Constraint Relaxation to Reduce Brittleness of Distributed Agent Protocols

M. Fadzil Hassan[1] and **David Robertson**[2]

Abstract. Incompatible goals among multiple agents working on domains involving finite constraints can be a source of conflict. This conflict, in the form of incompatible constraints established locally by the agents and imposed on the negotiated variables, may break the dialogue between these agents even though they could, in principle, reach an agreement. A common means of coordinating multi-agent systems is by using protocols to which are attached constraints on interaction; but protocols are brittle, in the sense that the constraints they contain must either succeed or fail, and if they fail the entire protocol may fail. We apply a constraint relaxation technique originally for automated negotiation using a distributed protocol language called the Lightweight Coordination Calculus (LCC), in order to overcome a class of conflicts, making protocols less brittle. This approach is illustrated in a scenario involving the ordering and configuration of a computer between the customer and vendor agents.

1 INTRODUCTION

One of the critical aspects of multi-agent systems (MAS) concerns with the coordination protocol between the agents involved in solving some prescribed tasks. To date, a number of frameworks have been proposed to address this aspect, for instance Electronic Institutions (EI) [1] and Conversation Policy (CP) [2]. However, there are a number of shortcomings within these approaches, as they are based on static state-based diagrams and require a centralised mechanism to manage the coordination between agents [3].

LCC, an agent protocol language, has been proposed to overcome this limitation [4]. This language, which is derived from process calculus, relaxes the static specification of agent protocols as state-based diagrams and allows protocols to be defined and disseminated in a flexible manner during agent interaction.

[1] Centre for Intelligent Systems and their Applications (CISA), School of Informatics, University of Edinburgh, Appleton Tower, Room 3.07, 11 Crichton Street, Edinburgh EH8 9LE, UK. Tel: (0) 131 650 2749, E-mail: s0090693@sms.ed.ac.uk

[2] CISA, Tel: (0) 131 650 2709, E-mail: dr@inf.ed.ac.uk

The protocol language has been applied to several domains, including a short but (by current standards) complex scenario that deals with the purchasing and configuration of a computer between the customer and vendor agents. The scenario, borrowed from [5], is as follows:

> An internet-based agent acting on behalf of a customer wants to buy a computer but doesn't know how to interact with other agents to achieve this, so it contacts a service broker. The broker supplies the customer agent with the necessary interaction information. The customer agent then has a dialogue with the given computer vendor in which the various configuration options and pricing constraints are reconciled before a purchase is finally made.

Within this given scenario, the agents' local goals, expressed in the form of finite constraints over the negotiated variables (e.g. memory_space(40..50), price(800..1100), monitor_size(15..21), etc.), may cause a conflict if no compatibility is found between the corresponding variables values set by the negotiated agents. This conflict will lead to a failure in the reconciliation process and break the prescribed protocol.

Therefore, to overcome this conflict, this paper proposes a constraint relaxation technique, described in [6], using the LCC protocol language. It is expected that through the proposed work, it will reduce the brittleness of the agent protocol, especially when applied to the domains involving finite constraint on variables.

The remainder of this section provides background to the related work involving agent coordination for constraint-based problems, an overview on the LCC interaction framework, and the approach used for expansion of the protocol clauses (the means by which LCC protocols are enacted). Section 2 describes the use of LCC interaction protocol for a computer ordering and configuration scenario, demonstrating the brittleness problem faced by the current work. In section 3, the constraint relaxation technique to reduce brittleness of the protocol is described and implemented. Lastly, section 4 will provide an example and the final section will discuss further work and potential shortcomings of this approach.

1.1 MAS coordination and constraint-based problems

MAS have been used to solve constraint-based problems in various domains, which include distributed meeting scheduling systems [7], organ transplant coordination for a hospital [8], and distributed timetabling systems [9]. In many of these works, a central agent that facilitates the sending and receiving of messages performs coordination among the distributed agents to reach a common goal. It is through this central agent that the process of obtaining the relevant variables, their associated domains and the required constraints from the various distinct agents is performed. A solution, globally consistent with the local goals of each of the involved agents, is then generated if there exists one.

