Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-014-9251-7

Dynamically generated commitment protocols in open systems

Akın Günay · Michael Winikoff · Pınar Yolum

Abstract Agent interaction is a fundamental part of any multiagent system. Such interactions are usually regulated by protocols, which are typically defined at design-time. However, in many situations a protocol may not exist or the available protocols may not fit the needs of the agents. In order to deal with such situations agents should be able to generate protocols at runtime. In this paper we develop a three-phase framework to enable agents to create a commitment protocol dynamically. In the first phase one of the agents generates candidate commitment protocols, by considering its goals, its abilities and its knowledge about the other agents' services. We propose two algorithms that ensure that each generated protocol allows the agent to reach its goals if the protocol is enacted. The second phase is ranking of the generated protocols in terms of their expected utility in order to select the one that best suits the agent. The third phase is the negotiation of the protocol between agents that will enact the protocol so that the agents can agree on a protocol that will be used for enactment. We demonstrate the applicability of our approach using a case study.

Keywords Commitment protocol · Generation · Ranking

1 Introduction

Interaction is a fundamental element of any multiagent system. Agents interact with each other in order to exchange information, coordinate their activities and collaborate on tasks.

A. Günay
School of Computer Engineering, Nanyang Technological University, Singapore, Singapore
e-mail: akingunay@ntu.edu.sg

M. Winikoff (✉)
Department of Information Science, University of Otago, Dunedin, New Zealand
e-mail: michael.winikoff@otago.ac.nz

P. Yolum
Department of Computer Engineering, Bogazici University, Istanbul, Turkey
e-mail: pinar.yolum@boun.edu.tr

Published online: 25 February 2014

These interactions are generally regulated by interaction protocols that specify the set of messages that can be exchanged among agents and their meanings.

Typically, interaction protocols are defined at design-time and embedded into agents' implementation [8,28,31]. However, there are a number of reasons why defining protocols at design-time is, sometimes, too limiting. For instance, in open systems, where agents enter the system at run-time, it is desirable to allow for run-time protocol construction. It is also possible, even in closed systems, for there to be changes in the environment, or in agent preferences, that suggest the need for run-time protocol construction. In other words, designing protocols up front is not sufficient for three major reasons:

(1) *Change in agents* If the system is open, new agents can enter the system. There is no guarantee that the new agents will use the same protocol as the agents inside the system. For example, a typical pay-for-delivery protocol would not work with a barter agent who would expect a different type of good, rather than money, to complete its transaction
(2) *Change in environment* Even if the agents are fixed, the *environment* can evolve in a way that requires run-time adaptation of protocols. For example, if the delivery service is temporarily unavailable, then the previous pay-for-delivery protocol will not suffice. Hence, the agents need to find a way to carry out a different protocol, such as pay-and-pickup, where the customer is expected to collect the goods after payment. Such cases occur because the environment is dynamic. However, it is difficult to account for all such cases when designing a protocol at design-time.
(3) *Change in agent preferences* It is also possible for agents' *preferences* to change over time. For example, an agent that runs out of cash may need to come up with a protocol in which the agent either delays its payment or uses a different commodity in return. Since the need becomes evident only at run-time, the agent should have the means to adapt and formulate a protocol that reflects its preferences.

For these reasons we argue that in open systems, and in some closed (but highly dynamic) systems, there is a need for agents to be able to derive new protocols at run-time, rather than relying on design-time protocols that are embedded into the agents' implementation.

There are two lines of work that are closely related to this problem. One line of work studies the evolution of protocols in light of changing requirements. That is, is it possible for designers to modify existing protocols in a systematic way such that the agents can just receive the new protocol and continue working? Gerard and Singh [11] show that using various refactoring mechanisms, this is possible. That approach assumes that there is already an existing protocol and a dedicated designer who modifies the protocol. In our case, we do not assume that there are any existing protocols available to the agents. The agents need to generate and agree on the protocol on their own. The second line of work does not assume an existing protocol but a centralized planning agency that knows the preferences, goals, and services of all the agents [25]. As a result, using HTN planning, a central agent can generate the right protocol to satisfy all agents. In our case, we do not assume that there is such an agent for planning that would know all the details of all the agents. Hence, we need to develop a method that will allow an agent to generate protocols using its own local knowledge and facilitate agreement of a protocol by its participants without a central agency.

One key tradeoff to be considered in run-time development of protocols is the *generality* of the protocols. At one extreme, we could consider protocols that are in effect more like plans. That is, they support only a single possible way of realising the desired interaction. Although such plans can work, since they are derived for a given context, they have a number of major drawbacks.

Firstly, they are rigid and, are thus not well aligned with the use of autonomous agents, which may possess a range of means for achieving a given goal [24]. Furthermore, a design process that focuses on messages tends to result in interaction protocols that are rigid and overly constrain the agents [4]. Secondly, they are brittle in that if things do not go according to the plan, the interaction is liable to fail, since it does not include any flexibility, or contingencies. Thirdly, they are not generic or reusable, which means that the agent cannot reuse a generated protocol, but has to repeatedly construct new protocols for any new situation, even where the situation may be very similar to a previous situation.

In other words, what we want is not a single use plan, but a more generic specification of the interaction that allows agents (some degree of) choice and freedom in their behaviour when interacting, that is able to handle contingencies, and that is usable in a range of situations (i.e. reusable).

Given these requirements, we propose to use protocols that are structured in terms of *social commitments* [3,21], rather than in terms of message exchanges. Using commitment-based protocols addresses the three concerns above:

- They allow agents to reason about and carry out their interactions in a flexible manner preserving their autonomy [9,15,32].
- They tend to allow a high degree of flexibility in interactions: instead of an interaction being defined as a specific sequence of messages with specifically defined deviations from the sequence, the interaction is defined in terms of the commitments that need to be achieved along the way. Any sequence of messages and actions that results in the commitments being achieved is acceptable. This flexibility allows interactions to deal with a range of contingencies, and a range of situations, thus enhancing reusability.

This paper therefore proposes to use run-time construction of commitment-based interaction protocols. Our contribution is not just a mechanism for creating a commitment-based protocol at run-time, but also a framework for the life-cycle of such protocols. In other words, we consider not just the creation of these protocols, but also what happens after protocol creation: how do agents reach agreement on a protocol to enact? How do agents reason about which potential protocols they prefer?

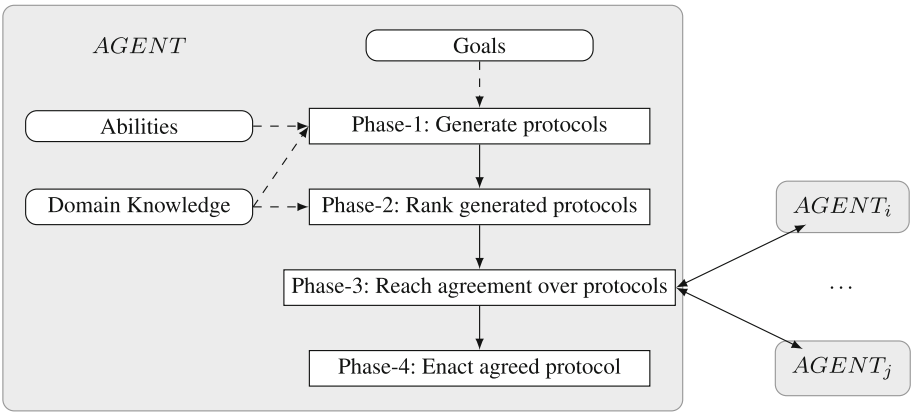


Fig. 1 Overall process of an agent

The general setting is depicted in Fig. 1. We assume that agents have some abilities, some goals, and that they also have some domain knowledge about the environment and about other agents' goals and the services that they offer. We propose a four step process:

(1) *Generation* For the reasons listed above, predefined protocols will not be sufficient. Hence, an agent that aims to achieve a certain goal state needs to generate a set of possible candidate protocols that could be used to realize the desired interaction. The generated protocols should support the goals of the generating agent and also take into account the goals and services of other involved agents.
(2) *Ranking* Since multiple protocols can be generated to achieve the same goals, the agent needs to evaluate the candidate protocols. A useful method is to rank the protocols based on some criteria. This ranking is subjective by nature and different parties may prefer different protocols over others. For example, a merchant may prefer a protocol in which he is paid first before delivery, whereas a customer may prefer a protocol in which he pays after delivery. For this purpose we develop a mechanism based on cost and risk metrics to rank commitment protocols.
(3) *Agreement* After the ranking, the agent then engages in negotiation with the other agents that will be involved in enacting the protocols in order to agree on which of the candidate protocols to use. We provide a simple agreement protocol that agents use to choose a protocol for enactment.
(4) *Enactment* After this step, the protocol that is chosen will be enacted by the agents. Enactment of protocols and their various properties have been extensively studied in the literature [26,33]. For that reason, our focus in this paper is on the first three steps of the process explained above.

The rest of this paper is organized as follows. Section 2 describes our technical framework in depth. Section 3 introduces our algorithms for generating commitment protocols based on agents' goals and abilities, compares their performances, and establishes formal results. Section 4 defines ranking of commitment protocols. Section 5 describes how the ranked protocols can be used in practice. Finally, Sect. 6 summarises, discusses our work in relation to recent work, and outlines directions for future work.

2 Technical framework

In this section we present the technical framework and a running example that we use in the rest of the paper. In the following we use the standard logical connectives $\top$ (true), $\wedge$ (and), $\Rightarrow$ (implication). We present the grammar of our framework's elements in Table 1. In the rest of the paper we use the variables x, x_i, y for agents, g, g_i for goals, a, a_i for abilities, s, s_i for

Table 1 Grammar of the framework elements

Goal	→	G_{AgentID}(Proposition)
Ability	→	A_{AgentID}(Conjunction, Proposition)
Service	→	S_{AgentID}(AgentID, Conjunction, Proposition)
Incentive	→	I_{AgentID}(AgentID, Proposition, Proposition)
Belief	→	Service | Incentive
Commitment	→	C(AgentID, AgentID, Conjunction, Proposition)
Conjunction	→	Proposition | Conjunction $\wedge$ Proposition
Proposition	→	*Any atomic propositional symbol*
AgentID	→	*Any agent identifier*

services, n, n_i for incentives, c, c_i for commitments, p, p', p_i for protocols, q, q_i, r, r', r_i, w, for propositions and d, d_i, d' for conjunctions.

An agent is an entity that acts in an environment to achieve its goals, has a set of abilities to perform certain tasks and has a set of beliefs about other agents.

Definition 1 (*Agent*) An *agent* is a three tuple $\langle \mathcal{G}, \mathcal{A}, \mathcal{B} \rangle$, where $\mathcal{G}$ is the agent's goals, $\mathcal{A}$ is the agent's abilities and $\mathcal{B}$ is the agent's beliefs.

A goal of an agent is a state of the world that the agent aims to achieve (e.g., a customer aims to purchase some goods). A goal of an agent is achieved when the environment evolves to that state.

Definition 2 (*Goal*) $G_x(r)$ denotes the *goal* of agent x to bring about the proposition r.

An agent has a set of abilities, which can be used by the agent to change the state of the world, if certain preconditions hold (e.g., a customer can make payments, if it has enough money).

Definition 3 (*Ability*) $A_x(d, r)$ denotes the *ability* of agent x to bring about the proposition r, if the precondition d holds.

An agent has beliefs about the services provided by other agents (e.g., a merchant sells goods) and possible properties that can be offered to those agents to create incentive for the provision of their services (e.g., offering payment to a merchant when it sells a good).

Definition 4 (*Belief*) $S_x(y, d, r)$ denotes that the agent x believes that the agent y can provide a service to bring about the proposition r, if the precondition d holds. $I_x(y, w, r)$ denotes that the agent x believes that the agent y accepts the proposition w as an incentive for its services to bring about r.

There is a conceptual difference between a service and incentive belief. A service belief defines an instance of a service, which requires a certain precondition to hold, to bring about a property. For instance, $s_i = S_{Customer}(Courier, Address\,Provided, Delivered)$ represents *Customer*'s belief about the delivery service that is provided by *Courier*, which requires provision of the delivery address as a precondition. Similarly, $s_j = S_{Customer}(Courier, Customer\,ID\,Provided, Delivered)$ represents *Customer*'s belief about another delivery service that is provided again by *Courier*. However, this service requires a user ID to be provided as a precondition, instead of the delivery address. On the other hand, $n = I_{Customer}(Courier, Delivery\,Paid, Delivered)$ represents that the *Customer* believes that she has to pay, in order to create incentive for the provision of a delivery service. Note that *Customer* can utilze n to create incentive for the provision of either s_1 or s_2.

An agent may acquire the information (i.e., beliefs) about the other agents' services and possible incentives for these services in various ways. For instance, in small and well-defined domains, this information may already be encoded as part of the agent's domain knowledge. In addition to this, such information is usually publicly advertised by the service providers. Even if the domain knowledge of an agent is incomplete or the other agents' services are not advertised, the agent may still infer this information by combining its partial domain knowledge with its observations from previous interactions. Accordingly, in the rest of the paper we assume that each agent has a set of beliefs about the services of the other agents and their expected incentives. However, these beliefs may be incomplete (e.g., an agent may not be aware of all provided services), out of date (e.g., some services may not be provided

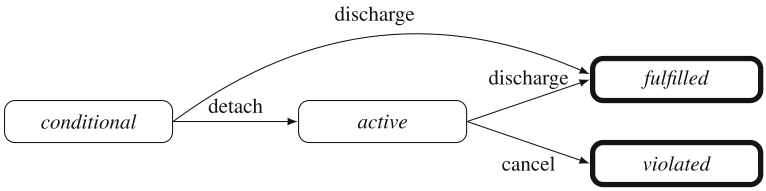


Fig. 2 Life cycle of a commitment

any more) or even be wrong (e.g., a provider may expect something different as an incentives than what the agent believes). We assume that agents are capable of revising their beliefs in the light of new information as is customary in the literature.

Agents interact with each other through their services and these interactions are regulated by *commitments*. A commitment is a social contract between a debtor and a creditor, where the debtor commits to the creditor to bring about a condition.

Definition 5 (*Commitment*) $C(x, y, d, r)$ denotes the *commitment* of the debtor agent x to the creditor agent y to bring about the consequent r if the antecedent d holds.

The lifecycle of commitments has been studied in detail in the multiagent systems literature [5,10,15,30,32]. Here we use a simplified lifecyle that we present in Fig. 2. A commitment is created in a conditional state. When the antecedent of the commitment starts to hold, the commitment is detached and the state of the commitment turns to active. A conditional or an active commitment is discharged once the consequent of the commitment starts to hold, which makes the commitment fulfilled. An active commitment is violated if the commitment is canceled by its debtor, indicating that the consequent of the commitment is not going to be brought about. The fulfilled and violated states are terminal states.

In general agents create several commitments in the context of an interaction. For example, in a purchase scenario several commitments are created between the customer and merchant to regulate different aspects of the whole transaction. Accordingly, a set of commitments in the context of an interaction is called a *commitment protocol*. Traditionally, a commitment protocol includes a set of messages in addition to the commitments, which are used by the agents to manipulate the commitments in the protocol [15,27,29,32]. Here, we omit these messages for readability. Our main goal is to generate the commitments. Once this goal is achieved, the messages can be defined systematically in a straightforward manner.

