Improvements of the Distributed Architecture for Assisted Annotation of video corpora

Rémi Dubot, Christophe Collet

IRIT
Université de Toulouse
remi.dubot@irit.fr, christophe.collet@irit.fr

Abstract

Progress on automatic annotation looks attractive for the research on sign languages. Unfortunately, such tools are not easy to deploy or share. We propose a solution to uncouple the annotation software from the automatic processing module. Such a solution requires many developements: design of a network stack supporting the architecture, production of a video server handling trust policies, standardization of annotation encoding. In this article, we detail the choices made to implement this architecture.

Keywords: Annotation, Sign Languages, Automatic Annotation, Distributed Architecture

Linguists need annotations of sign language video corpora. Video corpus annotations are mainly done manually on Annotation Tools (ATs). This work is really time-consuming and frequently repetitive. To support this claim, we can consider the literature regarding annotation tools of the last couple of years. There are two remarkable trends: collaborative annotation and automatic annotation. Works on collaborative annotation focus on the workflow (Hofmann et al., 2009; Brugman et al., 2004). About automatic annotation, we had a look on ELAN and ANVIL. ELAN (Auer et al., 2010) provides a plug-in interface, called "Recognizer API", for automatic processing. A trick is given to run the module on a different machine through the network but it is restricted to local network. ANVIL (Kipp, 2010) provides different kinds of processing based on co-occurrence measurements on annotations and motion characteristics extraction from motion capture.

As far as we know, most of automatic tools are developed as standalone prototyping software, finally remaining as in-house systems that continue to be used only by the team that have developed them. Barriers to their use in production are numerous: "temporary" privacy of sources and executables, deployment difficulty (run only in a specific environment, need a powerful machine), integration constraints (incompatible programming languages, running on different OS, etc.). We think that the architecture we have developed greatly simplifies this integration.

We have designed an open and distributed system for the integration of automatic processing in annotation tools. We consider that it will encourage collaboration initiatives. Our idea is to let every part of the system do what it does best. Each automatic processing tool will run on the module fitting the best. Videos are hosted by a dedicated server. And finally, manual annotation will be done on the annotator's computer.

What we provide is the solution to make all this working together. We see three parts in this problem:

- Service discovery,
- Security,
- Compatibility.

This article first shows the global architecture and the agents involved. Then, it details stage by stage the stack of protocols. Finally, it presents, as an evaluation, the automatic annotation modules already done and a preview of the potential offered.

1. Architecture

An early version of this architecture was presented in (Collet et al., 2010). At that time, the architecture was only partially implemented. The finalization leads us to make several adjustments and the new developed architecture is presented now. The system is based on four types of agents, from which three have been already presented in the introduction:

- The first type of agent is Annotation Tools (AT). Any annotation tool can be extended to allow its integration in the architecture. We already provide an annotation tool: AnColin.
- The second type of agent is Automatic Annotation Assistants (A^3).
- The third type of agent is the server hosting the videos (VFS: Video-File Server)
- The last type agent is a service directory, it is in charge to reference A^3s. When an annotator wants to use an automatic process, his AT will retrieve the list of all currently available functions from the A^3S.

These four agents are represented in figure 1.

We call an instance of this architecture a Distributed Annotation System (DAS).

2. Stack

2.1. Network

As agents are not necessarily located in the same local network, they communicate over the Internet. One of our objectives is to facilitate the deployment of automatic processing, this implies to let the implementation of the processing part as free as possible from constraints coming from the architecture. The consequence is a high heterogeneity

