Addressing Constraint Failures in Agent Interaction Protocol

Fadzil Hassan, Dave Robertson and Chris Walton

Center for Intelligent Systems and their Applications (CISA),
School of Informatics, University of Edinburgh, Scotland, UK.
E-mail: s0090693@sms.ed.ac.uk, {dr,cdw}@inf.ed.ac.uk Tel: +44-(0) 131-651-4155

Abstract. The field of multi-agent systems shifts attention from one particular agent to a society of agents; hence the interactions between agents in the society become critical towards the achievement of their goals. We assume that the interactions are managed via an agent protocol which enables agents to coordinate their actions in order to handle the dependencies that exist between their activities. An agent's failure to comply with the constraints attached within the protocol might cause a brittle protocol to fail. To address this problem, a constraint relaxation approach is applied using a distributed protocol language called the Lightweight Coordination Calculus (LCC). This paper describes the design and implementation of a constraint relaxation module to be integrated within the LCC framework. The working of this module is later illustrated using a scenario involving the ordering and configuration of a computer between the customer and vendor agents.

1 Introduction

In the area of multi-agent systems (MAS), constraints and rules are used to guide the cooperative, coordinative or even competitive behaviours of agents. As described in [1], the application of constraint-based specification within the domain of agent communication and coordination includes determining the allowed sequence of communicative acts between agents, commonly referred to as an Agent Interaction Protocol (AIP). AIPs are used to specify the set of allowed message types (i.e. performatives), message contents and the correct order of messages during the dialogues between agents [2], and become the basis for agent negotiation and cooperation [3]. Protocols provide a useful framework for coordinating agent conversation as agents using a particular protocol are obligated to obey the interactive rules specified by the protocols.

To date, a number of tools and languages have been proposed to model and implement agent interaction protocols, for instance Electronic Institutions [4] and Conversation Policy [5]. However, there are a number of shortcomings with these approaches. They are based on static state-based diagrams, and normally are not directly executable. They also require a centralised mechanism to synchronise the coordination between agents [6]. An AIP language, called the Lightweight Coordination Calculus (LCC), has been proposed to overcome these limitations [6-8]. This language, which is derived from process calculus, relaxes the static specification of agent interaction protocols as state-based diagram and allows the protocols to be defined and disseminated in a flexible manner during agent interaction. In LCC, coordination is achieved via an interaction model in which participating agents assume roles constrained by the social norms of their shared task; the state of the interaction reflecting the ways these constraints are mutually satisfied within some system for synchronisation that is open and distributed [8].

LCC interaction protocols are brittle, in a sense that the constraints that they contain must either succeed or fail, and if they fail the entire protocol may fail [9]. Consequently, protocol failure will cause the dialogue between agents to break, even though the interacting agents could in principle reach an agreement. Therefore, in this paper, we describe the design and implementation of a constraint relaxation approach within a particular AIP framework (LCC), in order to address the brittleness problem mentioned earlier. The approach is focused on finite-domain constraints involving bilateral, peer-to-peer multi-agent interaction patterns. This proposed approach is handled by the participating agents themselves without any reliance to a third-party mediator, thus ensuring that issues like invasion of privacy and the bottleneck problem can be addressed. To demonstrate this approach, it is applied to a short but (by current standards of web service interaction) complex scenario that deals with the purchase and configuration of a computer between the customer and vendor agents. The scenario, borrowed from [7], 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.

The remainder of this paper is organised as follows: Section 2 provides a review of the LCC interaction framework; the LCC protocol language and the mechanism used in implementing the framework. Using the scenario, in section 3 we describe the constraint handling aspect of the LCC interaction protocol, the brittleness problem faced by the current work and how this leads to the proposed research work. Section 4 provides a discussion on the design of the proposed constraint relaxation module to address the brittleness problem. Section 5 provides an example demonstrating the application of the approach to the mentioned scenario. Further work and potential shortcomings of the approach are discussed in Section 6, where this paper concludes.

2 Overview of LCC Interaction Framework

2.1 The LCC Protocol Language

LCC borrows the notion of role from agent systems that enforce social norms but reinterprets this in a process calculus. Figure 1 defines the abstract syntax of LCC. An interaction model in LCC is a set of clauses, each of which defines how a role in the interaction must be performed. Roles are described by the type of role and an identifier for the individual agent undertaking that role. The definition of performance of a role is constructed using combinations of the sequence operator (‘*then*’) or choice operator (‘*or*’) to connect messages and changes of role. Messages are either outgoing to another agent in a given role (‘$\Rightarrow$’) or incoming from another agent in a given role (‘$\Leftarrow$’). Message input/output or change of role can be governed by a constraint (‘*C*’) defined using the normal logical operators for conjunction, disjunction and negation. 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.

Framework	*:= {Clause,…}*
Clause	*:= Role::Def*
Role	*:= a(Type,Id)*
Def	*:= Role | Message | Def then Def | Def or Def | null* $\leftarrow$ *C*
Message	*:= M* $\Rightarrow$ *Role | M* $\Rightarrow$ *Role* $\leftarrow$ *C | M* $\Leftarrow$ *Role | C* $\leftarrow$ *M* $\Leftarrow$ *Role*
C	*:= Term|* $\neg$*C | C* $\wedge$ *C | C* $\vee$ *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.