Definition 6 (*Commitment protocol*) A *commitment protocol* p is a set of commitments.

A multiagent system is a system of agents and the commitments between these agents.

Definition 7 (*Multiagent system*) A *multiagent system* is two tuple $\langle \mathcal{AG}, \mathcal{C} \rangle$, where $\mathcal{AG}$ is a set of agents and $\mathcal{C}$ is a set of commitments between the agents of $\mathcal{AG}$.

Agents in a multiagent system act to achieve their goals either by utilizing their own abilities or participating in commitments to use other agents' services. Hence, in order to act effectively, agents should be able to decide whether they can achieve their goals given their abilities and existing commitments. In this context, we say that an agent *supports* a goal, if the agent can guarantee to achieve the goal. In general, an agent supports a goal in two situations: (1) the goal can be achieved by the agent itself using its own abilities, or (2) there is a commitment from another agent to provide a service that causes the agent to achieve

its goal. Let us explain the two conditions in detail in the context of our framework. For brevity, below we define support over propositions instead of over goals. This is appropriate since in our framework goals correspond to propositions. On the other hand, this does not affect the generality of the notion of support and it can easily be extended if a more complex representation of goal is used.

Definition 8 (*Support*) Given a conjunction of propositions $d' = r_1 \wedge \cdots \wedge r_n$, an agent $x = \langle \mathcal{G}, \mathcal{A}, \mathcal{B} \rangle$ and a set of commitments $\mathcal{C}$, x *supports* d' with respect to commitments $\mathcal{C}$, denoted as $x, \mathcal{C} \Vdash r_1 \wedge \cdots \wedge r_n$, with respect to the following conditions.

- $x, \mathcal{C} \Vdash d'$ if $d' = \top$
- $x, \mathcal{C} \Vdash d'$ if $d' = d_i \wedge d_j$ and $x, \mathcal{C} \Vdash d_i$ and $x, \mathcal{C} \Vdash d_j$
- $x, \mathcal{C} \Vdash r$ if $A_x(d, r') \in \mathcal{A}$ and $r' \Rightarrow r$ and $x, \mathcal{C} \Vdash d$

  or
  $C(y, x, d \wedge w, r) \in \mathcal{C}$ and $S_x(y, d, r') \in \mathcal{B}$ and $r' \Rightarrow r$ and
  $I_x(y, w, r) \in \mathcal{B}$ and and $x, \mathcal{C} \Vdash d \wedge w$

The first and second cases deal with the logical constructs $\top$ and $\wedge$. The third case considers support for a proposition in two different situations. In the first situation, the agent x supports the proposition r, if there is an ability of x (i.e., $A_x(d, r')$) and it is the case that $r' \Rightarrow r$. In other words, if x has the ability to bring about r' that implies the desired condition r, then r is supported. Furthermore, in order to be able to use its ability, x has to ensure that d, which is the precondition of the ability, holds. In other words, x should also support d. Note that d may be a conjunction. If this is the case, then x should support every proposition in the conjunction.

In the second situation, agent x is not able to bring about the desired condition on its own, so it needs to rely on another agent. Since other agents are, by definition, autonomous, and hence not subject to x's control, a commitment is needed to ensure that the other agent (y) will bring about r. Specifically, there needs to be a commitment from y to x that, under appropriate conditions, y will be committed to bring about r. We therefore require that $\mathcal{C}$ include a commitment of the form $C(y, x, d \wedge w, r)$. In addition to knowing that y has a (conditional) commitment to bring about r, we also need to ensure that y is actually able to bring about r (more precisely r', where $r' \Rightarrow r$), i.e. that it has a service to do so (formally: $S_x(y, d, r') \in \mathcal{B}$). However, having a (conditional) commitment to bring about r, and having a service to do so isn't enough: we also need to ensure that the antecedent of the commitment can be brought about by x (directly or indirectly), i.e. that the antecedent is supported. We also need to ensure that the precondition of y's service holds. We deal with both these conditions at once by defining the antecedent to be $d \wedge w$, where d is the precondition of the relevant service, and requiring that $x, \mathcal{C} \Vdash d \wedge w$. Pulling the pieces together, so far, we have the definition:

$$x, \mathcal{C} \Vdash r \text{ if } C(y, x, d \wedge w, r) \in \mathcal{C} \text{ and } S_x(y, d, r') \in \mathcal{B} \text{ and } r' \Rightarrow r \text{ and } x, \mathcal{C} \Vdash d \wedge w$$

There is one last part of the definition to explain. The definition so far is actually sufficient. However, what is missing is a reason for agent y to accept the proposed commitment (more generally, the protocol). In order to address this we introduce an *incentive*: in addition to the antecedent offering to make d true, agent x also offers an incentive w, where w is known to be an incentive of y for r (formally: $I_x(y, w, r) \in \mathcal{B}$).

We now show that Definition 8 is sufficient for ensuring that a protocol does allow agent x to achieve the desired condition.

Lemma 1 *Let $x, \mathcal{C} \Vdash r$, and assume that all active (i.e. non-conditional) commitments in $\mathcal{C}$ that can be fulfilled, are eventually fulfilled. Then there exists a way for agent x to act that results in r becoming true.*

Proof by induction. Trivial cases: (i) if r is $\top$ then it is trivially true, and the condition is satisfied; (ii) if r is achievable by agent x acting alone, using its ability $A_x(d, r')$ (where $r' \Rightarrow r$) then from $x, \mathcal{C} \Vdash r$ we know that the ability's precondition, d, is also supported ($x, \mathcal{C} \Vdash d$) and hence by induction agent x can ensure that d becomes true, and hence that it is able to use its ability to then make r true.

The other case is where agent x does not have the ability to make r true. Since $x, \mathcal{C} \Vdash r$ we know that: there exists an agent y which is committed to bringing about r (i.e., $C(y, x, d \wedge w, r)$), that y has the ability to do so ($S_x(y, d, r')$ where $r' \Rightarrow r$), and that the antecedent of the commitment is supported ($x, \mathcal{C} \Vdash d \wedge w$). Therefore, by induction, agent x can act in such a way to make $d \wedge w$ true. At this point the commitment becomes active. By assumption, agent y eventually fulfills the commitment (since it is able to do so), making r true as desired. □

Running example In our running example there are five agents, namely a customer (Cus), two builders (Bui_1 and Bui_2), a merchant (Mer) and a retail store (Ret). We examine the case from the customer's perspective. The customer wants to have a certain type of furniture. The first builder (Bui_1) offers a service to build custom furniture. However, as a precondition of this service the builder requires the materials to build the furniture to be provided. Similarly, the second builder (Bui_2) also offers a service to assemble furniture. However, the second builder, who is not a professional builder, requires both materials and also tools to be provided as a precondition. On the other hand, the merchant sells ready to use furniture and the retail store sells tools and materials for building furniture. From its domain knowledge, the customer believes that all agents expect to be paid for their services. The objective of the customer is to have a protocol that supports its goal. We summarize the propositions we use in our running example in Table 2.

The goal of the customer is $g_1 = G_{Cus}(HaveFurniture)$ (i.e., $\mathcal{G} = \{g_1\}$). We list the abilities of the customer (i.e., the contents of the set $\mathcal{A}$) in Table 3. a_1, a_2 and a_3 represent the ability of the customer to provide building materials and tools to builder agents. However, as a precondition it has to own the materials and tools beforehand. $a_4 - a_8$ represent different abilities of the customer to make payments to other agents for their services. We assume that

Table 2 Propositions of the running example and their meanings

HaveFurniture	The customer owns furniture
HaveMaterials	The customer owns materials
HaveTools	The customer owns tools
MaterialsPaid	The customer has paid the retailer for the materials
ToolsPaid	The customer has paid the retailer for the tools
FurniturePaid	The customer has paid the merchant for the furniture
$Bui_1 Paid$	The customer has paid the service cost to the first builder
$Bui_2 Paid$	The customer has paid the service cost to the second builder
$Bui_1 Materials Provided$	The customer has provided materials to the first builder
$Bui_2 Materials Provided$	The customer has provided materials to the second builder
ToolsProvided	The customer has provided the tools to the second builder

Table 3 Abilities of the customer

a_1	$A_{Cus}(HaveTools, ToolsProvided)$
a_2	$A_{Cus}(HaveMaterials, Bui_1MaterialsProvided)$
a_3	$A_{Cus}(HaveMaterials, Bui_2MaterialsProvided)$
a_4	$A_{Cus}(\top, MaterialsPaid)$
a_5	$A_{Cus}(\top, ToolsPaid)$
a_6	$A_{Cus}(\top, FurniturePaid)$
a_7	$A_{Cus}(\top, Bui_1Paid)$
a_8	$A_{Cus}(\top, Bui_2Paid)$

Table 4 Beliefs of the customer

s_1	$S_{Cus}(Ret, \top, HaveMaterials)$
s_2	$S_{Cus}(Ret, \top, HaveTools)$
s_3	$S_{Cus}(Mer, \top, HaveFurniture)$
s_4	$S_{Cus}(Bui_1, Bui_1MaterialsProvided, HaveFurniture)$
s_5	$S_{Cus}(Bui_2, Bui_2MaterialsProvided \wedge ToolsProvided, HaveFurniture)$
n_1	$I_{Cus}(Ret, MaterialsPaid, HaveMaterials)$
n_2	$I_{Cus}(Ret, MaterialsPaid, HaveTools)$
n_3	$I_{Cus}(Ret, ToolsPaid, HaveTools)$
n_4	$I_{Cus}(Ret, ToolsPaid, HaveMaterials)$
n_5	$I_{Cus}(Mer, FurniturePaid, HaveFurniture)$
n_6	$I_{Cus}(Bui_1, Bui_1Paid, HaveFurniture)$
n_7	$I_{Cus}(Bui_2, Bui_2Paid, HaveFurniture)$

the customer has enough money to make all the payments. Hence, none of these abilities has a precondition.

We list the customer's beliefs (i.e., the contents of the set $\mathcal{B}$) in Table 4. s_1 and s_2 denote the services of the retailer to sell materials and tools, respectively. s_3 is the service of the merchant to sell the furniture. s_4 and s_5 are the services of the first and second builders, respectively, to build the furniture. n_1–n_7 state that other agents expect payments as incentive for their services. The incentives include some that are fairly clear, such as n_1, stating that the customer has domain knowledge that the retailer is incentivised to provide materials (*HaveMaterials*) by the materials being paid for (*MaterialsPaid*). However, it can also be the case that the customer's beliefs about incentives and services may not reflect reality. An example to this is n_2, which states that the customer believes paying for materials is an incentive for providing tools. If this is not an accurate incentive, a protocol generated based on this incentive would be unpreferable by the agent providing the service. We return to this in Sect. 4.

Below, we provide an example commitment protocol that supports the goal (i.e., *HaveFurniture*) of the customer.

$$c_1 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$$
$$c_2 = C(Ret, Cus, ToolsPaid, HaveTools)$$
$$c_3 = C(Bui_2, Cus, Bui_2MaterialsProvided \wedge ToolsProvided \wedge Bui_2Paid, HaveFurniture)$$

The commitment c_3 states that the second builder will be committed to the customer to deliver the furniture (i.e., bring about *HaveFurniture*), if the customer provides the required

materials and tools, and also pays the costs (i.e., brings about $Bui_2MaterialsProvided$, $ToolsProvided$ and Bui_2Paid). Hence, once this commitment is fulfilled, the customer achieves its goal $HaveFurniture$. However, in order to make c_3 active by providing the tools and materials to the builder, the customer should have them first. Accordingly, the protocol includes c_1 and c_2, which, once fulfilled, ensure that the customer has materials and tools.

This protocol might be enacted and executed in a flexible manner. For instance, the customer could make c_1 and c_2 active in any order she finds appropriate. She might chose to make c_1 active by bringing about $MaterialsPaid$ first and might wait until c_1 is fulfilled before making c_2 active by bringing about $ToolsPaid$. Or the customer might chose to make both c_1 and c_2 active at the same time by bringing about $MaterialsPaid$ and $ToolsPaid$ concurrently. Also note that the customer might not need to make c_1 or c_2 (or both) active if she already had the necessary materials or tools. In this case the commitment stayed conditional and would not put the retailer under any obligation. Finally, once the customer possessed the materials and tools, she could make c_3 active, by providing these to the builder and paying the cost. Then, the builder would fulfil c_3 by delivering the furniture.

3 Protocol generation

In this section we develop two algorithms that generate commitment protocols to support the given goals of an agent, which corresponds to the first phase of our framework in Fig. 1. The first algorithm is a depth-first traversal algorithm that uses the definition of support (Definition 8) utilizing the agent's abilities and beliefs to generate the protocols. The second algorithm is also based on the definition of support and uses the agent's abilities and beliefs, however it takes advantage of the divide and conquer strategy to generate the protocols more efficiently. We examine and compare the execution of the algorithms over our running example. We also report on an empirical evaluation of the algorithms' performance, and show their formal properties.

3.1 ProtocolBased algorithm

We present our ProtocolBased algorithm in Algorithm 1. The algorithm uses a depth-first traversal strategy to generate the protocols. Basically, this algorithm creates a tree structure as follows. The root of the tree contains a set of unsupported initial goals of the agent that runs the algorithm. One goal from this set is selected and for each ability and service that can be used to support the selected goal, an edge is created. If the edge is created for a commitment, the commitment is set as the label of the edge. Otherwise, if the edge is created for an ability, the edge is not labeled. The destination node of each edge contains the remaining unsupported goals of the agent. If the used ability or service requires new propositions to hold (e.g., precondition of the service), then goals over these propositions are also added to the set of unsupported goals in the destination node. This process is repeated for each subsequent node until there is no unsupported goal left in any leaf node. As a result, the labels (i.e., commitments) on a path from the root to a leaf node correspond to a protocol that supports all the initial goals (i.e., goals that are given as input to the algorithm) and also all the goals that are created by the algorithm to use abilities and services.

