A Structural Synthesis System for Argument Protocol from High-Level Descriptions

2010

Abstract

As web service choreography matures, there is greater demand for multi-agent protocols to support argument and argumentation. There is not, however a single canonical argumentation protocol. Instead we find many different styles of argumentation depending on the contest for service interaction. This offers a need for rapid protocol syntheses. In this paper we propose a high level control flow specification language for designers to build interaction protocols by reusing Lightweight Coordination Calculus (LCC)-argumentation patterns thus accelerating the construction process and reducing manual effort. Our tool receives as input a Dialogue Interaction Diagram (DID) and returns the LCC protocol generated by LCC-Argument patterns. We illustrate our proposal with a persuasion dialogue use-case.

Key words: Structural Synthesis, Lightweight Coordination Calculus , Argument Interchange Format

1 Introduction

An argument offers a reason for believing a statement, taking an action, changing a goal, etc. Argumentation has for some time been an important area for research in natural language processing, knowledge representation and construction of automated reasoning systems. It also has attended to multi-agent systems (MAS), in particular to model the communication between agents, where it supports mechanisms for designing, implementation and analyzing models of the interaction among agents. However, a wide ranging approach of this kind carries with it various challenges. An important challenge is to ensure that agent arguments can be reliable communicated using argument-based protocols.

Argument Interchange Format (AIF) [1, 2] is an approach that has been used successfully to tackle this challenge. Recognizing that as single style of argumentation fits all circumstances, the AIF stipulates a layered style of specification in which a high level language is used to specify the argument and this is then implemented as a protocol.

Interaction protocols in AIF are represented using a protocol language called the Lightweight Coordination Calculus (LCC) [3, 4] an executable specification language which gives an overall architecture for coordination of MAS.

The goal of this research is to develop a useful tool that can enable designers to build an efficient LCC program in the easiest and quickest way. The aim is to build a high level control flow specification language for designers to build an agent by reusing common LCC argumentation patterns. The selection and instantiation of these patterns is performed automatically given a high level specification in the AIF.

The paper is organized as follows. Next, in Section 2, we summarize the basic concepts of AIF. In Section 3 we present a high level specification of interaction protocol. Then, in Section 4 we describe the Lightweight Coordination Calculus. In Section 5, provides an example of a dialogue protocol defined in LCC. In Section 6 we show how this is synthesized. Finally, section 7 presents conclusions and future work.

2 The Argument Interchange Format (AIF)

AIF [1, 2] is an international effort which has been proposed for representation and communication (data exchange) of argument resources between agents, research groups, argumentation tools, and specific domains. It provides an ontology which represents an argument as a network of linked nodes. This network consists of two types of nodes: Information nodes (I-nodes) that content specific data (such as claims, proposition and premises) that depend upon the domain of discourse, and scheme application nodes (S-nodes) that describe the domain independent patterns of reasoning. S-nodes come in three different types, including rule of inference application nodes (RA-nodes) that define the support or inference of argument, preference application nodes (PA-nodes) that define the value judgments or preference orderings of argument, and conflict application nodes (CA-nodes) that define the conflict of argument.

There are various restrictions on how nodes connected. For example, I-nodes cannot be connected to other I-nodes directly; they must be connected across S-nodes. On the other hand, S-nodes can be connected to other S-nodes directly. Basically, two types of edges can be added to connect any two nodes: scheme edges that support conclusions and start form S-nodes and end either in I-nodes or S-nodes, and data edges that supply data and start form I-nodes and end in S-nodes. Table 1 [1] summarizes smantics of support for node-to-node relationships.

AIF envisages three groups of concepts for communication in the context of argumentation:

- Locutions: Particular words, phrases or form of expressions which are used by agents.;
- Interaction Protocols: define communication between agents via a set of rules governing how two or more agents should interact in order to reach a specific goal.;
- Communication Context: includes elements which form the context of argumentation like: communication language which defines the set of possible locutions, participants ID, roles that the participants take on in the dialogue,

dialogue topic, and dialogue type such as persuasion or information seeking dialogue, commitment stores and commitment rules.

	to I-node	to RA-node	to PA-node	to CA-node
from I-node		I-node data used in applying an inference	I-node data used in applying a preference	I-node data in conflict with information in node supported by CA-node
from RA-node	inferring a conclusion in the form of a claim	inferring a conclusion in the form of an inference application	inferring a conclusion in the form of a preference application	inferring a conclusion in the form of a conflict definition application
from PA-node	applying a preference over data in I-node	applying a preference over inference application in RA-node	meta-preferences: applying a preference over preference application in supported PA-node	preference application in supporting PA-node in conflict with preference application in PA-node supported by CA-node
from CA-node	applying conflict definition to data in I-node	applying conflict definition to inference appliction in RA-node	applying conflict definition to pereference application in PA-node	showing a conflicr holds between a conflict defintion and some other piece of information

