

Agreements in a De-Centralized Linked Data Based Messaging System

Florian Kleedorfer¹, Heiko Friedrich¹, and Christian Huemer²

¹ Research Studio Smart Agent Technologies, Thurngasse 8/16, 1090 Vienna, Austria
([fkleedorfer](mailto:fkleedorfer@researchstudio.at)|[hfriedrich](mailto:hfriedrich@researchstudio.at))@researchstudio.at,

² Institute for Software Technology and interactive Systems, Vienna University of Technology, Favoritenstrae 9-11, 1040 Vienna, Austria, huemer@big.tuwien.ac.at

Abstract. People frequently use internet-based messaging systems to coordinate. In order to achieve that, it is sufficient for them to exchange natural language messages. The message history they generate can be seen as a shared database that can be tapped into by personal assistive systems; moreover, messaging is increasingly used for human-computer communication. However, if natural language understanding is required for such systems to function properly, the cost of developing them is high and only few market players will be able to compete. If, on the other hand, it is possible to mix machine-interpretable data with natural language conversations, assistive or conversational programs may be developed more easily. As a first important challenge, we tackle the problem of negotiating agreements and unambiguously represent what has been agreed upon in a machine-readable form. In this paper, we propose an extension of the Web of Needs, a de-centralized, Linked Data based matching and messaging system, to allow conversation partners to produce a mutually agreed-upon RDF dataset.

Keywords: Linked Data, messaging, agreements, e-negotiation

1 Introduction

Agreement is commonly considered as “a negotiated and typically legally binding arrangement between parties as to a course of action” [21]. When we look at the dominant ways to reach such agreements today, we see three clusters. The classic way is that people come to an agreement in some kind of oral or written conversation. With the advent of modern information technology, another way of agreeing on the specifics of a transaction evolved: technological artifacts (Websites, mobile apps) that require users to provide certain information needed to render a service. The common pattern in the second cluster is a fixed Web API on one side and a human who fills in the parameters on the other side. Here, the API exhibits little to no flexibility, and agreement depends on the person understanding the interface the way the designers intended them to.

A third option is gradually evolving in the form of chatbots and other conversational systems. The way such systems are typically built, they mix natural

language conversation with structured or semi-structured data exchange, for example by providing natural language patterns for allowing users to perform the equivalent of API function calls, or GUI widgets to ask users for specific information. This setup allows for more flexibility on the side of the service provider. However, the success of the transaction depends on the chatbot understanding the person correctly.

The field of electronic negotiations research (for a taxonomy of the field, see the work by Ströbl and Weinhardt [25]) is striving to develop a fourth option in which a communication medium produces agreement as an integrated functionality that can be relied upon by people and software agents alike. Such a medium might reduce the burden on intelligent conversational systems and, if available as an open, federated system, make it cheaper and easier for new market participants to enter.

Motivated by this vision, we describe an extension of a de-centralized, Linked Data based communication system, the Web of Needs (WoN) [13, 12], that helps users find cooperation partners based on mutual interest, and provides a communication channel. With the proposed extension, the communication partners can negotiate the contents of an RDF dataset containing mutually agreed-upon data that can be used for further coordination.

This paper is organized as follows. In Section 2 we explain how our work compares to existing approaches in the field of electronic negotiation systems. Section 3 establishes the technological background required to understand our contribution. The contribution itself is described in Section 4, which is followed by a discussion of important design decisions in Section 5.

2 Related Work

This section is divided into two parts. In the first part we compare our approach with other e-negotiation systems and protocols. Responding to the community’s interest, in the second part we compare agreements in WoN to blockchain-based solutions.

2.1 Negotiation Systems

The purpose of the protocol extensions described in this paper is to facilitate and organize electronic two-party negotiations in the Web of Needs. Systems that employ Internet technologies (e.g. open Linked Data) and are deployed on the Web are usually called e-negotiation systems (ENS) [11] and originate from negotiation support systems (NSS). Many e-negotiation systems that are based on game theoretic models or heuristic approaches restrict the negotiation domain, negotiation object, negotiation protocol, negotiation rules or communication language in order to enable autonomous agents for automatic negotiation processes [10, 16, 22]. This is also true for scenarios where automated agents negotiate with humans [17]. Some systems apply more flexible approaches by not

hard-coding negotiation protocols but expressing them by ontologies [3, 26] or by letting participants construct rules in open e-negotiation protocols [4].

According to the Montreal Taxonomy [25], which is the first attempt of a generic comprehensive classification scheme of e-negotiation processes and systems [5], electronic markets can be divided into different phases of interaction: Knowledge, Intention, Agreement and Settlement. So far the main focus of Web of Needs was on the Intention phase where supply and demand is specified by market participants. With the support of electronic negotiation processes, WoN can support users in the Agreement phase as well, in which terms and conditions of transactions are identified, and a contract can be signed.

In the Web of Needs the goal is to support negotiation between humans in the first place but keep the system open and structured enough for automated agent negotiation. Argumentation based negotiation approaches [1, 22] emphasize the importance of expressive communication (for exchanging information, resolving uncertainties, revising preferences, etc.) as an integral part of the negotiation process. In more complex scenarios this often enables agents (human or automated) to find an agreement in the first place. Therefore we want to keep the full flexibility and expressiveness of natural language in negotiating about arbitrary domains and objects while still having structured agreements as an outcome.

These structured agreements are more than just an interaction history of messages and can be compared to what is referred to as commitment stores [22]. Commitment stores are used to explicitly track statements of the participants of a conversation. There are commitment rules that decide if statements are added or removed from a commitment store. In contrast to plain interaction histories, statements in commitment stores are meant to have explicit consequences, for instance promising to execute certain actions automatically if specified conditions are reached. This notion of structured agreements is realized by referring to exchanged messages using RDF triples [18] and thereby marking them as issues of a proposal that can subsequently be accepted by the other participant so as to form an agreement. The content, number of issues and order of exchanged messages between participants is completely unrestricted and proposals/agreements can be formed (as well as retracted/canceled) directly from messages during the communication. Since these structured agreements can be automatically extracted from the message interaction history, the system represents a combination of communication-oriented and document-oriented approaches (e.g. [23]). This combination has the advantage of allowing informal communication to help avoid misunderstandings and errors, while still producing a document collection containing the result of the negotiations, and is particularly valuable in business-to-business (B2B) scenarios [28]. In many B2B scenarios (for instance public procurement [7, 19]), agreements could be used to create legal contracts[20].

