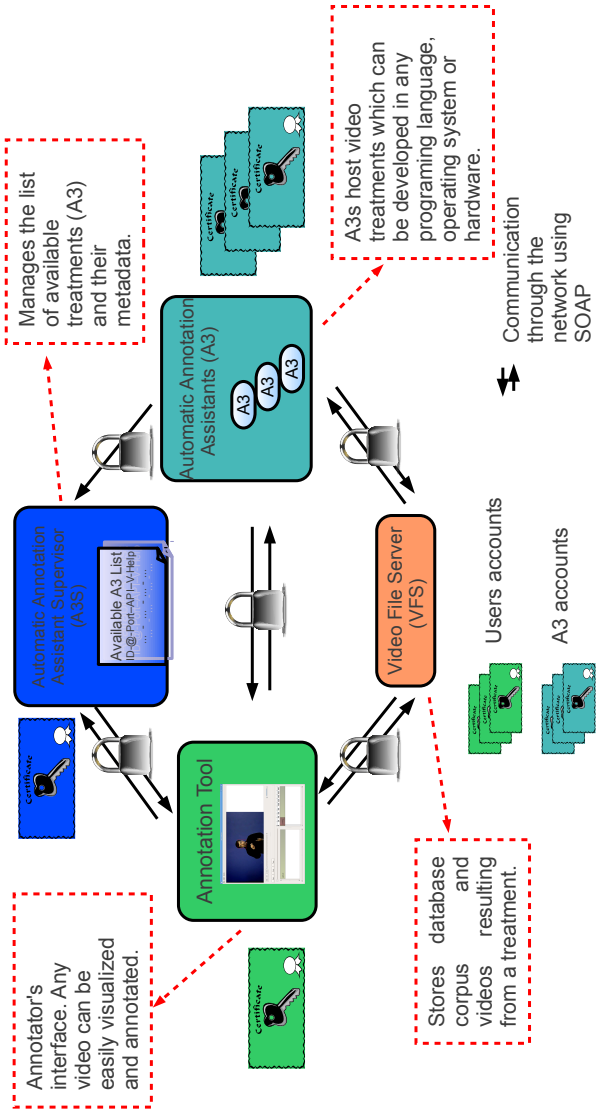


Figure 1.1: Overview of the system

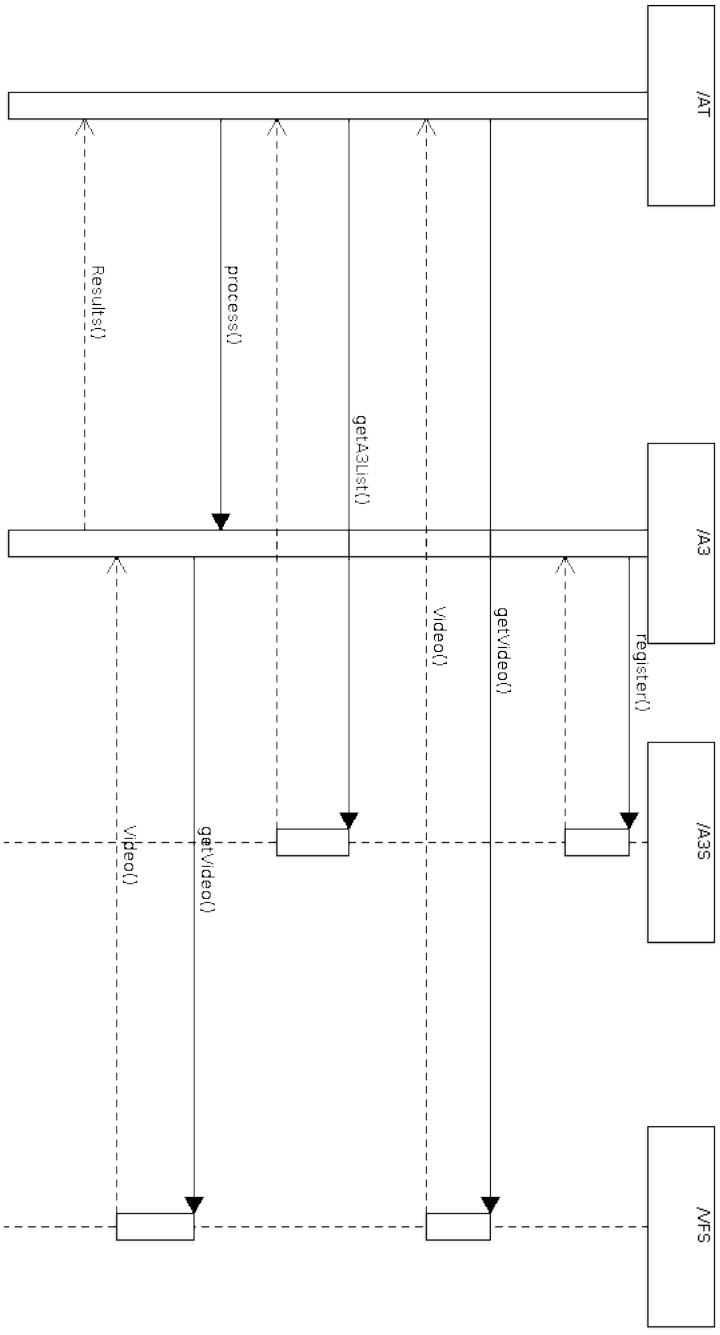


Figure 1.2: Overview of the collaboration between agents

2 Development and Implementation of Automatic Annotator Assistants A^3

In the WP1 and WP2 we have developed image processing algorithms to assist the annotation of SL video corpus. These algorithms have been implemented as A^3 to be used directly with an AT the meets all the requirements described before. The implemented A^3 are:

- Body Part Tracking
- Sign Segmentation

Also we have adapted some other methods developed in WP1 and WP2 of the project. Although these algorithms have not been developed to fulfil an annotation task, they can be implemented to be used with an AT. The adaptation consist on format the input and output to allow the AT and the A^3 to communicate. The methods that have been straight forward implemented are:

- Feature Extraction
- Signing or not

2.1 Body Part Tracking

This A^3 tracks hands and head using a particle filter based approach which robustly handles hand-over-head occlusion using a template before occlusion [4]. The results consist of the coordinates of the centre of each body part.

2.1.1 Principle

Tracking algorithms intend to estimate the state of the system over time by using a sequence of measurements. Bayes filter estimates the posterior probability density function $p(\mathbf{x}_t \mid \mathbf{z}_{1:t})$ of the current state $\mathbf{x}_t$ conditioned on all observations $\mathbf{z}_{1:t} = \mathbf{z}_1 \dots \mathbf{z}_t$ with $\mathbf{z}_t$ the observation vector. The probability density function $p(\mathbf{x}_t \mid \mathbf{z}_{1:t})$ for a first-order Markov process, i.e. the state $\mathbf{x}_t$ depends only on $\mathbf{x}_{t-1}$, is obtained in two stages: prediction and update. The prediction stage,

$$p(\mathbf{x}_t \mid \mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \cdot p(\mathbf{x}_{t-1} \mid \mathbf{z}_{1:t-1}) dx_{t-1}, \tag{2.1}$$

estimates the prior distribution for $t+1$ by the convolution of the posterior distribution $p(\mathbf{x}_{t-1} \mid \mathbf{z}_{1:t-1})$ and the transition probability distribution $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$, i.e. the dynamic model of the system. The updating stage,

$$p(\mathbf{x}_t \mid \mathbf{z}_{1:t}) = k \cdot p(\mathbf{z}_t \mid \mathbf{x}_t) \cdot p(\mathbf{x}_t \mid \mathbf{z}_{1:t-1}), \tag{2.2}$$

computes the posterior probability density using the observation likelihood $p(\mathbf{z}_t \mid \mathbf{x}_t)$ and the temporal prior distribution, $p(\mathbf{x}_t \mid \mathbf{z}_{1:t-1})$, over $\mathbf{x}_t$ given past observations.

Particle filter (PF) [5] presents a solution framework for tracking stochastic movements. It sequentially estimates the states $\mathbf{x}_t$ of the system by implementing a recursive Bayesian filter by Monte Carlo simulations. The posterior probability density $p(\mathbf{x}_t/\mathbf{z}_t)$ of the current state $\mathbf{x}_t$ is approximated by a weighted particle sample set, $\{\mathbf{s}_t^n, \pi_t^n\}_{n=1}^N$. PF maintain multiple hypothesis, i.e. each particle is a hypothetical state of the object, weighted by a discrete sampling probability $\pi_t^n \propto p(\mathbf{z}_t \mid \mathbf{x}_t = \mathbf{s}_t^n)$. Particle weights correspond to the observation generated by the hypothetical state and reflects the image feature relevance associated to each particle. The feature used to characterize head and hands is the skin colour. Particle filter algorithm consists essentially of iterating the following steps:

1. **Resample** N particles from the set $\{\mathbf{s}_{t-1}^n, \pi_{t-1}^n\}_{n=1}^N$ to $\{s'^n_t, \frac{1}{N}\}_{n=1}^N$. Particles are selected by their weight, large weight particles are duplicated while low weight particles are deleted.

2. **Propagate** particles using the dynamic model of the system $s_t^n \sim p(\mathbf{x}_t \mid \mathbf{x}_{t-1} = \mathbf{s}'^n_{t-1})$ to obtain $\{\mathbf{s}_t^n, \frac{1}{N}\}_{n=1}^N$.

3. **Weight** the new particles sample with the observation (image feature) $\mathbf{z}_t$ as $\pi_t^n \propto p(\mathbf{z}_t \mid \mathbf{x}_t = \mathbf{s}_t^n)$ and normalize so that $\sum_{n=1}^N \pi_t^n = 1$.