Incompatible local goals and constraints among the distributed agents may cause conflicts and failure to reach a consistent solution. Rather than terminating this whole process prematurely, it is common to resolve these conflicts using various conflict resolution strategies described in [10], usually mediated by the central agent. The use of a central agent for coordination purpose to resolve this conflict might be acceptable if confidentiality is not the main concern; so it is acceptable for agents to reveal their internal goals to the third parties. However, in some domains (e.g. buyer-seller negotiation), it is not practical for this private information to be completely revealed, as it might jeopardise the agents' individual strategies for obtaining an optimal outcome from the interaction process. Given this consideration, it is essential for the conflict resolution approach to be managed directly by the involved agents themselves, without any third-party mediator. Fundamentally, the use of central agent undermines the key principle of agency – that each agent can operate autonomously – since the use of central agent can be considered have removed part of that autonomy.

1.2 Overview of LCC

LCC borrows the notion of role from agent systems that enforce social norms but reinterprets this in a process calculus. The syntax of the protocol language is shown in Figure 1. Social norms in LCC are expressed as message-passing behaviours associated with roles. '$\Rightarrow$' and '$\Leftarrow$' mark messages being sent or received respectively. On the left-hand side of the double arrow is the message and on the right-hand side is the other agent involved in the interaction. The most basic behaviours are to send or receive messages, and more complex ones are constructed using the connectives *then*, *or* and *par* for sequence, choice and parallelisation respectively. A set of such behavioural clauses specifies the message passing behaviour expected of the social norm, and can be referred to as the interaction framework.

The LCC language ensures coherence interaction between agents by imposing constraints relating to the messages they send and receive in their chosen roles. Constraints are marked by '$\leftarrow$', which indicate the requirements or consequences for an agent on the performatives or roles available to it. The clauses of the protocol are arranged so that, although the constraints on each role are independent of others, the ensemble of clauses operates to give the desired overall behaviour. Further details of the LCC interaction framework are provided in [3-5].

```
Framework  := {Clause,…}
   Clause  := Agent::Def
    Agent  := a(Type,Id)
      Def  := Agent | Message | Def then Def | Def or Def | Def par Def |
              null ← C
  Message  := M ⇒ Agent | M ⇒ Agent ← C | M ⇐ Agent | C ← M ⇐ Agent
        C  := Term | C ∧ C | C ∨ C
     Type  := Term
        M  := Term
```

Where *null* denotes an event, which does not involve message passing; *Term* is a structured term in Prolog syntax and *Id* is either a variable or a unique identifier for the agent.

Figure 1. Syntax of LCC Interaction Framework

1.3 Using LCC protocol for coordination

An agent is capable of conforming to a LCC protocol if it is supplied with a way of unpacking any protocol it receives; finding the next moves that it is permitted to take; and updating the state of the protocol to describe the new state of the dialogue. There are many ways of doing this but perhaps the most elegant way is by applying rewrite rules to expand the dialogue state. This works as follows:

- An agent receives from some other agent a message with an attached protocol, *P*, of the form *protocol (S, F, K)*, where *S* is the dialogue state; *F* is the dialogue framework (a set of dialogue clauses); and *K* is a set of axioms defining common knowledge assumed among the agents. The message is added to the set of messages currently under consideration by the agent – giving the message set M_i.
- The agent extracts from *P* the dialogue clause, C_i, determining its part of the dialogue.
- The rewrite rules of Figure 2 are applied to give an expansion of C_i in terms of protocol *P* in response to the set of received messages, M_i, producing: a new dialogue clause C_n; an output message set O_n and remaining unprocessed messages M_n (a subset of M_i). These are produced by applying the protocol rewrite rules above exhaustively to produce the sequence:

$$\left\{ \begin{array}{l} C_i \xrightarrow{Mi,\ Mi+1,\ P,\ Oi} C_{i+1},\ C_{i+1} \xrightarrow{Mi+1,\ Mi+2,\ P,\ Oi+1} C_{i+2}, \\ \ldots,\ C_{n-1} \xrightarrow{Mn-1,\ Mn,\ P,\ On} C_n \end{array} \right\}$$

- The agent's original clause, C_i, is then replaced in *P* by C_n to produce the new protocol, P_n.

- The agent can then send the messages in set O_n, each accompanied by a copy of the new protocol P_n.