Table 1: Dialogue Types

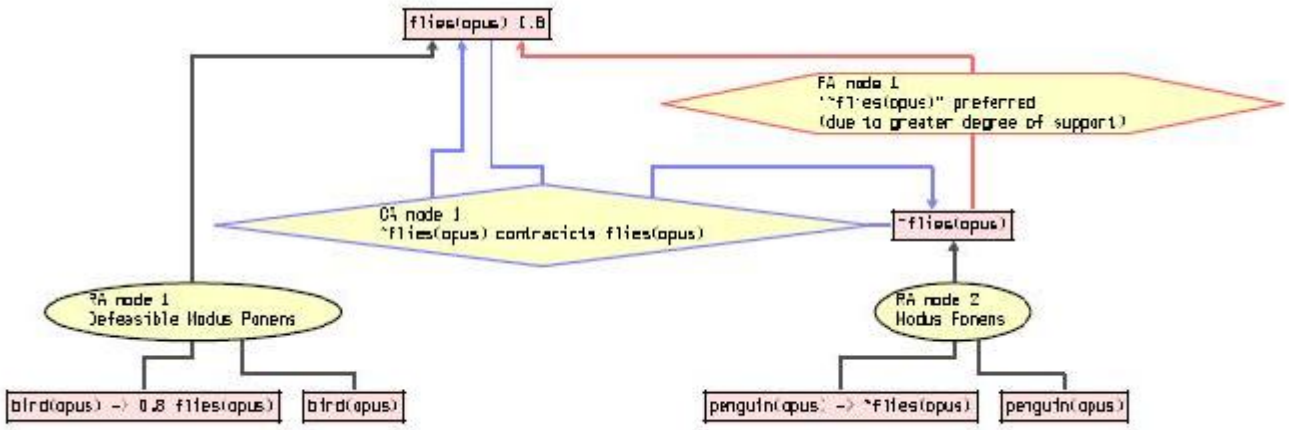

Figure 1: Flying abilities in birds: a concrete example of an argument network

AIF Example

The following Flying Abilities in Birds [1] is a simple example of an argument network (Figure 1):

- The argument for flies(opus) is composed of one RA-node, namely Modus Ponens and two child nodes (premises)
- The argument for flies(opus) is composed of one RA-node, namely defeasible Modus Ponens and two child nodes (premises)
- CA-node linking (bi-directionally) the two conflicting nodes: flies(opus) and flies(opus)
- The argument for flies(opus) has a higher degree of support because the premises support it with a higher degree of probability (1 degree). Whereas, the argument for flies(opus)) weak, because the premises support it with only 0.8 degree (a low probability). So flies(opus) is preferred to the argument for flies(opus). That's why the intermediate PA-node linking flies(opus) to flies(opus).

3 Interaction Protocol Representation

We use the traditional protocol of dialogue games [5, 6] where a dialogue is presented as a game in which each participant of dialogue make moves to attacks or surrenders the previous move of other participant. A Dialogue game protocol defines different rules such as:

- Locutions rules: represent the permitted moves,;
- Commitment rules (post-conditions): define the propositional commitments made by each participant with each move during the dialogue,
- Structural rules (reply rules): define legal moves in terms of the available moves that a participant can select to follow on the previous move.

See[5, 6] for further detail about the protocol different rules (locution rules, commitment rules and structural rules).

Figure 2 illustrates the overall structure of our Dialogue Interaction Diagram (DID). The illustration provides mechanisms to represent interaction protocol rules, by giving an overview of the permitted moves and their relationships. DID is a recursive diagram. It is a unique-move and immediate-reply protocol. The turn-taking between agents switches after each move, and each agent must reply to the move of the previous gent. An oval shape represents the locutions in our protocols and the links represent attack (reply) relations between arguments. There are five locutions: three attack locutions (claim, argue, and why), and two surrender locutions (concede and retract). Dot rectangles represent the locution type: Starting (claim and argue), Termination (concede and retract), and Recursive (why and argue) locution. In practice, the argument may start with either claim or argue and end with a concede or retract locution. A Rhombus shape represents conditions which apply to each move. KB (Knowledge Base list) represents the domain of discourse (e.g. claims, propositions, or data). CS (Commitment Store list) contains a set of propositions to which the player is committed in the discussion.