The weighted particle sample set, $\{\mathbf{s}_t^n, \pi_t^n\}_{n=1}^N$, is used to compute the expectation of the state at time t. The expectation is defined as

$$E[\mathbf{x}_t] = \sum_{n=1}^{N} \pi_t^n \mathbf{s}_t^n. \tag{2.3}$$

Multiple object tracking is challenging because of the presence of occlusions and the similarity of colour. Since the observation is computed considering skin colour, if no further constrains are considered for the filters, they might track the same object. In addition when hand overlaps the head, it is not possible to accurately determine the position of the hand. Theses problems are addressed using the exclusion principle [8]. A penalization scheme to avoid particles from one filter to influence other filter has been developed which consists on penalizing the observation image feature. The overall tracking algorithm, Fig. 2.1 consist of a sequence of detection and penalization. Firstly, head is detected and penalized using the template. After head penalization, hands are detected and penalized to correct their influence in head detection. Finally head and one hand are penalized to avoid influence. Penalization coefficients, when hand overlaps the face, are computed using a template before occlusion and are proportional to the luminance difference after and before the occlusion.

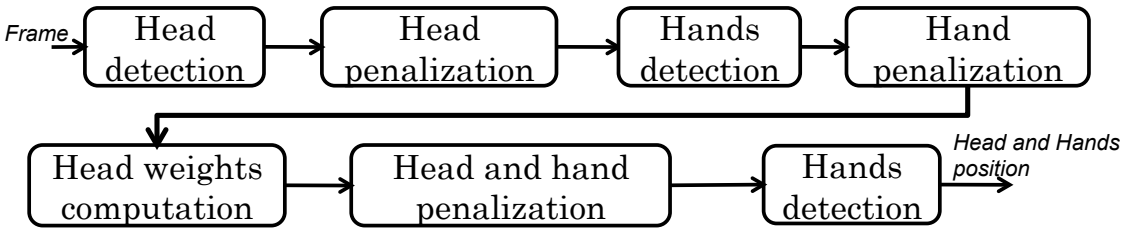


Figure 2.1: Tracking algorithm. Sequence of detection and penalization.

2.1.2 Evaluation

We have performed some experiments to evaluate our approach. Since head and hands were manually annotated in the LS-COLIN [7] corpus, we have selected A sequence of 3000 frames containing fast dynamics and several hand over face occlusions.

The results obtained from this A^3 are compared to the method proposed by Gianni *et al.* [2] and the approach introduced by Lefebvre [6]. Figure 2.2 shows the tracking results for a sequence where hand overlaps the head. In the case of rectangular skin regions [6], when the hand fully occludes the face one skin blob is missed, Figure 2.2 (top row). For a cloud of points model [2] the head and hand filter share the same skin region, thus filters do not accurately determine head and hand position, Figure 2.2 (middle row). On the other hand our A^3 accurately find the position of the head and the hand when hand overlaps the face, Figure 2.2 (bottom row).

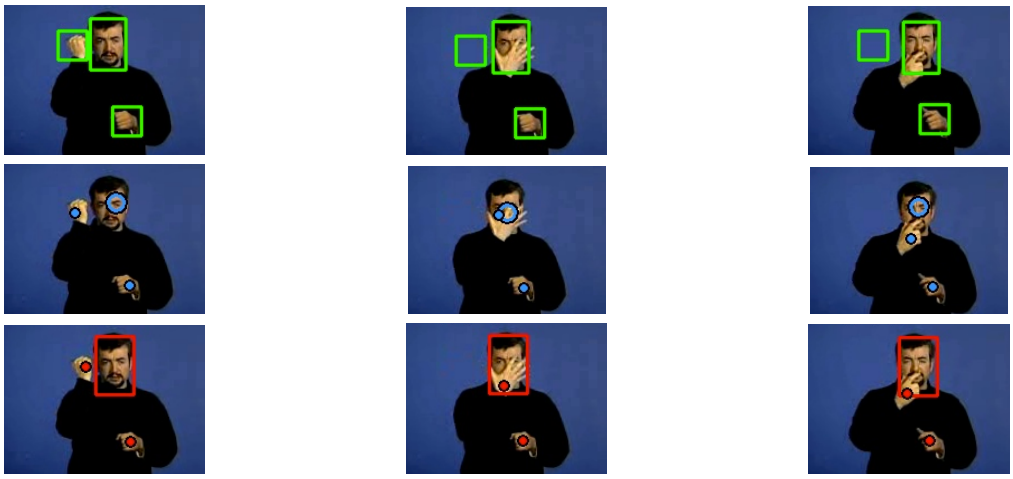
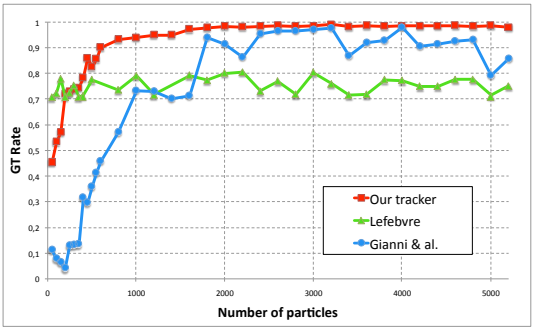


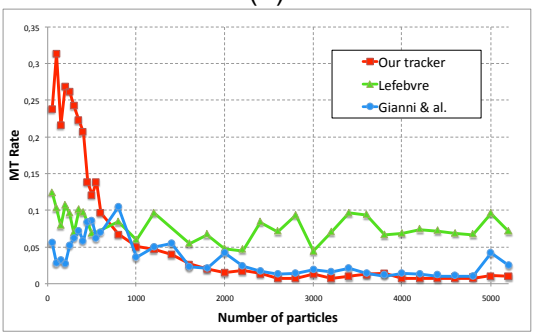
Figure 2.2: Tracking results for Lefebvre (top row), Gianni *et al.* (middle row) and the proposed approach (bottom row).

We have quantitatively evaluated tracking results through three rates: Good tracking rate (GTR), Miss tracking rate (MTR) and False tracking rate (FTR). GTR evaluates filters tracking the assigned skin object, MTR filters tracking another skin object represented by the same model, e.g. hand filters exchange, and FTR filters tracking none skin object or another object represented by a different model, e.g. hand and head filter exchange. Figure 2.3 shows the rates achieved by our A^3 in comparison to the algorithms proposed in [2] and [6]. We notice that our method significantly improves

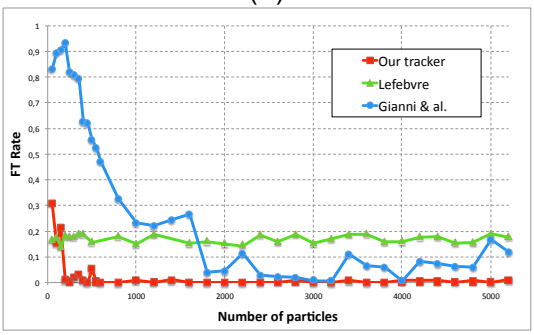
stability with respect to the other two methods. In addition we achieve the best GTR and the lowest MTR and FTR beyond approximately 300 particles.



(a)



(b)



(c)

Figure 2.3: (a) shows the GTR, (b) the FTR and (c) the MTR achieved by Lefebvre, Gianni *et al.* and our approach.

2.1.3 Application Programming Interface (API)

Parameters needed by this A^3 correspond to a track corresponding to the video segments to process. The output of this A^3 corresponds to a list of three point; head, right hand and left hand. Each point correspond to a pair of coordinates.

- **Input** : 1 track

  Void Segments of the video to process

- **Output** : 3 tracks

Point2D	Coordinates of the head centroid
Point2D	Coordinates of the right hand centroid
Point2D	Coordinates of the left hand centroid

2.2 Sign Segmentation

The second A^3 developed for this WP consist on the segmentation of continuous sign language. Sign segmentation is challenging because of the co-articulation problem. Figure 2.4 shows an example of the co-articulation between two signs in French Sign Language (FSL); *"United States"* and *"Tower"*. It is difficult to segment a video sequence into signs and transitions because one sign is influenced by the previous sign and itself influences the following sign. However, it is possible to approximate signs limits using manual and non-manual features. Although only manual features are considered we are aware that lot of information can be extracted from non-manual features.

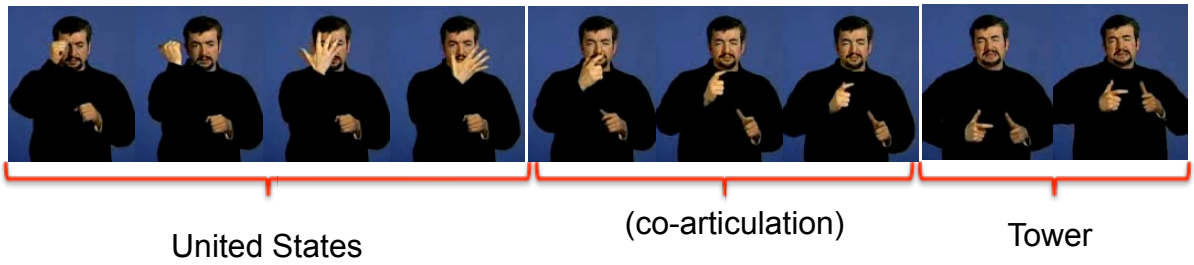


Figure 2.4: Example of *co-articulation*: gesture between the end of the sign "United States" and the beginning of the sign "tower" in Frech Sign Language

Since no linguistic information is used to perform the segmentation, it is impossible to classify sequences into *Signs (S)* and *Transitions (T)*. Sign segmentation is, then, modelled as an *Even Detection* approach where the limits between S and T are detected. Figure 2.5 illustrates an example of the annotation software *AnColin* where two tracks have been created: *Manual Segmentation* and *Auto Segmentation*. The former, Fig. 2.5 *Track: Manual Segmentation*, represents the results of manual sign segmentation where the annotator has selected the first and the last frame of a sequence corresponding to a sign. The latter, Fig. 2.5 *Track: Auto Segmentation*, illustrates the expected results from an automatic annotation approach.