$A::B \xrightarrow{Mi,\ Mo,\ P,\ O} A::E$	if $B \xrightarrow{Mi,\ Mo,\ P,\ O} E$
$A_1 \text{ or } A_2 \xrightarrow{Mi,\ Mo,\ P,\ O} E$	if $\neg closed(A_2) \wedge A_1 \xrightarrow{Mi,\ Mo,\ P,\ O} E$
$A_1 \text{ or } A_2 \xrightarrow{Mi,\ Mo,\ P,\ O} E$	if $\neg closed(A_1) \wedge A_2 \xrightarrow{Mi,\ Mo,\ P,\ O} E$
$A_1 \text{ then } A_2 \xrightarrow{Mi,\ Mo,\ P,\ O} E \text{ then } A_2$	if $A_1 \xrightarrow{Mi,\ Mo,\ P,\ O} E$
$A_1 \text{ then } A_2 \xrightarrow{Mi,\ Mo,\ P,\ O} A_1 \text{ then } E$	if $closed(A_1) \wedge A_2 \xrightarrow{Mi,\ Mo,\ P,\ O} E$
$A_1 \text{ par } A_2 \xrightarrow{Mi,\ Mo,\ P,\ O1 \cup O2} E_1 \text{ par } E_2$	if $A_1 \xrightarrow{Mi,\ Mo,\ P,\ O1} E_1 \wedge A_2 \xrightarrow{Mi,\ Mo,\ P,\ O2} E_2$
$C \rightarrow M \Leftarrow A \xrightarrow{Mi,\ Mi-\{M \Leftarrow A\},\ P,\ \varnothing} c(M \Leftarrow A)$	if $(M \Leftarrow A) \in M_i \wedge satisfy(C)$
$M \Rightarrow A \leftarrow C \xrightarrow{Mi,\ Mo,\ P,\ \{M \Rightarrow A\}} c(M \Rightarrow A)$	if $satisfied(C)$
$null \leftarrow C \xrightarrow{Mi,\ Mo,\ P,\ \varnothing} c(null)$	if $satisfied(C)$
$a(R,I) \leftarrow C \xrightarrow{Mi,\ Mo,\ P,\ \varnothing} a(R,I)::B$	if $clause(P,a(R,I)::B) \wedge satisfied(C)$

A protocol term is decided to be closed, meaning that it has been covered by the preceeding interaction, as follows:

closed(c(X))
closed(A or B) ← closed(A) $\vee$ closed(B)
closed(A then B) ← closed(A) $\wedge$ closed(B)
closed(A par B) ← closed(A) $\wedge$ closed(B)
closed(X::D) ← closed(D)

satisfied(C) is true if C can be solved from the agent's current state of knowledge.
satisfy(C) is true if the agent's state of knowledge can be made such that C is satisfied.
clause(*P*,X) is true if clause *X* appears in the dialogue framework of *P*.
.

Figure 2. Rewrite rules for expansion of a protocol clause

2 LCC PROTOCOL FOR FINITE-CONSTRAINTS PROBLEM

The LCC-based protocol for the scenario given in section 1 can be conceptually described in Figure 3, and further details can be found in [5]. There are two types of agent: a vendor agent and a customer agent. No limit is placed on the number of dialogues that may occur, although each such dialogue will be constrained by the LCC protocol.

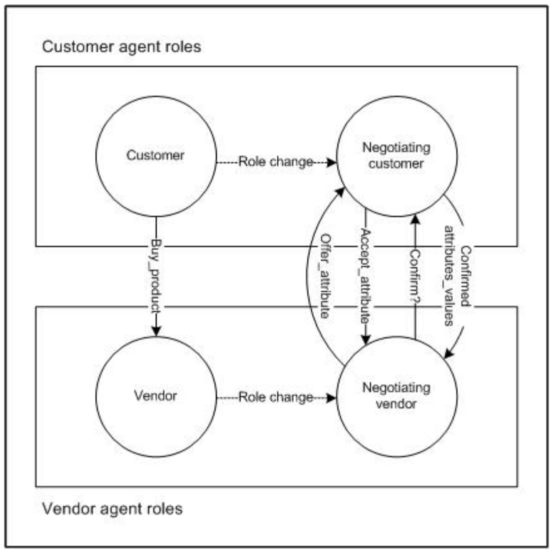

Figure 3. Roles and interactions diagram

Assuming that the customer agent has already obtained the necessary interaction information from a service broker, the agent (in the role of customer) may send a request to buy a computer to a selected vendor agent, and can then (in the role of negotiating customer) accept offers of each of the computer attributes values in turn.

