Software ENgineering

Construction of negotiation protocols for e-commerce applications

Z.V. Zlatev, N.K. Diakov, S.V. Pokraev

REPORT SEN-R0417 OCTOBER 2004

ISSN 1386-369X

Construction of negotiation protocols for e-commerce applications

ABSTRACT
A company doing e-business needs capabilities to negotiate electronically the parameters of its deals in order to fully utilize the potential of the Information and Communication technology (ICT). This paper focuses on construction of negotiation protocols for e-commerce with the following properties: (a) formal semantics, (b) compositionally constructed, and (c) dynamically reconfigurable. We apply the Reo coordination language to demonstrate that one can specify and implement negotiation protocols possessing the desired properties.

1998 ACM Computing Classification System: C.2.4, D.1.3, D.3.2, D.3.3, F.3, J.4
Keywords and Phrases: Automated negotiation, negotiation protocols, coordination, Reo language

Construction of Negotiation Protocols for E-Commerce Applications

Zlatko Zlatev
Centre for Telematics and Information Technology University of Twente,
P.O. Box 217 7500AE, Enschede,
The Netherlands
Tel.: +31 (53) 489 5334
Z.V.Zlatev@ewi.utwente.nl

Nikolay Diakov
Centrum voor Wiskunde en Informatica
P.O. Box 94079, 1090 GB Amsterdam,
The Netherlands
Tel.: +31 (20) 592 4073
nikolay.diakov@cwi.nl

Stanislav Pokraev
Telematica Instituut
P.O. Box 589, 7500 AN Enschede,
The Netherlands
Tel.: +31 (53) 485 0490
stanislav.pokraev@telin.nl

ABSTRACT

A company doing e-business needs capabilities to negotiate electronically the parameters of its deals in order to fully utilize the potential of the Information and Communication technology (ICT). This paper focuses on construction of negotiation protocols for e-commerce with the following properties: (a) formal semantics, (b) compositionally constructed, and (c) dynamically reconfigurable. We apply the Reo coordination language to demonstrate that one can specify and implement negotiation protocols possessing the desired properties.

Categories and Subject Descriptors

C.2.4, D.1.3, D.3.2, D.3.3, F.3, J.4

Keywords

Automated negotiation, negotiation protocols, coordination, Reo language

1. INTRODUCTION

As a result of the diffusion of Internet technology in the mid-90s, the business world encountered a new disruptive possibility [9][11] to exchange data by computer networks at low cost. Like any disruptive technology, computer-based networking has changed business activities and constellations of businesses significantly. The changes in the fundamental equations of business models require rethinking of these models.

One of the business activities that require such rethinking is the coordination among business partners. In the analysis of the potential impact of ICT on business, Malone, Yates and Benjamin [19] predict that in an e-business environment more transactions will be executed through markets than among departments within one company. Not assessing the validity of their prediction, new types of businesses based on execution of market transactions have appeared and sustained; eBay is a good example of such an innovative business.

In markets, business activities are coordinated through price, which is the value a business assigns to a resource. Since various businesses assign different values to resources, they need to negotiate to reach mutually acceptable agreements. For this reason, we consider it important to enable businesses to do negotiation in an e-business environment. Many fields of research exist that study negotiation in an electronic environment, such as: the Information Systems field with negotiation support systems; Multi-Agent Systems field with searching, trading and negotiating agents; and Market Design field with electronic auctions.

There exist many conceptualizations of negotiation. For the purpose of this paper, we adopt the framework of Jennings et al. [15], which decomposes negotiation into the following elements: *negotiation protocols*, *negotiation objects*, and *agents' decision-making models*. Negotiation protocols define the rules that govern the negotiation. Negotiation objects represent the matter the participants negotiate over; i.e. goods or services. Agents' decision-making models are the decision-making apparatus that the participants employ to achieve their objectives in accordance with the protocol. In this paper, we focus on the negotiation protocols.

One can classify negotiation according to various criteria [26], e.g., the number of negotiable parameters of a deal, negotiation environment, the types of participating businesses, and the number of participants. Most of these parameters affect the negotiation protocol. As a motivating case, we choose for an open and dynamic negotiation environment: one with varying number of participants of various types who may join and leave at arbitrary time during negotiation. An example of such a negotiation is an on-line auction, where: (1) anyone can take part in bidding; (2) a bidder can be a person or a software system; and (3) any combination of sellers, auctioneers or bidders is allowed.

We identify [21] three problems with negotiation protocols in an open electronic negotiation environment:

1. *Lack of a common understanding of the protocol*. Participants in a negotiation process should have a common understanding of the protocol they have to follow. We consider the source of this problem the use of protocol specification languages with poor formal semantics. One approach to this problem is to use a formal description of the negotiation protocol;
2. *Lack of support for temporary business constellations*. In a many-to-many negotiation, participants can form alliances. These alliances are not stable because the shared interests are only temporary; they can break in the course of the negotiation process. Moreover, participants may leave and join the negotiation process at an arbitrary moment in time. Supporting a negotiation protocol with these characteristics requires its implementation to have the ability to adapt to the changes in the negotiation environment;

3. *Inability to deal with nested negotiations*. Apart from the negotiation among participants, there is a negotiation of similar complexity prior to forming an alliance. Moreover, negotiation is required within an alliance to prepare each new agreement proposal. The alliance formation and proposal preparation are separate negotiation processes. Nested negotiation requires the ability to compose one negotiation protocol with other ones in a systematic way.