2.2.1 Principle

Our A^3 algorithm uses motion and hand shape information to perform the segmentation. Head and hands are tracked, using the A^3 *"Body Part Tracking"* describe above. Results from the tracking A^3 are used to compute velocity and detect events. Afterwards hand is segmented for each detected event. Hand segmentation is performed using an adapted algorithm depending if hand is occluding the face or not. Hand shape is systematically compared to the hand shape of adjacent detected events. This second step removes events for sequences that have been over segmented.

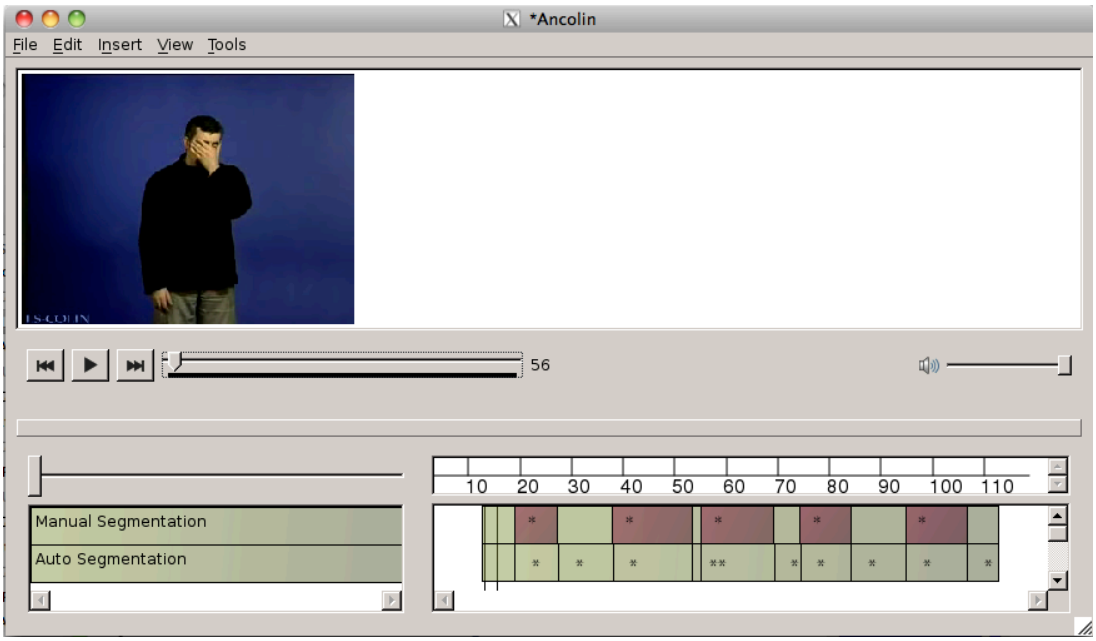


Figure 2.5: Annotation tool *AnColin*. Manual Segmentation. track represents the segmentation results performed by an human annotator and Auto Segmentation. track shows the expected segmentation results from automatic low level image processing techniques.

Table 2.1: Motion classification

Static Pose (S)	One Hand(1H)	Two Hands (2H)	
$v_r \approx 0$	$v_r > 0$	$v_r \approx 0$	$v_r > 0$
$(v_1 \approx 0 \ \ \& \ \ v_2 \approx 0)$	$(v_1 \approx 0 \oplus v_2 \approx 0)$	$(v_1 \neq 0 \ \ \& \ \ v_2 \neq 0)$	

Motion features are computed using a moving window of small size to avoid large signal smoothing (between 3 and 5 frames): velocity for right and left hand, $v_1(t)$ and $v_2(t)$ respectively. Velocity magnitude is used to determine relative velocity $v_r(t)$ between hands, i.e. velocity difference between right and left hand. It is derived as

$$v_r = \|v_1(t) - v_2(t+\tau)\|\,, \tag{2.4}$$

where τ represents the gap between hands velocity in symmetric movements. Indeed when hands move together there is a small gap between right and left hand velocities. Figure 2.6 shows *(on the left)* the sign *'shocked'* in French Sign Language which corresponds to a two-hand movement. Hands move together in a symmetric way from bottom to top. The velocity profile for right and left hand *(on the right)* is plotted. Although velocity profile for both hand looks very similar, one hand remains behind the other. In this example τ is about 1 frame.

Relative velocity $v_r(t)$ allows to identify movements performed with one or two hands. In addition by comparing velocity close to zero, it is possible to identify static poses. Table 2.1 shows the proposed motion classification tacking into account $v_1(t)$, $v_2(t)$ and $v_r(t)$. We propose to classify motion in three classes: static pose (S), one

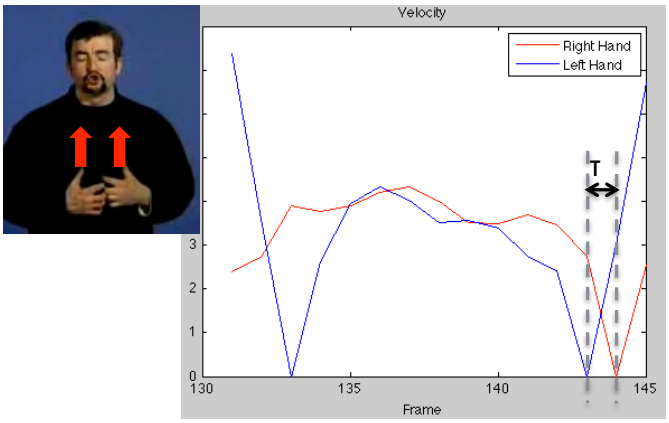
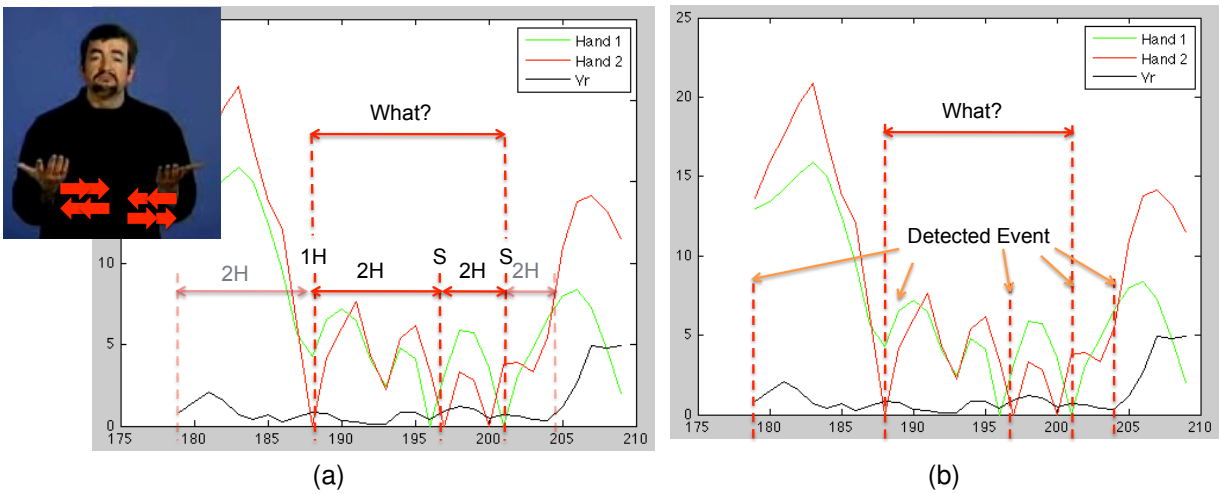


Figure 2.6: Illustrates velocity of right and left hand. *Left:* The sign *'shocked'* in French Sign Language. *Right:* Velocity profile for right and left hand.

hand gestures (1H) and two hands gestures (2H). Static pose (S), where no hand is moving, is detected when $v_r(t) \approx 0$ and right and left hand velocity are close to zero. One hand gestures are identified by $v_r(t) \approx 0$ as well as the velocity of one of the hands. For two hands classification, $v_1(t)$ and $v_2(t)$ different to zero, we can derive two subclasses. When $v_r(t)$ is close to 0, both hands move at the same speed *'symmetric gesture'* otherwise both hands move but there is no symmetry. From this classification events are detected as the changing from one class to another. A limitation of our approach is that it over-segments repetitive signs. Figure 2.7 shows an example of this



(a) (b)

Figure 2.7: Sign *'What?'* in French Sign Language is illustrated. Double arrow represent a repetitive gesture. (a) shows the hands velocities and the relative velocity. Motion classification results are illustrated. (b) shows the detected events.

limitation. On the top left the sign *'What?'* is illustrated. It is a repeated symmetrical sign. The sign is over-segmented because magnitude velocity comes to zero in the middle of the sign. We classify it as a static pose. Over-segmentation can be corrected considering other features that remain constant during repetitive gestures as is

the case of hand configuration.

Sign segmentation improvement is the next stage of our sign segmentation algorithm, it consists on the introduction of hand configuration to improve over-segmentation. Hand configuration recognition is a challenging task because of the high 2D hand shape variability from a mono-camera view. Same hand configuration may produce different hand shapes. For this A^3 we consider 2D hand shape regardless hand configuration.

Hand is systematically segmented at each detected event and compared, using similarity measurements, with the hand shape from the previous and following event. If hand shape is similar to the shape of previous and following events, we consider that this event has been detected in the middle of a sign and it is possible to remove it. Figure 2.8 show the *Automatic Detection* aligned to the *Manual Annotation* for the sign *What?* in French Sign Language. As it has been explained before the segmentation step using motion features over segments this sign. Now to correct the over-segmentation hands are segmented for each detected event, Fig. 2.8 *on the top*. Notice that the hand shape is similar to the hand shape of neighbouring limits. It is, then, possible to remove this limit to correct the segmentation.

