Trusted Contract Observation HTTP/TLS/JSON

Interface Design Description

Service ID: *"trusted-contract-observation"*

Abstract
This document describes the HTTP/TLS/JSON variant of a service useful for getting copies of negotiation sessions, as managed by the *Trusted Contract Negotiation* service.

ARTEMIS Innovation Pilot Project: Arrowhead
THEME [SP1-JTI-ARTEMIS-2012-AIPP4 SP1-JTI-ARTEMIS-2012-AIPP6]
[Production and Energy System Automation Intelligent-Built environment and urban infrastructure for sustainable and friendly cities]

Contents

1 Overview

This document describes the HTTP/TLS/JSON variant of the Trusted Contract Observation (TrCoOb) service, which allows for arbitrary Eclipse Arrowhead systems to obtain copies of the candidates part of negotiation sessions, as they are maintained by systems providing the Trusted Contract Negotiation (TrCoNe) sevice.

Readers of this document are assumed to be familiar with the TrCoNe and Contract Negotiation services. For more information about those services, please refer to their respective service description documents.

The rest of this document describes how to realize the TrCoOb service using HTTP [1], TLS [2] and JSON [3], both in terms of its functions (Section 2) and its information model (Section 3).

2 Service Interface

This section describes the one function that must be exposed by TrCoOb services. In particular, the below subsection first names the HTTP method and path used to call the function, after which it names an abstract function from the TrCoOb service description document, output type, as well as errors that can be thrown. The function is expected to respond with HTTP status code `200 OK` for all successful calls.

2.1 GET /trusted-contract-observation/negotiations ?name1={*name1*} &name2={*name2*} &negotiationId={*negotiationId*}

Interface: GetNegotiationByNamesAndId
Input: NegotiationRequest
Output: Negotiation

Called to acquire the candidate of some identified negotiation session, as well as some additional details. Examples of a valid invocation and response are given in Listings 1 and 2.

```
GET /trusted-contract-observation/negotiations?name1=A&name2=B&negotiationId=1593831 HTTP/1.1
Accept: application/json
```

Listing 1: A GetNegotiationByNamesAndId request. Some relevant headers have been omitted for the sake of brevity.

```
HTTP/1.1 200 OK

{
  "negotiationId": 4543,
  "offer": {
    "offerorName": "A",
    "receiverName": "B",
    "validAfter": "2019-12-06T12:07:22.14Z",
    "validUntil": "2019-12-06T14:07:22.14Z",
    "contracts": [{
      "templateName": "SimplePurchase",
      "arguments": {
        "buyer": "A",
        "seller": "B",
        "quantity": "12",
        "articleNumber": "ABC-123",
        "price": "3750"
      }
    }],
    "offeredAt":"2019-12-14T12:23:14.142Z"
  },
  "status": "OFFERING"
}
```

Listing 2: A GetNegotiationByNamesAndId request response, in this case being a Offer. Any Candidate variant could have been sent as response. Some relevant HTTP headers have been omitted for the sake of brevity.

3 Information Model

Here, all data objects that can be part of TrCoOb service calls are listed in alphabetic order. Note that each subsection, which describes one type of object, begins either with the *struct* or *enum* keywords. The former is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types, while the latter is used to signify that a JSON String matching any of the listed variants. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.7, which are used to represent things like hashes, identifiers and texts.

3.1 struct Argument

A concrete Contract parameter value. In other words, this data structure identifies a variable parameter in a template and provides a concrete parameter value. The type of the *Value* field should be dynamic enough to be able to take on at least a few concrete types, such as integer, float, string, list, associative array, and so on. Conceptually, the parameter specification in the template that matches the *Name* field could contain additional type information, such as range restrictions, required list lengths, or regular expressions.

As the *"arguments"* field of the Contract type is a JSON Object, which maps String names to String values, there is no need for a dedicated *Argument* type. This type exists only implicitly and is documented here only for the sake of completeness.

3.2 struct Contract

A parameterized reference to a *template*, representing certain rights and obligations accepted by two parties. In other words, a contract derives its meaning from both the template it refers to and the arguments it contains. Note that contract objects only become binding if they refer correctly to legally valid templates and become part of Acceptances.

Object Field	Value Type	Description
”templateName”	Name	Name of invoked *template*.
”arguments”	Object<String>	Arguments matching *parameters* defined in *template*.

3.3 struct Offer

A concrete offer of Contracts intended for a specific receiver. Note that in contrast to the CoNe service, a Name is used instead of a Hash to represent the offeror and receiver.

Object Field	Value Type	Description
”offerorName”	Name	Offeror’s common name.
”receiverName”	Name	Receiver’s common name.
”validAfter”	DateTime	The time after which the offer may be accepted.
”validUntil”	DateTime	The time after which the offer may no longer be accepted.
”contracts”	Array<Contract>	Offered contracts.
”offeredAt”	DateTime	The time at which this offer was made.

3.4 struct Negotiation

Data extracted from a negotiation session.

Field	Type	Description
”negotiationId”	RandomID	Negotiation session identifier.
”offer”	Offer	Last offer to become part of the session.
”status”	NegotiationStatus	Enumerates the state of the negotiation session.

3.5 struct NegotiationRequest

Identifies a requested Negotiation. As the fields of this type occur only as query parameters in the GetNegotiationByNamesAndId function, there is no need for it to representable as a JSON Object. This subsection exsits only for the sake of completeness.

3.6 enum NegotiationStatus

Identifies the status of a Negotiation.

Variant	Description
”OFFERING”	The last session offer can currently be replaced, accepted or rejected.
”ACCEPTED”	The last session offer has been accepted and the session closed.
”REJECTED”	The last session offer has been rejected and the session closed.
”EXPIRED”	The negoitation session has expired and is closed.

3.7 Primitives

As all messages are encoded using the JSON format [3], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information. Furthermore, the Object and Array types are given optional generic type parameters, which are used in this document to signify when pair values or elements are expected to conform to certain types.

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [4], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

With these primitives now available, we proceed to define all the types specified in the CoNeST service description document without a direct equivalent among the JSON types. Concretely, we define the CoNeST primitives either as *aliases* or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section. The types are listed by name in alphabetical order.

3.7.1 alias DateTime = String

Pinpoints a moment in type by providing a formatted string that conforms to the RFC 3339 specification [5] (ISO 8601). Naively, the format could expressed as ”YYYY-MM-DDTHH:MM:SS.sssZ”, where ”YYYY” denotes year (4 digits), ”MM” denotes month starting from 01, ”DD” denotes day starting from 01, ”HH” denotes hour in the 24-hour format (00-23), ”MM” denotes minute (00-59), ”SS” denotes second (00-59) and ”sss” denotes second fractions (0-999). ”T” is used as separator between the date and the time, while ”Z” denotes the UTC time zone. At least three fraction digits should be used, which gives millisecond precision. An example of a valid date/time string is ”2019-09-19T15:20:50.521Z”. Other forms or variants, including the use of other time zones, is adviced against.

3.7.2 alias Name = String

A String that is meant to be short (less than a few tens of characters) and both human and machine-readable.

3.7.3 alias RandomID = Number

An integer Number, originally chosen from a secure source of random numbers. When new RandomIDs are created, they must be ensured not to conflict with any relevant existing random numbers.

4 References

[1] R. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,” RFC 7230, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7230

[2] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC8446

[3] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[4] M. Cowlishaw, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, July 2019. [Online]. Available: https://doi.org/10.1109/IEEESTD.2019.8766229

[5] “Date and Time on the Internet: Timestamps,” RFC 3339, 2002, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC3339

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

5.2 Quality Assurance

No.	Date	Version	Approved by
1			