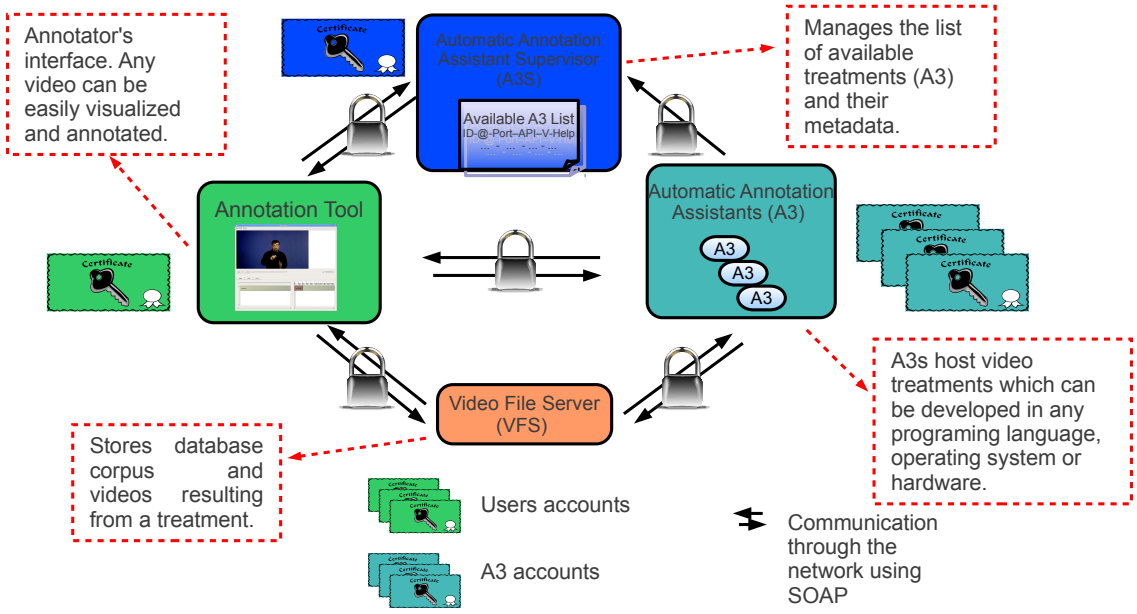

Figure 1: Overview of the system

in terms of programming language, operating system, etc. To insure the low level compatibility between elements we use SOAP (W3C, 2007). SOAP is an open source, multi-platform middle-ware. It uses the HTTP protocol for communications which is important as it allows to pass through proxies. Because some information traveling on the network might be confidential, a TLS (former SSL) layer is added under SOAP. This layer encrypts all the communications. We will see below that TLS provides us several more services.

The Internet, SOAP and TLS are enough to have flexible and secure end-to-end communication, now we have to look at how agents find one another on the network. An instance of the architecture must have one VFS and one A^3S. It is the solid part of the architecture. Instances of the two other agents, ATs and A^3s, have to be configured to use a VFS and an A^3S. Therefore the VFS and the A^3S must have static addresses. At launch, A^3s provide their status and their network addresses to the A^3S. ATs get the A^3s' addresses with their specifications when it retrieves the list of available functions. This sequence is summarized in the figure 2.

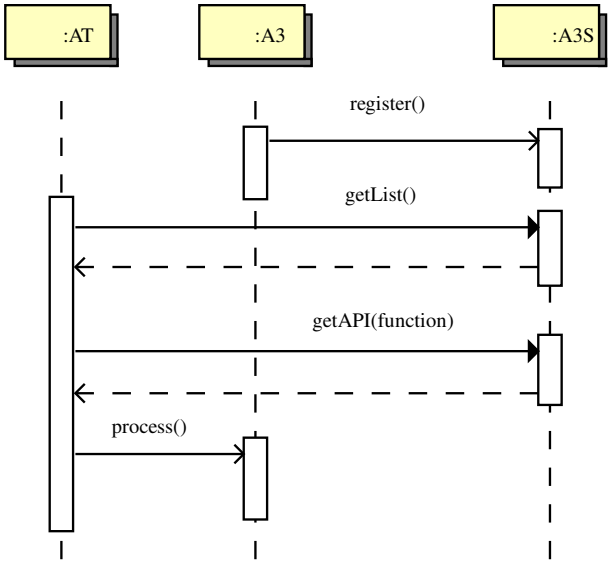

Figure 2: A^3S working sequence

2.2. Authentication

The authentication, in a communication, refers to the mean of identifying the end-users.

Most of our authorization/trust policy relies on identities consequently we must have a solution to identify agents.

On computers, there is a common solution for authentication which makes use of asymmetric cryptography. Concretely, each agent which has to identify itself must have a certificate, signed by a Certification Authority, which links its identity (name, email, etc.) to its public key. Such certificates are called X.509 certificates. The TLS protocol provides an authentication service using X.509 certificates.