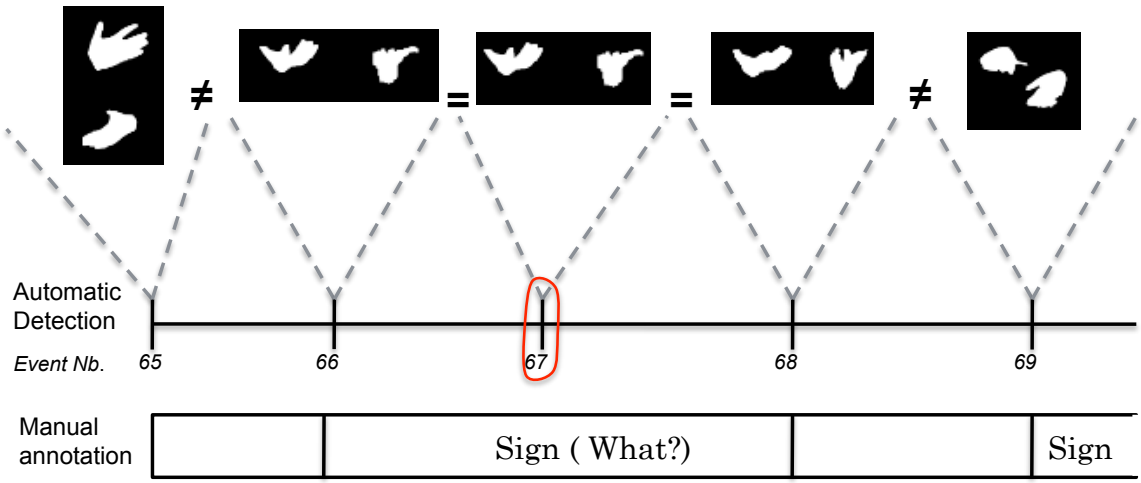
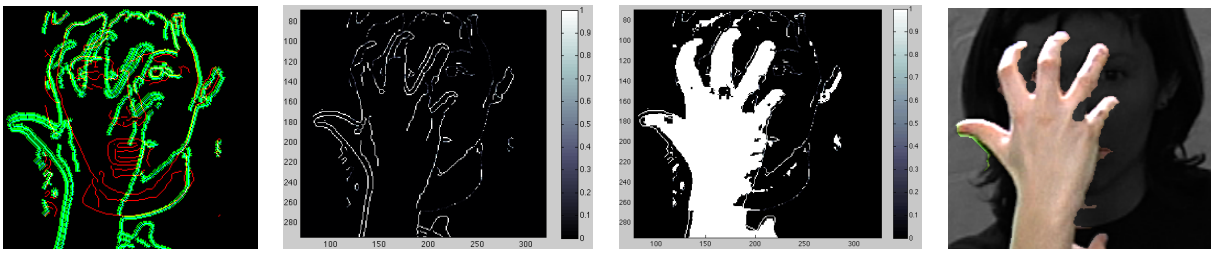


Figure 2.8: On the top it is shown the segmented hands at each detected event for the sign *'What?'*. Notice that hands shape for the same sign remains similar.

Hand segmentation is a challenging task because of the illumination changes and noise. We use a skin model to extract skin regions. Signer face is first segmented using an RGB skin model. Afterwards the model is build in YC_bC_r colour space. Since luminance and chrominance components are decoupled, the lumiance component can be ignored to be robust to illumination changes.

This specific model learned for each signer is used to extract skin regions for each frame corresponding to a detected event. The extracted skin regions are labelled to identify connected regions. The position obtained by the tracking algorithm is used to select the connected region that corresponds to each object, head and hands. For each position vector for head, right and left hand, $\mathbf{P}_h$, $\mathbf{P}_{h_1}$ and $\mathbf{P}_{h_2}$ respectively, the label assigned to one region is compared with labels from other objects to determine if there is an occlusion. If there is no occlusion connected region for right and left hand are extracted. Otherwise another segmentation approach is required to segment hand.



(a) (b) (c) (d)

Figure 2.9: (a) shows the registration results before and during occlusion, (b) the normalized edge orientation difference map and (c) the luminance component changes and edges orientation difference map merged. White pixels have greater probability to belong to the hand. (d) shows the hand segmentation result.

During occlusion, hand segmentation is challenging because of the similarity of colour between face and hands and the high hand shape variability, to achieve hand over face segmentation we propose a method based on the combination of two features: pixel colour and edges orientation [3].

A face template before occlusion is used to register the face during occlusion. Registration is performed using local gradient orientations taking into account translation and rotation. Figure 2.9a shows the contours of the face image, before and during occlusion, after the registration. Edges are classified as belonging to the hand or to the face by mapping edges orientation to the face template. For each pixel contour in the occluded image the corresponding contour is searched along the normal. The difference edge orientation map is, then, computed, Figure 2.9b. Colour information is extracted from changes on the luminance component before and during the occlusion. Finally by merging both features, Figure 2.9c, and by using an hysteresis threshold, which considers connectivity, a robust hand segmentation is reached, Figure 2.9d.

Hand shape comparison between detected events is necessary to extract shape features. Hand configuration classification is a challenging task and we prefer to use similarity measurements to determine hand shape similarity. We propose to use two measurements: equivalent diameter, ϵ_d and eccentricity ϵ. The former specifies the diameter of a circle with the same area as the region. The latter represents the eccentricity of the ellipse that has the same second-moments as the region. The eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length. The advantages of these measurements is that they are invariant to rotation and translation. However the inconvenient is the sensibility to scaling and noise.

2.2.2 Evaluation

We have conducted experiments on two sequence without any language constraints: LS Colin [7] and DEGELS [1], Figure 2.10. The segmentation algorithm has been performed on 2500 frames. Ground truth for both sequences has been manually performed by a native signer. The evaluation criteria is illustrated in Fig. 2.11. True positive rate (TPR) corresponds to the limits that have been correctly segmented with a tolerance of δ number of frames. False positive rate (FPR) corresponds to events that have been detected but that do not belong to any limit in the ground truth annotation. The

(a) (b)

Figure 2.10: (a) Ls-Colin corpus and (b) DEGELS corpus[1]

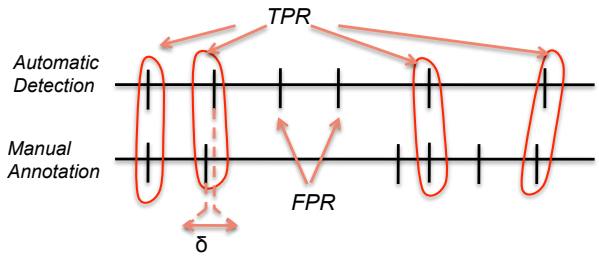


Figure 2.11: Evaluation criteria.

tolerance δ has been determined trough an experience where we have asked a native signer to perform the manual segmentation of the same sequence several times. We have noticed, as expected, that the same annotator have not selected exactly the same frame for each limit. Figure 2.12 illustrates the manual annotation obtained for three tiers. We have computed the average gap $\bar{\Delta} = 3.7\ frames$ from the gap at each limit Δ_j which corresponds to $\delta \approx 2$. Table 2.2 shows the TPR and the FPR for both sequences for $\delta = 2$. Notice that the TPR for motion and motion plus hand shape remains stable, 81.6% for LS-Colin and 74.5% for DEGELS, while the FPR decreases of about 3% for LS Colin and 10% for the DEGELS corpus when hand shape features are introduced. In these results we notice that over 74% of the events are detected with an accuracy of ± 2 frames which corresponds to the accuracy of a native signer performing the same annotation several times so we consider that this annotation is as good as an annotation manually performed. After correction we noticed that only 11.25% for LS-Colin and 7.1% for Degels from the FPR correspond to sign limits while

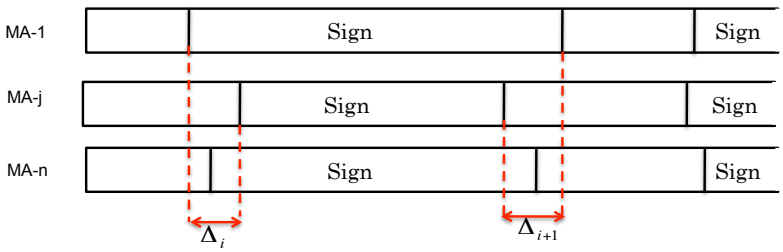


Figure 2.12: Manual annotation illustration.

Table 2.2: Segmentation Results

	Motion		Motion + Hand Shape	
	TPR(%)	FPR(%)	TPR(%)	FPR(%)
LS- Colin	81.6	46.2	81.6	44.9
DEGELS	74.5	54.2	74.5	44.7

the remaining detections can be deleted.

We have evaluated the usefulness of our approach by determining the time needed by a native signer to annotate a video sequence and correct the semi-automatic segmentation. The time needed is quite similar (10 min for LS-COLIN and 4.5 min for Degels). The manual annotation and the correction have been performed using the previous version of the annotation Tool AnColin which has been developed to perform manual annotation and was not adapted yet for correcting semi-automatic segmentation, since it is needed to select the beginning and the end of each sign.

This A^3 has shown promising results since the time needed to correct the results is not higher than the time needed to fully manual annotate. In addition the annotation becomes less dependent to the annotator's knowledge and is reproducible which are some of the drawbacks of the manual annotation.

2.2.3 Application Programming Interface (API)

This A^3 needs the results from the body tracker A^3. This is given in the input as one list of points track. Also the segments of the video to process are given as a track. As a result this A^3 returns a track of lengthless segments. Each segment correspond to a detected event.

- **Input** : 2 track

  List2DPoints List of coordinates of tracked body members

  Void Segments of the video to process

- **Output** : 1 track

  Void Detected events; lengthless segments

2.3 Facial Feature Tracker

This A^3 takes some examples of features to track in the video (coordinates in different images) and try to learn the aspect of these features to track them in the complet video (more details in Ong & al [9]).

2.3.1 Application Programming Interface (API)

This A^3 needs as input a list of coordinates for the points to track, this is required for each image in the learning dataset. It returns, as output, list of the tracked points for each frame. Each point type is a pair of coordinates.

- **Input** : 1 track

  List2DPoints List of coordinates for points to track

- **Output** : 1 track

  List2DPoints List of coordinates of tracked points

2.4 Signing Detection

This A^3 concerns the detection of signing activity. Indeed, in a corpus of dialog, there are many times where one of the participant is just listening. This A^3 will detect where in the sequence, the informant is actually signing and where isn't.