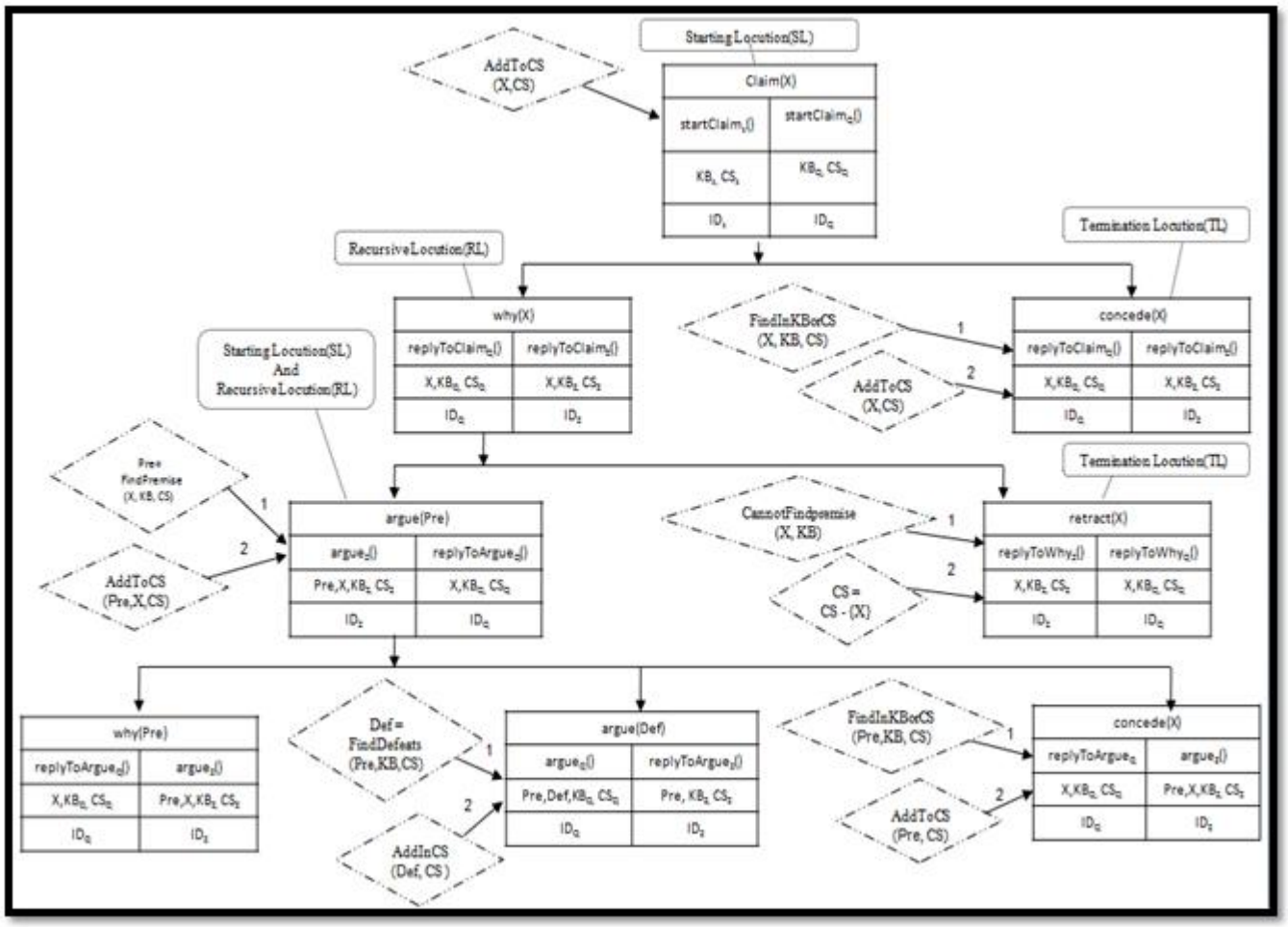

Figure 2: Dialogue Interaction Diagram

In this dialogue, Agent1 can open the discussion by sending either a claim(X) or argue(X) locution. Then, turn-taking switches to Agent2. Agent2 has to choose between two different possible reply locutions: why(X) or concede(X). Agent2 will make his choice using the conditions which appear in the Rhombus shape. In order to choose concede(X), Agent2 must be able to satisfy the two conditions which connect with concede. If he is not able to satisfy these conditions, Agent2 will send why(X). After that, the turn switches to Agent1, and so forth. The argument terminates once Agent1 sends retract(X) or Agent2 sends concede(X).

We obtained the idea of characterizing locutions and interaction protocols in the AIF's argument network ontology from paper [7]. The idea is to represent locutions as the content in I-nodes and interaction protocols as an additional scheme node type (Protocol Interaction Application (PIA) nodes).

We next need a simple protocol language that is sufficiently sophisticated to provide the capabilities of the AIF in a distributed system. For this we used the

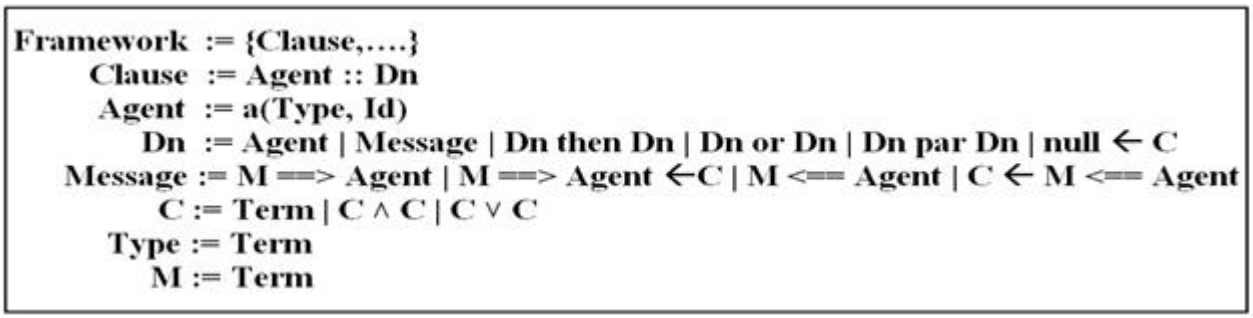

Figure 3: The abstract Syntax of LCC

lightweight coordination calculus (LCC), a declarative choreography language based on a process calculus is described in the following section.

4 The Lightweight Coordination Calculus (LCC)

The abstract syntax of LCC clause is shown in Figure 3 [8, 9]. It describes the interaction behavior of a role. The role is described by the role type and the agent unique identifier. The role behavior's is specified using of the sequence "then" or choice "or" operators to connect messages and roles.

Messages are the only way to exchange information between roles. There are two types of LCC messages: outgoing messages "M" from a role "⇒", and incoming messages "M" to a role "⇐". Message passing; role changing and recursion over role may be required to satisfy a constraint "C". Constraint represented by using "¡–" simple and defined by conjunction and disjunction operators. There are two types of constraint: pre and post condition. Pre-conditions specify the required conditions for an agent to send a message and for the receiver to accept it and process it. Post-conditions explain the states of the sender after sending a message and of the receiver after receiving a message. Agent can satisfy the constraints internally by using the agent's method or externally by using a set of Horn clauses which defines a shared knowledge. Null represents an event which is not related to message passing. Term represents constant or variable.