Fig. 1. Abstract syntax of LCC interaction framework

2.2 Implementing the Protocol Framework

The format of messages communicated between the agents within the LCC framework is as follows:

i. A message must contain (at least) the following information, which can be encoded and decoded by the sending and receiving mechanisms attached to each agent:

   - An identifier, *I*, for the social interaction to which the message belongs.
   - A unique identifier, *A*, for the agent intended to receive the message.
   - The role, *R*, assumed of the agent in identifier *A* with respect to the message.
   - The message content, *M*, expressed in an ontology understood by the agents.
   - The protocol, *P*, of the form *P*:=<*T*,*F*,*K*> for continuing the social interaction. *T* is the dialogue state. This is a record of the path of the dialogue through the conversation space and the current state of the dialogue for the agents. The second part is a set, *F*, of LCC clauses defining the dialogue framework (based on syntax in Figure 1); and the final part, a set *K*, of axioms consisting of common knowledge to be publicly known between the agents.

ii. The agent must have a mechanism for satisfying any constraints associated with its clause in the dialogue framework. Where these constraints can be satisfied from common knowledge (the set of *K* above) it is possible to supply standard constraint solvers with the protocol, in order to handle a more complex constraints, which will be described in details in section 3.1.

Given these assumptions about the message format, the basic operation an agent must perform when interacting via LCC is to decide what the next steps for its role in the interaction should be, using the information carried with the message it obtains from some other agent. 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. Rewrite rules can be applied to achieve these, and further details with regards to this mechanism and LCC in general can be found in [6-8].

3 Return to Scenario

For this work, the given scenario is formalised as an incremental Multiagent Agreement Problem (MAP) [10], where the process of reaching a mutual agreement requires each attribute (i.e. configuration options and pricing constraints) of the computer to be communicated on an attribute-by-attribute basis among the interacting agents. The interacting agents must jointly map elements from one set, which are modeled as the attributes or variables, to elements of the second set, which are modeled as values, satisfying both intra-agent and inter-agent constraints. In incremental MAP, agents are expected to choose values for variables to satisfy not only their own intra-agent constraints, but also inter-agent constraints with other agents. To ensure that inter-agent constraints are satisfied, agents must coordinate the choice values for variables through an interaction protocol. Further details with regards to the intra-agent and inter-agent constraints covered in this work are described in section 3.1

3.1 LCC Protocol for MAP

The LCC interaction protocol for the scenario introduced in section 1 are defined in expressions 1-4, which are borrowed from works described in [7]. As described in the given interaction protocol clauses, an agent, assuming the role of a customer, asks to buy an item of type *X* from the vendor, then enters into a negotiation with the vendor about the attributes required to configure the item to the customer requirements. The negotiation is simply a recursive dialogue between the vendor and customer with, for each attribute (*A*) in the set of attributes (*S*), the vendor offering the available attribute and the customer accepting it, as illustrated in expressions 2 and 4 respectively.

$$
\begin{aligned}
&a(customer, C) ::= \\
&\quad ask(buy(X)) \Rightarrow a(vendor, V) \leftarrow need(X) \wedge sells(X, V) \text{ then} \\
&\quad a(neg\_cust(X, V, []), C).
\end{aligned}
\tag{1}
$$

$$a(neg\_cust(X,V,As),C) ::= \qquad (2)$$

$$\begin{pmatrix} offer(A) \Leftarrow a(neg\_vend(X,C,\_),V) \ then \\ accept(A) \Rightarrow a(neg\_vend(X,C,\_),V) \leftarrow acceptable(A) \ then \\ a(neg\_cust(X,V,[att(A) \mid As]),C) \end{pmatrix}$$

$$a(vendor, V) ::= \qquad (3)$$

$$ask(buy(X)) \Leftarrow a(customer, C) \ then$$
$$a(neg\_vend(X,C,S),V) \leftarrow attributes(X,S)$$

$$a(neg\_vend(X,C,S),V) ::= \qquad (4)$$

$$\begin{pmatrix} offer(A) \Rightarrow a(neg\_cust(X,V,\_),C) \leftarrow S = [A \mid T] \ \wedge \ available(A) \ then \\ accept(A) \Leftarrow a(neg\_cust(X,V,\_),C) \ then \\ a(neg\_vend(X,C,T),V) \end{pmatrix}$$

Realising Inter-Agent Constraints. The protocol ensures coherence of interaction between agents by imposing constraints relating to the message they send and receive in their chosen roles. The clauses of a protocol are arranged so that, although the intra-agent constraints on each role are independent of others, the ensemble of clauses operates to give the desired overall behaviour, towards the realisation of inter-agent constraints. For instance, as defined in expressions 2 and 4, the protocol places two constraints on each attribute (*A*) in the set of attributes (*S*) of the computer to be purchased: the first (*available(A)*) of expression 4 is a condition on the agent in the role of negotiating vendor sending the message *offer(A)* and second (*acceptable(A)*) of expression 2 is a condition on the agent in the role of negotiating customer sending the message *accept(A)* in reply. By (separately) satisfying these intra-agent constraints the agents mutually constrain the attribute *A*.