The interaction protocols between the vendor and customer agents are defined by expressions 1-4, in Figure 4. In expression 1, a customer, *C*, can send a request to vendor, *V*, to buy an item, *X* that the customer needs and believes the vendor sells. Then the customer takes the role of negotiator with the vendor. Expression 2 consists of clauses to define a negotiating customer with a set, *S*, of negotiated attributes of the desired item, *X*, either receives an offer of a new attribute, *A*, and accepts that (continuing in the negotiating role with A added to *S*) or it receives a request to commit to the current set of negotiated attributes and replies with a commitment to the chosen attributes, *F*, from that set. In expression 3, a vendor, *V*, receives a request from a customer, *C*, to buy an item, *X*; then takes the role of negotiator with the customer over the attribute set, *S*, that applies to that item.

In expression 4, a negotiating vendor with a set, *S*, of negotiable attributes of the desired item, *X*, either takes the first element, *A*, of *S* and offers it to the customer for acceptance (continuing then in its negotiating role with the remaining attributes, *T*) or if *S* is empty it asks the customer to commit to the attributes they have discussed and receives confirmation of the commitment.

```
a(customer,C)::
  ask(buy(X)) ⇒ a(vendor,V) ← need(X) ∧ sells(X,V) then
  a(neg_customer(X,V,[]),C)                                            (1)
a(neg_customer(X,V,S),C)::
       ( offer(A) ⇐ a(neg_vendor(X,C,_),V) then
       ( accept(A) ⇒ a(neg_vendor(X,C,_),V) ← acceptable(A) then      )
       ( a(neg_customer(X,V,[att(A)|S]),C)                             )
or
       ( ask(commit) ⇐ a(neg_vendor(X,C,_),V) then                     )
       ( tell(commit(F)) ⇒ a(neg_vendor(X,C,_),V) ← choose(S,F)        )   (2)
a(vendor,V)::
         ask(buy(X)) ⇐ a(customer,C) then
         a(neg_vendor(X,C,S),V) ← attributes(X,S)                      (3)
a(neg_vendor(X,C,S),V)::
       ( offer(A) ⇒ a(neg_customer(X,V,_),C)← S=[A|T] ∧ available(A) then )
       ( accept(A) ⇐  a(neg_customer(X,V,_),C) then                        )
       ( a(neg_vendor(X,C,T),V)                                            )
or
       (  ask(commit) ⇒ a(neg_customer(X,V,_),C) ← S = [] then             )
       ( tell(commit(F)) ⇐ a(neg_customer(X,V,_),C)                        )  (4)
```

Figure 4. Interaction protocols between customer and vendor agents

2.1 Brittleness of current protocol

An important aspect of the coordination protocols between the vendor and customer agents defined in expressions 1-4 concerns with the message passing of the product attributes. The dialogue will continue in accordance to this predefined protocol as long as there exists a match between the ranges of attribute's values offered by the negotiating vendor with those required by the negotiating customer. To illustrate this point, consider the following example:

Assume that the current attribute value being negotiated between these two agents is the disk space size of the computer, and the following statements describe the knowledge and constraints private to the customer and vendor agents respectively:

Vendor: *available(disk_space(D)) ← D in 20..100 Gb*
Customer: *acceptable(disk_space(D)) ← D in 40..∞ Gb*

Upon negotiating these local constraints via the defined protocol, the disk space attribute value that meets the vendor offer, and also the customer requirement will be in the following range:

40 Gb <= disk_space(D) <= 100 Gb

Therefore, depending on the agents' strategies (e.g. choosing the maximum value within the agreed range, etc.), the disk space attribute will be assigned to a value within this agreed range.