Each agent must have its own certificate. Individual certificates are shown as keys in figure 1. The complete task of authentication is done by TLS.

Signed certificates are not necessarily free, in most cases Certification Authorities sell this service at high costs.

However, there are two low cost solutions:

- When many sites are involved, we recommend the use of certificates focusing only on the email address which are generally free.
- When one or few sites are involved, it is possible to setup a local Certification Authority.

2.3. Authorization

The authorization refers to rights policy, agents having the possibility of knowingly choose whether to serve another agent or not.
Videos are a sensitive point, mainly because they are under restrictive image rights. Consequently, the VFS has to manage trust in AT users and A^3s regarding the videos. Our implementation of the VFS allows a fine rights management for user access to videos and uses a mechanism of jobs to limit to its minimum the data given to A^3s. This is symbolized by certificate collections under the VFS in figure 1.
A^3S and A^3s might manage authorization too, but currently we do not see any reasons not to let them publicly available. Finally, as ATs do not provide any service to other agents, there is no authorization to manage on this side.

2.4. Encoding

We use Annotation Graphs (Bird and Liberman, 2001) as our standard annotation structure for exchanges. However, the AG model was designed to be as general as possible and consequently has gaps. To overcome this, we have made an extension of AGs. This extension is backward compatible with AGs. This means all ATs handling AGs (ELAN, ANVIL, etc.) are already able to deal with the files generated by an A^3. We will not detail all the features brought by this extension, only the two directly linked to the distributed architecture. First, we add the concepts of a track (identical to the concept of a tier) and a group of tracks forming a hierarchy (a tree or a lattice). Second, we add the concept of a type. Here, we talk about types for the content of annotations (the values inside segments). The need of track hierarchy is self-evident. We are going to address in detail the question of why and how to type the values.
Untill now, nothing has been done in the field of annotation encoding to encourage data homogeneity. Consequently, there is a high heterogeneity in data encoding. It leads to problems like "In these vectors, which component is the horizontal one?" or "How was this bounding box encoded? two corners? a corner and a size?", etc. And while it is annoying for humans, it is really problematic for automatic tools. The compatibility between A^3s depends on the homogeneity of data. An A^3 producing bounding boxes and an other processing bounding boxes must share the same encoding, even if there were developed fully independently. This is a constraint to allow users to process the results of the first A^3 with the second one. The homogeneity is also necessary to have a smart display and edition of annotations. Our solution is to affect types (as in computer languages) to annotation data. The type system we present below has been able to handle all annotations we use in our team currently.
We split the task in two parts:

- Making a type system able to deal with all kind of data appearing in annotations.
- Making types for all the common data and collect them all in a library.

2.4.1. Types

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 is intended to be a base to build a kind of boolean types. It is used when the information is contained in segments' positions and segments' values are meaningless.

The 4 construction rules are:

- `Copy`: Makes a copy of a type. Allows to makes synonyms.
  Example of definition:
  `Distance:= Copy(Float).`
  Example of valid instance of `Distance`:
  `12.3`.
- `List`: Describes a list of values sharing the same type.
  Example of definition:
  `Glose:= List(String).`
  Example of valid instance of `Glose`:
  `["to drive","car"].`
- `Struct`: Describes a set of named and typed fields.
  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`: Allows to take values between multiple types.
  Example of definition:
  `Head:= Union(HeadPose,String).`
  Example of valid instance of `Head`:
  `"profile".`
- `Constant`: Allows to make a constant of a given type.
  Mainly used coupled with `Union` for enumerations.
  Example of definition:
  `hello:= Constant(String,"hello").`
  Example of enumeration `Head`:
  `voc:= Union(Constant(String,"hello"),Constant`

The abstract type system may be extended if needed. But we want the type encoding to stay stable. The format we chose is XML Schema (Fallside and Walmsley, 2004). XML Schema is far more generic than our type system and allows many extensions of the abstract system.
We give translation schemes between abstract type system and XML Schema.

2.4.2. Common Type Library

To standardize the encoding of annotations, we maintain a standard type library which takes the form of an XML Schema Definition (XSD). It is intended to be online.
The detail of the current content of the CTL is the following:

- `Vector2D`.
- `Point2D`.
- `BoundingBox`: Encodes bounding box with the upper left corner as a `Point2D` and the size as a `Vector2D`.
- `List2DPoint`: Encodes a list of Point2D.
- `Hamnosys`: Encodes Hamnosys description.
- `Empty`.