Specifying Intra-Agent Constraints. Finite-domains formalism is used to assign a range of valid domain values that can be assigned to the set of attributes *S*. This means, that given a set of attributes $S = \{A_1,..,A_n\}$, there exists a set of domain values $D=\{D_1,..,D_n\}$: where each $D_i (1 \le i \le n)$ is a set of possible finite-domain values for attribute A_i. As described in [11], finite-domains can be formalised as constraint $A_i::D_i$ which means that the value for the variable A_i must be in the given finite-domain D_i. More precisely, if D_i is an:

- *Enumeration domain*, *List*, then A_i is a ground term in the *List*.
- *Interval domain*, *Min..Max*, then A_i is a ground term between *Min* and *Max*.

These specifications constitute what we call *unary constraints*. Finite-domain constraints can also be composed of *binary constraints* over pairs of variables that define the dependency relationship between them. For instance, the finite-domain constraint imposed on the price can be specified as an equation in the form of *price*={1000+((*monitor_size*-14)*100)+((*disk_space*-40)*10)}, which constitutes two parts; a fixed base price of 1000, and a non-fixed component that depends on the available finite-domain values of the attributes needed to configure a computer.

Accommodating Distributed Finite-Domain Constraint Solving. In providing dialogue coordination for distributed and mutual finite-domain constraint solving, [7] describes on how the basic clause expansion mechanism of LCC has been extended to preserve the ranges of restricted finite-domain on negotiated variables. This allows agents to restrict rather than simply instantiate these constraints when interacting, thus allowing a less rigid interaction. For instance, applying this to our example of mutual finite-domain constraints in expressions 2 and 4, if the range of values permitted for *A* by *available(A)* is {32,64,128}, while the range of values permitted for *A* by *acceptable(A)* is {64,128,256}, then were we to use finite-domain constraint solver, a constraint space of {64,128} is obtained – a range that would be attached to the variable returned in the *accept(A)* message.

The finite-domain constraints on variables, mutually defined by the distinct agents are entirely separate and private from each other. So, when a constraint is applied by one agent, the constraint will not propagate to the other agents unless carried by the protocol. This requires one addition to the protocol structure of section 2.2: a list of variable restrictions, *V*, for any variable that has been instantiated and constrained in the protocol. Figure 2 provides a general overview on the basic architecture and process flow on how this is accomplished.

As described in section 2.2, the components of the receipt message include a protocol *P*, of the form *P:=<T,F,K>*. Given this, set *V* contains the current restriction for each variable in the expanded clause of *T*. Once decoded, the set *V* is posted to the constraint store of a finite-domain constraint solver, and the rest of the message will be forwarded to the protocol expansion mechanism to determine the agent's next move in the dialogue. The expansion of an agent's role in a particular round of dialogue interaction requires the relevant variable and its intra-agent finite-domain constraint, associated with the interaction, to be instantiated with values from the agent's knowledge base and posted to the constraint store of the finite-domain constraint solver. Successful expansion of the agent's part in the interaction protocol is determined on whether the newly added constraint is consistent with existing set *V*; the process computationally performed by the finite-domain constraint solver. This process allows the distinct finite-domain constraints, mutually defined by the interacting agents on a particular variable contained within the interaction protocol, to converge to a new finite-domain range. Once completed, an updated state of the interaction protocol, a new message content, and updated set, *V'* are together encoded before being passed to the message passing media to be retrieved by its intended recipient.

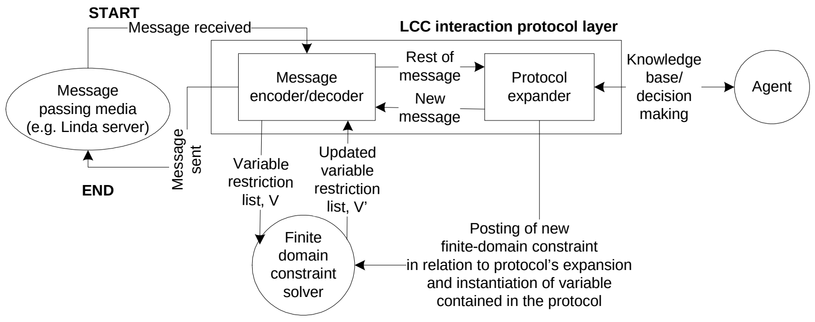

START
Message received
LCC interaction protocol layer
Message passing media (e.g. Linda server)
Message encoder/decoder
Rest of message
New message
Protocol expander
Knowledge base/ decision making
Agent
Message sent
END
Variable restriction list, V
Updated variable restriction list, V'
Finite domain constraint solver
Posting of new finite-domain constraint in relation to protocol's expansion and instantiation of variable contained in the protocol