An Example of LCC protocol

we now demonstrate LCC using the argumentation example of the previous section .Two agents, Z and Q, are having a conversion:

1) Z: My car is safe. (claim)
2) Q: Why is your car safe? (why)
3) Z: Since it has an airbag (argue)

LCC protocol for the first argument:

a(R1,Z)::
claim("My car is safe") ⇒ a(R2, Q)
a(R2,Q)::
claim("My car is safe")⇐ a(R1, Z)

This is read as: agent Z send claim message to the agent Q and then agent Q received claim message from agent Z.

LCC protocol for the second argument:

a(R4,Q)::
why("Your car is safe") ⇒ a(R3, Z)
a(R3,Z)::
why("Your car is safe") ⇐a(R4, Q)

This is read as: agent Q send why message to the agent Z and then agent Z received why message from agent Q.

We can write general LCC protocol that fits with the first and second argument:

a(R1,ID1)::
Locution (X) ⇒ a(R1, ID2)
a(R2,ID2)::
Locution (X) ⇐ a(R2, ID1)

Where Locution represents any permitted movies in the protocol, X represents the speech acts (topic) and ID1 and ID2 are agent's identifier.

5 Types of Dialogue

Walton and Krabbe [10] identify 5 different general types of dialogue in MAS based on their preconditions and goals. Table 2 shows a summary. We want to create a general framework in which we can synthesise the many different specific argument protocols that are within these general types. In order to achieve that, we will start by work in persuasion dialogue and then try to extend our work to fits with others types. In persuasion dialogues one participant is trying to persuade another participant to change their point of view. In the following sections we will present an example of persuasion dialogue and then we will use this example to generate the structure synthesis of persuasion dialogue games.

5.1 An Example Persuasion Dialogue

The following example persuasion dialogue, adapted from [11] will be used in this paper to illustrate the steps of structure synthesis:

Z: My car is safe. (making a claim)
Q: Why is your car safe? (asking grounds for a claim)
Z: Since it has an airbag, (argue: offering grounds for a claim)
Q: OK, your car is safe. (concede)

5.2 The Basic Scenario of Interaction Protocol of Persuasion Dialogue

Figure 4 represents the persuasion dialogue graph of care safe example represented in the AIF:

1. Dialogue takes place between two agents, Z and Q.
2. Z has KBZ and CSZ, and Q has KBQ and CSQ.
3. Z and Q can access CS Z and CSQ.
4. Z opens the discussion by sending claim ("My car is safe").

Dialogue typesn	Initial Situatiom	Participant's Goal	Goal of Dialogue
Persuasion	Confliect of opiniens	Persuade other party	Resolve or clarifiy issue
Inquiry	Need to have proof	Find and Verify evidence	Prove(disprove) hypothesis
Negotiation	Confilict of interests	Get what you most want	Reasonable settlement
Informaiton seeking	One party lacks information	Acquire or give informaion	Exchange inforamtion
Deliberation	Dilemma or pracitcal choice	Co-ordinate goals or actions	Decide best course of action

Table 2: Dialogue Types

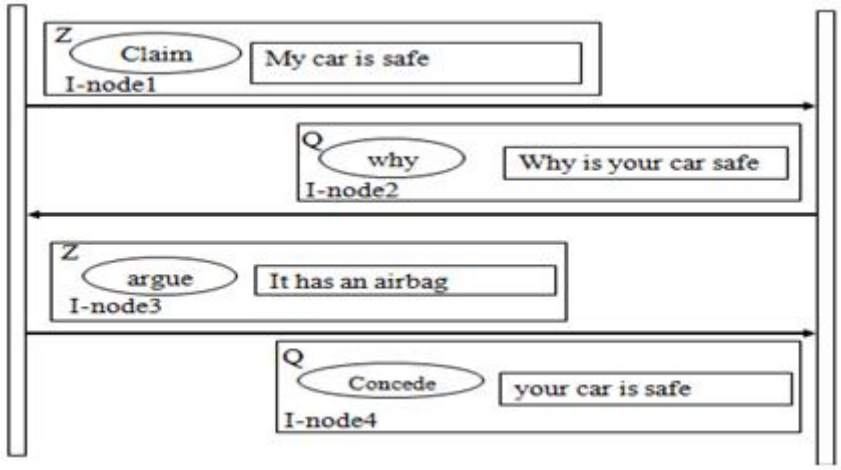

Figure 4: The persuasion dialogue graph represented in the AIF

5. Q checks with its argumentation system ASQ (ASQ = $\{KBQ, CSQ\}$) whether "My car is safe" is acceptable or not:
   - it finds that "My car is safe" is not acceptable, ;
   - Q challenges "My car is safe". In others words, it asks what the reason behind Z's proposal of "My car is safe". In this example, Q will challenges "My car is safe" by sending why("My car is safe") locution.
6. Z responds to the challenge by declaring the supporting premises Pre for "My car is safe". For Pre premises, Q checks with ASQ whether "My car is safe" is acceptable or not. In this example, Z offering grounds for a claim by sending argue("It has an airbag") locution.
7. Q checks with its argumentation system ASQ whether "if car has an airbag, car is safe" is acceptable or not. In this example Q finds that and concedes the main claims by sending concede ("your car is safe") locution.

