AaS Protocol – AaSP

Rebekka Hubert Michael Staniek Simon Will

December 6, 2016

1 Introduction[1]

1.1 Purpose

The Annotation as Search Protocol (AaSP) is an application-level protocol for communication of an AaS server and an AaS client. The purpose of this communication is described in the AaS specification.

1.2 Protocol compliance

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.[2]

An implementation of the AaSP is called compliant if and only if it satisfies all of the requirements marked by the key words "MUST", "SHALL" or "REQUIRED". Furthermore, implementations are encouraged to comply to the recommendations marked by the keywords "SHOULD" and "RECOMMENDED" if they are suitable to the respective application.

1.3 Overall Operation

The client initiates the AaSP communication by sending a message of the type *request* to the server. The server then build a forest of annotated trees from the request, generates a question, and reponds to the client by sending a message of the type *question*. The client answers the question as true or false and sends an *answer* message to the server.

When receiving an *answer*, the server filters its forest to only retain the trees which satisfy the given answer. If only one tree remains, the server sends the remaining tree to the client in a *solution* message. Otherwise it sends a *question* message again.

[1] This section heavily borrows from the Introduction to the specification of the HTTP as given in RFC 2616. `https://www.w3.org/Protocols/rfc2616/rfc2616-sec1.html`

[2] `https://www.ietf.org/rfc/rfc2119.txt`

At any time, the client may send an *abort* message to end the communication prematurely, or it may send an *undo* message to revoke one or more of the answers given previously.

If the server encounters an unexpected situation it may send an *error* message to the client.

2 AaSP Messages

2.1 Message format

AaSP messages are UTF-8-encoded[3] JSON-formatted[4] strings. Every message must consist of exactly one JSON object, which must contain a pair with the key `"type"` specifying the type of the message. The value of the `"type"` key shall be a string.

All other pairs in the message object depend on the type of the message. The available types are:

- `"request"` (sent by the client)
- `"answer"` (sent by the client)
- `"abort"` (sent by the client)
- `"undo"` (sent by the client)
- `"question"` (sent by the server)
- `"solution"` (sent by the server)
- `"error"` (sent by the server)

2.2 Message types

The following sections describe the purpose and semantics of the different message types, and specify what pairs a message has to contain in addition to the pair identified by the `"type"` key, which has to be present in every message.

2.2.1 Request

The client sends this message to initiate the communication between server and client, and the content of this message should be used by the server to build a forest of trees. However, there are two ways for the client to provide the necessary information: Either it supplies the whole forest or it supplies a sentence and tells the server to parse the sentence itself.

In the case of supplying the whole forest, the client shall provide two additional pairs:

[3] https://tools.ietf.org/html/rfc3629
[4] https://tools.ietf.org/html/rfc7159

`"use_forest"` A representation of the forest the server should use. The value should be a string. Servers may only accept different values – such as an array – of strings if the representation as a single string is not suitable.

`"forest_format"` A string specifying the format the forest is provided as. Servers should at least accept the format `"conllu"`[5].

In the case of supplying only the sentence, the client shall provide a `"parse_sentence"` field and may also provide a `"parser"` field:

`"parse_sentence"` A string representing the sentence the server is to parse.

`"parser"` **Optional.** A string specifying which parser the server is to use. If no `"parser"` is specified, servers should use their default parser.

2.2.2 Answer

The client sends this message after receiving a `"question"` message from the server, and the content of this message shall be used by the server to filter the forest for trees satisfying the condition inferred from the answer.

The client shall provide two additional pairs:

`"question"` The question wo which this message provides an answer. The value should be a question object.

`"answer"` A boolean value providing an answer to the question.

2.2.3 Abort

The client sends this message if it wants to terminate the annotation helping process, and the content of this message shall be used by the server to generate a mock `"solution"` to return to the client. The client can request the tree the server judges as being the best candidate for the correct tree or an incomplete tree containing only the edges that are present in all trees.

The client shall provide one additional pair:

`"wanted"` A string specifying what tree the client wants to have. Servers should support the options `"wanted"` for the best guess at the correct tree and `"fixed"` for the incomplete tree containing only fixed edges. Servers may, however, support additional options of their own.

2.2.4 Undo

The client sends this message if it wants to revoke previously sent `"answer"` messages, and the content of this message shall be used by the server to restore the state of the forest as it was before the answers had been processed.

The client can provide one additional pair:

[5] http://universaldependencies.org/docs/format.html

`"answers"` **Optional.** An integer value between zero and the number of given answers (both inclusive) specifying how many answers are revoked. If `"answers"` is not specified, servers should assume `1` as the default value.

2.2.5 Question

The server sends this message if there is more than one tree left in the forest and the client has not sent an `"abort"` message, and the content of this message shall be used by the client to provide the question to the user and prompt them for an answer. The question shall also provide futher information about the state of the annotation process for the client to display to the user.

The server shall provide four additional pairs:

`"sentence"` A string representing the sentence that is being annotated.

`"question"` A question object representing the question the user is to be asked.

`"remaining_trees"` An integer value specifying how many trees currently remain in the forest.

`"fixed_edges"` A tree object containing the fixed edges, i.e. the edges present in every tree in the forest.

2.2.6 Solution

The server sends this message either if only one tree remains in the forest or the client has sent an `"abort"` message. This message is supposed to be the last message in the annoation helping process, but the client shall still be able to send an `"undo"` message revoking previous answers leading to a reentry of the question-answer-cycle.

The server shall provide two additional pairs:

`"solution"` A tree object.

`"solution_type"` A string specifying what kind of tree is sent as the solution. Servers should support the values `"real"` for a tree which is the only remaining tree in the forest, `"best"` for a tree which is the server's best guess at the correct tree, and `"fixed"` for an incomplete tree only containing the edges present in every tree in the forest.

2.2.7 Error

The server sends this message if it encounters an unexpected situation.

The server shall provide two additional pairs:

`"error_message"` A string containing a human-readable message telling the user what went wrong.

`"recommendation"` A string specifying how the client should handle the error. Possible values are `"abort"` for recommending to abort the annotation helping process and `"retry"` for recommending to send the previous message again.

2.3 Custom objects

There are two custom JSON objects that are used in messages of more than one type. These are the question object and the tree object which are described in the following sections:

2.3.1 Question object

A question object represents a question the client can put to the user. There are two types of question objects, which are implicitly identified by the keys of the pairs present in the object.

The first type is the node question object. It asks if a given label is correct for a given node. A question object contains the following pairs:

`"node"` A string of the form `"token-index"` specifying the name of the token and the position in the sentence to identify the node.

`"label"` A string specifying a label.

`"label_type"` A string specifying the type of the label. Possible values are `"pos"` or `"morph"`.

The second type is the relation question object. It asks if a dependent node and a head node are connected by a specific relation.

`"head"` A string of the form `"token-index"` specifying the name of the token and the position in the sentence to identify the head node.

`"dependent"` A string of the form `"token-index"` specifying the name of the token and the position in the sentence to identify the dependent node.

`"relation"` A string specifying a relation.

`"relation_type"` A string specifying the type of the relation. A possible value is `"deprel"`.

2.3.2 Tree object

A tree object represents a complete tree or an incomplete tree. It contains two pairs:

`"tree_format"` A string specifying the format the tree represenation adheres to. Servers and clients should at least support the format `"conllu"`[6].

`"nodes"` An array of arrays. Each inner array represents a node. Each element of a node array is a string representing one of the values of the respective CoNLL format.

[6] http://universaldependencies.org/docs/format.html