In order to facilitate an open and dynamic negotiation environment, we consider the following requirements for a protocol specification language:

- Formal semantics – to accommodate heterogeneity of participants, i.e. businesses with different perception of the surrounding environment, we require the use of a language with formal semantics. This requirement addresses problem 1;
- Dynamic reconfigurability – this allows to accommodate dynamic changes in the negotiation environment, such as changing number or type of participants and forming or dissolving of alliances. This requirement addresses problem 2;
- Composability – this allows to design protocols in a modular style. Furthermore, composability allows to build a negotiation protocol that includes nested independent negotiations. Furthermore, composability during runtime enhances the previous requirement by allowing one to dynamically add new protocols, switch protocols, etc. This requirement addresses problem 2 and 3.

In this paper, we apply the Reo coordination language [5] to demonstrate that one can specify and implement negotiation protocols that possess the above-mentioned properties. Reo has formal semantics, but also a formal computational model that defines the rules according to which one can execute a particular specification in Reo. We refer to a formal computational model as an *operational semantics*.

The rest of the paper has the following organization. We introduce the Reo coordination paradigm and language. Then, we present an example auction protocol and construct its implementation using Reo. After that, we discuss our observations on using Reo with negotiation protocols. We overview related work. At the end, we summarize our results and outline future work.

2. THE REO COORDINATION PARADIGM

Reo presents a paradigm for composition of software components based on the notion of channels. Reo enforces a channel-based coordination model that defines how designers can build complex coordinators, called connectors, out of simpler ones. Application designers can use Reo as a 'glue code' language for compositional construction of connectors that orchestrate the cooperative behavior of component instances in a component-based system [5][6]. The Reo coordination language provides, among others, the following features:

- Loose coupling among components;
- Support for distribution and mobility of heterogeneous components;
- Exogenous coordination (i.e., by third parties);
- Dynamic reconfigurability that allows one to change a connector during runtime using topological operations;
- Formal semantics based on a coinductive calculus of flow [7][2][22] and (alternatively) on constraint automata [3].
- Formal operational semantics that defines the rules for computing Reo connectors in a distributed computing environment [13];
- A serialization of its visual notation in XML validated by XML Schema, for interoperability between analysis tools;
- A comprehensive visual notation.

2.1 Basic Concepts

From the point of view of Reo, a system consists of a number of *component instances*, interacting through *connectors*. Reo assumes that a component instance contains one or more active entities (e.g., processes, agents, threads, actors, etc.), which communicate with entities outside of their component instance only through connectors. Reo completely abstracts from the details of the communication within a component instance. Instead, Reo focuses on the communication between component-instances, which takes place exclusively through connectors. Reo allows compositional construction of a connector out of simpler connectors, where *channels* represent the *atomic connectors*.

A channel has precisely two *channel ends*. Reo introduces two types of channel ends: sink and source. A sink dispenses data out of its channel. A source accepts data into its channel.

Reo models a connector as a graph of nodes and edges, where zero or more channel ends may coincide on every node, every channel end coincides on exactly one node, and an edge exists between two nodes if and only if there exists a channel whose channel ends coincide on those nodes.

Reo has three types of nodes: *mixed*, *source*, and *sink*. A mixed node contains both source and sink channel ends. A mixed node serves as a pumping station for its coincident channel ends: it non-deterministically selects a data item available at one of its sink channel ends and replicates the data item to all of its source channel ends when all of them can accept the data item. A source node contains only source channel ends. If a component writes a data item to a source node, the node replicates the data item to all of its source channel ends when all of them can accept the data item. A sink node contains only sink channel ends. When a component tries to take a data item from a sink node, the node non-deterministically selects a data item available at one of its sink channel ends.

2.2 Reo Operations

Any active entity inside a component instance can perform Reo operations on a channel end. Reo defines two types of operations: topological – ones that allow manipulation of connector topology, and IO – ones that allow input/output of data. Due to space limitation, we discuss only operations that we use later in the text.

Topological operations include, among others, *join* and *split*. The join operation allows joining of two nodes identified by two channel ends, each coincident with one of the nodes. The split operation allows for splitting a node into two nodes by specifying the channel ends that the performer requires to coincide on the new nodes.

IO operations include, among others, *take* and *write*. The take operation allows the performer to read and remove a data item from a sink. The write operation allows the performer to write data to a source.