This list will grow quickly as soon as A^3s will be used in production.

3. Use

We maintain our own AT, AnColin, which fully integrates this architecture. Up to now, four A^3s has been developed: Signing detection, sign segmentation, body part tracker (Gonzalez and Collet, 2011) and facial feature tracker.
The first A^3 concerns the detection of signing activity. In a corpus of dialogue, the signers take turns to talk. This means that there are times where one of the participant is just listening. This A^3 detects where in the sequence the informant is actually signing. The core of this A^3 has been provided by Helen Cooper from University of Surrey.
The second A^3 regards the segmentation of continuous sign language. It uses hand movement analysis to detect limits between signs.
The third A^3 tracks hands and head using a particle filter based approach. It robustly handles hand-over-head occlusion using a template before occlusion (Gonzalez and Collet, 2011).
The last A^3 uses a small quantity of hand labelled images to learn a set of facial feature trackers, which can be applied across segments of video. The tracker is based on the linear predictor flock method as described in (Ong and Bowden, 2011).
A^3s produce a great amount of technical data. Rendered as text, data is difficult to visualize and to edit for humans. The use of automatic processing induces a new need for visualization and editing tools. The type feature is the base for such features. We have introduced two kind of tools to AnColin:

- Head-Up Display (HUD) modules which are in charge to display and edit annotations directly on the video. We currently have two modules: one for bounding boxes, the other for cloud of points.
- Segment display and edition modules which are in charge to display and edit annotations on the tracks. We currently have one generic module able to generate forms from types.

As an example of possible applications, we are able to chain A^3s and finally display results with HUDs, making workflows. Examples of workflows already possible:

- $SigningDetection \rightarrow FaceTracking \rightarrow HUD(BoundingBox)$
- $SigningDetection \rightarrow FaceFeatureTracking \rightarrow HUD(PointCloud)$

4. Conclusion

We have introduced an architecture for distributed annotations and the protocol stack of communications in this architecture. Everything presented here has been completely implemented. On the other hand, many parts are not stable enough to be used in production. Some elements which were written as prototypes are being rewritten and others are being stabilized. However we have experimented with using the architecture. The list of A^3s is currently short but presenting state-of-the-art tools and expected to grow.

5. References

E. Auer, A. Russel, H. Sloetjes, P. Wittenburg, O. Schreer, S. Masnieri, D. Schneider, and S. Tschöpel. 2010. ELAN as flexible annotation framework for sound and image processing detectors. In *International workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies (LREC)*, Valletta, Malte.

S. Bird and M. Liberman. 2001. A formal framework for linguistic annotation. *Speech Commun.*, 33(1-2):23–60.

H. Brugman, O. Crasborn, and A. Russel. 2004. Collaborative annotation of sign language data with peer-to-peer technology. In *4th International Conference on Language Resources and Evaluation*, pages 213–216.

C. Collet, M. Gonzalez, and F. Milachon. 2010. Distributed System Architecture for Assisted Annotation of Video Corpora. In *International workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies (LREC)*, pages 49–52, Valletta, Malte, May. European Language Resources Association (ELRA).

D. C. Fallside and P. Walmsley. 2004. XML Schema: W3C Recommendation. http://www.w3.org/TR/schema.

M. Gonzalez and C. Collet. 2011. Robust body parts tracking using particle filter and dynamic template. In *IEEE Int. Conf. on Image Processing*, pages 537–540, Brussels, Belgium.

C. Hofmann, N. Hollender, and D. Fellner. 2009. Workflow-based architecture for collaborative video annotation. *Online Communities and Social Computing*, pages 33–42.

M. Kipp. 2010. Multimedia annotation, querying and analysis in ANVIL. *Multimedia Information Extraction*, 19.

E.-J. Ong and R. Bowden. 2011. Robust Facial Feature Tracking Using Shape-Constrained Multiresolution-Selected Linear Predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:1844–1859.

W3C. 2007. Recommendation of the W3C. http://www.w3.org/TR/soap/.