Fig. 2. Basic architecture of LCC framework for distributed constraint solving interaction

3.2 Source of AIP Brittleness

As described earlier, for a multi-agent interaction involving a distributed constraint solving process over a set of variables, an expansion to the interaction protocol (i.e. moving to the successive states of the protocol) is only possible if the interacting agents can mutually satisfy their part of the finite-domain constraints imposed on variables contained within the protocol. Given the interaction protocol of expressions 1-4 for instance, the dialogue will continue in accordance to the defined interaction protocol as long as the ranges of attribute values offered by the negotiating vendor converge with those required by the negotiating customer. To illustrate this point, 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 finite-domain constraints private to the customer and vendor agents respectively:

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

Upon negotiating these private finite-domain constraints via the defined protocol, the disk space attribute value that meets the vendor offer, and also the customer requirement will be in the mutual range of $40\ Gb \leq disk\_space(D) \leq 100\ Gb$. However, as mentioned in [12], in the process of proposal exchange involving bilateral negotiations between two agents (i.e. customer and vendor), each agent has a *private border proposal*, which is the maximum (or minimum) limit that must be respected in reaching a deal. The intersection between the agents' border proposal defines what we call the *deal range*. If the deal range is empty, then the deal is impossible. This will lead to a failure in the product configuration process and break the prescribed protocol.

3.3 Constraint Relaxation to Reduce AIP Brittleness

Our approach to address this brittleness problem requires an agent to be able to adapt to the constraints on variables established by the other agents, achieved through constraint relaxation. Form of constraints relaxation considered in this work is focused on the revision of the initially assigned finite-domain intra-agent constraints by a single or many agents to ensure that a deal range is obtained.

Constraints relaxation is only possible if the agents participating in the interaction are cognitively and socially flexible to the degree they can handle (i.e. identify and fully or partially satisfy) the constraints that they are confronted with. As further emphasised in [13], a requirement for applying efficient mechanisms for (joint) constraint relaxation and propagation is that agents are able to reason about their constraints and involve other agents in this reasoning process. Thus, for the constraint relaxation process to be accomplished, the engineering requirements expected from the interacting agents include cognitive and social requirements.

The cognitive requirement concerns with the agent's internal reasoning capability that enables it to dynamically modify and redefine its own set of predefined constraints, an inherent functionality expected of agents involved in distributed constraint solving processes. The issue of the best computational approach or constraint relaxation strategy that an agent might employ to reach to this decision is still open, and its discussion extends beyond the scope of this paper. However, a generally accepted notion is that the decision taken should be to the agent's own advantage, leading to the realisation of the eventual goal of the agent (i.e. interacting agents reaching an agreement in solving a particular MAP). The second requirement (i.e. social requirement) obligates the participating agents to communicate and coordinate the constraint relaxation process with the other agents. This process, expected to be handled at the protocol level, is the focus of this work, and will be demonstrated within the LCC framework.

The application of constraint relaxation approach in MAS is not new, as it has been used to resolve conflicting constraints between agents [14]. Particularly within the area of multi-agent negotiation, research has been conducted that modelled negotiation as a constraint-relaxation process. The agents are self-interested in the sense that they would like to achieve an agreement that gives them the highest utility, but are also cooperative in the sense that they are willing to accept the lower offer in order to facilitate reaching an agreement. The agents communicate their constraints through proposals and counterproposals, achieved via a centralised agent who acts as a mediator to resolve any conflicting constraints established by the distinct agents. The central-agent approach is usually adopted in handling constraints, which involves multilateral interaction patterns (i.e. one-to-many or many-to-many) of distributed agents. The use of a central agent, though effective, has been associated with a number of drawbacks that include invasion of privacy [15] and the bottleneck problem [7].

As described in [16], the fundamental constraint-related issues that need to be considered when applying a constraint relaxation approach include:

i. How to relax a constraint?
ii. Which constraint to relax?
iii. When to relax – when exactly during computation that we have to relax the constraint?

By extending the constraint relaxation approach to the AIP domain, agent-related and protocol-related issues that need to be taken into account include:

iv. Who or which agent should be asked to relax a particular constraint?
v. How the agents coordinate their communicative acts when engaging in the constraint relaxation process?

However, not all of these issues can be tackled at the protocol level. Issue like (i), which involves customised and private constraint relaxation strategies, is expected to be internalised within the agent and individually defined by the engineer of the respective agent. The proposed constraint relaxation module that addresses the rest of the issue is illustrated in details in section 4.