The algorithm takes six parameters as input: the agent that runs the algorithm to generate protocols (x), (2) x's abilities ($\mathcal{A}$), (3) x's beliefs ($\mathcal{B}$), (4) a queue of goals that are aimed to be supported (G_p), (5) a set of goals that are already supported (G_s), and (6) the protocol generated so far by the algorithm (Δ). When the algorithm is first invoked G_p consists of a

Algorithm 1 P PROTOCOLBASED $(x, \mathcal{A}, \mathcal{B}, \mathsf{G}_p, \mathsf{G}_s, \Delta)$

Require: x, the agent that generates the protocols
Require: $\mathcal{A}$, abilities of x
Require: $\mathcal{B}$, beliefs of x
Require: G_p, the queue of goals pending for support
Require: G_s, the set of already supported goals
Require: Δ, protocol generated so far
if isEmpty(G_p) **then**
  return $\{\Delta\}$
else
  $g \leftarrow$ dequeue(G_p)
  $r \leftarrow$ getGoalProposition(g)
  $\mathsf{P} \leftarrow \emptyset$
  for all $\{A_x(d, r') : A_x(d, r') \in \mathcal{A}$ **and** $r' \Rightarrow r\}$ **do**
    $\mathsf{G}'_p \leftarrow \mathsf{G}_p$
    for all $q \in$ getConditions(d) **do**
      if not isSupported(q, G_s) **then**
        enqueue($\mathsf{G}'_p, G_x(q)$)
      end if
    end for
    $\mathsf{G}'_s \leftarrow \mathsf{G}_s \cup \{g\}$
    $\mathsf{P} \leftarrow \mathsf{P} \cup$ PROTOCOLBASED($\mathsf{G}'_p, \mathsf{G}'_s, \Delta'$)
  end for
  for all $\{S_x(y, d, r') : S_x(y, d, r') \in \mathcal{B}$ **and** $r' \Rightarrow r\}$ **do**
    for all $\{I_x(y, w, r'') : I_x(y, w, r'') \in \mathcal{B}$ **and** $r \equiv r''\}$ **do**
      $\mathsf{G}'_p \leftarrow \mathsf{G}_p$
      if not isSupported(w) **then**
        enqueue($\mathsf{G}'_p, G_x(w)$)
      end if
      for all $q \in$ getConditions(d) **do**
        if not isSupported(q, G_s) **then**
          enqueue($\mathsf{G}'_p, G_x(q)$)
        end if
      end for
      enqueue(G'_s, g)
      $\Delta' \leftarrow \Delta \cup \{C(y, x, d \wedge w, r)\}$
      $\mathsf{P} \leftarrow \mathsf{P} \cup$ PROTOCOLBASED($\mathsf{G}'_p, \mathsf{G}'_s, \Delta'$)
    end for
  end for
  return P
end if

set of x's goals for which x aims to find support, (i.e., $\mathsf{G}_p \subseteq \mathcal{G}$). Both G_s and Δ are initially empty. The algorithm returns the set of protocols $\mathsf{P} = \{p_1, \ldots, p_n\}$, where each protocol p_i is a set of commitments, such that each p_i supports all the goals given initially by G_p. If it is not possible to support every goal, the algorithm returns an empty set.

The auxiliary function *getGoalProposition* takes a goal and returns the proposition that is the satisfaction condition of the goal. The auxiliary function *getConditions* takes a conjunction and returns a set that includes every proposition in the conjunction. For example, getConditions$(r_1 \wedge r_2) = \{r_1, r_2\}$. The auxiliary function *isSupported* takes a proposition q corresponding to a goal, and G_s and returns true if the goal (or a more specific one that implies the goal) is already supported by the protocol generated so far, and false otherwise. We use the auxiliary functions *isEmpty*, *enqueue* and *dequeue*, for regular queue manipulation, where *dequeue* (G) modifies G by removing and returning the first item in the queue. Similarly, *enqueue* (G,g) modifies G by adding g.

The details of the ProtocolBased algorithm are as follows. It starts by checking whether the queue of pending goals (G_p) is empty (line 1).[1] If this is the case, then the algorithm returns $\{\Delta\}$(line 2), i.e., a set of protocols that includes only a single protocol Δ, which consists of the commitments generated so far. Otherwise, the algorithm gets the next pending goal g from G_p (line 4).

In order to find support for g the algorithm first considers x's abilities. For each matching ability $A_x(d, r')$, such that $r' \Rightarrow r$ (line 6), the algorithm first iterates over the precondition d of the ability by generating a new goal $G_x(q)$ for each proposition $q \in \text{getConditions}(d)$ and adding these new goals to the queue of pending goals G'_p, if the goal is not already supported (lines 8–12). Then, g is added to the set of supported goals G'_s (line 13). Finally, the algorithm recursively invokes itself by supplying the updated data structures G'_p, G'_s and Δ' in order to find support for the next goal (line 14). These steps are performed for each matching ability and accordingly an alternative protocol is generated for each such ability.

After considering x's abilities to support g, the algorithm also considers other agents' services to find alternative ways of supporting g. For this purpose, the algorithm checks x's beliefs to find appropriate services of other agents that can be used to support g. For each matching service $S_x(y, d, r')$, such that $r' \Rightarrow r$ (line 16) and for each corresponding incentive $I_x(y, w, r'')$, where $r \equiv r''$ (line 17), the algorithm first adds the corresponding incentive w and every precondition $q \in \text{getConditions}(d)$ of the service as a pending goal into G'_p, if they are not already supported (lines 19–26). Then, g is added to the set of supported goals G'_s (line 27). After that, the algorithm creates a new commitment $C(y, x, d \wedge w, r)$ and adds it to Δ' (line 28). Then, the algorithm recursively invokes itself by supplying the updated data structures G'_p, G'_s and Δ' in order to find support for the next goal (line 29). As result of this process the algorithm creates alternative protocols to support g considering all combinations of matching services and their corresponding incentives. Finally, once every matching ability and service is considered as explained above, the algorithm returns the set of generated protocols P (line 32). If the desired goals cannot be supported, then P will be empty, and hence the empty set is returned, indicating that no suitable protocols could be generated.

3.2 GoalBased algorithm

Although the ProtocolBased algorithm corresponds fairly directly with the definition of support (Definition 8), it is inefficient because it finds support for the same goal many times, if the goal has to be supported in the context of different protocols. For instance, in our running example the customer should support $HaveMaterials$ in order to use both the first and second builders' services that are part of different protocols. However, the ProtocolBased algorithm considers each protocol in isolation and finds support for $HaveMaterials$ redundantly for each protocol. To overcome this drawback, we develop a more efficient algorithm using the divide-and-conquer strategy, which we call GoalBased . This algorithm considers goals independently from protocols. Accordingly, for each individual goal of the agent the algorithm generates a set of sub-protocols that support only that goal. Then these sub-protocol are merged to come up with a complete protocol to support all the goals of the agent. This modular approach allows us to use memoization to reuse sub-protocols that are generated in the context of different protocols. Hence, the algorithm generates supporting sub-protocols for each goal only once and avoids redundant computation.

[1] Recall that a set containing an empty set, i.e. $\{\emptyset\}$, is distinct from the empty set $\emptyset$, for instance $\{a\} \cup \emptyset = \{a\}$ but $\{a\} \cup \{\emptyset\} = \{a, \emptyset\}$

We present our GOALBASED algorithm in Algorithm 2. The algorithm takes five parameters as input: the agent that runs the algorithm to generate protocols (x), (2) x's abilities ($\mathcal{A}$), (3) x's beliefs ($\mathcal{B}$), (4) a queue of goals that are aimed to be supported (G), and (5) a mapping M from goals to a set of protocols that supports them (i.e., M takes a goal g and returns a set of protocols P). When the algorithm is first invoked, G includes a subset of x's goals (i.e., $\mathsf{G} \subseteq \mathcal{G}$), for which x aims to find support. M is initially empty. The algorithm returns the set of protocols $\mathsf{P} = \{p_1, \ldots, p_n\}$, where each protocol p_i is a set of commitments, such that each p_i supports all the goals given initially by G. If it is not possible to support every goal, the algorithm returns an empty set.

The auxiliary function *getGoalProposition* takes a goal and returns the proposition that is the satisfaction condition of the goal. The auxiliary function *getConditions* takes a conjunction

Algorithm 2 P GOALBASED (x, $\mathcal{A}$, $\mathcal{B}$, G, M)

Require: x, the agent that generates the protocols
Require: $\mathcal{A}$, abilities of x
Require: $\mathcal{B}$, beliefs of x
Require: G, the queue of goals to be supported
Require: M, the mapping from goals to known protocols
if isEmpty(G) **then**
  return $\{\emptyset\}$
else if size(G) > 1 **then**
  $\mathsf{G}' \leftarrow$ dequeue(G)
  return merge(GOALBASED(G', M), GOALBASED(G, M))
else
  //size(G) $= 1$
  $g \leftarrow$ dequeue(G_P)
  $r \leftarrow$ getGoalProposition(g)
  if $g \in$ getKeySet(M) **then**
    return $\mathsf{M}[g]$
  end if
  $\mathsf{P} \leftarrow \emptyset$
  for all $\{A_x(d, r') : A_x(d, r') \in \mathcal{A}$ **and** $r' \Rightarrow r\}$ **do**
    $\mathsf{G}' \leftarrow \emptyset$
    for all $q \in$ getConditions(d) **do**
      enqueue(G', $G_x(q)$)
    end for
    $\mathsf{P}' \leftarrow$ GOALBASED(G', M)
    $\mathsf{P} \leftarrow \mathsf{P} \cup \mathsf{P}'$
  end for
  for all $\{S_x(y, d, r') : S_x(y, d, r') \in \mathcal{B}$ **and** $r' \Rightarrow r\}$ **do**
    for all $\{I_x(y, w, r'') : I_x(y, w, r'') \in \mathcal{B}$ **and** $r \equiv r''\}$ **do**
      $\mathsf{G}' \leftarrow \emptyset$
      for all $q \in$ getConditions(d) **do**
        enqueue(G', $G_x(q)$)
      end for
      enqueue(G', $G_x(w)$)
      $\mathsf{P}' \leftarrow$ GOALBASED(G', M)
      for all $p \in \mathsf{P}'$ **do**
        $p \leftarrow p \cup \{C(y, x, d \wedge w, r)\}$
      end for
      $\mathsf{P} \leftarrow \mathsf{P} \cup \mathsf{P}'$
    end for
  end for
  $\mathsf{M}[g] \leftarrow \mathsf{P}$
  return P
end if

and returns a set that includes every proposition in the conjunction. The auxiliary function *merge* takes two sets of protocols P' and P'' and merges them by forming all possible pairs ($p' \in P'$ and $p'' \in P''$) and forming their union ($p' \cup p''$). Formally $merge(P', P'') = \{p' \cup p'' \mid p' \in P' \text{ and } p'' \in P''\}$. We use the auxiliary functions *enqueue*, *dequeue*, *size* and *isEmpty* for regular queue manipulation as before. Additionally M is a mapping from a goal to a set of protocols that supports the goal. The function *getKeySet*(M) returns the key set of the mapping M (i.e., the set of goals). We use the notation $\mathsf{M}[g]$ to access the set of protocols that support g.

Algorithm 2 starts by checking whether the goal queue (G) is empty (line 1). If this is the case, then the algorithm returns a set of protocols that includes the empty protocol, to indicate that the base case is reached. Otherwise, if G includes more than one goal, then the algorithm divides the queue into two queues such that the first queue includes only the first goal in G and the second queue includes the rest of the goals. Then, for each queue a recursive call is made and the results are merged and returned (line 5).

If G includes only one goal (g), the algorithm first checks M to find whether the protocols that support g are already generated. If this is the case, the algorithm immediately returns the corresponding protocols (line 911). Otherwise, the algorithm checks x's abilities to find matching abilities that can be utilized to support g. For every matching ability $A_x(d, r')$, the algorithm first creates a queue (G') and adds a new goal $G_x(q)$ for each precondition $q \in \text{getConditions}(d)$ into G' (lines 15–17). Then the algorithm calls itself to find support for these goals (line 18) and the protocols returned by the recursive call are kept as alternatives to support g (line 19).

After considering x's abilities to support g, the algorithm also considers other agents' services to find alternative ways of supporting g. For this purpose, the algorithm checks x's beliefs to find matching services of other agents that can be used to support g. For each matching service $S_x(y, d, r')$ and for each corresponding incentive $I_x(y, w, r'')$, the algorithm first creates a goal queue G' (line 23) and adds every precondition $q \in \text{getConditions}(d)$ of the service and the corresponding incentive w as a goal into the queue (lines 24–27). Then, the algorithm calls itself to find support for these goals (line 28). Finally, the algorithm adds the commitment $C(y, x, d \wedge w, r)$ to each protocol in P' returned by the recursive call and adds these modified protocols to P (lines 29–32).

Once all the abilities of x and other agents' services have been considered, the algorithm adds an entry to M for g, noting that g can be achieved using the protocol set P, which means that if the algorithm has to consider g again later, it can reuse these protocols (line 35). Finally the algorithm returns P (line 36).

3.3 Formal results

We now consider formal properties of the algorithm. We prove these results for the GOAL-BASED algorithm (Algorithm 2), since this is the algorithm that we use. The two algorithms do, in fact, generate the same protocols (and hence the results below also apply to the first algorithm). However, we do not give a proof for the equivalence of the two algorithms.

We begin with soundness. Intuitively, the algorithm is sound if all protocols generated are able to support the generating agent's goals. Note that whereas the notion of support (Definition 8) takes a given protocol and assesses whether a given condition is supported with respect to the protocol, the algorithms are given the goals, and *generate* possible protocols, which contain commitments that allow the goals to be supported.

Definition 9 (*Soundness*) Algorithm 2 is *sound* iff any protocol that it generates contains sufficient commitments to support all of the propositions in the agent's goals. Formally: Let $\mathcal{P} = \textsc{GoalBased}\,(x.\mathcal{G}, \emptyset)$, then we have $\forall p \in \mathcal{P} : x, p \Vdash d$ where $d = \bigwedge_{G_x(r) \in x.\mathcal{G}} r$ is a conjunction of the agent's goals.

We now prove that the algorithm is sound.

Theorem 1 (Soundness) *Algorithm 2 is sound (see Definition 9).*

Proof see Appendix. □

We now turn to completeness. One slight complication is that Definition 8 allows for an infinite number of possible protocols that support a given condition, including all supersets of a given protocol: given any commitment set $\mathcal{C}$ such that $x, \mathcal{C} \Vdash r$, we also have that $x, \mathcal{C}' \Vdash r$ holds for any $\mathcal{C}'$ that is a superset of $\mathcal{C}$ ($\mathcal{C} \subset \mathcal{C}'$). We therefore define completeness relative to a *minimal* set of commitments that supports a given proposition r.

Definition 10 (*Minimal commitment set*) Given a condition r, we define $\mathcal{C}$ to be a *m*inimal commitment set supporting r if $x, \mathcal{C} \Vdash r$, where additionally there does not exist a (strict) subset of $\mathcal{C}$ ($\mathcal{C}' \subset \mathcal{C}$) such that $x, \mathcal{C}' \Vdash r$. We realize this restriction by defining a variant of definition 8, denoted $x, \mathcal{C} \,\widehat{\Vdash}\, r$, that constrains the set of commitments to be only as large as it needs to be. That is, in the first case ($x, \mathcal{C} \,\widehat{\Vdash}\, \top$) the only minimal commitment set is $\mathcal{C} = \emptyset$. Similarly, in the second case, if C_i and C_j are both minimal for respectively supporting d_i and d_j, then we construct a minimal commitment set for supporting $d_i \wedge d_j$ by merging C_i and C_j. The first part of the last case is straightforward: if $\mathcal{C}$ is minimal for supporting d, then it is also minimal for supporting r. Finally, if $\mathcal{C}'$ is a minimal commitment set that supports $d \wedge w$, then we can obtain a minimal commitment set for supporting r by adding to $\mathcal{C}'$ the commitment $C(y, x, d \wedge w, r)$.

$$
\begin{aligned}
&x, \mathcal{C} \,\widehat{\Vdash}\, \top \text{ iff } \mathcal{C} = \emptyset \\
&x, \mathcal{C} \,\widehat{\Vdash}\, d_i \wedge d_j \text{ iff } x, C_i \,\widehat{\Vdash}\, d_i \text{ and } x, C_j \,\widehat{\Vdash}\, d_j \text{ and } \mathcal{C} = C_i \cup C_j \\
&x, \mathcal{C} \,\widehat{\Vdash}\, r \;\text{ iff }\; A_x(d, r') \in \mathcal{A} \text{ and } r' \Rightarrow r \text{ and } x, \mathcal{C} \,\widehat{\Vdash}\, d \\
&\quad \text{or } S_x(y, d, r') \in \mathcal{B} \text{ and } r' \Rightarrow r \text{ and } I_x(y, w, r) \in \mathcal{B} \\
&\qquad \text{and } x, \mathcal{C}' \,\widehat{\Vdash}\, d \wedge w \text{ and } \mathcal{C} = \mathcal{C}' \cup \{C(y, x, d \wedge w, r)\}
\end{aligned}
$$

We now define completeness relative to the minimal commitment set. The intuition is that the algorithm is complete if it returns all minimal commitment sets.