However, the following local constraints would break the protocol:

Vendor: *available(disk_space(D)) ← D in 40..100 Gb*
Customer: *acceptable(disk_space(D)) ← D in 20..30 Gb*

In this particular situation, no match is found between the customer's required disk space value and the one that can be offered by the vendor. Rather than terminating the dialogue at this stage and wasting all the earlier effort of establishing and maintaining the coordination between agents, we might reduce brittleness by including a repair mechanism within the protocol. This approach allows customer and vendor to negotiate further by relaxing the specified constraint on the value of the attribute.

3 CONSTRAINT RELAXATION

When a customer and vendor negotiate, it is rarely the case that an offer is completely acceptable or completely inconsistent with their respective constraints. Rather, an offer usually satisfies the customer's constraints more or less. For example, an offer from the vendor consisting the following attributes values:

- *available(disk_space(D)) ← D in 40..80,*
- *available(monitor_size(M)) ← M in 15..18,*

Can only partially satisfy the customer's local constraints of:

- *acceptable (disk_space(D)) ← D in 60..80,*
- *acceptable(monitor_size(M)) ← M in 20..21,*

Except that there exists a conflicting range of values for the monitor size attribute. Resolving this conflict requires the customer agent to relax its constraint on this particular attribute. Depending on the internal decision strategy employed by the customer agent, the conflicting constraint might be relaxed or the rest of the negotiation process might be abandoned entirely.

It is not the focus of this work to cover the various computational approaches that a particular agent might employ to reach to a decision. However, it is assumed that the decision taken should be to the agent's own good, leading to the realisation of the eventual goal of the agent. This paper provides only a mechanism if the need arises, to provide an agent involved with the coordinating mechanism a way to relax the conflicting constraints. The focus of this work concerns on the inclusion of clauses into the existing protocol that allow the constraints relaxation to be coordinated and the remainder of this section provides a discussion on this.

The conceptual view of the agents' roles and interaction described in Figure 3 is extended in Figure 4, to include new roles that should be able to accommodate the negotiating agents with the coordination protocol in relaxing any conflicting constraints. The interaction protocols between the customer and vendor agents in the effort of relaxing the conflicting constraints are defined by expressions 5-8, in Figure 5.

Expression 5, which is an extended version of expression 2, includes clauses that allow the negotiating customer to inform the negotiating vendor of an unacceptable attribute value for A upon an occurrence of a conflict between the value offered, and the local constraints of the customer agent. Then, the customer takes up a new role (i.e. constraints handling customer) that is specially focused on handling these incompatible constraints between the agents. In expression 6, which is the extended version of expression 3, the vendor will take up a new role (i.e. constraints handling vendor) upon receiving the message of unacceptable constraint on the values of attribute A from the customer.

In expression 7, in the role of constraint handling customer, the request to relax the constraint on the values of attribute A is either entertained (i.e. by sending a message informing of a performed relaxation on the conflicting constraint), or rejected (i.e. by sending a message informing the vendor of a failure). The customer will then resume its prior role as a negotiating customer once the constraint relaxation is successfully performed.