4 The Constraint Relaxation Module

The module is composed of three fundamental components, namely:

i. **Extractor**
This component consists of two important processes. First, the constraint ex-

tractor is used to select the constraint(s) to be relaxed, and second, the recipient details extractor, is used to find the agents authorised to relax these constraints. The high level algorithms on how these are achieved are provided in section 4.1.

ii. **Constraint relaxation processor**
This component provides an interface with the agent internal reasoning layer, which allows the list of constraints to be relaxed obtained from (i) to be forwarded to this layer. Agent's feedback in form of finite-domain values on relaxed constraints is propagated against the variable restriction list using a finite-domain constraint solver. This allows constraint consistency to be checked.

iii. **Interaction sub-protocol for constraint relaxation**
This component allows the constraint relaxation process to be communicated and coordinated at the inter-agent level, achieved via the following two processes; first, the composition of sub-protocol to coordinate the agents' communicative acts when engaging in the constraint relaxation process, and second, the insertion of this sub-protocol into the existing protocol that allows the agents' interactions on constraint relaxation to be accommodated. Further details of these two processes are given in section 4.2.

4.1 Extractor

Constraint extraction and decomposition. Within the LCC framework as described in figure 2, an agent can proceed to the next interaction state if an instantiated finite-domain constraint, C_i of a variable A_i, associated with a particular interaction cycle is consistent with a set V containing the current finite-domain restriction for each variable in the expanded interaction protocol. If this is not the case, then it will cause the constraint relaxation module to be enacted. It begins with the process of identifying the set of constraints to be relaxed, and the high-level algorithm for this is as follows:

i. Add A_i into *List*, a list uses to store a possible set of variables, in which the finite-domain constraint imposed on A_i requires relaxation
ii. If C_i is a finite-domain constraint expression composed of N-binary constraints that define the dependency between A_i with N other instantiated variables (i.e. $A_1..A_N$) of already completed protocol states, then, each $A_k(1 \leq k \leq N)$ is added to *List* consecutively, given that A_k is not already exist in *List*

An agent might employ any reasonable finite-domain constraint relaxation strategy on this list of variables contained within *List* (i.e. removal of dependency, expansion of interval values, etc.). Each successive relaxation, $C_{Revised}$, on the content of *List*, is propagated against set V using a finite-domain constraint solver to ensure that the revised value is consistent with the values currently held in V. A successful constraint relaxation will enable the agent to recommence expanding its interaction state, ensuring the continuance of dialogue left out prior to the enactment of the constraint relaxation module.

Agent's details extraction. In case an agent fails to relax its part of the mutual finite-domain constraints associated with the set of variables contained in *List*, then the alternative is to compose a message requesting the other agents to relax their part. In finding the agent(s) authorised to relax these finite-domain constraints, it requires the dialogue state component of the protocol, *T*, to be searched. *T* provides a record of dialogue path followed by each of the interacting agents. Given *T*, the identifier of the agent, $X_{Receipt}$, mutually responsible for instantiating the finite-domain constraint for the variable A_i is obtained. $X_{Receipt}$ is then passed to the constraint relaxation interaction sub-protocol component to be used in a composition process, further described in section 4.2. A constraint relaxation request message, attached within a revised interaction protocol for coordinating the agents' communicative acts in performing the relaxation, is then forwarded to the LCC interaction layer.

4.2 Interaction Sub-Protocol for Constraint Relaxation

Composition of Sub-Protocol. The two roles determined to be important in the process of constraint relaxation are constraint relaxation initiator, and constraint relaxation responder. The constraint relaxation initiator is the one who initiates the constraint relaxation process, and is usually the one who fails to relax its part of the mutual finite-domain constraints imposed on variables. An agent that assumes this role can send a request for a constraint relaxation process to be performed by the other agents who mutually constrain the variable that is failed to be satisfied.

The constraint relaxation responder, on the other hand, is the prospective recipient of this request message. Upon receipt of a request message, the agent that assumes this role might reply with a message informing either the request for the constraint relaxation has been performed or it failed to relax. A composed interaction sub-protocol that defined the message passing behaviour of these two roles is instantiated with details (i.e. A_i, the variable in which its finite-domain constraint needs relaxation, and $X_{Receipt}$, the agents' identifier(s) authorised to perform finite-domain constraint relaxation on A_i) obtained from the extractor component.

Revision of Agent's Dialogue State. Once the composition and instantiation process is complete, it is necessary to integrate this sub-protocol with the interaction protocol currently followed by the agents. This requires the protocol's dialogue state component, *T*, of the agent(s) identified to be involved in the constraint relaxation interaction to be inserted with the composed sub-protocol. In a way, this insertion of sub-protocol can be seen as an interruption to the dialogue flow expected to be iterated among the interacting agents. This allows the agent(s) in receipt of the revised protocol to participate in the joint process of constraint interaction, as the sub-protocol defines on how the recipient of a message that consists of constraint relaxation related contents, can communicatively involve in the process.

A message, contained within a request for a constraint relaxation to be performed on a list of variable(s), a revised protocol and a revised set *V*, containing the restriction for each variable in the expanded interaction protocol, will be together encoded before being passed to the message passing media to be retrieved by the intended recipient. In order to properly interpret this message, the recipients need to ensure that the constraint relaxation module is locally defined on their side.