2.4.1 Application Programming Interface (API)

This A^3 requires one input track with the segment to process and returns the segments where the signer is signing.

- **Input** : 1 track

  void Segments of the video to process

- **Output** : 1 track

  void Segments where the signer is signing

3 Video File Server

The A3 architecture needs an agent called Video File Server (VFS). This agent's role is to host the videos of a corpus when a user is working on an annotation task using A3s. It must meet a requirement of data security.

We decided to make a normative specification of the interface between VFS and its clients (ATs and A3s). This specification primarily aimed to simplify the implementation of ATs and A3s. It should also allow the emergence of new VFS implementations compatible with existing ATs and A3s. Finally, we provide a server implementing this specification.

3.1 General specifications of VFS

3.1.1 Scenarios of use

- Scenario 1: The human annotator wants to work on a video from a corpus. His annotation tool (AT) retrieves the (hierarchical) list of videos accessible by the user on the VFS. The user selects the video on which he wants to work and his AT gets the video from the VFS. The user can then start his work on his terminal.

- Scenario 2: The human annotator wants to delegate some of his work to an A3. Seamlessly, the AT will inform the VFS that an A3 will claim the video. The A3 will then download the video from the VFS to work on it.

3.1.2 Needs

The needs about the VFS can be identified from the two previous scenarios and from the requirement of security. The list is the following:

- Access control

- Communications security

- Right management

- API for clients well documented and easy to use.

- Ease of deployment

3.2 Normative specification

This specification is intended to stabilize the interface between VFS and clients(AT and A3). In consequence, it does not cover every aspect of a VFS. The first step is to separate what should be standardized and what should not. The result is:

- Standardized:
  - The entire part about VFS-client(AT, A3) communication.
    - Security of communications.
    - Application level communication protocol.
    - Model of the data involved in communication.
- Free:
  - All the part around the administration of the server, like:
    - User management.
    - A3 management.
  - Every choices about the implementation, for example:
    - Internal data model.
    - Deployment solutions (hosting,etc.).
    - Video storage on the server.

Note: In this partition, the ”standardized” part is intended to be exhaustive, not the ”free” part.

3.2.1 Security layer

The task to secure access to VFS is given to the TLS (formerly know as SSL). TLS connexion is used to achieve confidentiality, authenticity and integrity.

To insure authenticity, the two ends of the communication must authenticate themselves with a certificate. Consequently, all the agents which will communicate with VFS (AT and A3) and VFS themselves must have a certificate signed by a certification authority.

Authentication of clients by VFS remains completely on the identity given by the TLS layer. The implementation is free to associate its internal representation of identity to either the identity embedded in certificates (Common Name field of certificates) or to the certificate itself (Public Key field).

The way information is secured through the communication is summarized in figure 3.1

3.2.2 Application layer

The expected behavior of the VFS to client requests is given informally by the scenarios. The two scenarios are two use cases.

Behavior-oriented rewriting of the scenarios:

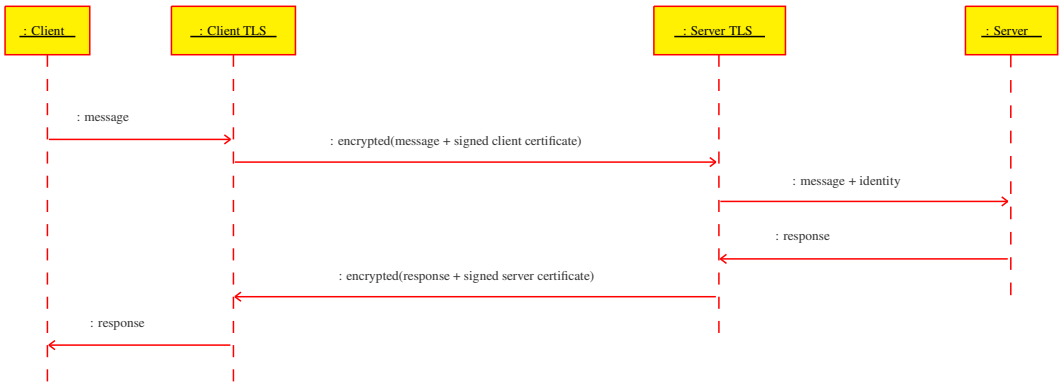


Figure 3.1: Schematic communication through the TLS layer

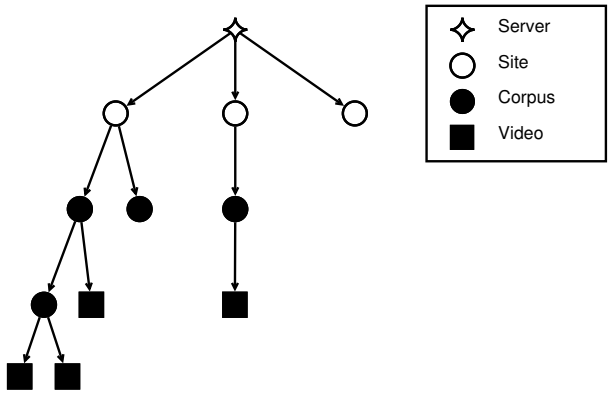


Figure 3.2: Schematic representation of the data model

- The AT request the video list. The VFS sends the tree list of videos back. This list is filtered with the user access right. The AT asks for the video the user wants to the VFS. The VFS respond with the video stream.

- The AT grants access right to an A3 for a video on the VFS. The VFS responds with a disposable URL

From these scenarios is extracted a data model and a set of operations (or actions).

Data Model

This VFS specification freezes the data model as visible by ATs and A3s. The internal data model is free to be completely different but, at the API, level it must be as specified below.

Videos are organized in a tree as shown in figure 3.2. The tree's root is the server. The layer under the root is made of some kind of field called "site". In "Site" can be set mixed videos and another kind of field called "corpus". Like a site, a corpus can contain (possibly mixed) videos and corpuses. This data model also specify to give an ID number and a URI to A3s and videos (see figure 3.3).

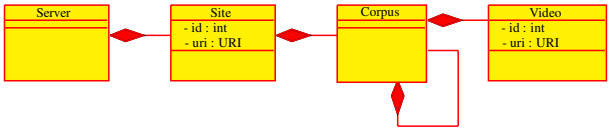


Figure 3.3: UML Class diagram of the data model

Operations

A set of operations has been extracted from the two use cases. These operations are initialized by two different agents: ATs and A3s. This criterion is used to cut the set in two.

ATs' operations are:

- Asking for the file tree.
- Download a video.
- Give an access to a file to an A3.

A3s' operations are:

- Download a video.

The solution adopted for the interaction is of type REST[1] with XML response.

Note: From now, for the needs of examples, the URL address of the VFS will be considered as "https://vfs.example.org".

The VFS have two interfaces, one for ATs and the other for A3s. Each has its own URL. These two URL are specified:

- AT API: "https://vfs.example.org/api/at "
- A3 API: "https://vfs.example.org/api/a3 "

General aspect of requests and responses

Requests Calling an operation on an API is done with an HTTP GET command. The URL of the command is the API's one or one of its extensions. The parameters of the HTTP command are used. An example of a request using the possibility to encode parameters in the URL is given here:

"https://vfs.example.org/api/at?action=download$video=dictasign.lsf.s1t1a"

Responses In most cases, the response is in XML. The exceptions are when the request is to download a video. In these cases, if there are no errors, the response is directly the video stream.

Here comes an informal description of the XML dialect used: The spacename used is "http://dictasign.eu/ns/vfs/atapi". The document root tag is <atapi>. There are two

[1] Representational state transfer (REST) is a style of software architecture. It defines some guidelines in the conception of APIs.

parts under the root: a part <errors> detailing encountered errors, and a part <results> detailing the results. In the <errors> part, the errors are embedded in <error> tags. The content of an <error> tag should be a description of the error but its form is free. The <results> tag enclose results in <result> tags. A <result> must have at least the two attributes “action” and “status”. The attribute “action” remind which operation the results is about. The “status” attribute value can be “ok” or “ko” depending on whether the operation has been passed on the state of the server.

API for ATs

From the two use cases, 3 operations initialized by the AT can be deduced:

- List videos: list.
- Download a video: download.
- Give a temporary acces right to an A3: grant.

Action ”list” This request is to get the tree of sites, corpuses and videos to which the user has an access right. There are no parameters for this request.

An example of a request is given here:

”https://vfs.example.org/api/at?action=list”.

The results is given in XML. An XML Schema definition is available for this XML dialect. An example of a result is given here:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<a:atapi xmlns:a="http://dictasign.eu/ns/vfs/atapi">
  <a:errors>
  </a:errors>
  <a:results>
    <a:result action="list" status="ok">
      <server>
        <site name="IRIT">
          <corpus name="lsf">
            <corpus name="S9">
              <video name="S9T4" uri="S9T4"/>
            </corpus>
            <video name="nasro" uri="nasro"/>
          </corpus>
          <corpus name="History">
            <corpus name="20s">
              <video name="boat" uri="boat"/>
              <video name="train" uri="train"/>
            </corpus>
            <video name="AnnieOakley" uri="AnnieOakley"/>
          </corpus>
          <corpus name="test">
            <corpus name="test2">
              <video name="vid2" uri="vid2"/>
              <video name="vid4" uri="vid4"/>
            </corpus>
            <video name="ptitevid" uri="ptitevid"/>
            <video name="vidnull" uri="vidnull"/>
```

```
            </corpus>
          </site>
        </server>
      </a:result>
    </a:results>
</a:atapi>
```

Action "download" This request is to download a video from the VFS. VFS have to control rights before to complete the request. To work, this request must have a parameter identifying the targeted video. It can be either the video's URI set in the parameter "video" or the video's ID set in the parameter "videoid".

An example of a request is given here:

https://vfs.example.org/api/at?action=download$video=nasro".

The result can be either the video stream or in case of error, an XML structure in the format seen above.

An example of a result is given here:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<a:atapi xmlns:a="http://dictasign.eu/ns/vfs/atapi">
  <a:errors>
      <a:error>ATAPI: No video with URI funnyvideo42 has been found in the
          database. </a:error>
  </a:errors>
  <a:results>
  </a:results>
</a:atapi>
```

Action "grant" This request is to give an access right to a given A3 for a given video on the VFS. VFS have to control rights before to complete the request. To work, this request must have two parameters: one identifying the targeted video and another identifying the targeted A3. For the video, it can be either the video's URI set in the parameter "video" or the video's ID set in the parameter "videoid". For the A3, it can be either the A3's URI set in the parameter "a3" or the A3's ID set in the parameter "a3id".