5.3 LCC Protocol for Persuasion Dialogue

Table 3 represents the persuasion dialogue protocol of care safe example represented in the LCC.

Agent Z	Agent Q
a(startClaimZ(KBZ,CSZ),IDZ):: claim(X) ⇒ a(startclaimQ(KBQ,CSQ),IDQ) ⟵ AddInCS(X, CSZ) then a(replyToClaimZ(X,KBZ,CSZ),IDZ). **a(replyToClaimZ((X,KBZ,CSZ),IDZ)::** concede(X) ⇐ a(replyToClaimQ ((X,KBQ,CSQ),IDQ) or (why(X) ⇐ a(replyToClaimQ ((X ,KBQ,CSQ),IDQ) then a(replyToWhyZ (X ,KBZ,CSZ),IDZ)). **a(replyToWhyZ(X ,KBZ,CSZ),IDZ)::** retract(X) ⇒ a(replyToWhyQ(X,KBQ,CSQ),IDQ) ⟵ CannotFindPremise(X,KBZ) and SubtractFromCS(X,CS) or a(argue(X,Pre, KBZ,CSZ),IDZ) ⟵ Pre=FindPremise(X,KBZ). **a(argueZ(X,Pre,KBZ,CSZ),IDZ) ::** (argue(Pre) ⇒ a(ReplyToArgueQ(X,KBQ,CSQ),IDQ) ⟵ AddToCS(Pre ,X,CSZ) then concede(X) ⇐ a(ReplyToArgueQ(X,KBQ,CSQ),IDQ) or ((why(Pre) ⇐ a(ReplyToArgueQ(X,KBQ,CSQ),IDQ) then a(ReplyToWhyZ(Pre ,KBZ,CSZ),IDZ))) or a(ReplyToArgueZ(Pre,KBZ,CSZ),IDZ). **a(ReplyToWhyZ(Pre ,KBZ,CSZ),IDZ)):: .....** **a(ReplyToArgueZ(Pre,KBZ,CSZ),IDZ)):: .....** --- **AddInCS(X,CSZ)**= agent Z will update its commitment store by adding X to CSZ (X represents the claim topic and CS represents agent believes list) **CannotFindPremise(X,KBZ)** =agent Z checks with its KBZ whether premises for X exist it or not. If agent Z cannot finds any premises, this constraint returns true **SubtractFromCS(X,CS)** =agent Z will delete X from its commitment store list (CS) **Pre= FindPremise (X,KBZ)**= agent Z checks with its KBZ whether premises for X exist it or not. If agent Z finds premises, it saves the premises in Pre	**a(startclaimQ(KBQ,CSQ),IDQ)::** claim(X) ⇐ a(startClaimZ((KBZ,CSZ),IDZ) then a(replyToClaimQ(X, KBQ,CSQ),IDQ). **a(replyToClaimQ((X,KBQ,CSQ),IDQ)::** concede((X) ⇒ a(replyToClaimZ (KBZ,CSZ),IDZ) ⟵ FindInKBorCS((X,CSQ,KBQ) and AddInCS(X, CSQ) or (why((X) ⇒ a(replyToClaimZ (KBZ,CSZ),IDZ) then a(replyToWhyQ((X,KBQ,CSQ),IDQ)). **a(replyToWhyQ(X,KBQ,CSQ),IDQ)::** retract(X)⇐ a(replyToWhyZ(X ,KBZ,CSZ),IDZ) or a(replyToArgueQ(X,KBQ,CSQ),IDQ). **a(replyToArgueQ(X,KBQ,CSQ),IDQ)::** argue(Pre) ⇐ a(argueZ (X ,Pre,KBZ,CSZ),IDZ) then concede(X) ⇒ a(argueZ (X,Pre,KBZ,CSZ),IDZ) ⟵ FindInKBorCS(X,Pre,CSQ, KBQ) and AddInCS(Pre,CSQ) or a(argue((Pre, Def, KBQ,CSQ),IDQ)) ⟵ Def= FindDefeat(Pre,KB,CS) and AddInCS(Def,CSQ) or why(Pre) ⇒ a(argueZ (X,Pre,KBZ,CSZ),IDZ) then a(replyToWhyQ (Pre ,KBQ,CSQ),IDQ)). **a(argueQ(Pre,Def,KBZ,CSZ),IDZ) ::** ..... **a(replyToWhyQ (Pre ,KBQ,CSQ),IDQ)):** ..... --- **AddInCS(Pre ,X, CS)** = agent Z will update its commitment store by adding Pre and X to CSZ (X represents the claim topic and Pre represents the X supporting Premise) **FindInKBorCS(X,CSQ,KBQ)**= agent Q checks with its argumentation system KBQ and CSQ whether X is acceptable or not. If X is acceptable, this constraint returns true **Def = FindDefeat(Pre,KB,CS)** =agent Q checks with its argumentation system KBQ and CSQ whether defeat premises for pre exist it or not. If agent Q finds defeat,it saves the defeat in Def

Table 3: LCC protocol for persuasion dialogue (care safety example)

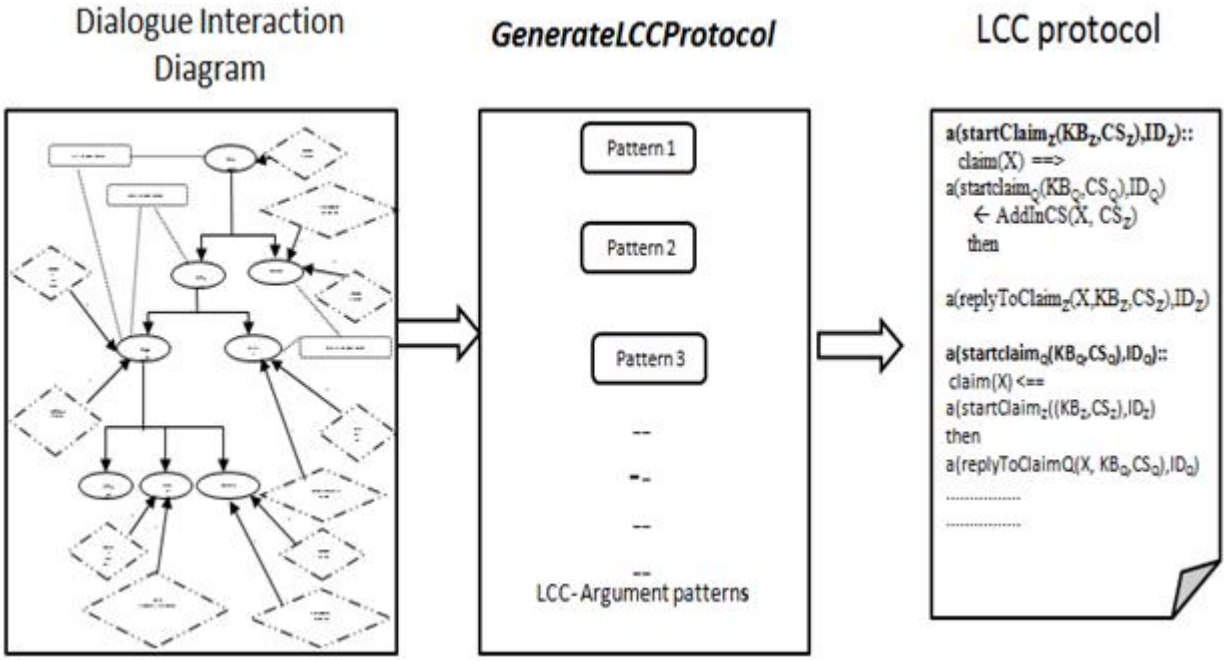

Figure 5: GenerateLCCProtocol

6 Structural Synthesis System for Argument Protocol

Our algorithm GenerateLCCProtocol, Figure 5, receives as input a DID such as Figure 2, and returns LCC protocol such as the one in Table 3 by using LCC-Argument patterns. In practice, representing interaction protocol as a DID and using LCC-Argument patterns can speed up the protocol's building process by providing guidance to the software engineer. By taking a closer look at the LCC protocol in Table 3, we realize that this protocol is quite complex, and therefore requires considering issues that the software engineer may not realize until later in the implementation process—such as synchronization of the role. Use of DID and LCC-Argument patterns helps to prevent subtle issues that can cause some problems. It also improves code readability and the efficiency of role synchronization mechanisms.

LCC- Argument Pattern (Structure Synthesis)

Our purpose is to find a common LCC argument pattern that can reused by different types of argument. The patterns are supported by the strutured design method from [4]. The general idea is that there are a very useful methods used in Prolog editors which can give as some good ideas for methods for LCC. However, we cannot use those methods in LCC directly because there are fundamental differences between LCC and Prolog.

Techniques editing [12] is a method used to synthesize Prolog clauses, which cannot used directly in LCC because LCC is a process calculus but where we can adapt using structural patterns to the LCC case. This approach of using LCC

patterns in argumentation seem to reduce the effort of building argument protocols quite well. In this section we will identify these patterns by using car safe example and Figure 2 as an example of DID which demonstrates the different type of LCC argument patterns.

Two patterns have been defined: Starter pattern (S-P) and Termination Recursive Pattern (T-R-P). The first one S-P is going to be used to start the argument between two agents (Z and Q) and the second one T-R-P is going to be used in other arguments.

First pattern: Starter pattern (S-P)

Let we consider the first role in agent Z and agent Q which are startClaimZ and startClaimQ.

a(startClaimZ(KBZ,CSZ),IDZ)::
claim(X) ⇒ a(startclaimQ(KBQ,CSQ),IDQ) ⟵ AddInCS(X, CSZ)
then
a(replyToClaimZ(KBZ,CSZ),IDZ).
a(startclaimQ(KBQ,CSQ),IDQ)::
claim(X) ⇐ a(startClaimZ((KBZ,CSZ),IDZ)
then
a(replyToClaimQ(X,KBQ,CSQ),IDQ).

From this example we can see that first role sends a locution and then changes its role, the second role receives a locution and then changes its role. We can consider that as pattern where the general idea is that both agents send/receive message and then change their roles so as is remain in dialogue.

a(RZ1(KBZ,CSZ),IDZ)::
SL(X) ⇒ a(RQ1(KBQ,CSQ),IDQ) ⟵ C1(X, CSZ)
then
a(RZ2 (KBZ,CSZ),IDZ).
a(RQ1(KBQ,CSQ),IDQ)::
SL(X) ⇐ a(RZ1(KBZ,CSZ),IDZ)
then
a(RQ2(KBQ,CSQ),IDQ).

where SL represents the starter locution and C1(X, CSZ) represents unspecified conditions between X and CSZ (Agent commitment store)

Transfer from Dialogue Interaction Diagram (DID) to Start pattern

1. The first step in the process of creating LCC protocol is to determine starting locution (SL). As we can see from Figure 2 SL could be either claim or argue. In the car safety example, SL = claim (which appears at the top of figure 2).
2. The next step is to apply the Starting Pattern. Before doing that, the system has to replace RZ1 with startClaimZ1, RQ1 with startClaimQ1, RZ2 with replyToClaimZ and RQ2 with replyToClaimQ. To this effect, the system is ready to apply the Starting Pattern. This will be done by matching the SL to claim (step1), matching C1(X, CSZ) to the acquire condition which represented in Rhombus shape in DID (step2). In car safe example, SL = claim, from DID

we can see that the acquire condition is addInCS(CS,X). So the systems will match C1(X, CSZ) to addInCS(X, CS)

Second Pattern: Termination Recursive Pattern (T-R-P)

a(replyToClaimQ(X,KBQ,CSQ),IDQ)::
concede(X) $\Rightarrow$ a(replyToClaimZ (KBZ,CSZ),IDZ) $\longleftarrow$ FindInKBorCS(X,CSQ,KBQ) and AddInCS(X, CsQ)
or
(why(X) $\Rightarrow$a(replyToClaimZ (KBZ,CSZ),IDZ)
then
a(replyToWhyQ(X,KBQ,CSQ),IDQ)).

a(replyToClaimZ(KBZ,CSZ),IDZ)::
concede(X) $\Leftarrow$ a(replyToClaimQ (X ,KBQ,CSQ),IDQ)
or
(why(X) $\Leftarrow$ a(replyToClaimQ (X ,KBQ,CSQ),IDQ)
then
a(replyToWhyZ (X ,KBZ,CSZ),IDZ)).

If we look at any other roles in the car safe example such as replyToClaimQ dReplyToClaimZ, we can see that the first role sender sends a locution to terminate the argument or sends a permitted locution and then recourses. The second role receives a locution which intends to terminate the argument or receives a permitted locution and then recurses. We can use this as a general recurseve skeletons. Skeleton describe the basic control flow of LCC argumentation clauses then we can develop the basic clause further by rewriting it to build more complex programs.

a(RZ1(KBZ,CSZ),IDZ)::
RZ1 $^{TL}\approx\succ$ RQ1
or
RZ1 $^{RL}\approx\succ$ RQ1

a(RQ1(KBQ,CSQ),IDQ)::
RQ1 $\prec\approx^{TL}$ RZ1
or
RQ1 $\prec\approx^{RL}$ RL RZ1

This Skeleton represents a generic recursive clause. The variable R in the definition above represents the role name. KB, CS are the role argument and ID is the agent identifier.TL represent Termination Locution (such as retract and concede) and RL represents a Recursive Locution (such as argue, why and claim). ' $\approx\succ$' represents outgoing messages from a role, and ' $\prec\approx$' represents incoming messages.

Rewriting of Termination Recursive Skeleton

First: Rewriting of " RZ1 $^{TL}\approx\succ$ RQ1"

The main function of rewriting is to allow generic relations between R Z1 and R Q1 to be rewritten into a specific way. There might be a direct or so complex or

indirect relation between them. If there is a general relation of " RZ1 $^{TL}\approx\succ$ RQ1" then it is possible to specialize within 3 different statements:

Rewrite 1:

We might specialize " RZ1 $^{TL}\approx\succ$ RQ1" to an interaction statement that sends TL (X) Termination message to agent IDz which is done under C1(X, KBQ,CSQ) and C2(X,CSQ) which represents condition between X, CS and KB.

TL (X) $\Rightarrow$ a(RQ1(KBQ,CSQ,CSZ),IDQ) $\longleftarrow$ C1(X, KBQ,CSQ) and C2(X,CSQ)

Rewrite 2:

We might specialize " RZ1 $^{TL}\approx\succ$ RQ1" to an interaction statement that sends TL (X) Termination message to agent IDz which is done under C1(X, KBQ,CSQ) and C2(X,CSQ) . Then, there is another termination relation between RZ1 and R Q1.

TL (X) $\Rightarrow$ a(RQ1(KBQ,CSQ,CSZ),IDQ) $\longleftarrow$ C1(X, KBQ,CSQ) and C2(X,CSQ)
or
RZ1 $^{TL}\approx\succ$ RQ1

Rewrite 3:

We might specialize " RZ1 $^{TL}\approx\succ$ RQ1" to an interaction statements that receive L(-) message from agent IDQ and then sends TL (X) Termination message to agent IDz which is done under C2(X, KBQ,CSQ) and C3(X,CSQ).

L(-) $\Leftarrow$a(RQ1(KBQ,CSQ,CSZ),IDQ) $\longleftarrow$ C1(X,CSQ)
then
TL (X) $\Rightarrow$ a(RQ1(KBQ,CSQ,CSZ),IDQ) $\longleftarrow$ C2(X, KBQ,CSQ) and C3(X,CSQ)

Second: Rewriting of " RZ1 $^{RL}\approx\succ$ RQ1 "

Rewrite 1:

We might specialize "RZ1 $^{RL}\approx\succ$ RQ1" to an interaction statement that sends message RL (X) to agent IDz which is done under C5(X, KBQ,CSQ) and C6(X,CSQ). Then it recurses.

(RL(X) $\Rightarrow$ a(RQ1(KBQ,CSQ,CSZ),IDQ) $\longleftarrow$ C5(X, KBQ,CSQ) and C6(X,CSQ)
then
a(RZ2 (KBZ,CSZ,CSQ),IDZ))

Rewrite 2:

We might specialize "RZ1 $^{RL}\approx\succ$ RQ1"to an interaction statement that sends message RL (X) to agent IDz which is done under C5(X, KBQ,CSQ) and C6(X,CSQ) and then it recurses. After that there is another Recursive relation between R Z1 and R Q1.

(RL(X) $\Rightarrow$ a(RQ1(KBQ,CSQ,CSZ),IDQ) $\longleftarrow$ C5(X, KBQ,CSQ) and C6(X,CSQ)
then
a(RZ2 (KBZ,CSZ,CSQ),IDZ)
or
RZ1 $^{RL}\approx\succ$ RQ1

Rewrite 3:

We might specialize "RZ1 $^{RL}\approx\succ$ RQ1"to change the role to RZ2 which is done under C5 (X, KBQ,CSQ).

a(RZ2 (KBZ,CSZ,CSQ),IDZ) $\longleftarrow$ C5 (X, KBQ,CSQ).

Transfer from DID to Termination Recursive Pattern

1. In order to apply the Termination Recursive Pattern, the system has to replace RZ1 with replyToClaimZ and RQ1 with replyToClaimQ.
2. As we can see from Figure 2 the legal move after claim could be either concede or why. Concede consider as Termination Locution and why as Recursive Locution, as shown in dot rectangles in Figure 2. To this effect, the system will match TL to concede and RL to why.
3. The next step after applying T-R pattern is the incremental rewriting of the clause:
   (a) Since we have only one TL the system will apply the first rewriting method of " RZ1 $^{TL}\approx\succ$ RQ1" by matching TL to concede (step1) and matching C1 (X, KBQ,CSQ) to the acquire condition " FindInKBorCS(X,CS,KB)" which represented in Rhombus shape in DID (step2).
   (b) Since we have only one RL the system will apply the first rewriting method of "RZ1 $^{RL}\approx\succ$ RQ1" by matching RL to why (step1) and replacing RZ2 with replyToWhy. (step2).

7 Conclusion and Future work

The contribution of this paper is to show by example how a form of structured syntheses from basic logic programming (techniques editing) can be applied to the problem of syntheses of argumentation protocols. A key insight in doing so is that the sort of specification used as the high-level language in the Argumentation Interchange Format (an example of a domain- specific specification language)can be used to drive the automatic synthesis algorithm.

There are a two interesting future directions in this area. Firstly, we have illustrated our proposal in the context of persuasion dialogue but in future, we want to investigate other kinds of dialogue, such as negotiation and deliberation dialogues. Secondly, our current approach is limited to two agents(which is the norm for argumentaion). In future, we aim to extend the work to handle a negotiation among N agents.

References

[1] Chesnevar, C., McGinnis, J., Modgil, S., Rahwan I., Reed, C., Simari, G., South, M., Vreeswijk, G., Willmott, S. : Towards an argument interchange format. The Knowledge Engineering Review 21(4), pp. 293–316, (2007)

[2] Willmott, S., Vreeswijk, G., Chesnevar, C., South, M., McGinnis, J., Modgil, S., Rahwan I., Redd C., Simari G. :Towards an Argument Interchange Format for Multi-Agent Systems, In Proceedings of the 3th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS2006), (2006)

[3] Robertson D.:Multi-agent coordination as distributed logic programming. In "Logic programming" 20th International Conference, Proceedings, vol. 3132 of Lecture Notes in Computer Science, pp. 416–430, (2004)

[4] Grivas A.: A Structural Synthesis System for LCC Protocols, PhD thesis, University of Edinburgh, (2005)

[5] Prakken, H.: On dialogue systems with speech acts, arguments, and counterarguments. In: Brewka, G., Moniz Pereira, L., Ojeda-Aciego, M., de Guzman, I.P. (eds.) JELIA 2000. LNCS (LNAI), vol. 1919, pp. 224–238. Springer, Heidelberg ,(2000)

[6] Prakken, H.: Coherence and flexibility in dialogue games for argumentation. Journal of logic and computation vol. 15, pp. 1009–1040, (2005)

[7] Modgil S., McGinnis J. :Towards Characterising Argumentation Based Dialogue in the Argument Interchange Format. In Proceedings of the 4th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS2007), pp. 80-93, (2007)

[8] Robertson D.: Multi-agent Coordination as Distributed Logic Programming. University of Edinburgh,(2005)

[9] Hassan F., Robertson D., and Walton C.: Addressing Constraint Failures in Agent Interaction Protocol. Center for Intelligent Systems and their Applications, University of Edinburgh, (2005)

[10] D.N Walton and E.C.Krabbe. : Commitment in Dialogue. Basic concepts of interpersonal reasoning. State university of New York Press, Albany, NY, (1995)

[11] Prakken H.: Formal systems for persuasion dialogue, Department of Information and Computing Sciences, Universiteit Utrecht Centre for Law and ICT, Faculty of Law, University of Groningen The Netherlands, (2006)

[12] Bowles A., Robertson D., VasconcelosW., Vargas-Vera M., and Bental D. : Applying prolog programming techniques. International Journal of Human-Computer Studies, 41(3), pp. 329–350, (1994)