Definition 11 (*Completeness*) Algorithm 2 is *complete* iff given the agent's goals and corresponding condition $d = \bigwedge_{G_x(r) \in x.\mathcal{G}} r$, for any minimal set of commitments $\mathcal{C}$ such that $x, \mathcal{C}$ supports the agent's goals according to Definition 10 (i.e. $x, \mathcal{C} \,\widehat{\Vdash}\, d$), the algorithm returns $\mathcal{C}$ in its set of protocols generated. Formally, given $x.\mathcal{G}$ and the corresponding d:

$$x, \mathcal{C} \,\widehat{\Vdash}\, d \;\Leftrightarrow\; \mathcal{C} \in \textsc{GoalBased}\,(x.\mathcal{G}, \emptyset)$$

We now show that the algorithm is complete.

Theorem 2 (Completeness) *Algorithm 2 is complete (see Definition 11).*

Proof see Appendix. □

3.4 Comparison of algorithms and experimental results

The ProtocolBased and GoalBased algorithms generate the same protocols. We present the protocols that are generated by these algorithms for our running example in Table 5. Protocol p_1 includes a single commitment from the merchant to the customer in order to support the customer's goal $HaveFurniture$. However, the customer should bring about $FurniturePaid$ to make this commitment active. In other words, the customer has to support $FurniturePaid$. The customer supports this condition by its ability a_6. Hence, there is no need for another commitment in this protocol. In p_2 and p_3, the customer's goal is supported by the commitment of the first builder to the customer to build the furniture. However, the customer has to support $Bui_1MaterialsProvided$ in order to make this commitment active, but the customer is not able to do that, since she does not have the necessary materials. Accordingly, p_2 and p_3 include a commitment from the retailer to the customer for provision of the materials, with alternative incentives $MaterialsPaid$ (n_1) and $ToolsPaid$ (n_4). Finally, protocols $p_4 - p_7$ include a commitment from the second builder to the customer to support the customer's goal. However, the customer has to provide tools ($ToolsProvided$) as well as materials to the second builder to make this commitment active. Since the customer is not able to provide either of them, these protocols include two commitments from the retailer to the customer, one to support provision of the materials and one to support provision of the tools. Each of these protocols uses different incentive in their commitments to stimulate the retailer to provide the materials and tools.

Although both ProtocolBased and GoalBased algorithms generate the same protocols for a given input, there are significant differences in their strategies to generate the protocols.

Table 5 Generated protocols for the running example

p_1	$C(Mer, Cus, FurniturePaid, HaveFurniture)$
p_2	$C(Ret, Cus, MaterialsPaid, HaveMaterials)$
	$C(Bui_1, Cus, Bui_1MaterialsProvided \wedge Bui_1Paid, HaveFurniture)$
p_3	$C(Ret, Cus, ToolsPaid, HaveMaterials)$
	$C(Bui_1, Cus, Bui_1MaterialsProvided \wedge Bui_1Paid, HaveFurniture)$
p_4	$C(Ret, Cus, MaterialsPaid, HaveMaterials)$
	$C(Ret, Cus, ToolsPaid, HaveTools)$
	$C(Bui_2, Cus, Bui_2MaterialsProvided \wedge ToolsProvided \wedge Bui_2Paid, HaveFurniture)$
p_5	$C(Ret, Cus, ToolsPaid, HaveMaterials)$
	$C(Ret, Cus, ToolsPaid, HaveTools)$
	$C(Bui_2, Cus, Bui_2MaterialsProvided \wedge ToolsProvided \wedge Bui_2Paid, HaveFurniture)$
p_6	$C(Ret, Cus, MaterialsPaid, HaveMaterials)$
	$C(Ret, Cus, MaterialsPaid, HaveTools)$
	$C(Bui_2, Cus, Bui_2MaterialsProvided \wedge ToolsProvided \wedge Bui_2Paid, HaveFurniture)$
p_7	$C(Ret, Cus, ToolsPaid, HaveMaterials)$
	$C(Ret, Cus, MaterialsPaid, HaveTools)$
	$C(Bui_2, Cus, Bui_2MaterialsProvided \wedge ToolsProvided \wedge Bui_2Paid, HaveFurniture)$

The PROTOCOLBASED algorithm uses a depth-first traversal strategy to create a tree structure, in which each edge from a goal to a sub-goal represents a commitment and each path from the root to a leaf corresponds to a complete protocol. Accordingly, the PROTOCOLBASED algorithm generates a complete protocol at a time. On the other hand the GOALBASED algorithm uses the divide-and-conquer strategy. Accordingly the GOALBASED algorithm divides its goal queue repeatedly until the queue contains only a single goal and then generates a sub-protocol for that individual goal. Then these sub-protocols are merged into larger sub-protocols repetitively until a complete protocol is generated.

The major advantage of the GOALBASED algorithm is reuse of the sub-protocols in different alternative protocols via memoization. Consider for example the protocols p_4 and p_6. To support the customer's goal $Bui_2MaterialsProvided$, both of these protocols include the commitment $C(Ret, Cus, MaterialsPaid, HaveMaterials)$. Once this commitment is generated in the context of one of these protocols by the GOALBASED algorithm, it can be reused whenever it is necessary to support $Bui_2MaterialsProvided$ in the context of another protocol. On the other hand, in the case of the PROTOCOLBASED algorithm such modularity cannot be achieved effectively, since every protocol is generated as a whole, without taking the relations between goals and sub-protocols into account. Another advantage of the GOALBASED algorithm is elimination of the auxiliary *isSupported* function, which is used in the PROTOCOLBASED algorithm, in order to determine whether a goal is already supported or not. In the GOALBASED algorithm this check is not necessary, since memoization handles already supported goals implicitly. More specifically, if an already supported goal was added to the goal queue and reconsidered by the algorithm, the sub-protocols that support the goal would be available in the mapping M and returned immediately by the algorithm.

In order to justify the above discussion and test the execution performance of our algorithms we conduct computational experiments. To the best of our knowledge there does not exist any large data set of commitment protocols for testing in the literature. Hence, we conducted our experiment on a data set we generated parametrically.

The basic idea of this generation process is that we begin with a number of top-level goals. We then repeatedly consider the goals, and for each goal generate services with preconditions that can be used to fulfil the goal's condition, and incentives that can be used to motivate the other agent to adopt the commitment to provide its service. Once services and incentives are generated, the propositions of the generated services and incentives are considered as new goals and in the next iteration these new goals are handled by other services and corresponding incentives. In other words the generation process follows the following outline:

(1) Generate initial goal(s).
(2) For each goal that has not yet been considered: generate services and incentives for it.
(3) For each generated service and incentive, create new goals to achieve the services' preconditions and the incentives' conditions.
(4) Go to step 2.

Now, this process clearly does not terminate, and so we modify it by only iterating a certain number of times, and then finishing the generation by creating for each goal that has not yet been considered an ability (with a true precondition) that allows the agent to achieve the goal.

This process generates a set of services (S), incentives (I), and abilities (A) that can be used to create support for a given number of initial goals of an agent x. In generating the services, incentives and abilities, we use a number of parameters to control the generation. These are summarised in Table 6. Most of the parameters control the number of entities generated at different points in the process (e.g. γ is the number of initial goals created in the first step above, σ is the number of services created in step 2 above, etc.). The exception

Table 6 Summary of parameters to generate the experimental data set

Parameter	Description
γ	Number of initial goals.
σ	Number of services that can be requested from other agents to achieve a goal.
ρ	Number of preconditions required to use a service.
ε	Number of possible incentives that can be offered to the provider for each service.
λ	Controls when a goal can be achieved using an ability that has no precondition, instead of a service.

Table 7 Example service and incentive data generated using $\gamma = 1, \sigma = 2, \rho = 1, \varepsilon = 1$ and $\lambda = 3$

Level	Services	Incentives	Abilities
1	$s_1 = S_x(y, r_2, r_1)$	$n_1 = I_x(y, r_4, r_1)$	None
	$s_2 = S_x(y, r_3, r_1)$		
2	$s_3 = S_x(y, r_5, r_2)$	$n_2 = I_x(y, r_7, r_2)$	None
	$s_4 = S_x(y, r_6, r_2)$		
	$s_5 = S_x(y, r_8, r_3)$	$n_3 = I_x(y, r_{10}, r_3)$	
	$s_6 = S_x(y, r_9, r_3)$		
	$s_7 = S_x(y, r_{11}, r_4)$	$n_4 = I_x(y, r_{13}, r_4)$	
	$s_8 = S_x(y, r_{12}, r_4)$		
3	None	None	$a_1 = A_x(\top, r_5)$
			$\vdots$
			$a_9 = A_x(\top, r_{13})$

is λ which specifies the number of iterations, i.e. if we consider the generated problem as a tree of goals, then λ specifies the depth of the tree.

Table 7, shows an example set of services, incentives and abilities generated using our data generation algorithm with the following parameter values: $\gamma = 1, \sigma = 2, \rho = 1, \varepsilon = 1$ and $\lambda = 3$. First we use γ to determine the number of initial goals, which is equal to one. Hence, initially we generate only one goal to achieve the proposition r_1. In the first iteration we generate two services s_1 and s_2 to achieve r_1, since σ is equal to two. Since ρ is equal to one, a single precondition is generated for s_1 and s_2, namely r_2 and r_3, respectively. Finally, since $\varepsilon = 1$, a single incentive over r_4 is generated for the services that bring about r_1. As result of the generation of services s_1 and s_2, new goals for propositions r_2 and r_3, which are the preconditions of these services, are also generated. Similarly, a goal for r_4 is generated, because of n_1. In the second iteration we generate the services and incentives for r_2, r_3 and r_4 as listed in Table 7 using the same method as in the first iteration. As a result of the generated services and incentives in the second iteration, new goals for propositions $r_5 - r_{13}$, which are the preconditions and incentives of the generated services, are introduced for the third iteration. However, in the third iteration we generate abilities instead of services for these goals, since iteration limit λ is reached.

Algorithm 3 defines our data generation process in detail, which returns a set of services (S), set of incentives (I), and set of abilities (A), generated using the process outlined above, and controlled by the parameters summarised in Table 6.

Algorithm 3 Data generation procedure with parameters γ, σ, ρ, ε, λ.

1: $\mathsf{S} \leftarrow \emptyset, \mathsf{I} \leftarrow \emptyset, \mathsf{A} \leftarrow \emptyset$
2: $currentGoals \leftarrow \{G_x(r_1), \ldots, G_x(r_\gamma)\}$
3: $i \leftarrow \gamma + 1$
4: **for** $l \leftarrow 1$ **to** $\lambda - 1$ **do**
5: $\quad nextGoals \leftarrow \emptyset$
6: $\quad$ **for all** $g \in currentGoals$ **do**
7: $\quad\quad //g = G_x(r)$
8: $\quad\quad start \leftarrow i$ // remember where we started generating propositions
9: $\quad\quad$ **for** $j \leftarrow 1$ **to** σ **do**
10: $\quad\quad\quad \mathsf{S} \leftarrow \mathsf{S} \cup \{S_x(y, q_i \wedge \ldots \wedge q_{i+\rho-1}, r)\}$
11: $\quad\quad\quad i \leftarrow i + \rho$
12: $\quad\quad$ **end for**
13: $\quad\quad \mathsf{I} \leftarrow \mathsf{I} \cup \{I_x(y, q_i, r), \ldots, I_x(y, q_{i+\varepsilon-1}, r)\}$
14: $\quad\quad i \leftarrow i + \varepsilon$
15: $\quad\quad nextGoals \leftarrow nextGoals \cup \{G_x(q_{start}), \ldots, G_x(q_{i-1})\}$
16: $\quad$ **end for**
17: $\quad currentGoals \leftarrow nextGoals$
18: **end for**
19: **for all** $g \in currentGoals$ **do**
20: $\quad //g = G_x(r)$
21: $\quad \mathsf{A} \leftarrow \mathsf{A} \cup \{A_x(\top, r)\}$
22: **end for**
23: **return** $\mathsf{S}, \mathsf{I}, \mathsf{A}$

The algorithm first generates the set of initial goals for x and keeps them in $currentGoals$ (line 2). The number of initial goals is determined by the parameter γ. Then the algorithm generates the services and incentives that can be used to support the goals in $currentGoals$. For each goal, the algorithm generates a set of services provided by some agent y (lines 6–12). The number of services is controlled by the parameter σ (line 9). For each generated service, the algorithm generates a set of preconditions and the number of preconditions is determined by the parameter ρ (line 10). Next, the algorithm generates a set of incentives for the current goal g, where the number of incentives is controlled by the parameter ε (line 13).

Remember that in our algorithms PROTOCOLBASED and GOALBASED, once a commitment is generated to guarantee provision of a service to support a goal in the context of a protocol, the preconditions of that service and of the incentive are added to the goal queue of the algorithm in order to find support for them. Accordingly, while generating our data set, we generate supporting services and abilities for preconditions and incentives of generated services. In order to achieve that, we generate a new goal for each precondition and incentive (line 15). These new goals are kept in a separate set called $nextGoals$. Once we generate services and incentives for all the goals in $currentGoals$, we replace the goals in $currentGoals$ with the goals in $nextGoals$ (line 17) and generate services and incentives to support these new goals in the next iteration. In this way the data generation algorithm iteratively creates services for the preconditions and incentives created in the previous iteration. In order to control the number of such iterations we use λ. Once the algorithm makes $\lambda - 1$ iterations, it stops generating new services for the preconditions and incentives of the previous iteration. Instead, the algorithm generates an ability, which has no precondition, in order to satisfy the preconditions and incentives of the previous iteration. Hence, these preconditions and incentives can be supported by the agent's abilities without requiring any other commitment.

Note that this algorithm generates a set of services and incentives for the worst possible case. That is, there is no duplication of preconditions and incentives for different services.