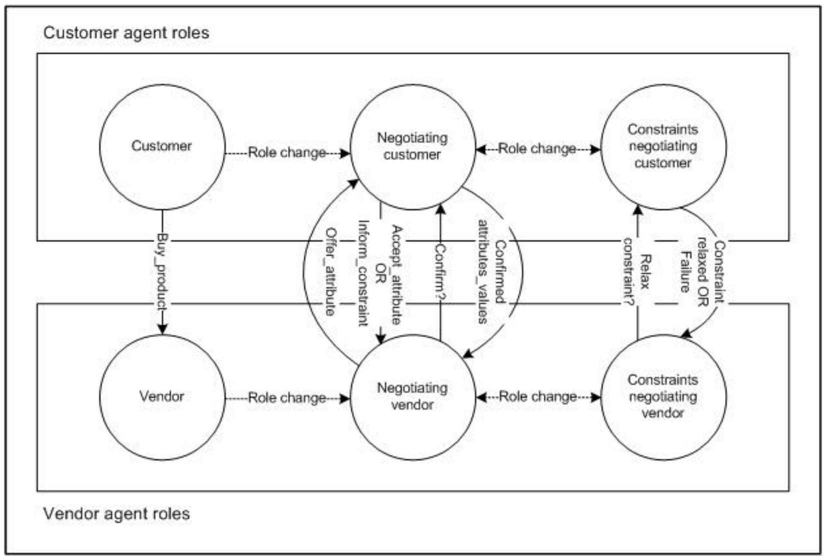

Figure 4. Roles and interaction diagram for constraints relaxation

```
a(neg_customer(X,V,S),C)::
        offer(A) ⇐ a(neg_vendor(X,C,_),V) then
          accept(A) ⇒ a(neg_vendor(X,C,_),V) ← acceptable(A) then
            a(neg_customer(X,V,[att(A)|S]),C)
          or
          inform(unacceptable(A)) ⇒ a(neg_vendor(X,C,_),V) ← ¬ acceptable(A)  then
            a(constraint_hand_customer(att(A),C))
        or
        ask(commit) ⇐ a(neg_vendor(X,C,_),V) then
        tell(commit(F)) ⇒ a(neg_vendor(X,C,_),V) ← choose(S,F)                          (5)
a(neg_vendor(X,C,S),V)::
        offer(A) ⇒ a(neg_customer(X,V,_),C)← S=[A|T] ∧ available(A) then
          accept(A) ⇐  a(neg_customer(X,V,_),C) then
            a(neg_vendor(X,C,T),V)
          or
          inform(unacceptable(A) ⇐  a(neg_customer(X,V,_),C) then
            a(constraint_hand_vendor(A,V))
        or

          ask(commit) ⇒ a(neg_customer(X,V,_),C) ← S = [] then
          tell(commit(F)) ⇐ a(neg_customer(X,V,_),C)                                      (6)
a(constraint_hand_customer(A,C))::
        request(relax_constraint(A)) ⇐  a(constraint_hand_vendor(A,V)) then
          inform(relaxed(A)) ⇒ a(constraint_hand_vendor(A,V)) ←  relax(A)  then
            a(neg_customer(X,V,S),C)
          or
          inform(failure_relax(A)) ⇒ a(constraint_hand_vendor(A,V)) ←  ¬ relax(A)         (7)
a(constraint_hand_vendor(A,V))::
        request(relax_constraint(A)) ⇒  a(constraint_hand_customer(A,C)) then
          inform(performed_relax(A)) ⇐ a(constraint_hand_customer(A,C)) then
            a(neg_vendor(X,C,S,V))
          or
          inform(failure_relax(A)) ⇐ a(constraint_hand_customer(A,C))                      (8)
```

Figure 5. Interaction protocol with constraint relaxation

In expression 8, the vendor will send a request to the customer to relax its local constraint on the values of attribute *A,* and expected afterwards for the customer to reply with a message stating that either the request has been entertained or turned down. If a successful constraint relaxation message is received, then the vendor will resume its prior role as a negotiating vendor, continuing with the dialogue that has been left out when the conflict on the agents' local constraint that caused deadlock to the interaction occurred.

4 EXAMPLE

In this section, the application of the new modified protocol clauses defined in expressions 5-8, to the scenario given in section 1 is demonstrated.

As an example of knowledge private to the customer agent, we define below the range of acceptable values for attributes of the personal computer under discussion. For instance, the customer would accept disk space of 40 or above. It is also defined how the specific values for attributes are chosen by the customer agent from the ranges agreed via earlier dialogue with the vendor: the maximum from the range being taken for every attribute.

need(pc)
sell(pc,s1)
acceptable(disk_space(D)) ← *D in 40..60*
acceptable(monitor_size(M)) ← *M in 19..21*
choose([att(Att)|T],[att(AttI)|R]) ← *choice(Att,AttI)* ∧ *choose(T,R)*
choose([],[])
choice(disk_space(D),disk_space(Dc)) ← *fd_max(D,Dc)*
choice(monitor_size(M),monitor_size(Mc)) ← *fd_max(M,Mc)*
(9)

The vendor agent's local constraints are defined in the similar way to that of the customer. The available ranges of the attributes needed to configure a computer are defined as the following:

attributes(pc,[disk_space(D),monitor_size(M)])
available(disk_space(D)) ← *D in 20..100*
available(monitor_size(M)) ← *M in 14..17*
(10)