5 Constraint Relaxation Application

The constraint relaxation module described in section 4 is implemented using SICStus Prolog, and the finite-domain constraint solver available in SICStus Prolog (i.e. clp(FD)) [17] is used to accommodate the handling of finite-domain constraints imposed on variables contained in the interaction protocol. The constraint solver restricts variables to integer ranges. The expression *V* in *L..U* restricts variable *V* to be in the range of *L* to *U*, where *L* and *U* are integers or the constant *inf* (for lower infinity) or *sup* (for upper infinity). These ways of defining and restricting ranges of variables are part of the specific constraint solver used in our example but different constraint solvers could be used. The more important issue is to demonstrate how a constraint relaxation performed locally by the vendor or customer agents is communicated consistently (i.e. at the intra-agent and/or inter-agent levels) throughout an interaction between agents. The working of the mechanism is demonstrated below but first some finite-domain constraints are introduced for our example.

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 a disk space attribute value in between 40 or above.

need(pc) (**5**)
sell(pc,s1)
acceptable(disk_space(D)) ← D in 40..sup
acceptable(monitor_size(M)) ← M in 17..sup
acceptable(price(_,_,P)) ← P in 1200..1600

The vendor agent's local constraints are defined in the similar way to that of the customer. We define the available ranges for the attributes needed to configure a computer and relate these to its price via a simple equation (the aim being to demonstrate the principle of relating constraints rather than to have an accurate pricing policy in this example).

attributes(pc,[disk_space(D),monitor_size(M), price(D,M,P)]) (**6**)
available(disk_space(D)) ← D in 40..80
available(monitor_size(M)) ← M in 14..21
*available(price(D,M,P) ← P #= 1500+((M-14)*100)+((D-40)*10)*

The finite-domain values for the price attribute of both agents are set to be conflicting with each other to demonstrate the working of the constraint relaxation module. The sequence of message passing that follows from the protocol expressions is shown in table 1. 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; 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 to five then are offers of ranges for attributes (*disk_space*, and *monitor_size*) each of which are accepted by the customer. However, in illocution six, an offer of *r(P*,[[1800|2600]]*)* for the price attribute by the vendor, is conflicting with the local finite-domain constraint of *r(P*,[[1200|1600]]*)* imposed by the customer, which causes a failure to expand the interaction protocol received with the message. This enables the constraint relaxation module to be enacted.

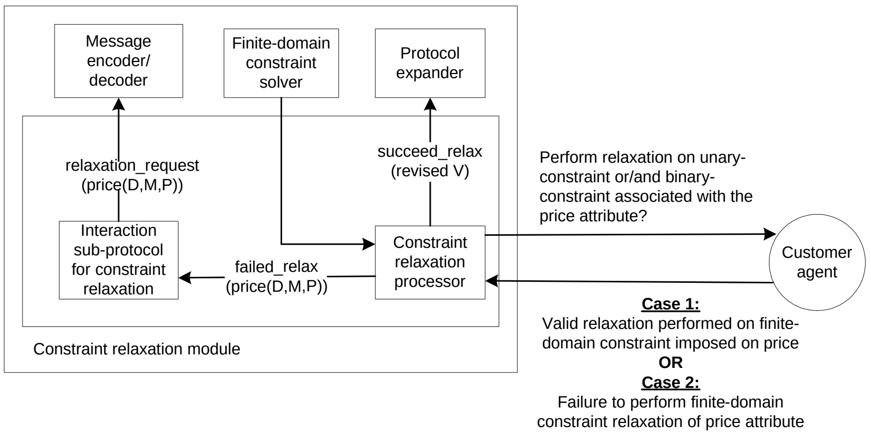
In this example, the constraint relaxation module is evaluated against two different scenarios. First, given the customer agent failure to reply with an *offer(price(D,M,P))* message in accordance to the prescribed interaction protocol, we consider a situation in which the customer agent is able or agreed to relax its part of the mutual finite-domain constraint on price locally. For this case, our primary aim is to show on how the intra-agent interactions involving components of the AIP layer, the constraint relaxation module, and agent's internal reasoning module are accomplished and coordinated, as illustrated in figure 3. Upon enactment of the constraint relaxation module, the constraint relaxation processor component will be involved in a repetitive interaction with the agent's internal reasoning module in order to obtain a revised finite-domain constraint on the price attribute given the failed unary constraint of *P in 1200..1600*, until a set of valid relaxation value is obtained or the agent decides against relaxation. Assuming that the customer agent decides to revise its finite-domain constraint to *P in 1500..2000* (i.e. case 1 of figure 3), using a finite-domain constraint solver, this value will be propagated against the current variable restrictions set *V*=[*r(P,*[[1800|2600]]*), r(M,*[[17|21]]*), r(D,* [[40|80]]*)*], causing a revision on its content to *V*=[*r(P,*[[1800|2000]]*), r(M,*[[17|21]]*), r(D,* [[40|80]]*)*]. The expansion of the interaction protocol will recommence once this local constraint relaxation process performed by the customer agent is complete.