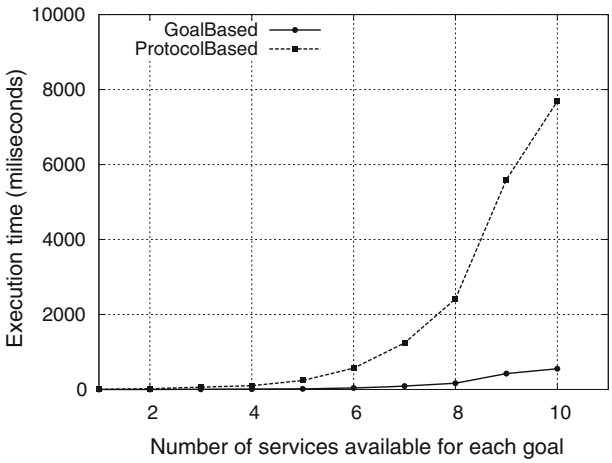


Fig. 3 Comparison of execution times of the PROTOCOLBASED and GOALBASED algorithms based on the number of available services for each goal

Hence, our protocol generation algorithms should always generate a new protocol for each utilized service, while working on this data.

In the rest of this section we present results of our experiments. We implemented[2] both PROTOCOLBASED and GOALBASED algorithms and the data generation algorithm we explained above in Java. We performed our experiments on an Intel i7-2620 2.7 GHz processor with 2 GB memory running Ubuntu 11.04 Linux. Each time measurement reported is the average of thirty runs.

In our first experiment we observed the effect of the σ parameter on the execution performance of our algorithms. This parameter controls the number of different services that can be used to support each goal. Since we generate an alternative protocol for each service, this parameter has a direct impact on the number of alternative protocols that can be generated. In order to investigate this issue further we conducted an experiment where we took the initial value of σ as 1 and then increased it up to 10, while measuring the execution time of our algorithms (in milliseconds). In this experiment we fix the other parameters as $\gamma = 1$, $\rho = 2$, $\varepsilon = 2$ and $\lambda = 3$. In other words, there is initially one goal. Each service has two preconditions, hence whenever a service is used, two new goals are introduced. Given a service, there are two alternative incentives, hence two alternative protocols are generated for each service. Note that we selected these particular values for other parameters after conducting trials with different values. For larger values of these parameters, the search space (i.e., number of services and incentives) grows rapidly and it is not possible for us to observe how the PROTOCOLBASED algorithm performs, since it does not terminate in a reasonable amount of time (e.g., in 10 min). On the other hand, for smaller values, the difference between the algorithms' performance is not adequate to make any conclusions (e.g., a few milliseconds).

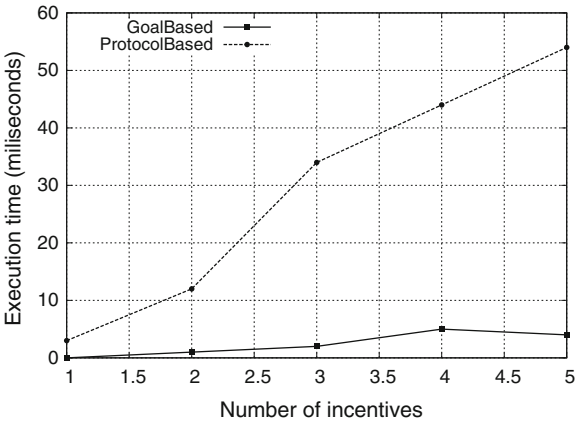
We show the results of the experiment in which we observe the effect of σ in Fig. 3. In the figure the x-axis is the number of services that can be requested for each goal and the y-axis is the execution time in milliseconds. The execution time of the PROTOCOLBASED algorithm grows exponentially. On the other hand the GOALBASED algorithm scales well even for 10 services per goal. Note that in practice we would expect to see that a given goal typically has

[2] The code is available from: http://mas.cmpe.boun.edu.tr/akin/protgen.html

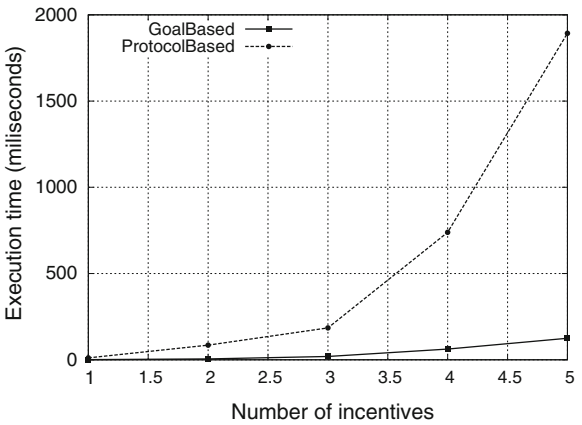
a relatively low number of different service types that could be used to achieve it. In other words, 10 possible service types per goal is quite a high value. In order to verify that the difference between the execution time of PROTOCOLBASED and GOALBASED is statistically significant, we conducted t-tests on the mean execution times (over thirty runs) of these algorithms for each different value of σ. In these t-tests our null hypothesis is "for a given σ, execution time of PROTOCOLBASED and GOALBASED are not different" and our alternative hypothesis is "for a given σ, execution time of GOALBASED is faster than PROTOCOLBASED". The results of the t-tests show that for $\sigma = 1$ we have $p < 0.05$ and for $\sigma \geq 2$ we have $p < 0.005$. Hence, we reject the null hypothesis for all the reported σ values and conclude that GOALBASED is significantly faster than PROTOCOLBASED.

By examining the execution traces of our algorithms, we conclude that the main reason for this performance difference between the algorithms is the reuse of sub-protocols by the GOALBASED algorithm. The PROTOCOLBASED algorithm generates the same sub-protocols many times in different protocols. On the other hand the GOALBASED algorithm reuses previously generated sub-protocols effectively to avoid redundant computation. In order to examine this issue in detail, we conducted another experiment, in which we fixed the number of alternative services and used different values for ρ and ε. These parameters have the following effects on the execution of our algorithms. If there are two incentives n_1 and n_2 for a service (i.e., $\varepsilon = 2$), then our algorithms generate two alternative protocols for each service, such that each protocol includes a commitment that contains one of these incentives to use the service. On the other hand, since both protocols use the same service, all the preconditions of this service should be supported by both protocols. For example, if there are two incentives n_1 and n_2 for a goal and two preconditions r_1 and r_2 for the service that brings about the goal, then there are two protocols, such that the first protocol supports n_1, r_1 and r_2, and the second protocol supports n_2, r_1 and r_2. Since the PROTOCOLBASED algorithm does not reuse sub-protocols, for this example it has to find support for r_1 and r_2 separately for each protocol. More generally, when ε grows, the number of redundant computations of the PROTOCOLBASED algorithm also grows, since more incentives means more protocols. On the other hand, ρ determines the total number of preconditions for a service and a higher value for ρ also results in more redundant computation done by the PROTOCOLBASED algorithm for each protocol, since there are more preconditions to be supported.

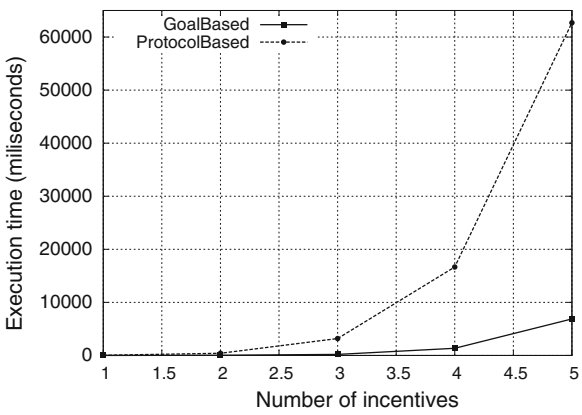
We present our results in Fig. 4. Each sub-figure shows the execution time of the PROTOCOLBASED and GOALBASED algorithms for different values of ε (i.e., number of incentives) between one and five. Additionally, in each sub-figure the value of ρ (i.e., number of preconditions) is different. Specifically, ρ is equal to one, two and three in Fig. 4a–c, respectively. For this experiment, we use the other parameters $\gamma = 1$, $\sigma = 3$ and $\lambda = 3$. In Fig. 4a the number of preconditions for each service is just one. Hence, the time required to find support for a service's precondition is short. Accordingly, even though the amount of redundant computation increases with the number of incentives, the PROTOCOLBASED algorithm's execution time still increases almost linearly. However, as Fig. 4b, c show, when the time required to find support for a service's precondition is longer, since there are more preconditions, the total execution time of the PROTOCOLBASED algorithm grows exponentially with the number of incentives. On the other hand, for all ε and ρ values, the GOALBASED algorithm scales well. In order to verify that the difference between the execution time of PROTOCOLBASED and GOALBASED is statistically significant, we conducted *t* tests on the mean execution times (over thirty runs) of these algorithms for each different combination of ε and ρ values. In these *t* tests our null hypothesis is "for a given ε and ρ value, execution time of PROTOCOLBASED and GOALBASED are not different" and our alternative hypothesis is "for a given ε and ρ value, execution time of GOALBASED is faster than PROTOCOLBASED". The results of the



(a) $\rho = 1, \varepsilon = 1, \ldots, 5$



(b) $\rho = 2, \varepsilon = 1, \ldots, 5$



(c) $\rho = 3, \varepsilon = 1, \ldots, 5$

Fig. 4 Comparison of execution times of the PROTOCOLBASED and GOALBASED algorithms based on the number of preconditions (ρ) and incentives (ε) for each service

t-tests show that for every combination of ε and ρ we have $p < 0.005$. Hence, we reject the null hypothesis for all the reported combinations of ε and ρ and conclude that GOALBASED is significantly faster than PROTOCOLBASED.

4 Protocol ranking

After the protocols are generated, the next phase in our framework is for the agent to evaluate these protocols and rank them to reflect its own preferences. The ranking results are used to guide the agent that generated the protocols (e.g., in our running example, the customer) in selecting and offering a protocol to the potential participants, since essentially the ranking captures how preferred each protocol is. Different agents might have different strategies to utilize these rankings. For example, an agent might begin by offering its most preferred option, and if it is rejected, it can proceed to offer its second most preferred option, i.e., the agent slowly concedes over his preferences. An alternative customer agent that is hoping to reach an agreement early on, may prefer to offer a protocol that is more likely to be accepted by the retailer. Various negotiation techniques can be applied at this step to decide on the protocol that will be chosen by the participants [14].

Ranking the protocols then means evaluating each protocol based on a set of criteria. We employ two important criteria: the *utility* of a protocol, how beneficial the protocol is when the benefit and the cost are considered, and the *risk-discounted utility* of a protocol, the utility taking into account risk. Note that since both of these criteria are subjective, the rankings of the same protocols for different agents might vary. That is, two different agents may have different rankings of a given protocol.

The *utility* of a protocol for an agent reflects how much an agent would gain by enacting the protocol. The utility is basically the difference between the benefit of the protocol and its cost ($utility = benefit - cost$). Intuitively, each agent would want to maximize its utility, given that two protocols achieve the same goals. There may also be some protocols that are not acceptable to a given agent, i.e., if the utility is negative (i.e. the cost exceeds the benefit).

The *risk-discounted utility* extends the utility by also considering risk. Since various agents are involved in enacting the protocol, it is possible for some of the agents to fail to fulfill their commitments. This means that the expected benefit may not be realised. We therefore extend the definition of utility by *discounting* the benefits based on the risk of the protocol. The risk of a protocol is based on how trustworthy the agents enacting the protocol are for the various services that they provide in the context of the protocol. If all agents are fully trustworthy, then there is no risk associated for a protocol. In that situation, risk-discounted utility is equivalent to utility. However, if any of the agents in the protocol are somewhat untrustworthy, then the protocol would be in risk, and the expected utility of the protocol may be jeopardized.

4.1 Utility of protocols

We define the utility of a protocol in terms of the benefit that the agent may derive from the protocol's successful enactment, discounted by the cost that the agent incurs in playing its part in the protocol (i.e. $utility = benefit - cost$). As noted earlier, this means that the utility of a protocol is specific to an agent: in assessing benefit vs. cost, an agent is considering the benefit that it derives, and the costs that it incurs.

We assume that we know the cost of each service [denoted $cost_x(A_x(d, r))$]. For example, the customer, may incur a cost of 5 units in providing tools [that is, $cost_{Cus}(a_1) = 5$], a cost

Table 8 Cost of services and benefit of relevant propositions from customer's perspective

Ability	Cost
$a_1 = A_{Cus}(HaveTools, ToolsProvided)$	5
$a_2 = A_{Cus}(HaveMaterials, Bui_1MaterialsProvided)$	1
$a_3 = A_{Cus}(HaveMaterials, Bui_2MaterialsProvided)$	1
$a_4 = A_{Cus}(\top, MaterialsPaid)$	2
$a_5 = A_{Cus}(\top, ToolsPaid)$	3
$a_6 = A_{Cus}(\top, FurniturePaid)$	12
$a_7 = A_{Cus}(\top, Bui_1Paid)$	4
$a_8 = A_{Cus}(\top, Bui_2Paid)$	5
Proposition	**Benefit**
HaveFurniture	15
HaveTools	8
HaveMaterials	0

of 1 in providing materials to either builder (a_2 and a_3), and a cost of 2 in paying for materials (a_4). These (and other) example costs are summarised in Table 8. In fact, when assessing the utility from a given agent's perspective, the costs are incurred from the services of that agent, i.e. from its abilities.

Given the costs of using services, we define the cost of a *proposition* as the maximal possible cost over the services that could be used to bring about the proposition. We use a maximum since in general we may not be able to decide or control which service is usable to achieve the desired property. Formally:

$$cost_x(r) = \max_{A_x(d',r') \in \mathcal{A} \wedge r' \Rightarrow r} cost_x(A_x(d', r'))$$

We also assume that each agent is aware of the benefit that it derives from each proposition [$benefit_x(r)$]. For example, the customer agent may derive a benefit of 15 units from having furniture [i.e. $benefit_{Cus}(HaveFurniture) = 15$], a benefit of 8 from having tools, and no benefit from having materials (see Table 8). Note that agents do not need to know each other's assigned benefits.

To calculate the overall cost (respectively benefit) of a protocol we need to consider the overall set of propositions involved, and then sum their costs (respectively benefits). We can't just calculate the cost for each commitment and then add them up, because a proposition that occurs in more than one commitment is only achieved once. For example, in protocol p_5 (see Table 5), the customer only pays for tools once, even though $ToolsPaid$ appears in two commitments in the protocol. In other words, we compute the cost (respectively benefit) of a protocol by accumulating the relevant propositions in the protocol, then working out what services are needed, and finally working out their costs (respectively benefits). For example, to derive the cost of protocol p_5 from the customer's perspective, the relevant propositions (that the customer has to bring about) are $ToolsPaid$, $Bui_2MaterialsProvided$, $ToolsProvided$, and Bui_2Paid. Each of these propositions in fact can only be achieved by a single ability (respectively a_5, a_3, a_1 and a_8). Given the example costs in Table 8, we have that the cost of p_5 from the customer's perspective is just: $cost_{Cus}(a_5) + cost_{Cus}(a_3) + cost_{Cus}(a_1) + cost_{Cus}(a_8) = 3 + 1 + 5 + 5 = 14$. On the other hand, the *benefit* of p_5 from the customer's perspective is the benefit of the relevant propo-

sitions, which are $HaveMaterials$, $HaveTools$ and $HaveFurniture$, giving a benefit of $0 + 8 + 15 = 23$, and the utility of p_5 is then $23 - 14 = 9$.