The values for the monitor size attribute of both agents are purposely set to be conflicting with each other, to test how well the protocol clauses defined in 5-8 are able to reduce the brittleness of the current interaction protocol defined in 1-4. The sequence of message passing that follows from the protocol expressions of 5-8 and the constraints of expressions 9-10 is shown below. The dialogue iterates between the customer, *b1*, and a vendor, *s1*. Each illocution shows: a numeric illocution identifier for reference (i.e. *1..n*); the type of the agent sending the message; the message itself; the type of agent to which the message is sent; and the variable restrictions applying to the message (the term *r(V,C)* relating a finite domain constraint *C* to a variable *V*). The first illocution is the customer making initial contact with the vendor. Illocution two consists of an offer for the range of values of the disk space attribute, which is accepted by the customer in illocution three. However, in illocution four, the offer for the range of values of the monitor size attribute is conflicting with the local constraint of the customer. Therefore, in illocution five, the customer will send a message to the vendor informing about the failure to comply with the offer. Illocutions six and seven consist of the agents' interactions to reconcile the conflicting constraints.

Assuming that the customer agent agreed to relax its constraint, a message will be send to the vendor informing about the performed relaxation in illocution seven. In illocution eight and nine, the agents resume their prior roles, and these illocutions consist of messages concern with the dialogue temporarily left out to give way to the conflict reconciliation. In illocution ten, the vendor that has worked through all its relevant attributes, asks for commitment from the customer. In reply, the customer submitted a list of committed values for all attributes.

Illocution identifier: *1*
Sender: *a(customer,b1)*
Message: *ask(buy(pc))*
Recipient: *a(vendor,s1)*
Restrictions: *[]*

Illocution identifier: *2*
Sender: *a(neg_vendor(pc,b1,[disk_space(D), monitor_size(M)]),s1)*
Message: *offer(disk_space(D))*
Recipient: *a(neg_customer(pc,s1,_),b1)*
Restrictions: *[r(D, [20|100])]*

Illocution identifier: *3*
Sender: *a(neg_customer(pc,s1,[]),b1)*
Message: *accept(disk_space(D))*
Recipient: *a(neg_vendor(pc,b1,_),s1)*
Restrictions: *[r(D, [40|60])]*

Illocution identifier: *4*
Sender: *a(neg_vendor(pc,b1,[monitor_size(M)]),s1)*
Message: *offer(monitor_size(M))*
Recipient: *a(neg_customer(pc,s1,_),b1)*
Restrictions: *[r(M,[14|17]), r(D, [40|60])]*

Illocution identifier: 5
Sender: *a(neg_customer(pc,s1,[att(disk_space(D))]),b1)*
Message: *inform(unacceptable(monitor_size(M)))*
Recipient: *a(neg_vendor(pc,b1,_),s1)*
Restrictions: *[r(M,[19|21]), r(D, [40|60])]*

Illocution identifier: 6
Sender: *a(constraint_hand_vendor(monitor_size(M)),s1)*
Message: *request(relax_constraint(monitor_size(M)))*
Recipient: *a(constraint_hand_customer(monitor_size(M)),b1)*
Restrictions: -

Illocution identifier: *7*
Sender: *a(constraint_hand_customer(monitor_size(M)),b1)*
Message: *performed_relax(monitor-size(M))*
Recipient: *a(constraint_hand_vendor(monitor_size(M)),s1)*
Restrictions: -

Illocution identifier: *8*
Sender: *a(neg_vendor(pc,b1,[monitor_size(M)]),s1)*
Message: *offer(monitor_size(M))*
Recipient: *a(neg_customer(pc,s1,_),b1)*
Restrictions: *[r(M,[14|17]), r(D, [40|60])]*

Illocution identifier: *9*
Sender: *a(neg_customer(pc,s1,[att(disk_space(D))]),b1)*
Message: *accept(monitor_size(M))*
Recipient: *a(neg_vendor(pc,b1,_),s1)*
Restrictions: *[r(M,[14|17]), r(D, [40|60])]*

Illocution identifier: *10*
Sender: *a(neg_vendor(pc,b1,_),s1)*
Message: *ask(commit)*
Recipient: *a(neg_customer(pc,s1,_),b1)*
Restrictions: *[]*

Illocution identifier: *11*
Sender: *a(neg_customer(pc,s1,[att(monitor_size(M)), att(disk_space(D))]),b1)*
Message: *tell(commit([att(monitor_size(M)),att(disk_space(D))]))*
Recipient: *a(neg_customer(pc,s1,_),b1)*
Restrictions: *[]*

5 CONCLUSION AND FUTURE DIRECTIONS

We have shown how brittleness of agent protocols, based on our LCC language, can be reduced via a constraint relaxation approach. A basic method for doing this was presented. This is achieved through the inclusion of new clauses into the existing protocol that allow the relaxation to be coordinated when conflicting local constraints over some negotiated variables cause the entire dialogue protocol to break. As the behaviour of an agent in a given role is determined by the appropriate LCC clauses, the introduced relaxation clauses spell out explicitly on how the agents should behave upon encountering a conflict involving incompatible constraints on variables established locally by these agents. An agent has the option of assuming new roles that are specifically designed to provide appropriate coordination measures for the agents to relax any conflicting constraints with its counterparts. This approach is later demonstrated in a scenario involving the ordering and configuration of a computer between the customer and vendor agents. Through a simple example of a conflicting constraint of attribute value between these agents, we explained how the extended LCC protocol clauses are able to coordinate the agents in resolving this conflict, and at the same time make the protocol less brittle. Future work includes the following:

- In this work, our emphasis is on extending the protocol clauses to include the capability to coordinate constraint relaxation among agents. Another approach, expected to result in a similar outcome, involves the integration of the relaxation approach within the rewrite rules distributed to each agent. Therefore, rather than focusing on establishing specialised agents' roles to deal with this issue, another viable option provides the state expansion mechanism (i.e. rewrite rules) with the conflict resolution strategies (i.e. constraints relaxation). This generic set of rewrite rules would define how a protocol clause should be expanded in case of a failure to satisfy any predefined constraint imposed on negotiated variables.

- Our approach currently applies to finite domain constraints, in which the constraints imposed on the respective variables are independent of each other. However, in the real scenario, it is usually the case that the relaxation performed on a particular variable's constraints, has an immediate effect on the others. Therefore, future work should include extending this approach to enable the coordination of constraint relaxation effort among agents with multiple-dependent local constraints.

ACKNOWLEDGEMENT

We would like to thank the referees for their comments, which helped improve this paper.

REFERENCES

[1] M. Estava, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos, "On the formal specifications of electronic institutions," *Lecture Notes in Artificial Intelligence*, pp. 126-147, 2001.

[2] M. Greaves, M. Holmback, and J. Bradshaw, "What is a conversation policy?," in *Issues in Agent Communication*, F. Dignum and F. Greaves, Eds.: Springer-Verlag, 1990, pp. 118-131.

[3] C. Walton and D. Robertson, "Flexible multi-agent protocols," University of Edinburgh, Technical Report EDI-INF-RR-0164, 2002.

[4] D. Robertson, "Distributed agent protocols," University of Edinburgh, Technical Report contact author for details(dr@inf.ed.ac.uk), 2003.

[5] D. Robertson, "Multi-agent coordination as distributed logic programming," paper submitted for the 20th International Conference on Logic Programming, Saint-Malo, France, Sept. 6-10, 2004.

[6] A. Sathi and M. S. Fox, "Constraint-directed negotiation of resource allocation," Carnegie Mellon University, Technical Report CMU-RI-TR-89-12, 1989.

[7] S. Macho-Gonzales, M. Torrens, and B. Faltings, "A multi-agent recommender system for planning meetings," presented at Workshop on Agent-based Recommender Systems (WARS' 2000), Barcelona, Spain, 2000.

[8] A. Aldea, B. Lopez, A. Moreno, D. Riano, and A. Valls, "A multi-agent system for organ transplant coordination," presented at VIII European Conference on AI in Medicine, Carcais, Portugal, 2001.

[9] A. Meisels and E. Kaplansky, "Distributed timetabling problems (DisTTP)," presented at 4th International Conference on Practice and Theory of Automated Timetabling, Gent, Belgium, 2002.

[10] T. H. Liu, A. Goel, C. E. Martin, and K. S. Barber, "Classification and representation of conflict in multi-agent systems," University of Texas, Austin, Technical Report TR98-UT-LIPS-AGENTS-01, 1998.