Second, we consider a situation, in which the customer agent fails to relax its part of the mutual finite-domain constraint, thus, requiring the involvement of the vendor agent in the constraint relaxation process. For this case, our primary aim is to show the enactment of the interaction sub-protocol for constraint relaxation component, together with the inter-agent interaction aspect that takes place during the relaxation process, and how these components are managed and coordinated. Assuming that the customer agent fails to relax its part of the mutual finite-domain constraint imposed on the price attribute (i.e. case 2 of figure 3), then the interaction sub-protocol for constraint relaxation component will be enacted. A message, contained within a request for the vendor agent to relax its part of the mutual finite-domain constraint imposed on the price attribute, will be composed and sent to the vendor agent, together with the revised interaction protocol, contained within a sub-protocol specifying the roles, and message passing behaviour expected to coordinate the constraint relaxation process.

The sequence of message passing that follows from an interaction between the constraint relaxation initiator (i.e. customer) and the constraint relaxation responder (i.e. vendor) concerning a constraint relaxation of the price attribute is shown in illocution 7 and 8 of table 2. In relaxing its part of the finite-domain constraint, the vendor, in the role of constraint relaxation responder, will undergo a similar intra-agent interactions process as described in figure 3. Since this relaxation involves a set of binary constraints, there exists a number of constraint relaxation strategies that can be employed by the customer agent (e.g. remove the dependency of price attribute on the other attributes). Assuming that the vendor agent agrees to relax its fixed base price component of the price attribute from 1500 to 1000, then a new finite-domain constraint of *r(P,*[[1300|2100]]*)* is obtained. A message confirming that a relaxation has been performed, as described in illocution 8, will be sent together with the revised variable restrictions set *V* of *V*=[*r(P*,[[1300|1600]]*), r(M*,[[17|21]]*), r(D,* [[40|80]]*)*]. Upon receipt of this message, the constraint relaxation message processor, local to the customer agent will be enacted, and the received set *V* will be applied. Once this is complete, that is the sub-protocol concerning the interaction protocol of constraint relaxation has been fully expanded, the agents resume their prior roles, and continue with the interaction that has been left out.

Table 1. Sequence of message passing

No	Sender	Message	Recipient	Restrictions
1	*a(customer,b1)*	*ask(buy(pc))*	*a(vendor,s1)*	[]
2	*a(neg_vend(pc,b1,[disk_space(D), monitor_size(M), price(D,M,P)]),s1)*	*offer(disk_space(D))*	*a(neg_cust(pc,s1,_),b1)*	[*r* (*D*, [[40|80]])]
3	*a(neg_cust(pc,s1,[]),b1)*	*accept(disk_space(D))*	*a(neg_vend(pc,b1,_),s1)*	[*r* (*D*, [[40|80]])]
4	*a(neg_vend(pc,b1,[monitor_size(M) price(D,M,P)]),s1)*	*offer(monitor_size(M))*	*a(neg_cust(pc,s1,_),b1)*	[*r (M,* [[14|21]]),*r (D,* [[40|80]])]
5	*a(neg_cust(pc,s1,[att(disk_space(D))]),b1)*	*accept(monitor_size(M))*	*a(neg_vend(pc,b1,_),s1)*	[*r (M,* [[17|21]]),*r (D,* [[40|80]])]
6	*a(neg_vend(pc,b1,[price(D,M,P)]),s1)*	*offer(price(D,M,P))*	*a(neg_cust(pc,s1,_),b1)*	[*r (P,* [[1800|2600]]), *r (M,* [[17|21]]) ,*r (D,* [[40|80]])]



Message encoder/ decoder
Finite-domain constraint solver
Protocol expander
relaxation_request (price(D,M,P))
succeed_relax (revised V)
Perform relaxation on unary-constraint or/and binary-constraint associated with the price attribute?
Interaction sub-protocol for constraint relaxation
failed_relax (price(D,M,P))
Constraint relaxation processor
Customer agent
Case 1:
Valid relaxation performed on finite-domain constraint imposed on price
OR
Case 2:
Failure to perform finite-domain constraint relaxation of price attribute
Constraint relaxation module

Fig. 3. Intra-agent interactions of finite-domain constraint relaxation

Table 2. Sequence of message passing involving constraint relaxation

No: *7* **Sender**: *a(const_initiator(pc,s1,_),b1)* **Message**: *relax_request(price(D, M, P))* **Recipient**: *a(const_responder(pc,b1,_),s1)* **Restrictions**:*V*=[*r(P*,[[1200|1600]]*),* *r(M*,[[17|21]]*), r(D,* [[40|80]]*)*]	**No**: *8* **Sender**: *a(const_responder(pc,b1,_),s1)* **Message**:*relax_performed(price(D, M ,P))* **Recipient**: *a(const_initiator(pc,s1,_),b1)* **Restrictions**:*V*=[*r(P*,[[1300|1600]]*),* *r(M*,[[17|21]]*), r(D,* [[40|80]]*)*]

6 Discussion and Future Work

Induced backtracking is another approach that has been applied to address constraint failures in distributed dialogue protocols [18]. It is considered limited because in multi-agent interactions we cannot assume that agents having received messages are able to backtrack, since they may be implemented in a language that does not support backtracking [7]. In addition, the work reported in [18] does not specifically focus on finite-domain constraints. Given that a failed constraint is dependent on a number of other constraints that have been satisfied by the interacting agents, then choosing an acceptable backtracking point within the expanded dialogue states to accommodate the agents' constraint relaxation strategies, might be a complicated matter.