Formally, we define the utility of a protocol p that was derived to achieve goals $\mathcal{G}$, from the perspective of agent x [denoted $utility_x(p)$] as follows:

$$\begin{aligned} utility_x(p) &= benefit_x(p) - cost_x(p) \\ benefit_x(p) &= \sum_{r \in m \cup g} benefit_x(r) \\ &\quad \text{where } m = \bigcup_{c \in p} rel_x^{benefit}(c) \text{ and } g = \{r \mid G_x(r) \in \mathcal{G}\} \\ cost_x(p) &= \sum_{r \in m} cost_x(r) \\ &\quad \text{where } m = \bigcup_{c \in p} rel_x^{cost}(c) \end{aligned} \tag{1}$$

Note that when calculating the benefit we also include the top-level goals g: these goals are achieved by all of the protocols, but are not always explicitly a condition within the commitments generated. Given the utility of a protocol, we conclude that a protocol is acceptable if it has a non-negative utility.

The definitions above make use of a notion of *relevant propositions* which we now define. The intuition is that when considering the cost of a commitment $C(x, y, d, r)$ from the perspective of agent x then, since x is responsible for bringing about r, the relevant proposition for cost is r. Similarly, from the perspective of agent y, the relevant proposition for cost calculations is d. On the other hand, when considering the calculation of *benefit*, this is reversed: the relevant proposition from the perspective of x when calculating benefit is d, and from y's perspective it is r. Formally, we define this using auxiliary functions $rel_x^{cost}(c)$ (for cost) and $rel_x^{benefit}(c)$ (for benefit). In essence, the function $rel_x^{cost}(c)$ extracts the propositions that agent x is responsible for bringing about, since these are the basis for the costs that the agent will incur when the protocol is enacted. If x is the debtor, then it is responsible for the consequent of the commitment, if it is the creditor then it is responsible for the antecedent, and if x is neither the debtor nor creditor then it does not have any responsibilities with respect to c (i.e. $rel_x^{cost}(c) = \emptyset$). Note that if the antecedent is a conjunction then we break it up (first case below). We also define a reversed function (denoted $rel_x^{benefit}$) that extracts the propositions that the agent will benefit from. Formally, we have:

$$rel_x^{cost}(C(x_1, x_2, q_1 \wedge \cdots \wedge q_n, r)) = \bigcup_{i=1,\ldots,n} rel_x^{cost}(C(x_1, x_2, q_i, r))$$

$$rel_x^{cost}(C(x_1, x_2, q, r)) = \begin{cases} \{r\}, & \text{if } x = x_1 \\ \{q\}, & \text{if } x = x_2 \\ \emptyset, & \text{otherwise} \end{cases}$$

$$rel_x^{benefit}(C(x_1, x_2, q_1 \wedge \cdots \wedge q_n, r)) = \bigcup_{i=1,\ldots,n} rel_x^{benefit}(C(x_1, x_2, q_i, r))$$

$$rel_x^{benefit}(C(x_1, x_2, q, r)) = \begin{cases} \{q\}, & \text{if } x = x_1 \\ \{r\}, & \text{if } x = x_2 \\ \emptyset, & \text{otherwise} \end{cases}$$

Example values for costs and benefits are given in Table 8. These values were picked to reflect realistic assumptions about our example. For instance, having furniture ($HaveFurniture$) has the highest benefit, since this is the ultimate goal of the customer. Having tools ($HaveTools$) has some benefit since tools can be reused but yields less benefit compared to having the furniture. Having materials by itself has no benefit. The scenario that we model assumes that tools are provided as a temporary loan, so the benefit from having tools ($HaveTools$) is realised even in protocols that involve providing (i.e. lending) the tools to the builder (i.e. making $ToolsProvided$ true).

In terms of costs, we select costs that make sense for the domain. Specifically, we select costs that make buying furniture from the merchant cheaper than the combined cost of obtaining tools, materials, and paying the second builder (but the first builder is still cheaper, since only materials need to be provided). However, if the agent already has tools and materials, then paying the second builder is cheaper than buying ready-made furniture from the merchant. Specifically, if the customer does not already have tools and materials, then the cost of obtaining the furniture is less than the costs of first obtaining materials ($a_4 = 2$) and tools ($a_5 = 3$), then providing the tools ($a_1 = 5$) and the materials to the second builder ($a_3 = 1$) and finally paying for assembly ($a_8 = 5$). However, if the customer already has materials and tools, then the cost of buying furniture ($a_6 = 12$) is greater than the combined costs of providing the materials to the builder ($a_3 = 1$), providing tools ($a_1 = 5$) and paying for assembly ($a_8 = 5$).

Table 9 provides an evaluation of the generated protocols. The first three labelled columns denote the benefit, cost and utility of the protocol, respectively and the remaining columns, the respective ranks (i.e., the protocol with the lowest cost has Cost Rank of 1, and the protocol with the highest utility has Utility Rank of 1; ties are split: e.g. p_4 and p_7 have the same cost, and so instead of ranking them as 6th and 7th, they are both given rank 6.5).

The cost by itself is an important indicator for a protocol, since it is a bound on the worse case. If other agents fail to fulfil their commitments, then this is the highest possible cost that the agent will pay whilst receiving no benefit. Note that this "worst case scenario" may actually be too pessimistic. For example, in p_4, if the retailer fails to fulfil its role (the first two commitments, making $HaveMaterials$ and $HaveTools$ true) then the customer will not be able to proceed to interacting with the builder. So in this case even though the cost of the protocol is 16, this cost can only be incurred if the retailer plays its role, in which case even if the builder fails, the customer still derives some benefit from having tools. The dual of this is the "best case scenario" when everyone fulfils their commitments. In that case, the agent will also receive some benefit and the overall gain will be reflected in the utility. For example, even though p_4 incurs the highest cost, it is not the worst protocol in terms of utility since it has a benefit of 23 and the overall utility comes out to be 7. On the other hand p_1

Table 9 Customer's evaluation of protocol utility

	Benefit	Cost	Utility	Cost Rank	Utility Rank
p_1	15	12	3	3	7
p_2	15	7	8	1	3
p_3	15	8	7	2	5
p_4	23	16	7	6.5	5
p_5	23	14	9	5	2
p_6	23	13	10	4	1
p_7	23	16	7	6.5	5

is the worst protocol in this respect, since even if all the agents fulfil their duties, the gain is only 3.

4.2 Incorporating risk

As expected, looking from the worst-case and the best-case ranks the protocols differently. The next obvious question is how likely is the worst-case? This intuitively will help us understand how risky a protocol is. In this section we extend the definition of benefit to include a risk assessment. In essence, we *discount* a given benefit by the risk that the benefit may not be realised.

Note that in the discussion below we are always calculating risk from the point of view of the agent that generates the protocol. This is because, in effect, the generating agent (e.g. customer) takes on all the risk. For other agents, e.g. builder, or retailer, all the commitments are of the form $C(me, Cus, d \wedge w, r)$, in other words, the builder or retailer is only committed to do something *after* the customer has done its part, including providing an appropriate incentive. This means that there is no risk: if the precondition and incentive are not brought about, then the agent is not committed to act, since the commitment is still conditional.

In order to quantify the risk of a protocol, we start from the trust relation among the agents that are involved in that protocol. In general, an agent's trust in another depends on the particular service in question. An agent might trust a retailer for delivering furniture but may not trust him or her for actually assembling the furniture. Hence, we consider the trust of an agent x for another agent y with respect to a particular service s, denoted as $T_x(y, S_x(y, d, r)) \in [0, 1]$. This trust value represents how likely y is to complete service $S_x(y, d, r)$ from agent x's perspective. As is customary in the trust literature, we assume that the higher values represent more trust, whereas lower values represent low trust. Hence, a trust value of 1 would mean that agent x believes y would definitely carry out the service, whereas a trust value of 0 would mean that x believes y will definitely *not* carry out the service. In general, the trust value is updated dynamically based on the outcomes of agents' interactions or new information coming in about the agent from other (trusted) sources. There are many existing mechanisms in the literature for managing and disseminating trust (e.g. [19,20,23]). Here, we assume that the agent has such a mechanism to maintain an accurate trust evaluation of other agents.

The actual values of $T_x(y, S_x(y, d, r))$ are subject to a number of constraints. First, since we consider trust between agents based on a particular service, agent x can only consider y trustful for a service s which x believes y provides, formally, if $S_x(y, d, r) \notin \mathcal{B}$ (recall that $\mathcal{B}$ is the agent's beliefs about other agents' services) then $T_x(y, S_x(y, d, r)) = 0$. Second, if the service is trivial, then trust is always 1, formally, $T_x(y, S_x(y, d, \top)) = 1$. Third, an agent always trusts itself, formally, $T_x(y, S_x(y, d, r)) = 1$ if $x = y$. Note that to simplify the definitions in this section we assume for convenience that for any ability that an agent has $A_x(d, r) \in \mathcal{A}$ there is also a corresponding implicit service belief: $S_x(x, d, r) \in \mathcal{B}$.

Given $T_x(y, S_x(y, d, r))$, we proceed to define the trust that an agent x has in agent y to bring about a proposition r (denoted $T_x(y, r)$). This can be defined in terms of the relevant services. One special case is that if y does not have any relevant services, then the trust is 0. Otherwise we define $T_x(y, r)$ by considering all the relevant services that y can use to bring about r' such that r' implies r:

$$T_x(y, r) = \begin{cases} 0, & \text{if } \neg\exists\ S_x(y, d, r') \in \mathcal{B} \text{ such that } r' \Rightarrow r \\ \bigoplus_{S_x(y,d,r') \in \mathcal{B} \wedge r' \Rightarrow r} T_x(y, S_x(y, d, r')), & \text{otherwise} \end{cases}$$

Thus, x considers all services that would enable r to be realized and combines the trust for these services using an auxiliary function $\oplus$. This auxiliary function can be defined using max, i.e. $T_x(y, s_1) \oplus T_x(y, s_2) = \max(T_x(y, s_1), T_x(y, s_2))$, meaning that the combined trust can at most be equal to the most trusted service.

However, this definition is not complete yet: given a service $S_x(y, d, r')$, in order to assess the risk that the service will not be successfully used to bring about r' there are actually two sources of risk: one is that agent y will fail to carry out its role, which is captured by the definition above; the other source of risk is that the precondition d may not be brought about. To capture this source of risk we define the function $T_x^p(y, r)$ [see Eq. (2) below], which is an extension of $T_x(y, r)$ (note the superscript p which is a parameter bound to the generated protocol). When considering a given service, we multiply the trust in that service by the trust that agent x has in the precondition d being achieved in the context of protocol p, denoted $T_x^p(d)$.

The function $T_x^p(d)$ [see Eqs. (3) and (4) below] takes the protocol p as an argument, as well as the identifier of the agent (x) and the condition that is desired (d). The function computes the trust that x has in the condition d being achieved by other agents operating with the protocol p. Equation (3) below simply decomposes conjunctions and accumulates trust using an auxiliary function ("$\otimes$") which can be defined in a number of ways. In the examples later we use the definition $\otimes \equiv \times$, where $\times$ is multiplication. Equation (4) uses the protocol: it identifies those agents that are relevant, according to the protocol, to achieve r, and uses $T_x^p(y, r)$ to assess for each such agent y the level of trust that x has in y's ability to bring about r in the context of p. In addition to considering agents that are relevant according to the protocol, we also consider the agent itself relevant, since the agent's abilities are available.

$$T_x^p(y, r) = \begin{cases} 0, & \text{if } \neg\exists\, S_x(y, d, r') \in \mathcal{B} \text{ such that } r' \Rightarrow r \\ \bigoplus_{S_x(y,d,r') \in \mathcal{B} \wedge r' \Rightarrow r} T_x(y, S_x(y, d, r')) \times T_x^p(d), & \text{otherwise} \end{cases} \tag{2}$$

$$T_x^p(q_1 \wedge q_2) = T_x^p(q_1) \otimes T_x^p(q_2) \tag{3}$$

$$T_x^p(r) = T_x^p(x, r) \oplus \bigoplus_{C(y,x,d \wedge w, r') \in p \ \wedge\ r' \Rightarrow r} T_x^p(y, r) \tag{4}$$

Now, given that the agent can assess the trust value for a proposition, we reflect this in the calculation of the utility, to derive an *expected value* on the utility. The intuition is to calculate how likely it is that the commitment will be fulfilled, thereby yielding its expected value. Hence, rather than using the utility measure defined earlier, we extend it by adding $T_x^p(r)$:

$$benefit_x(p) = \sum_{r \in m \cup g} benefit_x(r) \times T_x^p(r)$$

$$\text{where } m = \bigcup_{c \in p} rel_x^{benefit}(c) \text{ and } g = \{r \mid G_x(r) \in \mathcal{G}\}$$

The difference here, compared with the previous definition (Eq. (1) in Sect. 4.1), is that the benefit is being *discounted* based on the trust that agent x has in the ability of the condition r to be brought about by the relevant agents in the system. Note that if the trustworthiness ($T_x^p(r)$) is 0, then the benefit will be 0. Conversely, if the trustworthiness is 1, then the utility will reflect the best outcome, reflecting the "best case scenario".

Table 10 Customer's trust of services

Service	Trust
$s_1 = S_{Cus}(Ret, \top, HaveMaterials)$	0.7
$s_2 = S_{Cus}(Ret, \top, HaveTools)$	0.6
$s_3 = S_{Cus}(Mer, \top, HaveFurniture)$	0.9
$s_4 = S_{Cus}(Bui_1, Bui_1MaterialsProvided, HaveFurniture)$	0.8
$s_5 = S_{Cus}(Bui_2, Bui_2MaterialsProvided \wedge ToolsProvided, HaveFurniture)$	0.2

Let us now consider how this risk-based discount of benefit applies to our example. Suppose the customer has the following trust relations as reflected in Table 10, and consider protocol $p_4 = c_1, c_2, c_3$ in Table 5, where the commitments are:

- $c_1 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$
- $c_2 = C(Ret, Cus, ToolsPaid, HaveTools)$
- $c_3 = C(Bui_2, Cus, Bui_2MaterialsProvided \wedge ToolsProvided \wedge Bui_2Paid, HaveFurniture)$

The cost of p_4 is calculated as in the previous section. We first determine the set of relevant propositions for the customer, which is $m = \{MaterialsPaid, ToolsPaid, Bui_2MaterialsProvided, ToolsProvided, Bui_2Paid\}$. We then determine the cost of each proposition, and sum them. In this case each proposition has exactly one relevant ability (respectively a_4 with $cost_{Cus}(a_4) = 2$; a_5 with cost 3; a_3 with cost 1; a_1 with cost 5; and a_8 with cost 5), which gives a total cost of $2 + 3 + 1 + 5 + 5 = 16$.