2.3 A Useful Set of Primitive Channels

Reo assumes the availability of an arbitrary set of channel types, each with well-defined behavior. In this paper, we consider the following non-exhaustive set of channel types, each with some distinctive properties: Sync, Filter, SyncDrain, LossySync, FIFO1. A Sync channel has a source and a sink. Writing a message succeeds on the source of a Sync channel if and only if taking of a message succeeds at the same time on its sink. The Filter channel behaves like the Sync except that it loses all data that do not match the specified pattern of the Filter. A SyncDrain has two sources. Writing a message succeeds on one of the sources of a SyncDrain channel if and only if writing a message succeeds on the other source. A LossySync channel has a sink and a source. The source always accepts all data items. If the sink does not have a pending read or take operation, the LossySync loses the data item, otherwise the channel behaves as a Sync. The Sync, Filter, SyncDrain, SyncSpout and LossySync belong to the family of synchronous channels. The FIFO1 channel has a source and a sink. The FIFO1 channel maintains a buffer with capacity of one. Writing a message succeeds on the source of a FIFO1 if and only if its buffer does not contain any messages. Taking of a message succeeds on the sink of a FIFO if and only if its buffer already contains a message. The FIFO1 belongs to the family of asynchronous channels. Figure 1 shows the visual notation for the basic channels we introduced above.

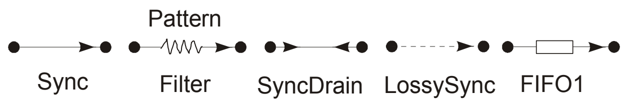

Figure 1. Visual notation for basic channels

2.4 Connector Encapsulation

In analogy with electrical circuits, we call a design a *circuit* in Reo. When designing large circuits, we find it useful to abstract from the details of a particular connector that we want to instantiate and reuse. In Reo, we do this through encapsulating the circuit of a connector. In a visual notation we represent an encapsulated circuit using a box. On the borders of the box, we position the nodes that the connector exposes to the outside. In effect, the box represents a new component, which internal behavior is entirely defined in Reo.

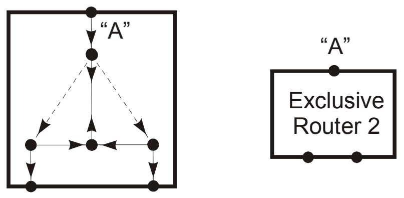

Figure 2. Exclusive Router 2 connector (left) and one of its instances (right)

Figure 2 shows how we define the Exclusive Router 2 connector [2]. The Exclusive Router 2 routes synchronously its input to precisely one of its outputs. We also show how we depict an instance of the Exclusive Router 2. Note that we may label nodes on the border of a connector, in order to assign some designer meaning to the messages passing through that node. This labeling serves only to simplify the presentation of a circuit, and to allow designers to distinguish the role of the nodes, i.e., input or output. It has no implications on the semantics of the circuit.

3. EXAMPLE: AN AUCTION FOR A TRANSPORTATION SERVICE

In this section, we apply Reo in an example business case that includes negotiations that require protocols with the properties listed in section 1.

3.1 Business Case

A company called DeRio produces a certain range of products. DeRio has recently closed a deal with a new partner situated at a distant location. To fulfill its contractual obligations, DeRio needs a transportation service from its factory to the customer address. Figure 3 shows a possible transportation route between point A, the factory and point E, the customer address.

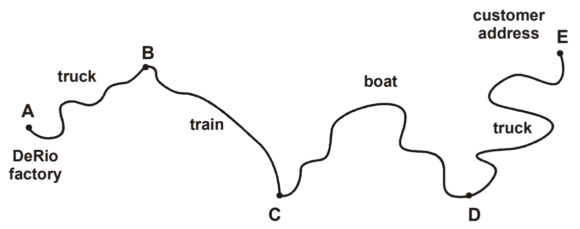

Figure 3. Transportation route

DeRio decides to set up an auction to determine the best service offer. All parameters of the services, such as delivery deadline and payment terms, are fixed. The price is the only negotiable parameter. Several companies respond to the announcement; among them are DHS and UPL, widely recognized players in the transportation domain and Neptun and Aviz, companies specializing in particular kind of transport, e.g. water or land transport. Neptun and Aviz cannot compete individually with DHS and UPL for some reason (e.g., Neptun only offers boat and train transport at good prices, while Aviz offers only truck transport at good prices). To gain competitive advantage, they decide to form an alliance called AlNeviz. Figure 4 shows a sch matic setup of the auction, the participants and the alliance.

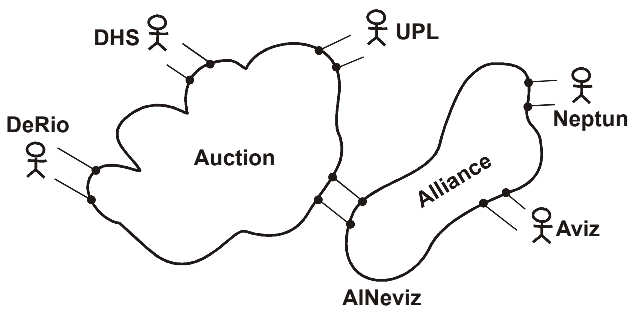

Figure 4. DeRio auction setup

Our business case includes two negotiations, namely: a negotiation in a form of auctioning among bidders and a negotiation among partners within an alliance. Due to space limitation, we specify and implement only the auction protocol.

3.2 Negotiation Protocol

In our example business case, we consider three distinct roles that participants can play:

- *Initiator* – there is only one instance per auction – played by DeRio;
- *Auctioneer* – there is only one instance per auction – played by DeRio or a third party owning some auction infrastructure;
- *Bidder* – there are no restrictions on the number of instances per auction – in this case DHS, UPL, and AlNeviz alliance perform as bidders.

Below, we list an informal specification of an interpretation on a procurement English auction [23] protocol. The numbering is an enumeration; it does not suggest a particular sequence in applying. When we introduce a term for the first time we show it in *italic*:

1. The initiator determines an *initial* price. This is the maximum price that the initiator is willing to pay for the service. The initial price is positive real number;
2. The initiator determines a *bid-step*. The bid-step is a positive real number;
3. The auctioneer initializes and starts the auction;
4. During initialization, the auctioneer sets the auction's current price to the initial price determined by the initiator;
5. During initialization, the auctioneer sets the auction's *valid-bid* constraint to the bid-step indicated by the initiator;
6. Once per auction and at any time, the auctioneer may remove the valid-bid constraint on bids by setting the bid-step to zero;
7. The change of the bid-step is announced to all bidders;
8. Before participating in the auction, every bidder must register. A registration includes an approval by the initiator and payment of entrance fee;
9. Bidders may register at any time;
10. Upon registration a bidder is informed about the current price and bid-step;
11. Registered bidders may bid at any time and with any bid;
12. Only *valid* bids are taken into account. A valid bid is defined below;
13. A valid bid is a bid that is lower than the current price minus the bid-step. (Note that the bid-step may equal zero);
14. Upon acceptance of a valid bid two consecutive steps are made: (1) the current price is set equal to the valid bid and (2) the new current price is broadcasted to all bidders;
15. The auctioneer may close the auction at an arbitrary moment in time;
16. Upon auction closing, the auctioneer consequently (1) informs all bidders except for the winner that they have lost the auction and (2) informs the winner that it has won the auction.

3.3 Protocol Specification and Implementation

Using the specification from the previous section, we construct a Reo circuit that specifies the auction protocol. The specification also serves as an implementation, because Reo has a formal operational semantics. First, we introduce basic components; then, the larger auxiliary connectors; then, an external library of connectors; and at the end, we introduce the auction protocol circuit.

3.3.1 Basic Components

In addition to the basic channels introduced earlier, we need several basic components that can operate on the data passing through channels (Figure 5).

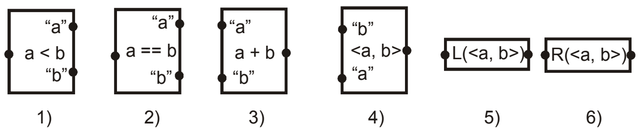

Figure 5. Basic components

The first component offers two source (input) nodes and one sink (output) node. We have labeled the sources with "a" and "b". When we write two messages *a* and *b* representing integers to "a" and "b" respectively, the component outputs "true" if $a < b$ and "false" otherwise, through its sink. The second component has the same characteristics as the first one; however, it outputs "true" if $a = b$ and "false" otherwise. The third component differs in that it outputs $a + b$. The fourth component differs in that it outputs a pair $<a, b>$ constructed of *a* and *b*, and that it does not require integers as input data type. The fifth and sixth components each offers one source and one sink node. The fifth component takes as input a pair and outputs the first elements of the input pair, while the sixth component differs in that it outputs the second element of the input pair. One can find the formal definition of the behavior of the third component in [6] using the Reo algebraic semantics [7]. One can easily define the other five components in a similar way.

3.3.2 Larger Connectors

Using the six basic components, we build three larger connectors that we use in the protocol implementation.

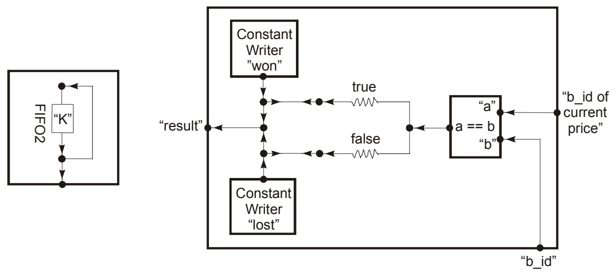

Figure 6. Constant Writer "K" (left) and Result Generator (right)

The Constant Writer connector (Figure 6) offers a single sink node. The Constant Writer "K" represents a component that always offers a message containing the constant *K* to anyone that makes a take operation on the component's sink. We use this component in the implementation of the Result Generator connector. The Result Generator connector (Figure 6) offers two source nodes and one sink. The Result Generator represents a component that takes as input two integers and if the two integers are equal it outputs a "won" message, otherwise it outputs a "lost" message. We use this component when we want to notify bidders about the outcome of the auction.