2.2 Differentiation from Blockchain Technologies

The agreement protocol layer is built on the Web of Needs message protocol layer, which generates a signed message interaction history for the two partici-

pants of the communication. This guarantees authentication, integrity and non-repudiation for agreements or contracts (cf. [9]) as well as for the whole message history. Every message references the signature of previous non-referenced messages therefore creating a structure which has some similarity to a blockchain. However in contrast to a blockchain this structure represents a consensus that affects only two participants and is not distributed to nodes of the whole network. The use of blockchains for storing negotiations and contracts has already been proposed [27, 29]. Identified trade offs were the limitation of data that can be stored on the blockchain as well as the communication latency that might result in poor user experience if too much data is communicated over the blockchain. In the following we list differences between WoN and a blockchain approach:

Maturity. WoN is still being developed, the protocol can be expected to change. It has not undergone an independent security review yet, nor is the current protocol documented in full. To the best of our knowledge, nobody has ever seriously tried to hack WoN. Blockchains, on the other hand, are quite mature and widely considered secure.

Network structure. Blockchains are distributed whereas WoN is decentralized. Some WoN nodes are bound to become more important than others (i.e., host more needs). Outage of a WoN node means that all needs it hosts are unavailable.

Data distribution. The blockchain essentially is a transaction log shared by all parties that want to have transactions or sign blocks. In WoN, conversations are data structures shared only between the participants and the WoN nodes they use.

Currency. Blockchain-based smart contracts require some kind of currency. In contrast, a clear design goal in WoN was to build a technological layer for expressing reasons for interaction, establishing contacts, and communicating and to separate this layer from an accounting layer on which payments are made. This decision has two reasons: first, WoN is not tied to any specific payment system and may well be able to interface with all of them. Second, for some interactions, payment is not necessary, or may even be detrimental in the sense that they may never occur if they were tied to payment.

Strength of guarantees. The guarantees that can be made for a WoN-based agreement are weaker than a blockchain based smart contract, at least if the blockchain has enough mutually independent miners. In WoN, one must always be prepared to be communicating with an attacker who controls the WoN node she is using. Some of the possible attacks are: a) An attacker can at all times remove all traces of the needs she creates or the conversations she has via those needs at any time from her WoN node. In that case we are left with the complete conversation history on our WoN node, but no counterpart to talk to, or to report to the police, for example. Therefore, additional trust mechanisms may be required. b) An attacker may choose to ignore messages. We will never be able to prove that they received the message if they do not want to. In WoN, you can only prove the message history that both parties have approved. c) An attacker may control her own WoN node as well as the one we are using. As we

sign all our messages, our WoN node cannot fake our messages, but it can decide to drop them, or to drop messages coming from the counterpart. From our point of view, the conversation will consist of all messages that were let through.

Fake interactions. Because no proof of work or proof of stake is required in WoN, it is always possible to set up fake conversations, which may be used in an attack to trick users into thinking a participant has been around for a long time and received many positive reviews (another feature we’re planning), but in fact has just set up a new fake need. In contrast, the data in a blockchain cannot be faked.

Self-execution of contracts. In the ethereum system, contracts are written in a programming language, such that they react to state changes in the blockchain. In WoN, agreements as proposed in this paper are just arbitrary RDF datasets that the participants agree upon, i.e., static data. However, it does seem possible to achieve similar functionality by embedding e.g. SHACL shapes, SPARQL queries, or Javascript in a WoN agreement and defining how that code is to be evaluated.

3 Background

Our approach for the negotiation of agreements over Linked Data is based on the Web of Needs (WoN) architecture. In this section, we explain the aspects of WoN that are required to understand our approach.

The central idea of WoN is to connect people or software agents based on their intentions, or as we call them, *needs*, in a de-centralized infrastructure built on top of Linked Data and related Web technologies. When their needs have been matched, users of different Web domains can establish a communication channel. The architecture does not limit users in the description of their needs, nor is the content of the messages they exchange restricted in any way. One core benefit of this approach is that matchmaking, arguably a highly centralized functionality in today’s Web, can be realized in a de-centralized fashion. When connected, users have conversations by exchanging messages. In the simplest case, a conversation is a chat session. However, the messages can contain arbitrary RDF data, and they are stored online in an immutable fashion, effectively providing an add-only RDF store shared between users. They can thus collaboratively build a shared RDF data model of their relationship and use it for aligning their expectations and coordinating their actions.

3.1 Components and Responsibilities

Users (or software agents) publish a Linked Data resource on the Web describing their interest in an interaction. This resource is referred to as a *need*, the agent that created it is its *owner*, who used an *owner application* to create the resource, which in turn published it on a server that supports the WoN protocol, called a *WoN node*. The need description can contain a self-description and a description of the need it should be matched with. Independent *matching services*

can subscribe to changes on the WoN node and crawl their content. Whenever a matching service finds a suitable pair of needs, it informs them of each other’s existence. Consequently one could describe a need as a persistent, self-describing search query that can be discovered by other such queries, and that serves as a bi-directional communication proxy.

WoN nodes fulfill two purposes. On the one hand, they store and serve all the data, and on the other hand, they serve as message brokers, routing messages directed at needs and offering updates via a publish-subscribe system.

3.2 Message Composition and Delivery

Messages are realized as RDF datasets that are created by owners or WoN nodes, and that are de-referencable under their *message URI*, minted by the sender, on the sender’s WoN node. Message datasets contain three types of graphs: *content graphs*, *envelope graphs* and a *signature graph*. Content graphs can contain any kind of RDF data (including higher-level protocol data, see Section 4) that the sender wants to communicate to the recipient. Envelope graphs contain meta data about the message like the addressing information or references to other envelopes that were created during processing of messages. Both kinds, content and envelope graphs are cryptographically signed during the process of sending and receiving messages. Envelope graphs are added consecutively by each party processing the message, linking them up in a chain. The signature of the last (‘outermost’) envelope graph in the chain is placed in a signature graph. The first (‘innermost’) content graph of a message links to the content graphs and includes their signatures. The structure of a message which is sent from an owner to a WoN nodes is depicted in Figure 1.

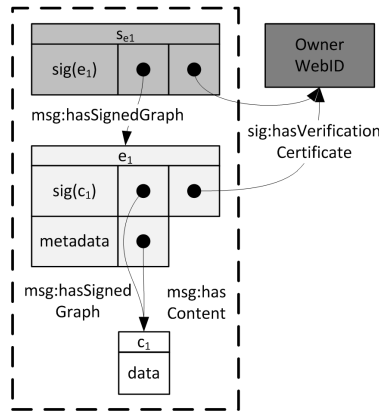


Fig. 1: Basic structure of message which is send from owner to WoN node. The envelope e_1 contains meta data and a signature of the content graph c_1 , linking to the owner’s WebID. The signature graph s_{e_1} contains a signature of e_1 . The message URI is minted by the sender in their WoN node’s URI space where it will be accessible.

A message that is delivered to another need is sent to that need’s WoN node, stored there, and dereferencable on that node under an URI that the sender’s node mints in the recipient’s node’s URI space. The two copies, the *local copy*

and the *remote copy* reference each other so as to make the message delivery tractable. All participants sign the data they add to the message using their respective WebID [14]. The dataset representing a message has an empty default graph to avoid mixing triples from different messages when aggregating data.

We define \mathcal{D} to be the set of all possible RDF datasets. The boolean-valued function $\text{isMsg} : \mathcal{D} \rightarrow \{\text{true}, \text{false}\}$ indicates whether a given dataset is a processable message. $\mathcal{M} = \{m \in \mathcal{D} \mid \text{isMsg}(m) = \text{true}\}$ is the set of all possible messages. As messages do not have default graphs, messages are sets of named graphs, i.e. $\mathcal{M} \subset \mathcal{P}(I \times \mathcal{G})$, where \mathcal{P} denotes the powerset, I denotes the set of all possible IRIs, and \mathcal{G} denotes the set of all possible RDF graphs.

A WoN node always responds to a message it receives with a SUCCESSRESPONSE or a FAILURERESPONSE message, which is itself delivered to the sender of the original message and stored on the WoN node. The sender of a message may either be an owner, a remote node, or the WoN node itself. A message is referred to as *failed* if any of the WoN nodes involved responds with a FAILURERESPONSE message. If at least one of the WoN nodes does not respond to the message at all, it is referred to as *ignored*. We use the boolean valued functions $\text{failed} : \mathcal{I} \times \mathcal{P}(\mathcal{M}) \rightarrow \{\text{true}, \text{false}\}$ and $\text{ignored} : \mathcal{I} \times \mathcal{P}(\mathcal{M}) \rightarrow \{\text{true}, \text{false}\}$ to express that a message with a given IRI has failed or was ignored in a given set of messages.

3.3 Creating Needs and Establishing Connections

A need is created by sending a CREATE message containing the need description along with a public key to a WoN node. The need creator mints two URIs in the URI space of the WoN node in the process, the message URI (as explained above) and the *Need URI*. The latter is used as the root resource of the need description. Upon receiving the message, the WoN node makes the need description available as Linked Data and responds with a SUCCESSRESPONSE.³

Need owners can ask other needs to establish a *Connection* by performing a handshake of a CONNECT and an OPEN message (or a CLOSE message to deny the connection). Matching services can suggest a connection by sending a HINT message to one or both of the needs involved. In that case, the connection is created by the WoN node, but the owners still have to establish the connection by an OPEN/OPEN handshake. In an established connection, the need objects can freely exchange CONNECTIONMESSAGE messages.³ The exchanged messages form two parallel signature chains on both WoN nodes as new messages refer to signatures of earlier (formerly unreferenced) messages. Moreover, these chains reference each other since each transferred message contains a reference to its remote copy. The message chain structure resulting from connecting two needs is depicted in Figure 2.

³ In an earlier work [12] we show sequence diagrams of the message exchange taking place upon need creation (Figure 5) and connecting (Figure 6)

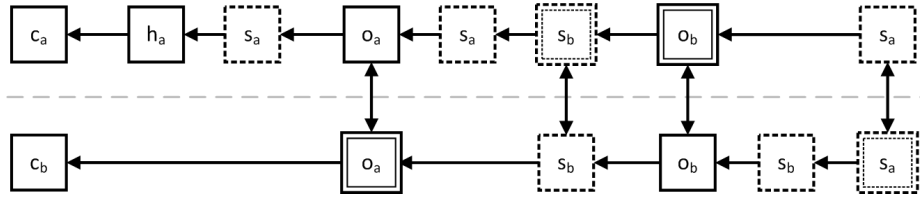


Fig. 2: Diagram showing the messages visible to both owners in a conversation. The beginning of the conversation is shown here with the two CREATE messages (c), followed by a HINT, (h), and an OPEN/OPEN handshake (o). The two sides of the conversation are separated by the dashed line. The subscript a or b indicates which side the message was sent from. SUCCESSRESPONSES are represented by s . Dashed boxes represent responses, double boxes represent remote copies. The arrows indicate references to other messages.

Needs and connections have *event containers*, which are modeled after pageable LDP containers[24].⁴ When stored, a message is added to the event container of the object it belongs to. CREATE, ACTIVATE, and DEACTIVATE messages are stored in the respective need's event container. All other messages are added to the respective connections's event container.

When processed by the WoN node, an envelope graph is added to the message. Besides other message metadata, references to previous messages in the same event container are added in the form of signature references citing the signature value. By doing so, all messages become part of a signature chain, making message modification prohibitively hard, as signatures depend on signatures created by the two owners and up to two nodes involved in a conversation. When a connection is created for a need, the first message in the connection references the need's create message, thereby tying the need's messages to the connection's.

4 Protocol Layers: From Conversations to Agreements

In the following, we consider a conversation between two need owners, o_1 and o_2 that connects their needs n_1 and n_2 via their connections c_1 and c_2 .

The conversation is the union of all messages that are accessible to o_1 in the conversation with o_2 , at the point in time when m is the last message seen by o_1 . This point in time is defined as either the reception of the remote response if o_1 is the sender of m . If o_1 is the recipient, the point is defined as the time the response to m in c_1 is sent.

The set of all possible conversations is defined as the powerset of all possible messages $\mathcal{C} = \mathcal{P}(\mathcal{M})$ (which, as stated earlier, is equal to $\mathcal{P}(\mathcal{I} \times \mathcal{G})$). We define the *raw conversation dataset* $C_r(o_1, o_2, m)$ as the union of all messages in the

⁴ Note that the current implementation of WoN is not based on LDP, it only uses the paging interface of LDP containers.

event containers of n_1, n_2 and c_1 . As we will regard o_1 and o_2 as fixed for the remainder of the paper, describing the situation from the point of view of o_1 , we will write $C_r(m)$ instead of $C_r(o_1, o_2, m)$.

The raw conversation dataset consists of all messages exchanged, or attempted to be exchanged. It is used to verify the integrity of the message history based on the chain of message signatures. It can be seen as a monotonically growing RDF store that the participants can only manipulate by adding messages. In the context of WoN, this dataset is intended to be used as a device that allows users to coordinate, that is, to build a shared, and at least partly agreed-upon model of their relationship. In order to allow for the latter, we define higher-level conversation protocols based on C_r . For doing this, we first make some auxiliary definitions:

The function $\text{iri} : \mathcal{M} \rightarrow \mathcal{I}$ gives the IRI of the input message.

The function $\text{iris} : \mathcal{C} \rightarrow \mathcal{I}$ yields the IRIs of all all message in the specified conversation dataset.

The function $\text{msg} : \mathcal{I} \times \mathcal{C} \rightarrow \mathcal{M}$ yields the message dataset of the message with the specified IRI in the specified conversation dataset.

The function $\text{sender} : \mathcal{I} \times \mathcal{C} \rightarrow \mathcal{I}$ yields the IRI of the sender of the input message.

The function $\text{happenBefore} : \mathcal{I} \times \mathcal{I} \times \mathcal{C} \rightarrow \{true, false\}$ is true if there is a signature reference path in C from the message identified by the second IRI and all of its Responses to the message identified by the first IRI and all of its responses, respectively.

The function $\text{content} : \mathcal{I} \times \mathcal{C} \rightarrow \mathcal{G}$ yields the union of all content graphs of the message with the specified IRI in the specified conversation dataset.

The function $\text{strip} : \mathcal{M} \rightarrow \mathcal{M}$ removes all content graphs from the input message.

In the following we define a number of conversation protocols based on these defintions.

4.1 Acknowledgement Protocol

As stated earlier, message delivery can fail, and messages can be ignored. In both cases, we are dealing with messages that were not provably delivered to the conversation partner. In this protocol layer, we want to present only the content that was provably delivered, therefore we exclude the content of these messages in the *acknowledged selection* $\mathfrak{S}_{\text{ack}} : \mathcal{C} \rightarrow \mathcal{C}$:

$$\mathfrak{S}_{\text{ack}}(C) = \bigcup_{m \in \text{iris}(C)} \text{cleanup}(m, C) \quad (1)$$

where

$$\text{cleanup}(m, C) = \begin{cases} \text{strip}(\text{msg}(m)) & \text{if } \text{failed}(m, C) = true \\ & \vee \text{ignored}(m, C) = true \\ \text{msg}(m) & \text{otherwise} \end{cases} \quad (2)$$

4.2 Modification Protocol

We intend to allow participants to specify SHACL shapes [15] defining the information they require their counterpart to provide. Moreover, executing SPARQL queries [2] over the conversation dataset can be useful for a number of use cases. However, for accurately representing a shared model of the conversation content, it is necessary to allow participants to change their mind or to correct mistakes, which means, there must be a way to modify past messages. Modifying is only allowed in one way: by marking one's own earlier messages as no longer to be considered or, as we will call it in the remainder of this work, as *retracted*.

For modelling the modification of messages, we introduce the Modification ontology⁵, prefixed 'mod'. It specifies only one ObjectProperty, `mod:retracts`, that is used in triples linking two message IRIs, the subject being the retracting message, the object being the retracted message.

The function `retracts` : $\mathcal{I} \times \mathcal{C} \rightarrow \mathcal{P}(\mathcal{I})$ returns all message IRIs linked to from the input message via `mod:retracts` in any of its content graphs. The function `isRetracted` : $\mathcal{I} \times \mathcal{C} \rightarrow \{true, false\}$ is defined as follows:

$$\begin{aligned} \text{isRetracted}(m, C) = & \\ & m \in \text{iris}(C) \\ & \wedge \exists r \in \text{iris}(C) : & (3) \\ & \quad m \in \text{retracts}(r, C) \\ & \quad \wedge \text{sender}(m, C) = \text{sender}(r, C) \\ & \quad \wedge \text{happenBefore}(r, m, C) = true \end{aligned}$$

The modification functionality is provided by the modification selection $\mathfrak{S}_{mod} : \mathcal{C} \rightarrow \mathcal{C}$ as

$$\mathfrak{S}_{mod}(C) = \bigcup_{m \in \text{iris}(\mathfrak{S}_{ack}(C))} \text{modifyMessage}(m, \mathfrak{S}_{ack}(C)) \quad (4)$$

where

$$\text{modifyMessage}(m, C) = \begin{cases} \text{strip}(\text{msg}(m)) & \text{if } \text{isRetracted}(m, C) = true \\ \text{strip}(\text{msg}(m)) & \text{if } \text{retracts}(m, C) \neq \emptyset \\ \text{msg}(m) & \text{otherwise} \end{cases} \quad (5)$$

The effect of this selection is that the content graphs of retracted messages and those of the retracting messages are removed in the result. The selection is applied to \mathfrak{S}_{ack} , hence modifications only have an effect if they do not fail and are not ignored.

The messages referenced through the `mod:retracts` property are said to be *retracted*.

⁵ See <http://purl.org/webofneeds/modification> [2017/07/23].

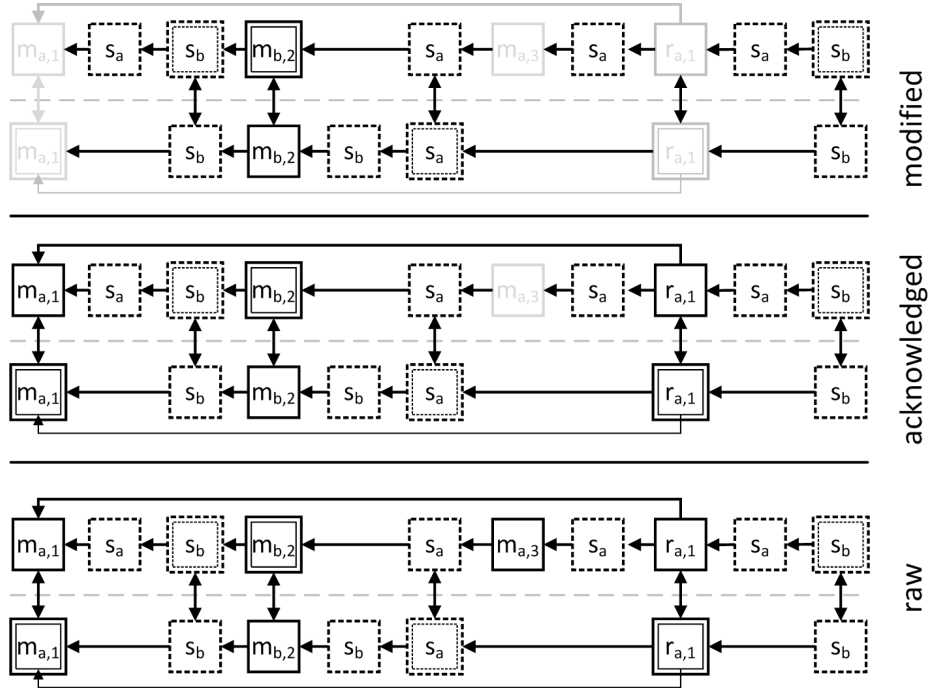


Fig. 3: Diagram showing a acknowledgment and modification protocol layers using the same visualization style as Figure 2. Here, a later part in a conversation is shown with three consecutive `ConnectionMessage` messages (m). Message $m_{a,3}$ is ignored by its intended recipient. In the acknowledged selection (middle layer), its content graph is therefore removed, which is indicated by its light grey color. Message $m_{a,1}$ is retracted by message $r_{a,1}$. The content graphs of both messages are removed in the modified selection (top layer), leaving message $m_{b,2}$ as the only message that still has a content graph.

4.3 Agreement Protocol

The fact that the message history can be modified does not mean that both participants agree on anything. It just allows them to revise earlier statements, which is not sufficient to coordinate actions between agents. In order to coordinate, it is required to come to a shared understanding about facts, which requires that the agents state the facts and that they signal each other that they agree on them. In the Semantic Web, the core of such an agreement naturally is a set of triples. Consequently, the agreement protocol allows the participants to select a set of triples as agreed-upon.

The result is formally defined as the agreement function $\mathfrak{F}_{agr} : \mathcal{C} \rightarrow \mathcal{D}$, yielding a dataset in which each graph corresponds to one agreement in the

conversation C and the associated graph name is an IRI that identifies the agreement.⁶ In the following, we define \mathfrak{F}_{agr} .

In order to represent agreements, we allow participants to propose the content graphs of a set of earlier messages as the content of an agreement, to accept a proposed agreement, and to cancel an accepted agreement.

Again, we introduce an ontology, the Agreement ontology⁷, prefixed 'agr' that defines the properties `agr:proposes`, `agr:proposesToCancel`, and `agr:accepts`, and the classes `agr:Proposal` and `agr:Agreement`.

`agr:proposes`, domain `agr:Proposal`, is used to link the IRIs of two messages p and c in a triple `iri(p) agr:proposes iri(c)`. Such a triple only has an effect in this protocol if it occurs in a content graph of p , making p a proposal that can be accepted. We call the message c an *clause* of the proposal. There is no limit on the number of clauses in a proposal.

`agr:accepts`, domain `agr:Agreement`, is used to link the IRIs of two messages a and p in a triple `iri(a) agr:accepts iri(p)`. Such a triple only has an effect in this protocol if it occurs in the content graph of a if p is actually a proposal. In that case, we say that a accepts the proposal p . There is no limit to the number of proposals that can be accepted by one message a .

`agr:proposesToCancel`, domain `agr:Proposal` is used to link the IRIs of two messages a_2 and a_1 in a triple `iri(a2) agr:proposesToCancel iri(a1)`. Such a triple only has an effect in this protocol if it occurs in a content graph of a_2 , if a_1 is an earlier agreement that has not been canceled yet

There are no restrictions concerning the combined use of these properties in one content graph, that is, one message can accept any number of proposals, propose any number of other messages, and propose to cancel any number of agreements. As will be explained in the following, the effects of these statements do not influence each other. It is thus possible to propose an agreement that replaces another one by mixing `agr:proposes` and `agr:proposesToCancel`. It is even possible to make a proposal and agree to another proposal in one message.

The agreement protocol depends on the acknowledgement protocol and on the modification protocol: We want to make sure that only provably delivered messages can play a role in the agreement protocol, and we want to allow that until accepted, proposals and clauses can be retracted. However, after its acceptance, an agreement must remain unaffected by later retractions - the only way to get rid of an agreement must be to make a new one that cancels it.

For defining the agreement function we need to make some auxiliary definitions:

`accepts` : $\mathcal{I} \times \mathcal{C} \rightarrow \mathcal{P}(\mathcal{I})$ returns all message IRIs linked to from the input message via `agr:accepts` in any of its content graphs.

`proposes` : $\mathcal{I} \times \mathcal{C} \rightarrow \mathcal{P}(\mathcal{I})$ gives all the message IRIs the input message links to via `agr:proposes` in any of its content graphs.

⁶ Note that \mathfrak{F}_{agr} , in contrast to \mathfrak{S}_{mod} and \mathfrak{S}_{ack} , does not simply select named graphs from a dataset. Rather, it selects them and recombines their triples, hence we do not refer to it as a selection.

⁷ See <http://purl.org/webofneeds/agreement> [2017/07/23].

`proposesToCancel` : $\mathcal{I} \times \mathcal{C} \rightarrow \mathcal{P}(\mathcal{I})$ gives all the message IRIs the input message links to via `agr:proposesToCancel` in any of its content graphs.

`hasContent` : $\mathcal{I} \times \mathcal{C} \rightarrow \{true, false\}$ indicates if the message with the specified IRI in the specified conversation dataset has at least one content graph.

The function `isProposal` : $\mathcal{I} \times \mathcal{C} \rightarrow \{true, false\}$ is defined as follows:

$$\begin{aligned}
\text{isProposal}(m, C) = & \\
& m \in \text{iris}(C) \\
& \wedge \forall c \in (\text{proposes}(m, C) \cup \text{proposesToCancel}(m, C)) : & (6) \\
& \quad c \in \text{iris}(C) \\
& \quad \wedge \text{happenBefore}(c, m, C) = true \\
& \quad \wedge \text{hasContent}(c, C) = true.
\end{aligned}$$

`isProposal` is true for message m in the conversation dataset C if C contains all messages that are proposed or proposed to be canceled. These messages need to happen before m (thereby also disallowing m to propose itself) and need to have at least one content graph. This last condition ensures that such a structure is not a proposal if evaluated over $\mathfrak{S}_{mod}(C)$ and one or more of its clauses have been retracted, because such a clause does not have a content graph in $\mathfrak{S}_{mod}(C)$.

The function `isAgreement` : $\mathcal{I} \times \mathcal{C} \rightarrow \{true, false\}$ is defined as follows:

$$\begin{aligned}
\text{isAgreement}(m, C) = & \\
& m \in \text{iris}(C) \wedge \forall p \in \text{accepts}(m, C) : & (7) \\
& \quad \text{isProposal}(p, C) = true \\
& \quad \wedge \text{happenBefore}(p, m, C) = true \\
& \quad \wedge \text{sender}(m, C) \neq \text{sender}(p, C).
\end{aligned}$$

`isAgreement` is true for message m in C if that message accepts a proposal that was made earlier by the counterpart of the sender of m .

`isProposal` and `isAgreement` only regard messages before their input message m , hence they can be evaluated over $C(m)$ instead of C without changing the result. Doing that is actually required if we allow retraction of messages (using \mathfrak{S}_{mod}), because messages added to C after m may change the result of `isProposal` and `isAgreement`. Therefore, for evaluating agreements over \mathfrak{S}_{mod} , we have to do it at point m in the conversation, or $\mathfrak{S}_{mod}(C(m))$, which is used in the following definitions.

As we want to allow agreements to be cancelled later, we have to distinguish between valid and canceled agreements. The function `isValidAgreement` : $\mathcal{I} \times \mathcal{C} \rightarrow \{true, false\}$ is defined as follows:

$$\begin{aligned}
\text{isValidAgreement}(m, C) = & \\
& \text{isAgreement}(m, \mathfrak{S}_{mod}(C(m))) = true \wedge \neg \exists c \in \text{iris}(C) : & \\
& \quad \text{isAgreement}(c, \mathfrak{S}_{mod}(C(c))) = true \wedge \exists p \in \text{iris}(\mathfrak{S}_{mod}(C(c))) : & \\
& \quad \quad p \in \text{accepts}(c, \mathfrak{S}_{mod}(C(c))) \wedge m \in \text{cancels}(p, \mathfrak{S}_{mod}(C(c))). & (8)
\end{aligned}$$

`isValidAgreement` is true if agreement m was not canceled by a later agreement c , which would have to accept a message p that proposes to cancel m . Note that c is itself not checked for validity this way, so if c does cancel m , and c is itself cancelled later, m remains cancelled. There is no way to restore a cancelled agreement.

Now that we have defined valid agreements, we can proceed to show how to calculate the content of one agreement, and how to calculate all contents of all agreements in the conversation, which is the output of \mathfrak{F}_{agr} .

The function `clauses` : $\mathcal{I} \times \mathcal{C} \rightarrow \mathcal{P}(\mathcal{I})$, yielding the IRIs of all clauses in an agreement (valid or invalid), is defined as follows:

$$\begin{aligned} \text{clauses}(m, C) &= \\ &= \begin{cases} \{c \in \mathcal{I} \mid c \in \text{proposes}(p, \mathfrak{S}_{mod}(C(m)))\}, \\ \quad p \in \text{accepts}(m, \mathfrak{S}_{mod}(C(m))) & \text{if } \text{isAgreement}(m, \mathfrak{S}_{mod}(C(m))) \\ \emptyset & \text{otherwise} \end{cases} \end{aligned} \quad (9)$$

The function `agreementContent` : $\mathcal{I} \times \mathcal{C} \rightarrow \mathcal{D}$ is defined as follows:

$$\text{agreementContent}(m, C) = \bigcup_{c \in \text{clauses}(m, \mathfrak{S}_{mod}(C(m)))} \text{content}(c, \mathfrak{S}_{mod}(C(m))) \quad (10)$$

This is to say that the content of an agreement is one RDF graph, namely the union of all content graphs of all clauses in the agreement at the point m in the conversation, applying \mathfrak{S}_{mod} . Note that neither the content of the proposing message nor the content of the accepting message are added to the content of the agreement.

We can now define the agreement function $\mathfrak{F}_{agr} : \mathcal{C} \rightarrow \mathcal{D}$ as follows:

$$\mathfrak{F}_{agr}(C) = \bigcup_{a \in \{m \mid \text{isValidAgreement}(m, C)\}} \langle a, \text{agreementContent}(a, C) \rangle \quad (11)$$

The agreement function \mathfrak{F}_{agr} , applied to C_r , yields a dataset in which each graph corresponds to one valid agreement in the conversation. The name of each such graph in the result dataset is the IRI of the message that accepted the agreement.⁸ The triples in each such graph are the union of the triples of all content graphs in all clauses of the agreement. Note that if the agreement has no clauses (i.e., it only cancels other agreements), the corresponding graph is empty.

This construction allows for identifying the messages pertaining to an agreed-upon graph by looking up the agreement IRI and following `agr:accepts` and `agr:proposes`. Likewise, it is simple to ask for canceling an agreement: the IRI required to do that is the IRI associated with the agreement's graph in $\mathfrak{F}_{agr}(C_r)$.

⁸ Thus, the IRI of the accept message denotes two different graphs in two different datasets - it refers to the accept message in the conversation dataset and to the agreement in the agreement dataset. We choose to accept this ambiguity.

5 Discussion

The Web of Needs protocols have been implemented⁹ and can be tested on a public demonstrator¹⁰ The work at hand is part of an effort to apply WoN in the transportation domain, in which we plan to use agreements for the coordination of transportation jobs (e.g., setting/changing a pick-up or delivery appointment). At this point, we have not implemented agreements yet, so we cannot provide an experimental evaluation of the approach at hand. We will proceed to implementing the extensions described here and evaluate their applicability for that use case in simulations, a case study, and finally in field study. In this work, we reflect on our solution by discussing some of our design decisions, which we do in the following.

Cascading retracts. The option to retract earlier messages requires the special case to be considered in which the retracted message is itself a retracting one. In designing the protocol, we considered the options to a) to disallow such messages (leading to a failure of the message) b) to let such messages not have any cascading effect, or c) to let a retract have a cascading effect. Moreover, we considered enabling a restore operation for retracted messages in combination with a) and b).¹¹ Our decision was guided by simplicity of computation and by an intuitive evaluation of the importance for users. We decided not to support restoring retracted messages as this does not seem to be an important feature of a communication application (judging from state of the art messaging applications), and because such a feature may enable deceiving an unsuspecting user by retracting a message and later silently restoring it. Consequently, we decided to avoid cascading retracts. The computationally simplest solution we found was to define a message as retracted if there is a later message retracting it, without checking if that message is retracted, and in addition to that to remove the retracting message itself from the modified selection. The same reasoning was applied for designing the cancelling of agreements: it is not possible to restore a cancelled agreement, and cancelling has no cascading effect.

Granularity of modifications. When considering the modification of the conversation content, the decision on the granularity of the modifications has to be made. We are aware of current discussions of the related patch functionality¹² in LDP[24], which is to allow triple-level modifications. We decided only to support retraction of whole messages for two reasons. For one, the approach is simple to implement yet sufficient for all changes, and second, it is easy to understand what happened for a human user. Modification on the triple level may be much harder to keep track of, possibly adding an attack vector, e.g. for introducing unnoticed triples into an agreement.

⁹ See <https://github.com/researchstudio-sat/webofneeds/> [2017/07/28].

¹⁰ See <https://matchat.org/> [2007/07/28].

¹¹ Interested readers are referred to the discussion on the topic of negotiations on the semantic Web Mailing list for more background. See <https://lists.w3.org/Archives/Public/semantic-web/2017Jul/0004.html> [2017/07/07].

¹² See <https://dvcs.w3.org/hg/ldpwg/raw-file/ldpatch/ldpatch.html> [2017/07/13].

Retraction may fail. All messages can fail or be ignored. Worse, an authenticated byzantine participant [8] operating a WoN node may deliberately choose to lose certain messages or let them fail. This may lead to unfair situations. For example, let an owner make a proposal, then notice a mistake and send a retract, and an authenticated byzantine remote WoN node, colluding with the owner’s counterpart, ignore the retract message. The result is that the proposal is not retracted, and the counterpart can still accept the proposal. One strategy to counter this attack is to have the owner’s WoN node ignore the accept message from the counterpart as long as the SUCCESSRESPONSE for the retract message has not been received. In a non-fraudulent setting, however, the owner has no way to make the WoN node do that. We currently do not have a solution to this problem, other than that one should double check a proposal before sending it, especially if one cannot be sure counterpart’s WoN node is uncompromised.

Proposing proposals. Technically, it is possible to propose a proposal or an already agreed-to agreement. The effect is that the content graph of the proposal or agreement message can become the content of an agreement. The protocol could be adapted to prevent such agreements, but we do not see any harm done by them.

No special message type for retraction and agreement. One may argue for at least some of the messages we introduce in this paper to be realized with new top-level message types. We decided against this and rather opted for a clear separation of communication layers. The basic protocol is used for making connections and exchanging messages. Retraction as well as reaching agreements is about a client-side interpretation of the message history, and consequently we decided to realize that functionality in a separate higher layer.

6 Conclusion and Outlook

In the work at hand, we propose a new way to interpret a Linked Data based conversation between agents in the Web of Needs as a shared, add-only RDF database. We introduce new protocol layers as views of the raw conversation data available to each participant. These layers provide a) a cleaned-up view, removing the content of failed and ignored messages and b) a modified view, allowing participants to retract messages they sent earlier. Using these two layers, we introduce c) an agreement view, enabling the participants to produce a mutually agreed-upon RDF dataset consisting of the RDF payload of selected earlier messages.

The contributions of this work represent one step toward a generic, decentralized (or federated) protocol that supports natural language conversations as a special case of the exchange of arbitrary RDF structures between participants. Therefore, the agreements introduced in this paper may consist of natural language clauses or any other RDF content, supporting negotiations between humans, of humans and software agents, and among software agents.

A related extension currently being designed is an additional protocol allowing agents to define information requirements using SHACL shapes, thereby

allowing to ask for certain information in a non-ambiguous, machine-processable way. We envision the use of intelligent client-side assistants that may assume different responsibilities such as filling in already known data like the user’s home address etc., or organizational tasks like managing appointments on behalf of the user.

We are planning to implement the proposed protocol extensions and evaluate them in the context of transportation by connecting Web APIs of transportation companies to WoN via specifically designed bots bridging between the two systems.

Acknowledgements

We would like to express our appreciation for valuable criticism and ideas contributed by Soheil Human, Fabian Suda and Kevin Singer, members of the team at SAT and to Eike Walsdorff for guidance on improving the formal definitions. We would also like to thank the participants of a discussion on the semantic Web mailing list on negotiations for sharing their expertise.¹¹ Finally, this work would not have been possible without the funding in project *Open Logistics Networks (OLN)*, funded by the Austrian Research Promotion Agency in the COIN program.

References

1. L. Amgoud, Y. Dimopoulos, and P. Moraitis. A unified and general framework for argumentation-based negotiation. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 158. ACM, 2007.
2. C. B. Aranda, O. Corby, S. Das, L. Feigenbaum, P. Gearon, B. Glimm, S. Harris, S. Hawke, I. Herman, N. Humfrey, N. Michaelis, C. Ogbuji, M. Perry, A. Passant, A. Polleres, E. Prud’hommeaux, A. Seaborne, and G. T. Williams. Sparql 1.1 overview, March 2013. [Last accessed on 2016/06/06].
3. C. Bartolini, C. Preist, and N. R. Jennings. Architecting for reuse: A software framework for automated negotiation. In *AOSE*, pages 88–100. Springer, 2002.
4. M. Bichler, G. Kersten, and S. Strecker. Towards a structured design of electronic negotiations. *Group Decision and Negotiation*, 12(4):311–335, 2003.
5. M. Bichler, G. Kersten, and C. Weinhardt. Electronic negotiations: Foundations, systems and experiments—introduction to the special issue. *Group Decision and Negotiation*, 12(2):85–88, 2003.
6. J. Brandts, M. Ellman, and G. Charness. Lets talk: How communication affects contract design. *Journal of the European Economic Association*, 14(4):943–974, 2015.
7. T. Bui, A. Gachet, and H.-J. Sebastian. Web services for negotiation and bargaining in electronic markets: design requirements, proof-of-concepts, and potential applications to e-procurement. *Group Decision and Negotiation*, 15(5):469–490, 2006.
8. X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, Dec. 2004.

9. J. Gu and X. Zhu. Designing and implementation of an online system for electronic contract negotiation based on electronic signature. *Journal of Software*, 9(12), 2014.
10. N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, M. J. Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.
11. G. E. Kersten and H. Lai. Negotiation support and e-negotiation systems: An overview. *Group Decision and Negotiation*, 16(6):553–586, 2007.
12. F. Kleedorfer, C. M. Busch, C. Huemer, and C. Pichler. A linked data based messaging architecture for the web of needs. *Enterprise Modelling and Information Systems Architectures*, 11(3):1 – 18, 2016.
13. F. Kleedorfer, C. M. Busch, C. Pichler, and C. Huemer. The case for the web of needs. In *Business Informatics (CBI), 2014 IEEE 16th Conference on*, volume 1, pages 94–101. IEEE, 2014.
14. F. Kleedorfer, Y. Panchenko, C. M. Busch, and C. Huemer. Verifiability and traceability in a linked data based messaging system. In *Proceedings of the 12th International Conference on Semantic Systems, SEMANTiCS 2016*, pages 97–100, New York, NY, USA, 2016. ACM.
15. H. Knublauch and D. Kontokostas. Shapes constraint language (shacl) - proposed recommendation, 6 2017. [Last accessed on 2017/07/06].
16. S. Kraus. Automated negotiation and decision making in multiagent environments. *Easss*, 2086:150–172, 2001.
17. R. Lin and S. Kraus. Can automated agents proficiently negotiate with humans? *Communications of the ACM*, 53(1):78–88, 2010.
18. F. Manola, E. Miller, and B. McBride. Rdf 1.1 primer, 2014. [Last accessed on 2017/07/26].
19. M. Nečaský, J. Klímek, J. Mynarz, T. Knap, V. Svátek, and J. Stárka. Linked data support for filing public contracts. *Computers in Industry*, 65(5):862–877, 2014.
20. M. Parkin, D. Kuo, and J. Brooke. A framework & negotiation protocol for service contracts. In *Services Computing, 2006. SCC'06. IEEE International Conference on*, pages 253–256. IEEE, 2006.
21. O. U. Press. Agreement. Online, 2017.
22. I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. Mccburney, S. Parsons, and L. Sonnenberg. Argumentation-based negotiation. *The Knowledge Engineering Review*, 18(4):343–375, 2003.
23. M. Schoop, A. Jertila, and T. List. Negoisst: a negotiation support system for electronic business-to-business negotiations in e-commerce. *Data & Knowledge Engineering*, 47(3):371–401, 2003.
24. S. Speicher, J. Arwe, and A. Malhotra. Linked data platform 1.0, 2013.
25. M. Ströbel and C. Weinhardt. The montreal taxonomy for electronic negotiations. *Group Decision and Negotiation*, 12(2):143–164, 2003.
26. V. Tamma, M. Wooldridge, I. Blacoe, and I. Dickinson. An ontology based approach to automated negotiation. In *International Workshop on Agent-Mediated Electronic Commerce*, pages 219–237. Springer, 2002.
27. H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, and J. J. Kishigami. Blockchain contract: A complete consensus using blockchain. In *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on*, pages 577–578. IEEE, 2015.
28. H. Weigand, M. Schoop, A. de Moor, and F. Dignum. B2b negotiation support: The need for a communication perspective. *Group Decision and Negotiation*, 12(1):3–29, 2003.

29. X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen. The blockchain as a software connector. In *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*, pages 182–191. IEEE, 2016.