Now let us consider the discounted benefit of protocol p_4. We firstly determine the set of relevant propositions $m = \{HaveMaterials, HaveTools, HaveFurniture\}$ and the set of top-level goals $g = \{HaveFurniture\}$. For each of the three resulting propositions, we look up the benefit (Table 8) and compute the risk-based discount:

$$\begin{aligned} benefit_{Cus}(p_4) &= (benefit_{Cus}(HaveMaterials) \times T^{p_4}_{Cus}(HaveMaterials)) \\ &\quad + (benefit_{Cus}(HaveTools) \times T^{p_4}_{Cus}(HaveTools)) \\ &\quad + (benefit_{Cus}(HaveFurniture) \times T^{p_4}_{Cus}(HaveFurniture) \\ &= 0 + (8 \times T^{p_4}_{Cus}(HaveTools)) + (15 \times T^{p_4}_{Cus}(HaveFurniture)) \end{aligned}$$

We now consider the two instances of $T^p_x(r)$. For $T^{p_4}_{Cus}(HaveTools)$ we consider the commitments in p_4 and find that only c_2 is relevant, and hence that the retailer is the only relevant agent. Similarly for $T^{p_4}_{Cus}(HaveFurniture)$ there is only a single relevant commitment (c_3) and only Bui_2 is relevant. In both cases, the agent itself does not have a relevant ability, and so the $T^p_x(x, r)$ term reduces to 0. It's worth noting that although in general the first builder and the retailer are also relevant to bringing about the proposition $HaveFurniture$, in the context of p_4, they are not relevant. We therefore have:

$$\begin{aligned} &T^{p_4}_{Cus}(HaveTools) \\ &\quad = T_{Cus}(Ret, S_{Cus}(Ret, \top, HaveTools)) \times T^{p_4}_{Cus}(\top) = 0.6 \times 1 = 0.6 \\ &T^{p_4}_{Cus}(HaveFurniture) \\ &\quad = T_{Cus}(Bui_2, S_{Cus}(Bui_2, Bui_2MaterialsProvided \\ &\qquad \wedge ToolsProvided, HaveFurniture)) \\ &\qquad \times T^{p_4}_{Cus}(Bui_2MaterialsProvided \wedge ToolsProvided) \\ &\quad = 0.2 \times T^{p_4}_{Cus}(Bui_2MaterialsProvided) \otimes T^{p_4}_{Cus}(ToolsProvided) \end{aligned}$$

We then need to determine $T_{Cus}^{p_4}(Bui_2MaterialsProvided)$ and $T_{Cus}^{p_4}(ToolsProvided)$. In both cases there are no relevant agents according to the protocol, but the customer itself is relevant, and we therefore consider $T_{Cus}^{p_4}(Cus, r)$ which, since agents trust themselves (recall the constraint on trust), and there is in each case a single relevant ability, reduces to $T_{Cus}^{p_4}(d)$ where d is the precondition for the ability that achieves r.

$$\begin{aligned}
&T_{Cus}^{p_4}(Bui_2MaterialsProvided)\\
&\quad = T_{Cus}^{p_4}(HaveMaterials) = T_{Cus}(Ret, s_1) \times T_{Cus}^{p_4}(\top) = 0.7 \times 1 = 0.7\\
&T_{Cus}^{p_4}(ToolsProvided) = T_{Cus}^{p_4}(HaveTools) \;=\; 0.6
\end{aligned}$$

We therefore have that

$$\begin{aligned}
&T_{Cus}^{p_4}(HaveFurniture)\\
&\quad = 0.2 \times T_{Cus}^{p_4}(Bui_2MaterialsProvided) \otimes T_{Cus}^{p_4}(ToolsProvided)\\
&\quad = 0.2 \times 0.7 \times 0.6 \;=\; 0.084\\
&benefit_{Cus}(p_4)\\
&\quad = 0 + (8 \times T_{Cus}^{p_4}(HaveTools)) + (15 \times T_{Cus}^{p_4}(HaveFurniture))\\
&\quad = 0 + (8 \times 0.6) + (15 \times 0.084) \;=\; 6.06
\end{aligned}$$

Since the cost of p_4 is 16 and the risk-discounted benefit is 6.06, we have that the overall expected value of the utility is $6.06 - 16 = -9.94$. In other words, since the customer's trust in the retailer is not that high, and the customer's trust in the second builder quite low, protocol p_4 is too risky to be considered acceptable.

Table 11 shows the results for all of the protocols. It includes the risk-discounted benefit, the cost (unchanged from Table 9), the expected value for the utility (risk-discounted benefit minus cost), and the rank of the protocol by expected value for utility.

We see that the rank of a protocol is dependent on both the utility of the protocol and the trustworthiness of the participating agents. Even though a protocol has a potential to create a high utility for an agent, if others do not play their parts, the outcome might not be as desired. By factoring in the trustworthiness of the agents, we can compute the expected values for the utilities, which reflect a more realistic outcome. For example, if we look at p_1 (in Table 9) we see that it is the worst protocol in terms of utility; however, it is the best one in terms of expected value for utility (Table 11). The high trustworthiness of the merchant in the protocol means a low risk for the protocol, yielding a high overall expected value for utility. Contrast that with p_6, which is the best protocol in terms of utility (Table 9). However, the players in

Table 11 Customer's evaluation of protocol's expected value for utility

	Risk-discounted benefit	Cost	Expected value for utility	Expected rank
p_1	13.5	12	1.5	1
p_2	8.4	7	1.4	2
p_3	8.4	8	0.4	3
p_4	6.06	16	−9.94	6.5
p_5	6.06	14	−7.94	5
p_6	6.06	13	−6.94	4
p_7	6.06	16	−9.94	6.5

that protocol are untrustworthy for the service they provide, yielding a lower expected value for utility (ranked 4th in Table 11).

5 Using generated protocols

In the two previous sections we studied how an agent can generate a set of protocols to support its goals and how the agent can rank these generated protocols combining trust and benefit. The next phases are reaching agreement on one of these protocols to use it, and enactment of the agreed protocol.

In order to reach an agreement on a protocol we propose a procedure in Algorithm 4 which can be used by the agent that has generated and ranked the protocols. This section presents an example of how the third phase, reaching agreement, can be realised. There are many alternatives possible to the procedure that we present below.

Algorithm 4 Agreement procedure to reach an agreement on a protocol.

Require: $\mathcal{P}$, list of generated protocols
1: $\mathcal{P}' \leftarrow order(\mathcal{P})$
2: **for all** $p \in \mathcal{P}'$ **do**
3: $\quad \mathcal{C} = \emptyset$
4: $\quad$ **for all** $C(x, y, d, r) \in p$ **do**
5: $\quad\quad response \leftarrow propose(x, C(x, y, d, r))$
6: $\quad\quad$ **if** $response = ACCEPT$ **then**
7: $\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{C(x, y, d, r)\}$
8: $\quad\quad$ **else if** $response = REJECT$ **then**
9: $\quad\quad\quad$ **for all** $C(x, y, d, r) \in \mathcal{C}$ **do**
10: $\quad\quad\quad\quad informCancel(x, C(x, y, d, r))$
11: $\quad\quad\quad$ **end for**
12: $\quad\quad\quad$ **goto** 2 // *next protocol …*
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad$ **return** p
16: **end for**
17: **return** *null*

In this procedure, the agent first selects one of the generated protocols. Then, the agent proposes creation of the commitments of the protocol to the corresponding debtors. If all the debtors accept to create the proposed commitments, an agreement is established over the protocol. Note that the debtor only accepts to create the commitments, but they do not actually create the commitments at this stage. Actual creation of the commitments happens in the enactment phase.

A question that we should answer is how the agent selects the protocols that it offers to other agents. This can be done in several ways depending on the design of the agent. For example, the agent may first select the protocol, which has the highest utility, to offer to the other agents. If the selected protocol is rejected (i.e., at least one debtor rejects an offered commitment), then the agent may concede and selects the next protocol that has a lower utility than the previously offered protocol. The agent concedes until either an agreement is reached on a protocol (i.e., every commitment in the protocol is accepted by its debtor) or there is no protocol left that has a positive utility for the agent. In our procedure, we do not specify a fixed method for the ordering of protocols. Instead we use the $order$ function that

takes a set of protocols $\mathcal{P}$ and returns an ordering of these protocols with respect to some criteria specified in the agent's design.

Let us explain the agreement procedure we present in Algorithm 4 in detail. The procedure requires the list of protocols ($\mathcal{P}$) as input and returns the agreed protocol, or *null* if no agreement is established. The function $order(\mathcal{P})$ is an agent specific function, which takes a set of protocols and returns the protocols in a specific order to offer the other agents.

The procedure iterates over $\mathcal{P}'$ and performs the following operations for each protocol p that has a positive utility. For each commitment $C(x, y, d, r)$ in p, the agent offers the commitment to its debtor x using the $propose$ function. If the debtor accepts the commitment, it is added to the list of accepted commitments ($\mathcal{C}$). Finally, if every commitment in a protocol is accepted (i.e., there is agreement on the protocol), then the protocol is returned.

Otherwise, if at least one of the commitments in p is rejected by its debtor, the agent omits p and concedes to the next protocol in $\mathcal{P}$, which has a lower utility than p. Also note that, when a protocol is rejected, the agent informs other debtors, who have already accepted a commitment in the context of the protocol, about the cancellation of the protocol using *informCancel* function. Finally, if the agent offers all the protocols and none of them is accepted, then the procedure returns *null*, which indicates that there is no agreement on any protocol.

Our agreement procedure is inspired by the monotonic concession procedure proposed by Rosenschein and Zlotkin [18]. One difference in our case is that the offers are only made by the agent that generates the protocols and the other agents only reply to the offers by accepting or rejecting the proposed commitments.

For our running example, the customer agent orders the protocols as they are ranked in Table 11. Accordingly, the first protocol that it would propose to the other agents is p_1. In this protocol, the only other involved agent is the merchant. If the merchant agrees to participate, the protocol is set to be p_1. If the merchant rejects the proposed protocol, then the agent would offer its second best alternative, p_2 to the retailer and the first builder. If both agree to participate, only then p_2 would be chosen as the protocol. If either one rejects, the customer would move on to its third best choice, and so on. Once the protocol is agreed upon, then the agents will enact the protocol as explained in Section 2.

Note that the other agents involved would themselves evaluate each proposed protocol using an appropriate metric, such as the definition of utility that we propose here. Using such a mechanism, each agent can decide if it would be beneficial for itself to take part in a protocol and act accordingly. For example, from the merchant's perspective, the utility of protocol p_1 is the benefit that the merchant derives from being paid minus the cost of providing the furniture to the customer. This would be normally greater than zero (otherwise the merchant would be making a loss on every sale).

6 Discussion

This paper has made a number of contributions. Firstly, we proposed that agents should be able to generate commitment protocols at run-time and accordingly developed a framework to enable agents to generate, rank and negotiate over protocols. Secondly, for the generation of protocols we developed two algorithms that use the generating agent's abilities and beliefs about other agents' services. A distinctive feature of these algorithm is consideration of incentives to persuade other agents to provide their services. We showed, using an experimental evaluation, that the GoalBased algorithm, which is based on the divide-and-conquer strategy, is more efficient than the ProtocolBased algorithm that uses depth-first

traversal. We also proved the soundness and completeness of the GoalBased algorithm. Thirdly, for the ranking of protocols we proposed a utility based calculation, using cost and benefit. We then extended this to include risk-based discounting, using trust relations between the agents. Applying these ranking metrics to the case study showed that even if a protocol has a high utility, the protocol may not be as desired when the participating agents are untrustworthy. Finally, for negotiating over protocols, we provided a concession procedure.

This paper significantly extends our earlier work [12] which proposed an algorithm for generating commitment protocols, but did not consider ranking. Additionally, the earlier work did not include formal results (soundness and completeness), or an experimental evaluation. Furthermore, this paper defined the more efficient GoalBased algorithm. In addition, the older algorithm incorporated the agent's current state into protocol generation. Thus, the generated protocols would not always be applicable in other settings. Here, the generated protocols are independent of the current world state. This enables the agent to choose the protocol that best suits its current state, enabling flexibility in carrying out the interactions.

We now review related work in three areas: protocol generation, ranking and trust, and more broadly, the relationship between commitments and goals.

There have been a few papers that have considered, in various forms, the issue of how to plan an effective interaction between agents. Telang, Meneguzzi and Singh [25] use Hierarchical Task Network (HTN) planning: given a set of agents and their goals, their objective is to come up with a global plan to satisfy these goals. The resulting plan is a set of commitments and the operations that are required to fulfil these commitments, which lead to the achievement of the agents' goals. By contrast, in our approach we do not aim to generate a complete plan that specifies the operations that should be carried out by the agents. Instead, our objective is to come up with a protocol, without enforcing agents to take certain actions. Their approach is centralized both in terms of knowledge representation, and plan generation. In other words they assume that there is a central planner, which knows the goals and capabilities of all agents. Besides, it is assumed that the agents' preferences are available to the central planner. On the other hand, in our approach the agent that generates a commitment protocol uses only the locally available knowledge.

Alberti et al. [1] study service discovery and automatic contracting in semantic Web services. They assume that the services advertise for their functionalities through behavioral interfaces, which are declarative specifications. The system can then match agents to services based on the compatibility of the agents' expectations and the functionality of the services. While their aim is to create matching between services and agents, our aim is to generate protocols to fulfil a set of goals. Contrary to them, we provide a set of alternatives and rank them based on risk and cost criteria.

Pham and Harland [17] also derive protocols to achieve a given goal based on the services of other agents. A key difference is that their derivation of protocols is interleaved with the protocol's enactment. This means that it is possible for the agents to find themselves pursuing dead ends in their interaction. Their approach does not generate all candidate protocols (although it probably could be extended to do so) and thus does not rank the protocols. Additionally, as they note, their framework requires expertise in temporal linear logic (an extension of Girard's linear logic with temporal operators, not to be confused with linear temporal logic), which is complex, and not widely-known.

Artikis develops an infrastructure for dynamic adaption of protocols at run-time [2]. In the proposed infrastructure a protocol is specified as a set of core rules, which are static, and a set of dynamic rules, which can be modified at run-time dynamically by the agents. To

modify a dynamic rule, agents initiate a meta-protocol and as a result of the meta-protocol's execution the modification is either applied or rejected depending on the decisions of the agents. The proposed infrastructure does not specify how an alternative for a dynamic rule is generated by an agent, which is our main focus in this study. Artikis discusses evaluation of a modification using metric space, however this evaluation depends on shared definition of a desired specification. We consider evaluation of a protocol from an agent's perspective. On the other hand, the proposed infrastructure can be used to realize our agreement procedure on a protocol. In this respect, Artikis' infrastructure is complementary to our work.

We now turn to considering trust and ranking of protocols. Johnson et al. [13] study when two dialgoue-game protocols can be considered equivalent. To do this, they compare protocols in terms of their length, the utterances they allow, and what they achieve in the end. In our case, all protocols are generated to satisfy the same set of goals, and hence intrinsically achieve end equivalence. Even then, we argue that protocols will vary based on the value they offer to their users and show that protocols can be preferred over each other based on how costly or risky they are.

Mallya and Singh [15] attack a similar problem for commitment protocols. They compare and evaluate commitment protocols based on the runs that the protocols can generate. To do this, they provide a commitment algebra that can reason on the various relations of runs, such as subsumption. Through this, they can identify whether two protocols are similar or whether one can be used in place of another one. However, they do not provide an explicit mechanism to rank protocols as we have done here.

Yolum and Singh [33] develop a valuation function that compares commitments with each other in terms of benefit and risk. They use the valuation function to design concession rules that enable agents to enact an already available protocol in concession. Our focus here is not on enactment but in ranking and agreeing on generated protocols. Hence, in this work, once the agents have ranked the protocols, they concede over protocols (rather than single commitments) to agree on a protocol to enact.

Singh [22] represents trust as dependence between two agents and develops a logic-based approach to understand and reason on trust. Singh provides various reasoning postulates on trust and compares them to those on commitments. Through this analysis, he identifies principles for engineering agent-based software. While we are also concerned with commitment and trust, we use the latter to decide which commitments are more likely to be carried out. Hence, our notion of trust between two agents is only meaningful if the two agents are involved in a commitment relation.

Finally, considering the broader question of the relationship between goals and commitments, Chopra et al. [6] study what it means for an agent's goals and commitments to be compatible and whether a set of commitments can realize a goal. There, a set of commitments is given and the check is performed. Here, we are generating the set of commitments to realize a goal from scratch. There is also no ranking in their work as we have done here. The work of Dalpiaz et al. [7] builds on this, developing a notion of adaptation, where an agent adapts to threats or opportunities by considering *variants* of its goal model that are supported by its capabilities and commitments. The paper (which unfortunately is somewhat light on details) appears to consider subsets of the goal model, and consequently selects a subset of the known commitments. By contrast, we provide an algorithm to generate possible commitment protocols from the goals and domain knowledge.

Marengo et al. [16] define control of a proposition and safety of a commitment. An agent has control over a proposition, if it can realize the proposition. They define two types of control: innate, which means that the agent can bring about the proposition by itself, and social, which means that the agent has a commitment to bring about the proposition from

an agent that controls the proposition. Then, if an agent has control over the proposition(s) that it is responsible for, due to a commitment, then the commitment is safe. Our notion of support is analogous to this notion of control and safety. However, our notion of support is more general since it considers both propositions and commitments together. Hence, it does not require two separate notions. Further, we can generate support for a set of goals and rank the produced protocols.

Our work here opens up interesting directions for future research. Perhaps the most significant is developing a wider range of larger examples. Unfortunately, there does not exist any repository of commitment-based protocols, or interaction problems that we could use to benchmark our generation algorithm and ranking metrics (which is why our experimental evaluation had to resort to synthetic generated problems). Our agreement procedure provides a basic mechanism for agents to reach agreement on one of the generated protocols. However, our procedure does not consider some important issues, such as how agents can negotiate when more than one agent generate protocols, and how agents can make counteroffers. Accordingly, we plan to develop a complete negotiation protocol in the future to handle such issues.

There are also a number of extensions to the work. One direction is considering variations to the benefit calculation that reflect other concerns about protocol executions. For example, an agent that would want minimal loss in any protocol can compute all subsets of a protocol, calculate each subset's benefit, and require that the benefit be positive for all subsets. Another interesting question is to explore to what extent the rankings proposed are robust against the agent's domain knowledge not being entirely correct. Finally, another direction could be in incorporating rankings from different metrics.

Acknowledgments Akın Günay is partially supported by TÜBİTAK Scholarship (2211) and Turkish State Planning Organization (DPT) under the TAM Project, 2007K120610. Part of this work was done while Akın Günay was a PhD student in the Department of Computer Engineering of Bogazici University and visiting PhD student in the Department of Information Science of University of Otago. This work is partially supported by Bogazici University Research Fund under grant 13A01P2.

Appendix: formal proofs

This appendix provides the formal proofs for the theorems in Section 3.3.

Theorem 3 (Soundness) *Algorithm 2 is sound (see Definition 9).*

Proof we need to show that the algorithm, when given a set of goals, generates commitments that are sufficient to ensure support for these goals. The algorithm closely follows the structure of Definition 8, and we prove the desired result by induction over the structure of the algorithm's first argument.

- Lines 1–2 of the algorithm correspond to the base case: if there are no propositions to be supported, then the agent's goals ($\top$) are trivially supported by the empty protocol.
- Lines 3–5 correspond to the second case: the condition $d_1 \wedge d_2$ is supported if both d_1 and d_2 are supported, (and more generally, $d_1 \wedge \cdots \wedge d_n$ is supported if all of the d_i are supported). By the induction hypothesis, the recursive calls to GoalBased yield sets of protocols P_1 and P_2 such that for d_1 we have that for any $p_i^1 \in P_1 : x, p_i^1 \Vdash d_1$, and respectively for d_2 we have that for any $p_j^2 \in P_2 : x, p_j^2 \Vdash d_2$. Consider now $p_{ij} = p_i^1 \cup p_j^2$ (recall that protocols are just sets of commitments, so they can be merged using set union). We have that $x, p_{ij} \Vdash d_1$ and $x, p_{ij} \Vdash d_2$ and therefore, by the second case of Definition 8, that

$x, p_{ij} \Vdash d_1 \wedge d_2$ as desired. This holds for any selection of p_i^1 and p_j^2 which is exactly what the merge in line 5 does.

- Lines 13–20 correspond to the first part of the last case of Definition 8 (agent using its own ability): The loop in line 13 finds all cases where the first part of the condition in Definition 8 is satisfied. Line 14 creates a fresh goal queue, and lines 15–17 decompose d into propositions. The recursive call in line 18 generates protocols P' such that (by inductive hypothesis) for all $p \in P' : x, p \Vdash d$. Given this, and that (as per line 13) $A_x(d, r') \in \mathcal{A}$ where $r' \Rightarrow r$, by Definition 8 we conclude that for all $p \in P' : x, p \Vdash r$ as desired.
- Lines 21–34 correspond to the second part of the last case of Definition 8: The loops in lines 21–22 find all cases where $S_x(y, d, r') \in \mathcal{B}$ (where $r' \Rightarrow r$) and $I_x(y, w, r) \in \mathcal{B}$. The next few lines create a fresh goal queue, and decompose d into propositions, inserting them into the goal queue, and then (line 27) also inserting w into the goal queue. The recursive call in line 28 thus, by induction hypothesis, generates a set of protocols P such that for any $p \in P : x, p \Vdash d \wedge w$. We then define $P' = \{p \cup \{C(y, x, d \wedge w, r)\} \mid p \in P\}$. By Definition 8 we then conclude that for any $p' \in P'$, we have that $x, p' \Vdash r$ as desired. □

Theorem 4 (Completeness) *Algorithm 2 is complete (see Definition 11).*

Proof Algorithm 2 follows Definition 10, and the proof, by induction over the structure of the first argument to the algorithm, follows the algorithm's structure.

- The first case of Definition 10 corresponds to lines 1–2: if there are no conditions to be supported, then a single empty protocol is returned. This meets the completeness requirement: the only minimal commitment set in this case is the empty set, and this is exactly what the algorithm returns.
- The second case corresponds to lines 3–5: given $d_i \wedge d_j$ the algorithm is called recursively to obtain support for each of the conjuncts. By the induction hypothesis we have that P_i and P_j (respectively the set of protocols that support d_i and d_j) contain all (and only) minimal commitment sets. The merge in line 5 selects $p \in P_i$ and $p' \in P_j$ and merges them ($p \cup p'$) in line with Definition 10. Hence the algorithm, which considers all possible combinations of p and p', generates exactly all possible minimal protocols that support $d_i \wedge d_j$.
- The first part of the last case corresponds to lines 13–20: by induction hypothesis the recursive call generates all minimal commitment sets that support d, and, by Definition 10 these are exactly the ones that support r in this case, which is what the algorithm returns.
- The final part of the last case corresponds to lines 21–34: by the induction hypothesis the recursive call generates all minimal commitment sets that support $d \wedge w$. The algorithm then considers these commitment sets, and adds the extra commitment $C(y, x, d \wedge w, r)$ to each one, yielding exactly the complete collection of minimal commitment sets for r.

□

References

1. Alberti, M., Cattafi, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., et al. (2011). A computational logic application framework for service discovery and contracting. *International Journal of Web Service Research*, *8*(3), 1–25.
2. Artikis, A. (2009). Dynamic protocols for open agent systems. In *Proceedings of the eighth international conference on autonomous agents and multiagent systems (AAMAS)*, (Vol. 1, pp. 97–104).
3. Castelfranchi, C. (1995). Commitments: from individual intentions to groups and organizations. In *Proceedings of the international conference on multiagent systems*, (pages 41–48).

4. Cheong, C., & Winikoff, M. (2009). Hermes: designing flexible and robust agent interactions (chapter 5). In V. Dignum (Ed.), *Multi-agent systems—Semantics and dynamics of organizational models* (pp. 105–139). Hershey, PA: IGI.
5. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2013). Representing and monitoring social commitments using the event calculus. *Autonomous Agents and Multi-Agent Systems*, *27*(1), 85–130.
6. Chopra, A. K., Dalpiaz, F., Giorgini, P., & Mylopoulos, J. (2010). Reasoning about Agents and Protocols via Goals and Commitments. In *Proceedings of the 9th international conference on autonomous agents and multi-agent systems (AAMAS)*, (pp. 457–464).
7. Dalpiaz, F., Chopra, A. K., Giorgini, P., & Mylopoulos, J. (2010). Adaptation in open systems: Giving interaction its rightful place. In J. Parsons, M. Saeki, P. Shoval, C. C. Woo, Y. Wand, (Eds.), *Conceptual modeling - ER 2010, proceedings of the 29th international conference on conceptual modeling, Vancouver, BC, November 1—4, 2010*, (Vol. 6412, pp. 31–45). Lecture Notes in Computer Science: Springer.
8. Desai, N., Mallya, A. U., Chopra, A. K., & Singh, M. P. (2005). Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, *31*(12), 1015–1027.
9. El-Menshawy, M., Bentahar, J., Qu, H., & Dssouli, R. (2011). On the verification of social commitments and time. In *Proceedings of the tenth international conference on autonomous agents and multiagent systems*, (pp. 483–490).
10. Fornara, N., & Colombetti, M. (2002). Operational specification of a commitment-based agent communication language. In *Proceedings of the first insternational conference on autonomous agents and multiagent systems*, (pp. 536–542).
11. Gerard, S. N., & Singh, M. P. (2013). Evolving protocols and agents in multiagent systems. In *Proceedings of the twelfth international conference on autonomous agents and multiagent systems (AAMAS)*, (pp. 997–1004).
12. Günay, A., Winikoff, M., & Yolum, P. (2013). Commitment protocol generation. In M. Baldoni, L. Dennis, V. Mascardi, & W. Vasconcelos (Eds.), *Declarative agent languages and technologies X* (Vol. 7784, pp. 136–152)., Lecture Notes in Computer Science Springer: Heidelberg.
13. Johnson, M. W., McBurney, P., & Parsons, S. (2003). When are two protocols the same? *Communication in multiagent systems, agent communication languages and conversation polocies* (Vol. 2650, pp. 253–268)., Lecture Notes in Computer Science Heidelberg: Springer.
14. Jonker, C. M., Hindriks, K. V., Wiggers, P., & Broekens, J. (2012). Negotiating agents. *AI Magazine*, *33*(3), 79–91.
15. Mallya, A. U., & Singh, M. P. (2007). An algebra for commitment protocols. *Autonomous Agents and Multi-Agent Systems*, *14*(2), 143–163.
16. Marengo, E., Baldoni, M., Baroglio, C., Chopra, A. K., Patti, V., & Singh, M. P. (2011). Commitments with regulations: reasoning about safety and control in REGULA. In *Proceedings of the 10th international conference on autonomous agents and multi-agent systems (AAMAS)*, (pp. 467–474).
17. Pham, D. Q., & Harland, J. (2007). Temporal linear logic as a basis for flexible agent interactions. In E. H. Durfee, M. Yokoo, M. N. Huhns, & O. Shehory (Eds.), *6th International joint conference on autonomous agents and multiagent systems (AAMAS 2007), IFAAMAS, Honolulu, Hawaii, May 14–18, 2007*, (pp. 124–131).
18. Rosenschein, J. S., & Zlotkin, G. (1994). *Rules of encounter*. Cambridge: MIT Press.
19. Sabater, J., & Sierra, C. (2001). Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on autonomous agents*, (pp. 194–195).
20. Sensoy, M., Zhang, J., Yolum, P., & Cohen, R. (2009). Poyraz: Context-aware service selection under deception. *Computational Intelligence*, *25*(4), 335–366.
21. Singh, M. P. (1999). An ontology for commitments in multiagent systems. *Artificial Intelligence and Law*, *7*(1), 97–113.
22. Singh, M. P. (2011). Trust as dependence: A logical approach. In *Proceedings of the 10th international conference on autonomous agents and multi-agent systems (AAMAS)*, (pp. 863–870).
23. Teacy, W., Patel, J., Jennings, N., & Luck, M. (2006). TRAVOS: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems*, *12*(2), 183–198.
24. Telang, P. R., & Singh, M. P. (2012). Comma: A commitment-based business modeling methodology and its empirical evaluation. In: V. Conitzer, M. Winikoff, L. Padgham, & W. van der Hoek, (Eds.), *Proceedings of the eleventh joint international conference on autonomous agents and multi-agent systems (AAMAS 2012)*, (pp. 1073–1080).
25. Telang, R. P., Meneguzzi, F., & Singh, M. P (2013). Hierarchical planning about goals and commitments. In *Proceedings of the twelfth international conference on autonomous agents and multiagent systems, AAMAS*, (pp. 877–884).
26. Udupi, Y. B., & Singh, M. P. (2006). Contract enactment in virtual organizations: A commitment-based approach. In *Proceedings of the twenty-first national conference on artificial intelligence*, (pp. 722–727).

27. Venkatraman, M., & Singh, M. P. (1999). Verifying compliance with commitment protocols. *Autonomous Agents and Multi-Agent Systems*, *2*(3), 217–236.
28. Winikoff, M. (2006). Implementing flexible and robust agent interactions using distributed commitment machines. *Multiagent and Grid Systems*, *2*(4), 365–381.
29. Winikoff, M. (2007). Implementing commitment-based interactions. In *Proceedings of the sixth international joint conference on autonomous agents and multiagent systems, AAMAS*, (pp. 1–8).
30. Winikoff, M., Liu, W., & Harland, J. (2004). Enhancing commitment machines. In J. Leite, A. Omicini, P. Torroni, & P. Yolum (Eds.), *Declarative agent languages and technologies II* (Vol. 3476, pp. 198–220)., Lecture Notes in Artificial Intelligence Heidelberg: Springer.
31. Yolum, P. (2007). Design time analysis of multiagent protocols. *Data and Knowledge Engineering*, *63*(1), 137–154.
32. Yolum, P., & Singh, M. P. (2002). Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the first international conference on autonomous agents and multiagent systems*, (pp. 527–534).
33. Yolum, P., & Singh, M. P. (2007). Enacting protocols by commitment concession. In *Proceedings of the sixth international conference on autonomous agents and multiagent systems*, (pp. 116–123).