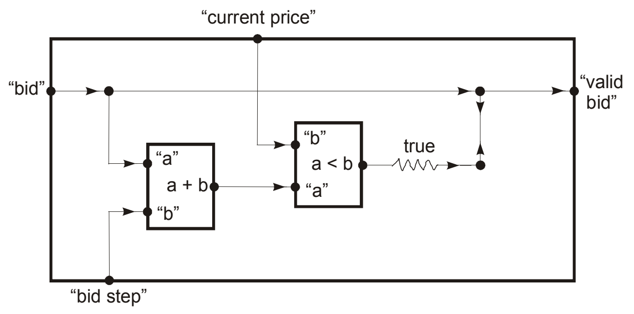

Figure 7. Bid Validator

The Bid Validator connector offers three source nodes and one sink node. The Bid Validator represents a component that takes as input the current price in the auction, the next bid, and the current bid step, and outputs the bid it received as input, if and only if the value of the bid meets the requirements described in rule 13 of the auction protocol. We use this component to determine the validity of a bid made by bidders.

3.3.3 Library of Connectors

We depict in gray color all components in the auction protocol circuit that we refer to [2] for their definition. These components constitute: Initially Closed Valve (ICV), Initially Opened Valve (IOV), Shift Lossy FIFO1 (SLFIFO1), and Exclusive Routers with more than two outputs. Both valves regulate the flow of data; however, the ICV initially does not allow flow, while IOV initially allows flow. Both valves offer a node through which one can toggle a valve's state from opened to closed and the other way around. The SLFIFO1 behaves similarly to FIFO1 except that when full, the SLFIFO1 throws away its current data item to accept in its buffer any new data item. An exclusive router of order higher than two, routes to more than two nodes.

3.3.4 Auction Protocol Circuit

Figure 8 shows the visual specification of the auction protocol circuit. Using tools, one can obtain an XML serialization of this specification. Furthermore, one can also obtain a complete algebraic specification by applying the Reo algebraic semantic rules for channel composition to all channels and component instances in the circuit.

The Auction protocol circuit consists of one instance of the Initiator connector, one instance of the Auction connector, and one or more instances of a Bidder connector. The Initiator connector coordinates the activities of the initiator – DeRio. The Auction connector coordinates the activities of the auctioneer – DeRio or a third party. The Bidder connector coordinates the activities of a bidder – DHS, UPL, and AlNeviz alliance. For simplicity, Figure 8 depicts only one Bidder instance.

The initiator uses the labeled nodes exposed by the Initiator connector to submit values for initial price and bid step (in any order). These two values constitute the necessary conditions for starting of the auction by the auctioneer.

The Initiator connector automatically authenticates bidders known to the initiator, namely: DHS, UPL, and AlNeviz. For this, we use a Filter channel with a pattern that can recognize the three bidders. The initiator can change the bidders that its circuit can authenticate by using Reo dynamic reconfigurability feature to replace the filter with another one (with a new pattern) at runtime. First, the initiator splits the node at the beginning of the filter to disconnect the filter from the rest, and then joins the nodes of a new filter with the previously split node. Reo takes care to maintain the circuit operation before, during, and after this change. While disconnected the initiator circuit cannot authenticate anyone. One can envisage a more sophisticated authentication mechanism, e.g., including look-up in an external database. We prefer a simpler solution, because supporting an elaborate authentication scheme does fall into the scope of this work.

A bidder uses the labeled nodes exposed by the Bidder connector to register and to pay the fee of 200 payment units. Upon successful registration, the bidder can place bids. The bidder receives auction information about the bid step, the current price, and upon closing of the auction – whether it won or lost the auction.

In the case a new bidder appears, the auctioneer instantiates an additional Bidder instance. Then, the auctioneer joins the respective sink and source nodes of the Bidder instance with nodes A, B, C, D, and F (using auxiliary Sync channels). With E, we denote the set of ninety-nine nodes offered by the Auction connector using an Exclusive Router 99. Technically, this limits the current circuit to 99 bidders that can joint the auction; however, if needed, the auctioneer can use Reo dynamic reconfigurability to dynamically upgrade the Exclusive Router 99 to one with capacity 100. The auctioneer can do this by adding an Exclusive Router 2 instance to one of its available sink nodes, using the join operation. The auctioneer can perform this operation many times during runtime. Each Bidder instance connects one of its nodes to precisely one of the nodes in E, and every node in E can have zero or one Bidder instance connected to it.

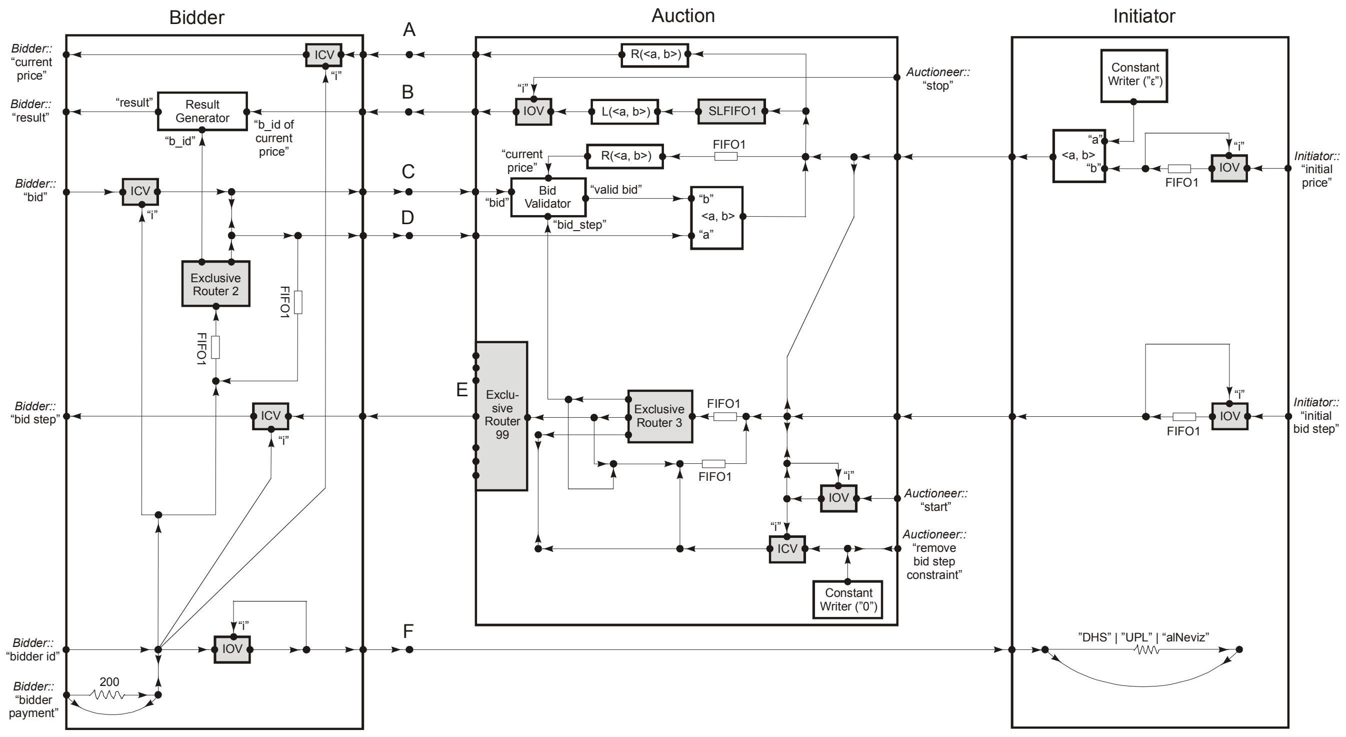

Figure 8. The Auction protocol circuit

The auctioneer uses the labeled nodes exposed by the Auction connector to start and stop the auction. Furthermore, the auctioneer can (once) remove the bid step constraint by setting it to zero.

In this implementation, we have made several technical choices that the informal protocol description does not specify in detail. When facing such technical choices, we prefer the simpler ones in terms of Reo primitives used to implement the circuit. Furthermore, because we want to keep the implementation comprehensible, we do not check for proper input values nor do we perform error handling.

4. OBSERVATIONS ON USING REO WITH NEGOTIATION PROTOCOLS

In this section, we present several observations, we made during working with Reo on the auction protocol. Furthermore, we give an idea about the additional possibilities that become available to protocol designers, should they decide to select Reo as their specification and implementation language.

The Reo coordination paradigm allows for strict separation of coordination from data processing. Therefore, we can assess the amount of pure coordination in comparison with data manipulation in the auction protocol. Gray components and all channels from Figure 8 (the auction protocol circuit) represent coordination elements. The white components perform some data processing. One can see the significant amount of coordination that occurs in this implementation of the auction protocol. Using a coordination language, such as Reo, which treats coordination as a first class modeling concept, allowed us to deal more efficiently with the coordination issues in the auction protocol. This observation illustrates that designers can use Reo also for modeling of other e-commerce applications that involve significant amount of coordination.

The strong formal apparatus behind Reo gives the following additional advantages:

- Simulator for Reo – a tool that implements a non-distributed version of the Reo operational semantics, to allow running and testing protocol prototypes;
- Distributed coordination middleware for Reo – a tool (under development) that implement the full Reo operational semantics to allow one to run Reo circuits directly in a distributed environment. The Reo coordination middleware allows plugging in of different component models for structuring the user applications into entities, and of transport protocols for the communication between the entities at the level of channels;
- Model checker – Time Scheduled Data Stream Logic [4] represents an existing theory that a model checker tool (under development) can use to check properties, such as, liveness, reachability, and deadlock conditions.

5. RELATED WORK

Several coordination languages have been proposed and used for negotiation protocol specification. Two such languages from the Multi-Agent Systems field are AgenTalk [18][17] and COOL [8]. They are based on Finite State Machines (FSM) and used in negotiation protocol specification. Nevertheless, they are limited in their expressiveness and not suitable for the negotiation environment described in section 1. The AgenTalk language does not have formal semantics [18] and does not support dynamic reconfigurability and composability. AgenTalk has an inheritance mechanism as means for reuse of design but this is not dynamic. The COOL language uses FSM as its formal basis, where the transitions are exchanges of speech-act-based messages. COOL does not have formal operational semantics and does not have language primitives for dynamic reconfiguration.

Another group of specification languages originates from the Web services initiative. One example is the Business Process Execution Language for Web Services (BPEL4WS)[12]. BPEL4WS combines the formalisms used in its predecessors, namely the support for graph oriented processes from WSFL and the structural constructs for processes from XLANG [25]. BPEL4WS is designed for composition of Web services; although, it lack the dynamic reconfigurability. This and other drawbacks are discussed by Kim et al. [16] and Van der Aalst [1].

Web Services Conversation Language (WSCL) [24] is language that specifies the documents exchanged and the sequence the document exchanges. WSCL is limited in its expressiveness; it limits the number of the participants in a conversation to two, does not support parallel activities, and decision activities can only use as conditions the output of the a preceding activity. WSCL does not have language primitives for dynamic reconfiguration of the conversation protocol.

Business Process Modeling Language (BPML) [10] is language similar to XLANG. BPML has a well-defined but not formal semantics. It provides composability, transactions, executable specifications, dynamic participation, etc. However, BPML does not have language primitives for dynamic reconfiguration.

General formal modeling techniques exist, such as π-calculus, data-flow models, Kahn-networks, and Petri-nets. We view them as specialized channel-based models that incorporate certain specific primitive coordination constructs [6]. Recent work [14] on comparing Reo and Petri-nets, showed that one can relatively easy transform existing Petri-net models into a Reo circuit, while the opposite proves to be difficult. In our view, the inherently dynamic topology of connectors and the very liberal notion of channels make Reo more general, and hence more powerful.

6. CONCLUSIONS

In this paper we have presented an approach for specification of negotiation protocols using the Reo coordination language. We use the inherent features of Reo, such as composability and dynamic reconfiguration, to produce a specification of an auction protocol that can also serve as a protocol implementation in a dynamic and open negotiation environment. In this we blur the fine line between a protocol model and a protocol implementation, because given a proper coordination middleware, Reo gives the implementation for free.

The formal apparatus for reasoning about Reo circuits includes algebraic and co-algebraic methods [7][22], constraint automata theory [3][4], structured operational semantics [20]. With this, we solve the problem regarding the equivalent interpretation of a protocol specification by every participant in a negotiation. Informal specification techniques, such as AgenTalk, WSCL, and BPML, cannot provide specifications with these properties.

Reo offers topological operations that one can use to modify a Reo circuit during runtime. These allow to switch negotiation protocols in runtime or to modify an operating negotiation protocol. We provide three examples of the latter case in section 3.3.4: (1) changing the authentication filter, (2) upgrading the capacity of an exclusive router, and (3) adding a new bidder to the protocol circuit. The dynamic reconfigurability feature in principle (we do not show it in this paper) allows support of temporary business constellations; e.g., when an alliance appears, we can modify the protocol to add support for the alliance, and when it disappears, we can modify it again to remove the alliance support. To our knowledge, only specialized channel-based models allow for some specific forms of dynamic reconfigurability.

The Reo composability and dynamic reconfigurability allow to design and to implement large protocols in a modular style. In such a way, we isolate and reuse coordination designs. This relates to both scalability (compositionality) and flexibility (dynamic reconfiguration) of system. As system grows (compositionally), its overall behavior remains predictable. As such, Reo composability allows for support of nested negotiation protocols. Non-composable languages, such as AgenTalk, COOL, BPEL4WS, and WSCL, do not allow one to derive properties of a composition from its constituent nested protocols.

7. FUTURE WORK

As future work, we plan to investigate the modeling of negotiation protocols that allow dynamic switching of coordination behavior. Furthermore, we intend to implement several alliance protocols and compose those with a negotiation protocol. In the case of auctions, we find interesting the following two types of alliance protocols: (a) an alliance of participants that appear as one participant to the auctioneer, and (b) an alliance that coordinates the direct interactions of participants with the auctioneer.

We want to deploy and study negotiation protocol implementations with and without alliances in a distributed environment. This will contribute to the point we make that Reo does not only constitute modeling paradigm, but also an executable paradigm in a distributed environment.

8. ACKNOWLEDGMENTS

We would like to thank Pascal van Eck, David Clarke, Farhad Arbab and Roel Wieringa for their comments.

9. REFERENCES

[1] Aalst van der, W.M.P., Don't go with the flow: Web services composition standards ex-posed, *Trends and Controversies*, IEEE Intelligent Systems, Jan/Feb, 2003.

[2] Arbab, F., Abstract Behavior Types: A Foundation Model for Components and Their Composition, In: *Proceedings of the First International Symposium on Formal Methods for Components and Objects (FMCO 2002)*, LNCS 2852, 2003, pp.33-70.

[3] Arbab, F., Baier, C., Rutten, J., Sirjani, M., Modeling Component Connectors in Reo by Constraint Automata, *Electronic Notes in Theoretical Computer Science*, vol. 97, No. 22, July, 2004, pp. 25-46.

[4] Arbab, F., Baier, C., de Boer, F., Rutten, J., Models and Temporal Logics for Timed Component Connectors, *Proc. of the IEEE International Conference on Software Engineering and Formal Methods (SEFM '04)*, Beijing, China, 26-30 September, 2004.

[5] Arbab, F., Mavadatt, F., Coordination through channel composition, In: *Coordination Languages and Models: Proc. Coordination 2002*, volume 2315 of Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 21-38.

[6] Arbab, F., Reo: A Channel-based Coordination Model for Component Composition, *Mathematical Structures in Computer Science*, Cambridge University Press, Vol. 14, No. 3, June 2004, pp. 329-366.

[7] Arbab, F., Rutten, J.J.M.M, A Coinductive Calculus of Component Connectors, In: *Proceedings of 16th International Workshop on Algebraic Development Techniques (WADT 2002)*, Lecture Notes in Computer Science 2755, Springer, 2003, pp. 35-56.

[8] Barbuceanu, M. and Fox, M.S., COOL: A language for Describing Coordination in Multi Agent Systems, *Proc. First International Conference on Multi-Agent Systems (ICMAS'95)*, 1995, pp. 17-24.

[9] Bower, J. L. & Christensen, C. M., *Disruptive technologies: catching the wave*, Harvard Business Review, 73, 1 (January-February) , 1995, pp. 43—53.

[10] BPMI.org, BPML 1.0 Specification, online available at: http://www.bpmi.org/specifications.esp

[11] Christensen, C.M., *The Innovator's Dilemma*, Harvard Business School Press, Boston, Mass, 1997.

[12] Curbera, F., Goland, Y.,Klein, J., Leyman, F., Roller, D., Thatte, S. and Weerawarana, S., Business Process Execution Language for Web Services (BPEL4WS) 1.1, May 2003, online available at: http://www.ibm.com/developerworks/library/ws-bpel/

[13] Everaars, K., Costa, D., Diakov, N.K., Arbab, F., Reo Rewrite Rules: Functional Specification, work in progress, presented at The Amsterdam Coordination Group (ACG), May, 2004.

[14] Guillen-Scholten, J., A Translation from Reo to Petri Nets and Vice-Versa, work in progress presented at the The Amsterdam Coordination Group (ACG), June, 2004.

[15] Jennings, N., Faratin, P., Lomuscio, A., Parsons, S., Sierra, C. & Wooldridge M. , Automated Negotiation: Prospects, Methods and Challenges, *Int Journal of Group Decision and Negotiation*, vol. 10, no. 2, 2001, pp. 199-215.

[16] Kim, J.B., Segev, A., Patankar, A.K., Cho, M.G., Web Services and BPEL4WS for Dynamic eBusiness Negotiation Processes, In: Zhang, L. (ed.), *Proceedings of the International Conference on Web Services*, ICWS '03, Las Vegas, Nevada, USA. CSREA Press 2003, 2003, pp. 111-117, ISBN 1-892512-49-1.

[17] Kuwabara, K., Ishida, T., and Osato, N., AgenTalk: Describing Multiagent Coordination Protocols with Inheritance, *Proc. 7th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '95)*, 1995, pp. 460-465.

[18] Kuwabara, K., Ishida, T., and Osato, N.,AgenTalk: Coordination Protocol Description for Multiagent Systems, *Proc. First International Conference on Multi-Agent Systems (ICMAS '95)*, 1995, p. 455.

[19] Malone, T.W., Yates, J., Benjamin, R.I., Electronic Markets and Electronic Hierarchies, *Communications of the ACM*, vol. 30, June, 1987, pp. 484-497.

[20] Mousavi, M.R., Sirjani, M., Arbab, F., Specification, Simulation, and Verification of Component Connectors in Reo, *Technical Report No. 04-15*, Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, June 2004.

[21] Pokraev, St., Zlatev, Z., Brussee, R. & Eck, P. van, Semantic Support for Automated Negotiation with Alliances. In Seruca, E. [et al.] (eds.): *Proceedings of the Sixth International Conference on Enterprise Information Systems*, ICEIS 2004, Porto INSTICC, Portugal, April 14-17, vol.4, 2004, pp. 244-249.

[22] Rutten, J.J.M.M., Kwiatkowska, M., Gethin, N., Parker., D., Chapter 5: Component Connectors, In *Mathematical Techniques for analysing concurrent and probabilistic systems*, CRM Monograph series Volume 23, American Mathematical Society, 2004, p. 215.

[23] Shor, M.(admin.), Game Theory .net, Owen Graduate School of Management, Vanderbilt University, 2004, online available at: http://www.gametheory.net/Dictionary/Auctions/ProcurementAuction.html

[24] W3C, Web Services Conversation Language (WSCL) 1.0, W3C Note, March, 2002, online available at: http://www.w3.org/TR/wscl10/

[25] Weerawarana, S. and Curbera, F. P., Concepts in business processes, 2002, online available at: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/

[26] Zlatev, Z. & Eck, P. van, An Investigation of the Negotiation Domain for Electronic Commerce Information Systems. In: Camp, O., Filipe, J., Hammoudi, S. and Piattini, M. (eds.) *Proceedings of the Fifth International Conference on Enterprise Information Systems*, ICEIS 2003, Angers, France, April 23-26. Volume 4, 2003, pp. 386-391 ISBN: 972-98816-1-8.