Generalising argument dialogue with the Dialogue Game Execution Platform

Floris BEX [a], John LAWRENCE [b], Chris REED [b]
[a] *Information and Computing Sciences, Utrecht University, The Netherlands*
[b] *School of Computing, University of Dundee, UK*

Abstract. In this paper, we present the Dialogue Game Execution Platform (DGEP), which is able to process and execute any dialogue game specified in a general description language and build arguments and dialogue histories in the language of the AIF ontology. Thus, DGEP allows us to generalise techniques for generation and investigation of dialogues and, through a set of web services, connect a wide variety of multi-agent systems and human-computer interfaces for dialogue.

Keywords. dialogue specification, dialogue execution, argument web services

1. Introduction

Dialogue games are attracting increasing attention both for their formal properties and connections to proof theoretic procedures [10], as well as for their ability to support, structure and evaluate dialogical processes [17]. Though there are a number of dialogue games for argumentation in the literature [8], and a number of general frameworks for representing games [5, 16], there are no general purpose mechanisms for handling high-level executable definitions of arbitrary games including all the usual features of realistic argument dialogue games, including commitment handling, turn-taking rules, attacks on dialogue participants or the dialogical construction of underlying argument structures. In order to fill this gap, we have developed a general framework for describing and executing dialogue games, with a further aim of providing its functionality as a robust, implemented platform accessible through a straightforward set of web services.

One of the difficulties of the research on argument dialogue systems is that it is fragmented: each dialogue system has its own definitions and is embedded in its own context, so it is difficult to specify, analyse and empirically compare the existing protocols. Work that does take a more generic [10] or comparative [9] approach does not concern itself with aspects of implementation and execution, and existing implementations [15, 17] build the protocol rules into the program itself, which means that the program will only ever be able to execute a single predetermined protocol.

A solution to the above-mentioned lack of flexibility is to design a language for specifying different protocols in a machine-readable way, so that the protocol specifications can be expressed separately from the program that executes them. Such specification languages have been designed for generic multi-agent dialogue protocols (e.g. LCC [12] and the FIPA ACL[1]). Extensions have been proposed for dealing with argumentation [8], and recently two languages aimed specifically at argumentation dialogues have been developed [16, 5]. However, what is still missing is a flexible execution platform, that is not only able to execute these dialogue protocol specifications, but also handles connections to software agents, argument structures, large knowledge bases and human-computer interfaces. Hence, argument dialogue systems have largely stayed within the confines of the academic lab, in contrast to more static applications for opinion and argumentation, which have seen relatively wide uptake for educational and commercial purposes[2].

In order to advance the theoretical study as well as the practical applicability of dialogue games, we propose a generic and flexible approach which is able to capture and execute not only protocols from computational argumentation (e.g. [10, 9]), but also protocols from philosophy and communication theory (e.g. [14]). At the heart of this approach sits the Dialogue Game Execution Platform (DGEP). DGEP is able to interpret dialogue game specifications expressed in an amended version of the Dialogue Game Description Language (DGDL) [16]. Based on a protocol specification, DGEP produces *dialogue templates* [3], schematic representations of a single move in a dialogue, its reply and the connections to the underlying argument structure in terms of the AIF ontology [7]. Dialogue histories with an explicit reply structure can be formed by combining multiple templates and thus existing argument structures in the Argument Web [1], which also adhere to the AIF ontology, can be navigated and updated using dialogues.

In section 5, we explain how DGEP executes a machine readable specification of a dialogue protocol (as defined in section 3), thus building formal dialogue histories and their underlying argument structures in the language of the AIF (section 4)

2. Dialogue games for argumentation

We start with a simple example to illustrate some of the concepts of dialogical argumentation that will be used throughout the paper. Dialogues consist of a series of locutions (utterances) made by the participants. As a simple example of a dialogue, take the following exchange between Bob and Alice on the UK's Trident nuclear missile programme:

1. Bob: Britain should stop the Trident Programme!
2. Alice: Why?
3. Bob: It is expensive!

In this dialogue, multiple participants make claims and pose questions. There is coherence to the way the participants react to one another, and the connections between locutions represent functional transition relations rather than, for example, temporal relations. That is, there is an (implicit) reply structure in the dialogue that contains the functional connections between the locutions in a dialogue. These connections can be informally interpreted as a 'responds to' relation between two locutions. The relationship between Alice asking 'Why [should Britain stop Trident]' (2) and Bob's response 'It is expensive' (3) is an example of such a functional relationship.

[1]`www.fipa.org/repository/aclspecs.html`

[2]As evidenced by the large number of users of visualisation software like Araucaria (`araucaria.computing.dundee.ac.uk`) and Rationale (`www.rationaleonline.com`).

During dialogue, participants (implicitly) construct and navigate an underlying argument structure [16][23] of claims and reasons. For example, in the above dialogue the argument '[Trident] is expensive *therefore* Britain should stop Trident' is made. Thus, we distinguish between arguments as a static structure of premises that are reasons for or against conclusions (as in, 'he prepared an argument') and arguments as debates or dialogues (as in, 'they had an argument').

Dialogue and arguments are connected via the illocutionary force relations of the speech acts in the dialogue. A speech act can be analysed as a locution (the actual utterance, e.g. 'Why?'), but also as an illocutionary act which consists of the illocutionary force (the intention of uttering a locution: one may say p with an intention of asserting p, asking p, challenging p, promising p and so on) and the propositional content, the proposition(s) the act refers to. In our example, speech acts (1) and (2) have the same propositional content – 'Britain should stop Trident' – but differing illocutionary force – asserting and challenging, respectively. Furthermore, relations between propositions in argument structures can also be generated by, or *anchored in*, transitions between one locution and the next in a dialogue [11]. Take, for example, Alice's question (connected to its propositional content) and Bob's assertion (also connected to its propositional content). When imagined in isolation, even as occurring in different dialogues then, ceteris paribus, there would be no link between 'Trident is expensive' and 'Britain should stop Trident'. It is only in virtue of the fact that Bob's assertion of Trident's high cost is responding to Alice's challenge of 'Britain should stop Trident' that there is an inferential link here. Hence, the relationship between (2) and (3), which captures the notion of responding, has as its illocutionary force 'arguing' and as its content the inferential relationship from 'Trident is expensive' to 'Britain should stop Trident'.

3. The Dialogue Game Description Language

The domain specific language DGDL (Dialogue Game Description Language) was presented in [16] and, after having been extended, refined and simplified, is now being released as DGDL+[3]. We will not discuss the entire specification but rather an example of a dialogue protocol expressed in DGDL+ (Specification 1). This example presents a slightly simplified version of Walton's CB game [14].

Lines 1-6 define general characteristics of the game. A player can make a single move per turn, there is a strict progression of alternating moves and the maximum number of turns is defined by the user (line 2). There are always 2 players, identified as `black` and `white` (line 3). These players can play the role of a speaker, a listener or (ultimately) a winner during the dialogue (line 4). Each player has a commitment store (`CS`, lines 5, 6) which is visible to all.

Protocol rules that are not tied to a specific move in the dialogue, such as rules for initiation, termination and winning, are defined using `rules`, which check the game state at set stages in the dialogue – at the start of the dialogue (`initial`), after every turn (`turnwise`) or after every move (`movewise`). The `interaction` elements of a specification define the types of speech acts that can be performed in the dialogue. Both `rules` and `interactions` contain at least one element that states the effects

[3]The complete DGDL+ specification and protocols are available at `www.arg.dundee.ac.uk/dgdl`

Specification 1 The simplified CB protocol [14] expressed in DGDL+

```
CB{
{turns, magnitude:single, ordering:strict, max:$UserDefined$} ;
{players, min:2, max:2}; {player, id:black}; {player, id:white};
{roles, speaker, listener, Winner} ;
{store, id:CS, owner:black, structure:set, visibility:public} ;
{store, id:CS, owner:white, structure:set, visibility:public} ;

{transforce, {<challenge, {p}>}, {<statement, {q}>}, arguing,
              {<q, p>, Inference} }

{rule, StartingRule, scope:initial,
 {assign(black, speaker) &
  move(add, next, statement, {p},
       {inspect(in,{p},CS,black,initial}})}};

{rule, SpeakerWins, scope:turnwise,
 {if {foreach {p, {CS,speaker,initial},
                  extCondition(Conseq{p, CS, listener, current})}}
   then {status(terminate,CB) & assign(speaker, winner)}}} ;

{rule, ListenerWins, scope:turnwise,
 {if {foreach {p, {CS,listener,initial},
                  extCondition(Conseq{p, CS, speaker, current})}}
   then {status(terminate,CB) & assign(speaker, listener)}}} ;

{interaction, statement, asserting, {p}, "State",
 {store(add, {p}, CS, speaker) &
  move(add, next, statement, {q}) &
  move(add, next, challenge, {p}, inspect(!in,p,CS,listener)) &
  foreach{q, {CS,listener}, move(add, next, Withdraw, {q},
           extCondition(NotConseq{q, CS, listener})}}};

{interaction, challenge, challenging, {p}, "Why?",
 {move(add, next, statement, {q}, extCondition(Conseq,{{q},{p}})&
  move(add, next, Withdraw, {p}}}};

{interaction, withdraw, withdrawing, {p}, "No commitment",
 {store(remove, {p}, CS, speaker) &
  move(add, next, statement, {q}) &
  foreach{q, {CS,listener}, move(add, next, Withdraw, {q},
           extCondition(NotConseq{q, CS, listener})}}};
}
```

of applying the `rule` or `interaction` uttered, and zero or more requirements, which capture the conditions for the effects to be applicable.

The effects of a `rule` or an `interaction` can be of several types. Common ones are store operations (`store`), which define manipulations on the contents of commitments stores, role updates (`assign`) to indicate changes of roles and status updates, which indicate changes in global dialogue status (`status`). The most important type of effect is indicated by the `move` predicate, which adds moves to a set of *legal moves* for one or more of the players. These moves then become either mandatory at the `next` turn (challenges must be met immediately, for example), or possible at some point in the future (a claim can be questioned at some point, for example).

The predefined requirements can be used to, for example, the presence or absence of elements within a commitment store (`inspect`) or whether the current (or some other) player is in a particular role. In addition to these standard requirements, there exist an essentially limitless number of properties of the dialogue or the underlying argument structure that can be checked or calculated. For example, many games (including CB) include a limit on the number of turns in a dialogue. Requirements can also be based on comparisons – for example, that the number of moves of one party is more than that of another party. And they need not be numerically based – there may be constraints based on string manipulations (that a textual contribution is fixed at a number of characters, say), or logical properties (e.g., that an utterance should be outside the deductive closure of what has already been said) or argument properties (e.g. that an utterance must address an argument which is undefeated under grounded semantics), and so on. Clearly DGDL+ cannot anticipate all the possible requirements of a dialogue that may be pertinent, lest it turn into a general purpose programming language. DGDL+ therefore provides a small number of requirements that commonly appear in the literature as inbuilt predicates (i.e. commitment store check, role check), and a general purpose predicate `extCondition` is provided with which arbitrary functions not defined in the protocol specification can be indicated. Note that the `requirements` in an `interaction` are not the requirements for moving *that* interaction. Rather, the requirements tell us which conditions need to be satisfied for *another* interaction to be added to the legal moves set.

In the CB specification, `StartingRule` (lines 11-14) is the initial rule: `black` is assigned as the first speaker and that just one move is possible, namely `statement`. Note the extra requirement is enclosed at the end of the `move(add,...)` structure: black can only state something that is in his `initial` commitment set. Further note that it is not indicated for which player the move is added to the set of legal moves: the strict turn order combined with the fact that moves that are added are always mandatory in the next turn means that this is clear from the specification.

`SpeakerWins` (lines 16-19), resp. `ListenerWins` (lines 21-24) check every turn whether all the elements that were in the speaker's (listener's) initial commitment set follow from the listener's (speaker's) current commitment set; if this is the case, the speaker (listener) wins and the dialogue terminates. Notice how DGDL refers to some the external condition `Conseq`. Clearly, specifying how to compute whether some `p` is a (logical) consequence of some `q` lies outside of the scope of DGDL, so this is referred to an outside function.

Like all elements of DGDL+, `interactions` have a standard form. The second element of an `interaction` indicates the name of the interaction (`statement`), the third element indicates its illocutionary force (`asserting`), the fourth element indicates the content of the speech act (a proposition `p`) and the fifth element is gloss, giving an informal indication of the utterance (`'State'`). The rest of the interaction contains the rules for what should be done after a player uses the interaction in the dialogue.

When a player makes a `statement p`, (lines 26-31) `p` is added to the commitment store of both the speaker, and the listener (i.e. the next player to move) can perform one of three moves: they can make a new `statement q`, they can challenge `p` as long as `p` is not in their commitment set, or they can `withdraw` some `q` from their commitment store (as long as `q` does not follow from their commitment store).

A `challenge p` move (lines 33-35) forces the next player to either `withdraw p` or provide a reason `q` for `p`, that is, state some `q` of which `p` is a consequence. A `withdraw p` move (lines 37-41) deletes `p` from the speaker's commitment set and can be followed by a (possibly unrelated) `statement` or `withdraw` move from the next player.

The only part left to explain of the example protocol is the idea of transitional illocutionary force, or `transforce`. This element of a specification defines what the illocutionary force and content of a reply of one `interaction` to another `interaction` in a dialogue is. In the example protocol (lines 8,9), replying to a `challenge p` with a `statement q` has the illocutionary force of `arguing` and as its content an `Inference` from `q` to `p`.

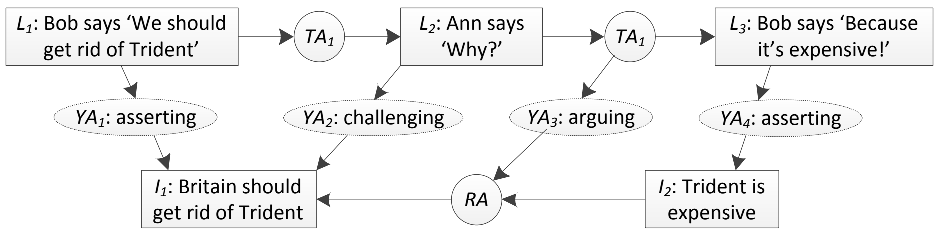
4. AIF graphs

The Argument Interchange Format (AIF) [7] is an abstract core ontology that encapsulates the common subject matter of the various (logical, linguistic, graphical) approaches to argumentation. AIF acts as an interlingua between different theoretical and practical approaches to argumentation which allows, for example, arguments constructed in visualization packages such as Araucaria to be evaluated using the various argumentation theoretic semantics that are available [2]. The AIF also serves an important practical purpose as the underlying language of the Argument Web [1], a linked data Semantic Web structure containing more than 15,000 claims and 1500 arguments with different (inferential, conflict, illocutionary) relations between them. This Argument Web is implemented in a number of machine readable formats (DOT, RDF/XML, SQL, Prolog) and can be interacted with in a number of different ways; existing and bespoke tools are available for text annotation, visualisation, search, evaluation, social media and dialogue[4].

Whereas DGDL+ makes it possible to express protocol rules for dialogue games, we also need to be able to express structures of actual argument and dialogue. More specifically, we need a representation of a dialogue history or trace. The AIF's graph-theoretic basis allows dialogue histories and their underlying arguments to be represented as *AIF argument graphs*. The ontology places a distinction between *information*, the propositions and sentences, and *schemes*, general patterns of reasoning. Accordingly, there are two basic types of nodes: I-nodes, to represent information, and S-nodes, to denote applications of schemes, the relations between the information nodes. Rule application nodes (RA-nodes) denote applications of an inference rule and conflict application nodes (CA-nodes) denote specific conflicts. In Figure 1, which shows the history of our example dialogue and the corresponding argument structure as an AIF graph, there are two standard I-nodes, I_1 and I_2, and one RA-node representing that I_2 is inferred from I_2.

The original AIF specification [7] has been expanded to capture dialogue [11, 3]. Locutions are represented by L-nodes, which are a special type of I-node, and these L-nodes are connected by transition application nodes (TA-nodes), which represent the functional connections between locutions. Note that, where RA-nodes represent applications of inference rules or schemes, TA-nodes are also based on general *transition schemes*. For example, replying to a 'Why *p*' challenge by giving a reason for *p* is a typical pattern one expects to encounter in an argumentative dialogue and can hence be captured by a scheme. Figure 1 shows the history the dialogue: $L_1 – TA_1 – L_2 – TA_2 – L_3$. Note that while this example is a simple chain, dialogues that allow for backtracking (i.e., replying to another locution than the last) will lead to more tree-shaped reply structures.

[4]`argumentinterchange.org/library`



L_1: Bob says 'We should get rid of Trident'
TA_1
L_2: Ann says 'Why?'
TA_1
L_3: Bob says 'Because it's expensive!'
YA_1: asserting
YA_2: challenging
YA_3: arguing
YA_4: asserting
I_1: Britain should get rid of Trident
RA
I_2: Trident is expensive

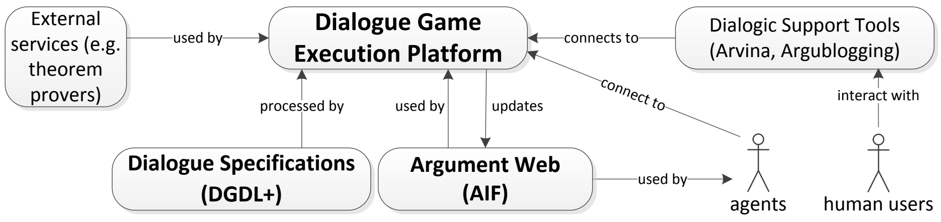
Figure 1. A dialogue (top) and argument (bottom) as an AIF graph, connected by illocutionary force

The last type of relation that needs to be discussed is the illocutionary relation between L-nodes and I-nodes. This relation can be of a number of types, depending on the type of illocutionary force, and thus capture a specific type of linguistic relation. As a result, we refer to these rules as illocutionary schemes or Y schemes, and specific applications of these schemes are represented by YA-nodes. In the dialogue (figure 1), the *asserting* and *challenging* relations between the L- and I-nodes are shown. Notice also the response of L_3 to L_2, captured by TA_2, which has an illocutionary force of *arguing* and as its content the inference rule application captured by the RA-node.

It is usual for the dialogue history to change only monotonically as a dialogue proceeds (that is, once something has been said it cannot be unsaid), and AIF graphs are therefore monotonic (i.e., dialogical updates will not remove material from the graph). However, in addition to dialogue history we also need a representation of the (shared) information stores of the participants, such as commitments or knowledge bases. In representation of these stores, monotonicity is less common: we would not wish to prohibit retraction or withdrawal. So the specific information stores as defined in the DGDL+ specification of a dialogue cannot be directly captured in the AIF graph and need to be defined separately in DGEP (see section 5).

5. Dialogue Game Execution Platform

The Dialogue Game Execution Platform, DGEP, aims to execute games specified in DGDL+ (section 3), thus building AIF graphs (section 4) and expanding the Argument Web. Figure 2 shows the interactions between DGEP, its users and its support services.



External services (e.g. theorem provers)
used by
Dialogue Game Execution Platform
connects to
Dialogic Support Tools (Arvina, Argublogging)
connect to
processed by
used by
updates
interact with
Dialogue Specifications (DGDL+)
Argument Web (AIF)
used by
agents
human users

Figure 2. The Dialogue Game Execution Engine and its associated systems and users

5.1. DGEP execution cycle

DGEP processes a DGDL+ specification rather as if it were a compiler[5]. DGEP then starts an execution with a set of participants and a given starting claim. The participants may be human or software, and DGEP makes no distinction (that is, it implements the ideal of *mixed initiative argumentation*). Many games also require initial specification and initialisation of various parameters (e.g., the number of turns permitted) and information states (e.g., the players' starting commitment stores or knowledge bases), so DGEP also processes these in the initial phase. The connection between agents and the Argument Web means that agents can optionally use (a part of) the Argument Web as their knowledge base.

From the initial turn onwards, DGEP develops the *legal move list* for each participant. It does this by firing `rules` and `interactions`: a `rule` fires according to its `scope`, and an `interaction` when it is moved by a player during the game. When a `rule` or an `interaction` fires, DGEP first checks whether the possible `requirements` are true. Most of the standard requirements (e.g. involving commitment stores, number of turns, player roles) can be checked against the parameters and stores initialised in DGEP. For external conditions (`extCondition`) a general purpose interface is provided by which arbitrary functions, provided as web services which return Booleans, can be called by DGEP. For example, in a run of the CB game, an external theorem prover can be called to find out whether some `p` is a consequence of some `q`.

Once all `requirements` of a rule can be confirmed, DGEP instantiates the `effects`, possibly binding variables with, for example, matches from the store. As mentioned in section 3, the `effects` may be store operations, role and status updates and, perhaps most importantly, move availability. A given set of `effects` may give rise to many possible updates to the list of legal moves available for a given participant, indicating moves that become either mandatory or possible at some point in the future.

For all the legal moves at a given turn, there may be many instantiations (depending on, for example, the commitments and knowledge base of the participants). DGEP generates all of these moves and delivers them to participants (that is to say, to autonomous agents and to graphical interfaces used by humans) as a 4-tuple consisting of a `moveID`, `opener` (an informal indication of the utterance), `reply` (the formal structure of the move) and `aif` (a fragment of AIF that corresponds to the move). The fragment of AIF informs the agent as to the structure of his move in terms of the AIF, and thus gives an agent the correct structure for, for example, easily creating queries on AIF (recall that software agents use the Argument Web as a knowledge base from which they can extract arguments that constitute appropriate instantiations of the required move). However, the agent may very well choose to disregard this AIF and call on another knowledge base.

Finally, a participant selects an appropriate move from its legal move list and executes it by passing the move identifier back to DGEP. Artificial agents need to do some sort of processing to select between alternative legal moves – dialogues strategies have not yet been implemented so currently this selection is random. Note participants' replies are handled asynchronously both because of increased robustness and because some di-

[5]Specifically, an ANTLR (`http://www.antlr.org`) generated parser is used to convert DGDL+ to an Abstract Syntax Tree (AST). The AST produced is then converted to a Python data structure representing the hierarchy of elements within the DGDL+ specification.

alogue games do not impose rigid turn taking rules (those in which interruptions are permitted, and those, such as auctions, where many agents take on identical roles).

In addition to updating the legal move list for participants, the execution of a move also updates AIF structures. A theoretical account of how move executions update AIF structures was first discussed in [3], where we introduced the idea of a dialogue game specification expressed as *dialogue templates*. Dialogue templates are chunks of uninstantiated AIF representing a single transition relation (reply) in a dialogue, including the *start* (replied to) and *end* (replying) locutions of the transition, the type of illocutionary force of both locutions (and possibly of the transition) and the argument structure that the locutions refer to. Templates can be instantiated to form transitions (i.e. a step in a dialogue), and these transitions can then be chained to form a dialogue history, an AIF graph. As an example of dialogue templates, consider figure 3, which shows two templates based on the CB specification. The elements inside the dotted box have to be present in the Argument Web for the template to be applied; the elements outside the dotted line will be added to the Argument Web after the application of the template. Template A represents replying to an asserting move with a challenge, and template B shows how one can reply to a challenging move by asserting something, and how this reply is an instance of arguing. Note how instantiating these two templates with the appropriate information and chaining them together gives us the example from 1.

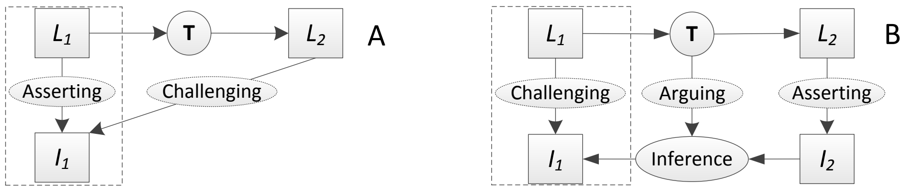


Figure 3. Dialogue templates for capturing updates to the Argument Web

Dialogue templates show how a dialogue game specification can be represented as a series of explicit reply structures (cf. [10, 9]). DGEP implements the idea of dialogue templates in its own way. The *start* locutions of a template (i.e., the elements inside the dotted box in figure 3) are in the Argument Web. The *end* locutions of the template, those that will be added on execution of the move, are captured by the respective DGDL+ `interaction`, which defines the illocutionary force and content. The (possible) illocutionary force of a transition is expressed by the `transforce` structure in the specification. Whenever a move is executed in reply to another move, DGEP first checks the Argument Web for the presence of the *start* locutions (with their associated illocutionary force and content). It then applies the dialogue templates, adding the *end* locution, the illocutionary force and propositional content (as defined by the `interaction` the AIF structure.

5.2. DGEP API

DGEP provides a simple set of interfaces allowing clients for both autonomous agents and human interfaces to connect and play instances of dialogue games. In order to provide an open platform, and to be consistent with other AIF and linked data, