Contract Negotiation

Service Description

Service ID: *"contract-negotiation"*

Abstract
This document describes an abstract service, which, if implemented by an application system, can be used to negotiate about and enter into legally binding contracts with other systems implementing the same abstract service, potentially located in other local clouds and being owned by other stakeholders.

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 an abstract Eclipse Arrowhead service meant to enable distinct parties to digitalize their contractual interactions. Examples of such interactions could be taking on the obligation to deliver a good, negotiating insurance, agreeing to pay by invoice in exchange for access to a particular Eclipse Arrowhead service, among many other. The advantages of digital contractual interactions include

1. increased room for **transparency** within and without organizations, as contracts and proofs can be distributed digitally and exist in a standardized format;
2. improved opportunity for real-time **analysis**, which be used to make or adjust financial or contractual decisions immediately as relevant; and
3. allowing for new levels of contractual **automation**, as the acceptance of new contracts can be made to trigger computer or machine action.

Concretely, the *Contract Negotiation* (CoNe) service, described in this document, facilitates collaboration by providing a means of negotiating about and signing Contract objects, which are parameterized references to legal texts in the form of *templates*. Whenever new contracts are entered into, the system providing the CoNe service could trigger events that lead to the fulfillment of any contracts entered into. For example, parties A and B might both accept a Contract that requires A to provide a certain good to B, and B to pay A. The Contract object itself would identify a template that describes the rights and obligations of A and B in legal language. The same object would also contain variables, such as quantities, prices, or delivery deadlines, as well as identifying the roles of A and B in the contract. By merit of being properly signed by both A and B, as well as referring to a sound legal text, the Contract can become useful as evidence in a court of law, in the event of a later dispute between the parties.

Consequently, the three primary concerns of this service becomes (1) *templates*, which provide legal definitions, (2) Contracts, which immutably record agreements, and (3) *performances*, through which any agreements entered into may be fulfilled. Out of those three, only the second, immutable contract records, are of direct concern to this service, as depicted in Figure 1.

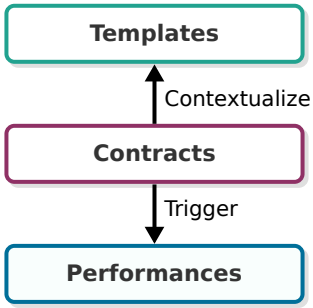

Figure 1: The three contract layers. The first layer represents concrete contractual documents where certain parameters might be changeable, such as prices, quantities, or payment options. The second layer, consisting of Contract objects, refers to contractual clauses in the first layer, provides arguments and contains the digital signatures of the contracting agents. The third layer represents the fulfillment of the contracts by the progation of events.

The rest of this document is organized as follows. In the remainder of this section we consider significant prior art, describe how this service is meant to be used and comment on the status of this document. In Section 2, we relate what contracts and templates are, as well as how they enable collaboration. In Section 3, we explain how the service facilitates negotiation and non-repudiable contract acceptances. In Section 4, we outline major delimitations of this service, which is a work-in-progress. In Section 5, we describe the abstract interface, in terms of functions invoked by messages, provided by this service. Finally, in Section 6, we present the data types used by those functions.

1.1 Significant Prior Art

CoNe services are meant to be useful for forming so-called *contract networks*, which are social networks where the most relevant theme is collaboration via contracts.[1] The *contract network* concept is very much related to the *exchange network* concept, which has been the subject of significant prior art [1] [2]. Work on the so-called *exchange network architecture* eventually revealed that the negotiation and exchange of *tokens*, its primary primitive, is an awkward abstraction for key kinds of legal agreements, even if working well for physical components, liquid assets and other ownable entities. In particular, modelling the obligation to deliver a good as a token proved to require details that would have been redundant if using a traditional kind of contract, in which roles and obligations are explicitly stated. To avoid potential confusion and increase the generality of the solution, a *contract-first* model was conceived as a replacement to the *ownership-first* model of the exchange network architecture. This Arrowhead Core service proposal is an important first step to evaluating the new model.

1.2 How This Service Is Meant to Be Used

For the CoNe service to become practically useful, it must be implemented by a system that can make it initiate and react to contractual events. In other words, the CoNe service only provides a means through which contracts can be expressed, it does not solve tasks such as creating, offering and choosing what contracts to accept. To use the CoNe service, an Eclipse Arrowhead system must be used that provides both (1) a concrete realization of the interface described in this document and (2) the behavorial logic required to make meaningful use of that interface. Such a system could, for example, provide a graphical user interface, act according to sophisticated policies, or use fixed routines tailored for a specific use case.

One way to use this service is via the so-called *Contract Negotiation Proxy* system, which is part of the same Arrowhead Core proposal as this service. That system allows for multiple Arrowhead systems to reuse the same contractual identity, which can be important for legal or practical reasons, as well as providing a standardized way of distributing events about changes to contracts in a local cloud via the *Event Handler* system, which could be important for logging, analysis and automation purposes. The Contract Negotiation Proxy system is recommended for most evaluation purposes at this point[2]. Certain use cases might, however, benefit from integrating the CoNe service directly into custom application systems.

1.3 Status of this Document

Designing an adequate infrastructure for digitized collaboration has proven to require a lot more thought and effort than initially perceived. The criticality and genuine complexity of the problem being addressed by this service means that there still are too many unknowns and unsures for this document to be close to finalization. This document, and all other such part of the same Eclipse Arrowhead Core proposal, are still to be considered early drafts and might to have to undergo several significant revisions before becoming sufficient for most kinds of industrial deployments. If evaluating this service, and any implementations that may come with it, please report back your findings and experiences to Emanuel Palm <emanuel.palm@ltu.se>, who will ensure they contribute to the further refinement of this service.

[1] The term *contract network* is not meant to refer exclusively to any particular kinds of technologies, but rather to the situation this service and other components are meant to help put your local cloud in. In other words, the CoNe service allows for a given system to become *part* of a contract network, in which systems owned by distinct stakeholder can enter into legally binding agreements.

[2] The source code of a working prototype can be retrieved from https://github.com/emanuelpalm/arrowhead-contract-proxy.

2 Contracts and Templates

From a legal perspectice, a contract is a set of rights and obligations two parties have with respect to each other. Contracts may be written down on signed documents, be articulated in front witnesses, or be recorded in any other way permitted by the court of law, arbitrator or other kind of adjudicator to be used in case of any dispute. By implication, a contract is not, in and of itself, a tangible or digital artifact. Rather, both tangible and intangle artifacts, such as conventional paper contracts, promises, or digitally signed agreements, together make up *proof* that a contract has been entered into.

The primary objective of the CoNe service is *not* to define or interpret contract documents, but to provide a means whereby contracts can be *proved to have been entered into*. Defining, interpreting and performing contracts are very much related activities, and may also be performed by a system providing the CoNe service, but those activities are not essential to the CoNe service itself. The entering into contracts is facilitated by having distinct parties Offer and Accept Contract objects, which are concretely represented by parameterized references to *templates*. The Contract itself functions only as a record of agreement, while the template it refers to contains all necessary legal texts.[3] An example of a Contract refering to a template is given in Figure 2.

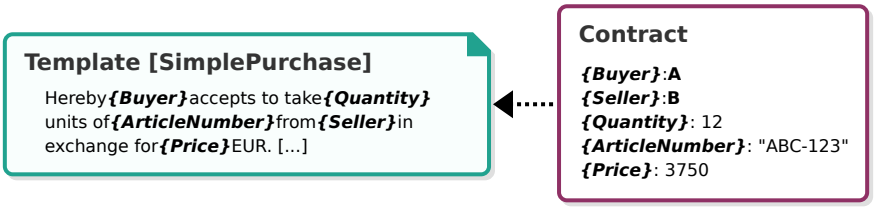

Figure 2: A Contract object referring to a *template*. Note how the arguments in the contract parameterize the template, in terms of integers, strings and role assignments. The combination of the contract and template is what allows for the creation a meaningful proof of agreement. To make such a template/contract combination legally binding, however, it must be offered and accepted via the Offer and Accept functions described in Section 5, which add digital signatures to the contract.

2.1 Caveats About Digital Signatures

The Offer and Acceptance data types in Section 6 both contain a *Signature* field. The first signature is provided by the offeror of the Contract in question, while the latter by the acceptor. While it may be convenient to use the certificates [8] and corresponding private keys that are mandated for secured Eclipse Arrowhead clouds, it may not always be practical. For example, certain legal frameworks may require a certain kind of certificate to be used and that a certain party endorses them or signs them. It may, consequently, become necessary to use one set of keys only for legal signing, while another is used for most or all other purposes. When collaborating across several legal juristictions, different certificates may have to be used for different counter-parties. This may complicate things if wanting to be able to show signed Acceptances to third parties, as it has to be ensured beforehand that the third party will be able to trust all contained signatures. The adequate use of different kinds of certificates in certain contexts is likely going to be a significant subject of future research. Please have this in mind as you evaulate this service, as any insights you gain, and share with us, may prove indispensible to its future evolution.

[3] As we see it, there are three major legal document formats that could be considered relevant to use with this service. Firstly, we have the traditional paper contract, which can be scanned to make available for digital use. It has the advantage of already being acceptable to traditional courts of law. Making such compatible with our solution would only require our Contracts to adhere to existing legislation, such as [3] in Europe. Secondly, we have the *Ricardian contracts* [4], which lately have had a renaissance in e.g. the work of Clack, Bakshi and Braine [5] [6]. Ricardian contracts are unique in that they are technically simple and appear and function similarly to paper contracts, with the exception that they make it convenient to programmatically extract certain contract details. Thirdly, we have what we refer to as the *Australian School* of contracts, which most notably has produced the OASIS *LegalRuleML* standard [7]. The standard can be used to formulate contracts in an entirely programmatic fashion, which has obvious technical advantages while likely being more difficult to use in a court of law, at least at the time of writing.

3 Negotiation

From the perspective of the CoNe service, Contracts are always the result of successful *negotiations*. The simplest possible negotiation would be one party presenting an Offer that is immediately accepted by a counter-party. However, the receiver of an offer could also make a counter-offer or decide to reject it. This idea of negotiation preceeding contracts is a reflection of how most legal traditions regard contracts as the product of an *offer* and an *acceptance* [9]. While some offers may not be negotiable due to legal circumstances demanding their being accepted, the CoNe service itself never makes that assumption.

The CoNe service keeps track of on-going negotiations by maintaining so-called *sessions*, each of which represent a single negotiation between one pair of stakeholders.[4] Bear in mind that as negotiations always are concretely realized by creating sessions, those sessions are often referred to simply as *negotiations*. A session is created when a party sends an Offer containing a *negotiation identifier* that does not identify any other session currently or previously shared by the negotiating parties. Sessions are permanently closed when offers are accepted, rejected, or expire.

3.1 Logging

Conceptually, each maintained session consists of a *message log*. The log is created when a party makes its first Offer. Any further counter-offers, or other messages, are appended to the log as they are sent and received. The most recent offer in the log that is neither accepted, rejected or expired is considered the negotiation *candidate*, and is what any subsequent counter-offer, acceptance or rejection must either replace or refer to. If the candidate does expire, is accepted or is rejected, the session log is considered *closed* and can no longer be appended to.

3.2 Progression

Sessions are created, maintained and destroyed as described by the finite state machine in Figure 3. Valid and successful service function calls, described in Section 5, and the **IsExpired** assertion, which is true if and only if the current time is after the *ValidUntil* field of the current candidate Offer, make up the preconditions of all possible transitions.

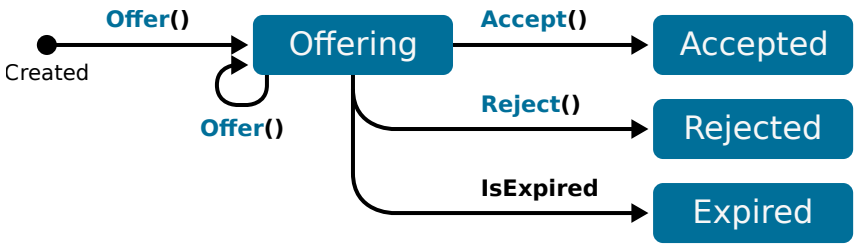

Figure 3: A negotiation session described as a finite state machine.

No distinction need to be made between the two parties when progressing through the state machine, as the parties must take turn to send messages and thereby trigger state transitions. Errors occuring while trying to formulate or send messages do not affect state machine transitioning.

3.3 Events and Automation

Whenever a session ends in the acceptance of an Offer, that Offer should, in whatever manner decided by the system implementing the service, be published in such a way that it can trigger automation routines. Whether it be to add tasks to billing systems or to start manufacturing ordered components. If relevant, also other transitions of the negotiation state machine could trigger events being published.

[4] Multi-party negotiation and making proposals are not directly supported at this point. See Section 4 for a discussion.

4 Important Delimitations

The CoNe service, as it is currently defined, is meant to constitute a *minimum viable product*. Making the CoNe service any more complicated than it has to be would make it less straightforward to implement and evaluate. For this and educational reasons, the following delimitations have been superimposed on this service:

1. No direct support is provided for multi-party negotiations, or for negotiations that must be attested by third parties before finalization.
2. Proposals, which are useful when not enough details are known to formulate an Offer,[5] are not supported. For example, there is no way to offer multiple commitment sets from which one can be chosen.
3. No way is provided for CoNe services to represent multiple legal identities.
4. There is currenly no direct support for sending contractual data, such as templates, directly between CoNe services, which would enable starting negotiations about templates that were previously unknown to one party. In fact, templates are not directly handled by the service at all. What templates can be used must be established prior to this service being used.
5. Making the CoNe service more aware of certain template details, such as whether an argument identifies a role or a previous Acceptance, or if a particular Contract makes another such void, could make it more straightforward to automate more aspects of contract handling.

[5] In legal terms, an offer is only considered valid if it leaves no ambiguities regarding what needs to be performed in order for the offer to be fulfilled. Proposals, on the other hand, never contain enough information to be accepted as contracts, but serve as starting points for negotiation about the unspecified details.

5 Service Interface

This section lists the *functions* that must be exposed by CoNe services in alphabetical order. Each function is meant to represent one thing the CoNe service can *do* in response to the function being invoked via a message from an authorized sender. In particular, each following subsection names an abstract function, an input type and an output type, in that order. The input type is named inside parentheses, while the output type is preceded by a colon. Input and output types are only denoted when accepted or returned, respectively, by the function in question.

All abstract data types named in this section are defined in Section 6. To better understand how these functions are meant to be used, please refer to Section 3.

5.1 function Accept (Acceptance)

Called to accept a previously received Offer, putting its Contracts into effect. After this interface has been sucesssfully called, both the caller and the callee will have their own identical copies of the same Acceptance, which includes both of their signatures. The call closes the session in which the accepted offer was registered, as described in Section 3.

5.2 function Offer (Offer)

An offer for the caller and callee to enter into one or more Contracts. If the callee deems the Offer acceptable, it may proceed by wrapping the Offer that contains it into an Acceptance and call the Accept function of the caller. The call either creates a new session or updates an existing, as described in Section 3.

5.3 function Reject (Rejection)

Called to signal the desire to immediately terminate an identified negotiation session, as described in Section 3. The call is to be interpreted as if the caller is no longer interested in continuing the negotiation. If wanting to continue the negotiation, a counter-offer should be made instead.

6 Information Model

Here, all data objects that can be part of CoNe service calls are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is used to denote a collection of named fields, each with its own data type. As a complement to the explicitly defined types in this section, there is also a list of implicit primitive types in Section 6.6, which are used to represent things like hashes and identifiers.

An overview of the data object types is illustrated in Figure 4.

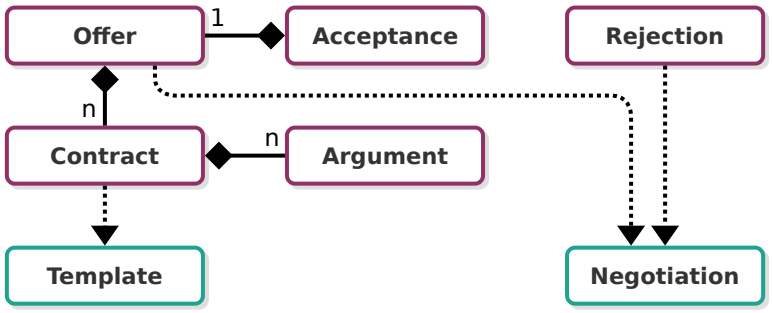

Figure 4: Information model as a UML class diagram. Black rhombuses signify data ownership, while dotted arrows denote references. The type closest to each rhombus is the one containing the reference or owning the data. Numbers are used to specify how many objects are referred to or owned, where n is to be regarded as an arbitrary positive number or zero. Note that *Template* and *Session* are only indirectly part of the information model, but are included for the sake of completeness.

6.1 struct Acceptance

An accepted and doubly signed Offer, meant to be legally binding. Note that there is a signature also in the Offer itself, which is why there is only one signature field in this structure.

Field	Type	Description
negotiationId	RandomID	Negotiation session identifier.
offerorFingerprint	Hash	Fingerprint of offeror's certificate.
acceptorFingerprint	Hash	Fingerprint of acceptor's certificate.
offerHash	Hash	Hash of accepted offer.
signature	Signature	Acceptor's signature.

6.2 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, in a non-naive implementation, 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.

Field	Type	Description
name	Name	Human and machine-readable name of parameter.
value	Custom	Concrete value.

6.3 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 properly signed Acceptances.

Field	Type	Description
templateHash	Hash	Hash of invoked *template*.
arguments	List<Argument>	Arguments matching *parameters* defined in *template*.

6.4 struct Offer

A concrete offer of Contracts intended for a specific receiver.

Field	Type	Description
negotiationId	RandomID	Negotiation session identifier.
offerorFingerprint	Hash	Fingerprint of offeror's certificate.
receiverFingerprint	Hash	Fingerprint of intended receiver's certificate.
validAfter	DateTime	Instant when offer becomes acceptable.
validUntil	DateTime	Instant when offer expires.
contracts	List<Contract>	Offered contracts.
signature	Signature	Offeror's signature.

6.5 struct Rejection

Identifies a negotiation session a sending party wishes to terminate, as described in Section 3.

Field	Type	Description
negotiationId	RandomID	Negotiation session identifier.
offerorFingerprint	Hash	Fingerprint of offeror's certificate.
rejectorFingerprint	Hash	Fingerprint of rejector's certificate.
offerHash	Hash	Hash of rejected offer.
signature	Signature	Rejector's signature.

6.6 Primitives

Types and structures mentioned throughout this document that are assumed to be available to implementations of this service. The concrete interpretations of each of these types and structures must be provided by any IDD document claiming to implement this service.

Type	Description
Custom	Any suitable type chosen by the implementor of the service.
DateTime	Pinpoints a specific moment in time.
Hash	The data required to represent a cryptographic hash produced with a certain function.
List <A>	An *array* of a known number of items, each having type A.
Name	A string identifier that is intended to be both human and machine-readable.
RandomID	An identifier produced using a cryptographically secure random function.
Signature	A timestamped signature produced with a well-defined scheme.

7 References

[1] E. Palm, O. Schelén, U. Bodin, and R. Hedman, "The Exchange Network: An Architecture for the Negotiation of Non-Repudiable Token Exchanges," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, 2019, *accepted for publication*. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-74043

[2] E. Palm, U. Bodin, and O. Schelén, *The Performance, Interoperability and Integration of Distributed Ledger Technologies*. Luleå University of Technology, 2019, ch. Paper D, pp. 111–141. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-74046

[3] Council of the European Union, "Regulation (EU) no 910/2014 of the European Parliament and of the Council of 23 july 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing directive 1999/93/EC," 2014. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2014/910/oj

[4] I. Grigg, "The ricardian contract," in *Proceedings of the First IEEE International Workshop on Electronic Contracting*. IEEE, 2004, pp. 25–31. [Online]. Available: https://doi.org/10.1109/WEC.2004.1319505

[5] C. D. Clack, V. A. Bakshi, and L. Braine, "Smart contract templates: foundations, design landscape and research directions," *arXiv preprint arXiv:1608.00771*, vol. abs/1608.00771, 2016. [Online]. Available: http://arxiv.org/abs/1608.00771

[6] ——, "Smart contract templates: essential requirements and design options," *arXiv preprint arXiv:1612.04496*, vol. abs/1612.04496, 2016. [Online]. Available: http://arxiv.org/abs/1612.04496

[7] T. Athan, G. Governatori, M. Palmirani, A. Paschke, and A. Wyner, *LegalRuleML: Design Principles and Foundations*. Cham: Springer International Publishing, 2015, pp. 151–188. [Online]. Available: https://doi.org/10.1007/978-3-319-21768-0_6

[8] D. Cooper *et al.*, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, 2008. [Online]. Available: https://doi.org/10.17487/RFC5280

[9] M. Giancaspro, "Is a 'smart contract' really a smart idea? insights from a legal perspective," *Computer Law & Security Review*, vol. 33, no. 6, 2017. [Online]. Available: https://doi.org/10.1016/j.clsr.2017.05.007

8 Revision History

8.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

8.2 Quality Assurance

No.	Date	Version	Approved by
1			