On the other hand, the approach reported in this paper provides a mechanism that allows an interaction sub-protocol on constraint relaxation to be initiated and incorporated with the currently executed interaction protocol. This is considered an improved and extended version of our previous work involving constraint relaxation as reported in [9]. In our previous work, the communicative acts for the joint process of constraint relaxation are predefined in the interaction protocol deployed to each of the agents participating in the interaction. This requires the agents' roles in the constraint relaxation interaction to be determined in advance, as such, it does not provide the necessary support for a flexible constraint relaxation strategy to be incorporated. In addition, a number of processes which could be accomplished at the protocol layer are assumed to be internalised within the interacting agents.

At this stage, our constraint relaxation module can only accommodate a minimum interaction requirement, in which the agents can propose or request for a constraint relaxation to be performed, and respond to such requests by either acceptance or rejection. However, with a simple accept or reject reply, the constraint relaxation initiator has no idea in which direction of search space should it move in order to find a converging mutual finite-domain constraint range. This could be time consuming and inefficient, and might also lead to an infinite loop. Therefore, an ideal remedy requires the agents in receipt of a constraint relaxation request to provide more information in their response to help direct the initiator. This is possible if the agents' responses to a request could include *critiques* or *counter-proposals* [19], which enable the agents to exercise a more flexible constraint relaxation strategy. To accommodate this type of flexible interaction at the protocol level, the interaction protocol currently executed by the interacting agents needs to undergo a dynamic revision process, controlled within a certain set of parameters. The described extension is one of the important focuses of our further research work.

References

[1] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "Specification and verification of agent interaction using social integrity constraints," *Theoretical Computer Science*, vol. 85, pp. 23, 2004.

[2] J. Odell, H. V. D. Parunak, and M. Fleischer, "Modeling agents and their environment: the communication environment," *Journal of Object Technology*, vol. 2, pp. 39-52, May-June 2003.

[3] B. Chen and S. Sadaoui, "A generic formal framework for multi-agent interaction protocols," University of Regina, Canada, Technical report TR 2003-05, 2003.

[4] 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.

[5] 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.

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

[7] D. Robertson, "Multi-agent coordination as distributed logic programming," presented at 20th International Conference on Logic Programming, Saint-Malo, France, Sept. 6-10, 2004.

[8] D. Robertson, "A lightweight coordination calculus for agent social norms," presented at Declarative Agent Languages and Technologies (AAMAS), New York, USA, 2004.

[9] F. Hassan and D. Robertson, "Constraint relaxation to reduce brittleness of distributed agent protocols," presented at Coordination in Emergent Agent Societies Workshop (CEAS 2004), held in conjunction with the 16th European Conference on Artificial Intelligence (ECAI' 04), Valencia, Spain, 2004.

[10] P. J. Modi and M. Velose, "Bumping strategies for the multiagent agreement problem," presented at Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecht,Netherland, July,2005.

[11] T. Fruhwirth, "Theory and practice of constraint handling rules," *The Journal of Logic Programming*, vol. 37, pp. 95-137, 1998.

[12] G. E. d. Paula, F. S. Ramos, and G. L. Ramalho, "Bilateral negotiation model for agent-mediated electronic commerce," presented at Agent-Mediated Electronic Commerce (AMEC 2000) Workshop, Barcelona, Spain, 2000.

[13] G. Weib, "Congnition, sociability, and constraints," in *Balancing Reactivity and Social Deliberation in Multi-Agent Systems: From RoboCup to Real-World Applications*, vol. 2103 (LNAI), M. Hannebauer, J. Wendler, and E. Pagello, Eds.: Springer-Verlag Heidelberg, 2001.

[14] K. P. Sycara, "Multiagent systems," in *AI Magazine*, vol. 19, 1998, pp. 79-92.

[15] D. Pruitt, *Negotiation behaviour*. New York: Academic Press, 1981.

[16] N. Jussien and P. Boizumault, "Implementing constraint relaxation over finite domains using assumption-based truth maintenance systems," in *Lecture Notes in Computer Science (LNCS)*, vol. 1106, M. Jumper, E. U. Freuder, and M. J. Maher, Eds.: Springer, 1996, pp. 265-280.

[17] *SICStus Prolog User's Manual*. Stockholm: Swedish Institute of Computer Science (http://www.sics.se/sicstus.html), 1999.

[18] N. Z. Osman, "Addressing constraint failures in distributed dialogue protocols," University of Edinburgh, MSc. Thesis 2003.

[19] S. Parsons, C. Sierra, and N. R. Jennings, "Agents that reason and negotiate by arguing," *Journal of Logic and Computation*, vol. 8, pp. 261-292, June 1998.