An example of a request is given here:

"https://vfs.example.org/api/at?action=grant$video=nasro$a3id=42".

The result is an XML structure in the format seen above.

An example of a result is given here:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<a:atapi xmlns:a="http://dictasign.eu/ns/vfs/atapi">
  <a:errors>
  </a:errors>
  <a:results>
      <a:result action="grant" status="ok">
        https://vfs.example.org/api/a3?action=download&jobid=42
      </a:result>
  </a:results>
</a:atapi>
```

API for A3s

About the A3, there is only one use case implying one unique operation on the VFS:

- Download a video from a one-use URL.

The request is just the HTTP GET on the given URL. VFS have to control the coincidence between identity of the authentified A3 and the recipient of the URL.

3.3 Implementation

A reference implementation has been done at the IRIT[2]. This implementation is called Wanda. Wanda is fully functional, cross-platform and easy deployable. It is a Java Web Application which can be runned by Apache Tomcat. There are limits to this implementation: the scalability has not been tested and the administration interface is not really user friendly.

In Wanda, there are two kinds of identities:

- Users which should match a human being.
- A3s which are instances of A3s on a given machine.

There are three roles for clients, each has its own interface:

- Administrator: It is a user connected on the administration web interface.
- AT: It is an AT connected with the identity of its user on the AT API.
- A3: It is an A3 connected as an A3 on the A3 API.

3.3.1 Setup instructions

Java

Wanda needs Java's VM in a version >= 6. Wanda has only been tested on Sun's JVM.

Tomcat

Wanda needs Apache Tomcat in a version >= 7.0.

Wanda

Setup from sources You will need Bazaar VCS to download the sources.

- Open a terminal and browse to $ TOMCAT/webapps
- Run:

[2]Computer Science Laboratory of the Université Paul-Sabatier

```
bzr branch lp:wanda/release
mv release wanda
cd wanda
chmod u+x install.sh
./install.sh
```

- Follow instructions.

3.4 Conclusion and prospects

The existant implementation fulfills the requirements of the specification. Its use is integrated in the AT Ancolin and in all existing A3. The global system (AT, A3, A3S, VFS) meets the needs to uncouple human annotation from automatic annotation on different machines while keeping transparency for the user.

Since the first specification, needs have evolved. It could be interesting to extend the specification of the VFS. First example of new need: Some A3 have binary results, in particular video results. In this case, the A3 cannot send the result back to the AT directly. A video is far too heavy to be sent through XML. It is easy to imagine that the VFS could host the result temporary. Second example of new need: The role of the VFS is originally to allow to work with distant A3 by hosting the video for the A3. Now it seems interesting for users to use it as a library for themselves. This extension of the role of VFS implies many consequences on the specification (for example integrating metadata).

4 Common Type Library

In project Dicta-Sign, we use several ATs and A3s developped by different teams to fill different needs. For many reasons, there were no intention to fuse all these piece of software in one. However there was a need to be able to share data and more specifically annotation data. The delivrable D5.3 explored most of the aspects of annotations sharing and proposed some solutions for each problematic.

As a reminder, the choice has been done to use Annotation Graph extended for the occasion. The extension was there to fulfill different needs. This chapter interests two parts of the D5.3.

The first one is the need to help ATs to handle complex values in annotations. More precisely, it is the need to enable ATs to provide nice presentation and edition of values and to check user inputed values. This feature is close to automatic interpretation of free data. It is a really large problem which can be addressed at several different levels. At a low level, a kind of interpretation can be done by typing values. The solution adopted in D5.3 is to use *XML Schema* to express our types and extend annotation by typing tracks (or tier) with Schema types.

The second part is the need to uniformize as much as possible encoding of information. Said differently, it is the wish to share one encoding for data with a same meaning. The solution was to use the type notion introduced to make a common type library (CTL).

Since the publication of D5.3 (which was a specification), the solution has been integrated in all A3s developped and in (at least) the AT AnColin. This chapter details in a first section the ajustements done to the specification in regards to the experience of implementation. A second section comes next two details the actual content of the CTL.

4.1 Specification update

XML Schema is a language to describe XML grammars. It has to be able to, as much as possible, describe any XML dialect with its grammar, its constraints, its semantic, etc. The expressive power of XML Schema exceeds our needs, even in term of grammar. The problem with this is that a large part of the types expressible with XML Schema makes no sense in the context of annotations. This is the reason why an abstract model of types has been done. This model is designed for annotations.

4.1.1 Model definition

To describe types, the model defines 3 atomic types and 4 construction rules. The 3 atomic types are:

- `Integer`: An integer.
- `Float`: An approximation of a real number.
- `String`: A string.
- `Empty`: This type has an unique value which is `Null`.

The 4 construction rules are:

- `Copy`: Makes a copy of a type. Allow to makes synonyms.
  Example of definition: `Distance := Copy(Float).`
  Example of valid instance of `Distance`: `12.3`.
- `List`: Describes a list of values sharing a same type. The number of values is free.
  Example of definition: `Glose := List(String).`
  Example of valid instance of `Glose`: `["to drive","car"].`
- `Struct`: Describes a set of named and typed fields. In a structure, each field must have a unique name.
  Example of definition: `HeadPose := Struct(roll:Float, yaw:Float, pitch:Float).`
  Example of valid instance of `HeadPose`: `{roll:3.14, yaw:42.0, pitch:2.72}`.
- `Union`: Allow to take value between multiple types.
  Example of definition: `Head := Union(HeadPose,String).`
  Example of valid instance of `Distance`: `"profile"`.

About the type Empty The “Empty” type role is less obvious. It is intended to be a base to build a kind of binary (boolean) types. Tracks of this type (or a copy of it) describe when, in the video, a property is true. The most common case of use is in the context of communications AT-A3. Many A3 work without any “technical” parameters. In this case, there is at least one trivial parameter which should say what parts of the video the user want to be annotated. Such a track can be called “segment of interest”, all the information is contained in the presence of a segment itself and segment values have no meaning.

Since, the XML Schema is still the format used in communications and in the CTL. It is imperative to have an injective translation function from the abstract model to XML Schema. Indeed, it is the existence of such a function which ensure the ability to serialize and unserialize types in Schema without alterations. However, there is no bijections (respecting the meaning). Consequently, whatever the function choosed, there will be Schema definitions (which are valid in an XML Schema point of view) which are not expressible in the abstract data model.

4.1.2 Translation scheme

There are several reasons to fix the translation scheme in this document: It proves the existence of an injective scheme, it helps integration of types in ATs, etc. But, first of all, it defines the subset of XML Schema which is interpretable in abstract types. This subset has to be common to keep the interest of the serialization and unserialization.

In the scheme below, we use:

- X : The type defined.
- Y, Z : Types used for definition.
- X.name : Name of type X.

Copy `X := Copy(Y)` $\Longrightarrow$

```
<xsd:complexType name="X">
  <xsd:complexContent>
    <xsd:extension base="Y"/>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="X.name" type="X"/>
```

List `X := List(Y)` $\Longrightarrow$

```
<xsd:complexType name="X">
  <xsd:sequence>
    <xsd:element name="Y.name" type="Y" minOccurs="0" maxOccurs="unbounded"
        />
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="X.name" type="X"/>
```

Struct `X := Struct(FieldY:Y, FieldZ:Z)` $\Longrightarrow$

```
  <xsd:sequence>
    <xsd:element ref="Y" type="Y"/>
    <xsd:element ref="Z" type="Z"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="X.name" type="X"/>
```

Union `X := Union(Y, Z)` $\Longrightarrow$

```
<xsd:complexType name="X">
  <xsd:choice>
    <xsd:element ref="Y"/>
    <xsd:element ref="Z"/>
  </xsd:choice>
</xsd:complexType>
<xsd:element name="X.name" type="X"/>
```

Empty `X := Empty` $\Longrightarrow$

```
<xsd:complexType name="X" />
<xsd:element name="X.name" type="X" />
```

Atomic types

- `Integer` $\Longrightarrow$ `xsd:int`
- `Float` $\Longrightarrow$ `xsd:float`
- `String` $\Longrightarrow$ `xsd:string`

4.2 Content

The detail of the types defined in the CTL is the following:

- `Vector2D`: Encodes a 2 dimension vector. Without anything else, it cannot be drawed on video frames.
- `Point2D`: Encodes a 2 dimension point. It can be draw as a point on video frames.
- `BoundingBox`: Encodes bounding box with the upper left corner as a `Point2D` and the size as a `Vector2D`. It can be draw as a box on video frames.
- `List2DPoint`: Encodes a list of Point2D. It can be draw as a cloud of points on video frames.
- `Hamnosys`: Encodes Hamnosys description. It is a string which should be displaid with the hamnosys font.
- `Empty`: Is used to type a segment empty of data.

4.3 Conclusion and prospects

The typing system described in D5.3 and extended above and the CTL are already in use. It has been integrated in at least one AT (AnColin) and in all existing A3. AnColin makes a demonstration of how this system enable ATs to provide nice presentation, edition and verification.

In this system, it still lacks of a way to encode enumerated types. One solution could be to add a rule to create constants. An enumerated type could so be build with an union of constants.

The current content of the CTL match the needs of currently existing A3. It will be extended when required.

5 AnColin

The annotation tool AnColin has been enhanced to be able to process the new kind of data provided by the A3s. The modifications are based on D5.2 and D5.3 specifications. Mainly, it implements the secured communication with A3 and VFS, the Common Type Library management and the extended AG–XML structure as an annotation data file format. The update of AnColin is a deep rewriting of the original code. To do this, we decided to adapt the Graphical User Interface (GUI) of the programme, using the design pattern Model-View-Controller (MVC).

5.1 The Model:

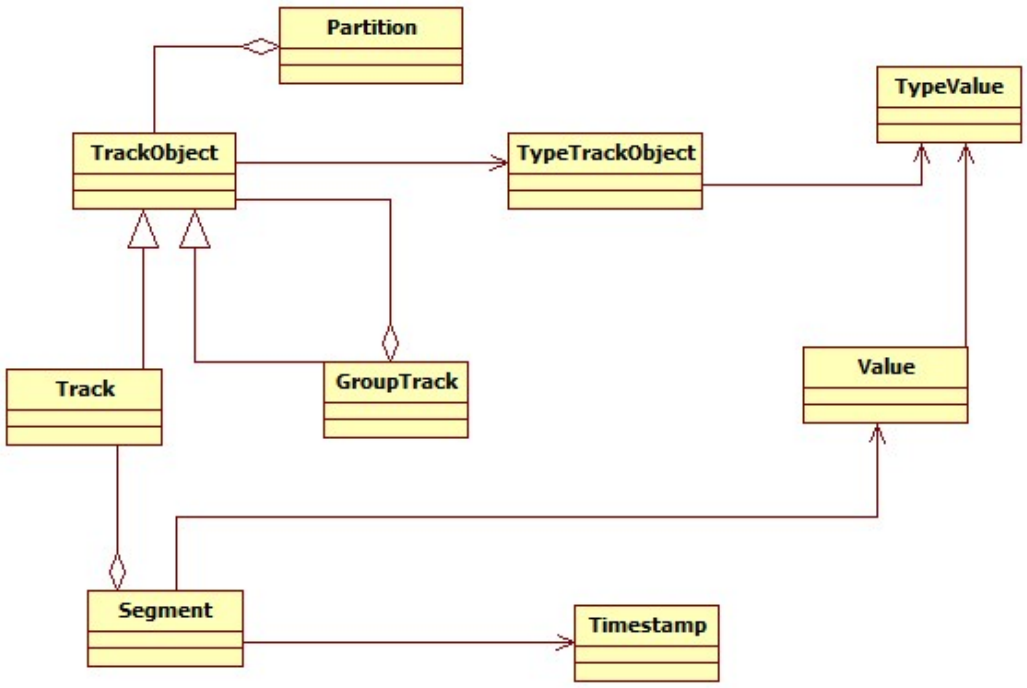


Figure 5.1: Model of hierarchy.

It is the same as in D5.1. There are four main classes in the model (fig. 5.1): Partition, GroupTrack, Track and Segment. A Partition is an aggregation of TrackObjects.

A TrackObjects can be either a GroupTrack or a Track. A GroupTrack is also an aggregation of TrackObjects. A Track has a collection of Segments. A Segment has a Type and a Value. The type of a segment is fixed by its Track. It is the segment which stores the information of annotations.

5.2 Types and values:

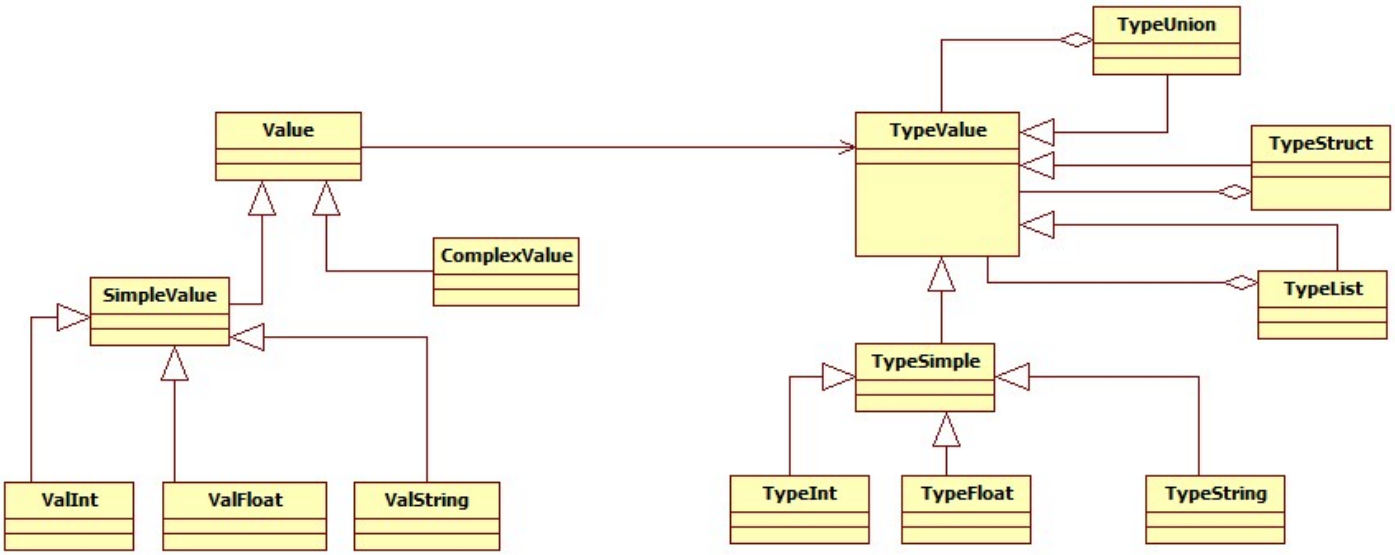


Figure 5.2: class diagram of types and values.

To enable enhanced annotations, AnColin implements the types of 4.1.1 (page 32). The type system is implemented as shown in figure 5.2. This type system allows users to create their own types as needed. Each track has a type, and each segment in a track has a value of the same type. Types and values edition is integrated to the GUI.

5.3 The View:

To each one of the four classes of the model (presented in 5.1) corresponds a class intended to display it. These classes are called views. Views are managed by controllers as shown in figure 5.3. Thanks to this principle, there can be several classes that display the same object in different ways.

5.4 The Controllers:

For each main class of the model, there is a controller (fig. 5.4). Each controller respects the same structure as the model. To keep models independant from views, a class Updatable has been introduced. All view controllers inherit from Updatable. This

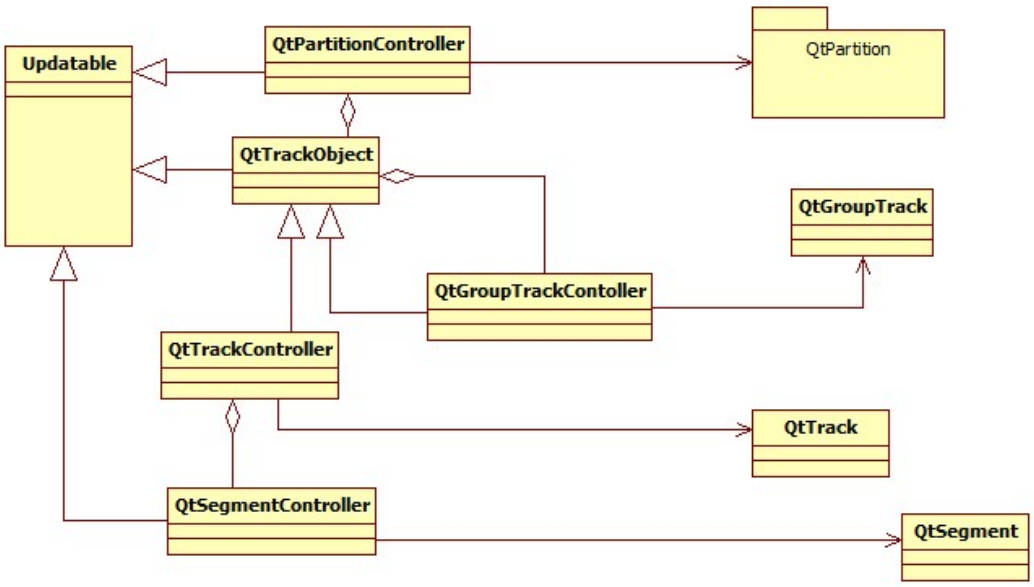


Figure 5.3: class diagram of the viewers.

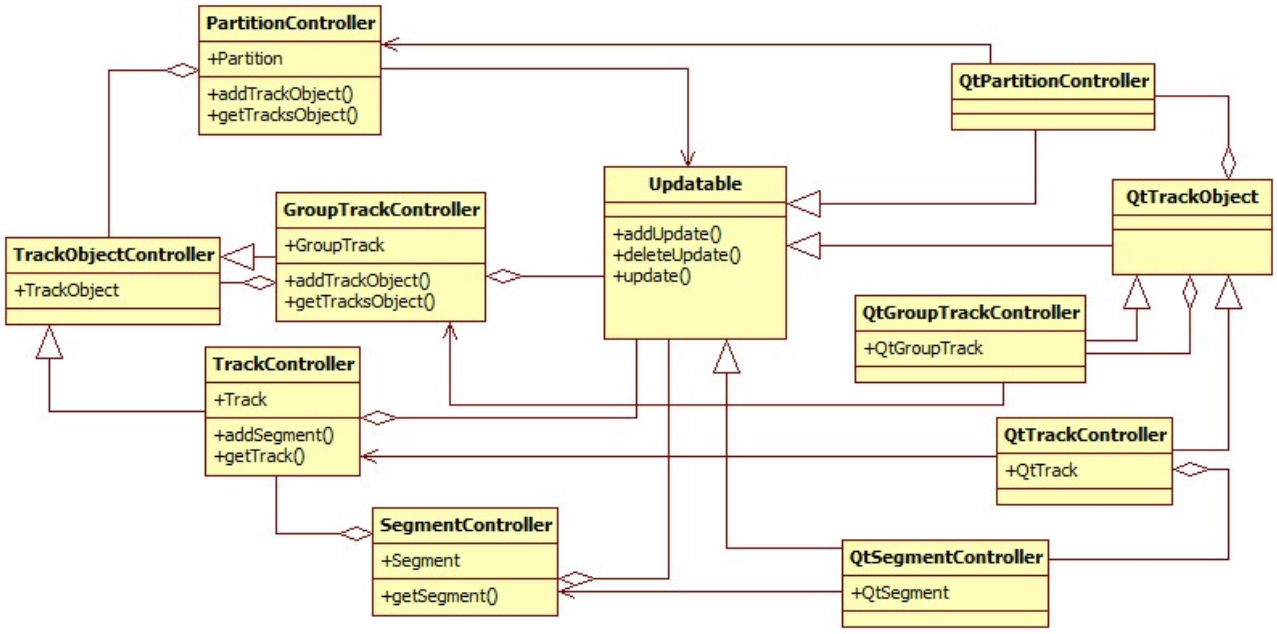


Figure 5.4: class diagram of the Controllers.

allows to add new views which will be refreshed everytime the model is updated. Examples of operations between the view and the model:

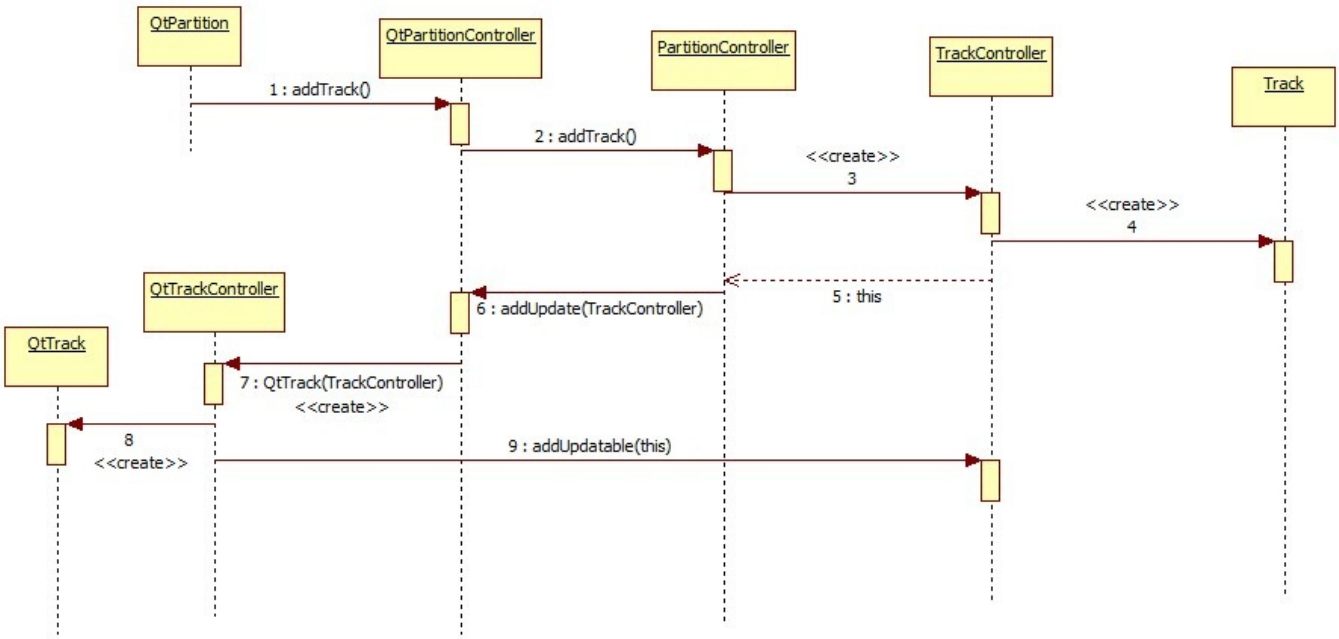


Figure 5.5: sequence diagram for addTrack.

5.4.1 addTrack:

When the user clicks to add a new Track, the event is reported to the GUI controller (fig. 5.5). The GUI controller calls the model controller by a direct link (a pointer in C++). Then, the model controller processes the information to create a TrackController. It is this TrackController which creates a Track model.
The PartitionController calls addUpdate() on all GUI controllers to add the track model. The controller of GUI create a QtTrackController that is creating a QtTrack in turn. QtTrackController keep a link with TrackController. And TrackController keep a link with all ”Updatable” that display the Track.

5.4.2 SetName:

When the user starts to change the name of a track, the GUI calls the method setName of the GUI controller which calls the method setName of the model controller (fig. 5.6). The model controller applies the change on the track and then calls the update method on all the Updatable.

5.4.3 Actions:

Save/Load:

To save an annotation, AnColin serializes a Partition to Xml (AG struture) and saves it in a file. To load an annotation, AnColin parses the Xml and creates a tracks’ hierarchy in a partition.

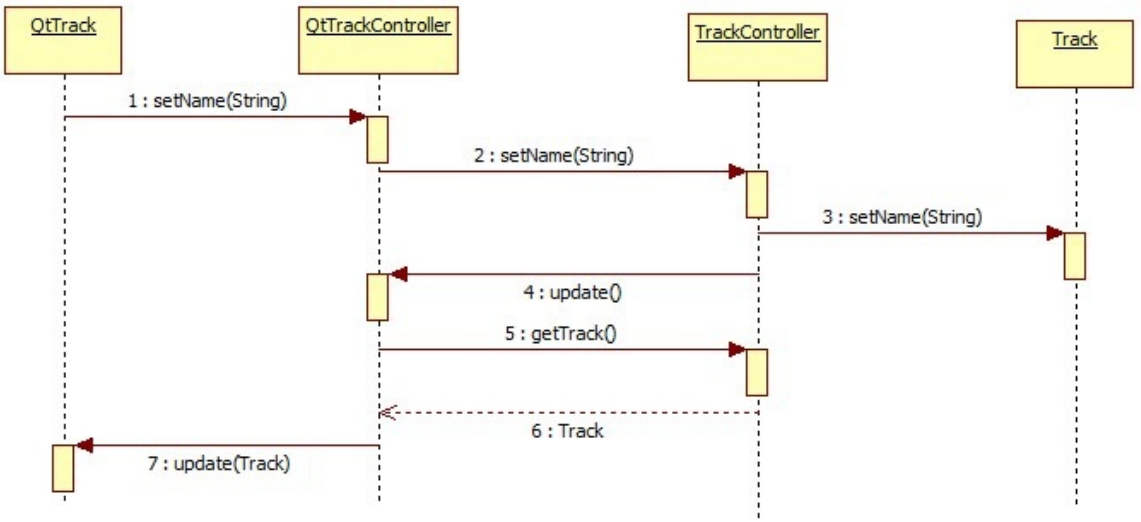


Figure 5.6: sequence diagram for setName.

Cut/Copy/Paste:

Clipboard functionalities use the serialization presented above. The only difference is that it serialize only a track or a segment instead of the whole partition. By this way, AnColin can share annotations with other softwares

5.5 Conclusion and prospects

The AnColin annotation tool has been deeply enhanced during the project Dicta-Sign. Nevertheless, there is still to do to get a stable version. It can be considered now as demonstration prototype to evaluate the A3 distributed architecture. It include all the enhancement that have been specified during the project: secured communication with A3, A3S and VFS ; API of A3 integrated ; VFS communication management ; CTL management ; and the user interface of AnColin has been rebuilted to integrate all this new features.

Conclusion

In this Deliverable, we have described the progress made in the specification and the development of tools that can help annotation task. We propose a Distributed System Architecture that includes security mechanism to ensure annotation data, video files and programs code integrity. All the specifications of API for A3 and the source code of A3 template and A3S are publicly availables (VCS server opened soon). The program code of the VFS and the AnColin annotation tool are opensource and available for the community. A VFS, an A3S and all the A3 that have been developped during the project Dicta-Sign, will be available on our server in the near futur. All this material makes possible to integrate A3 features in others annotation tools. This allows people who wish to give the potential of their program to the community but not the code of the program to do it.

Bibliography

[1] Boutora (L.) et Braffort (A.). – Degels. *DEfi Geste Langue des Signes. Corpus DEGELS1. Corpus ID oai:crdo.fr:crdo000767 : Video en LSF, informateur A*, 2011.

[2] Gianni (F.), Collet (C.) et Dalle (P.). – Robust tracking for processing of videos of communication's gestures. *Gesture-Based Human-Computer Interaction and Simulation*, 2009, pp. 93–101.

[3] Gonzalez (M.) et Collet (C.). – Head tracking and hand segmentation during hand over face occlusion in sign language. *In : International Workshop on Sign, Gesture, and Activity (SGA) in Conjunction with ECCV*, éd. par Pub. (Springer Verlag).

[4] Gonzalez (M.) et Collet (C.). – Robust body parts tracking using particle filter and dynamic template. *In : IEEE Int. Conf. on Image Processing*. – Belgum, 2011.

[5] Isard (M.) et Blake (A.). – Condensation-conditional density propagation for visual tracking. *International journal of computer vision*, vol. 29, n° 1, 1998, pp. 5–28.

[6] Lefebvre-Albaret (F). – *Traitement automatique de vidéos en LSF, modélisation et exploitation des contraintes phonologiques du mouvement,*. – Phd thesis, University of Toulouse, October 2010, 95–132p.

[7] LS-COLIN. – *13:08 - 15:15 le 11 septembre 2001 par Nasredine Chab, http://corpusdelaparole.in2p3.fr/spip.php?article30&ldf_id=oai:crdo.vjf.cnrs.fr:crdo-FSL-CUC020_SOUND*, 2002.

[8] MacCormick (J.) et Blake (A.). – A probabilistic exclusion principle for tracking multiple objects. *International Journal of Computer Vision*, vol. 39, n° 1, 2000, pp. 57–71.

[9] Ong (E J), Lan (Y), Theobald (B J), Harvey (R) et Bowden (R). – Robust facial feature tracking using selected multi-resolution linear predictors. *In : IEEE International Conference Computer Vision ICCV09, Kyoto, Japan.* IEEE, pp. 1483–1490.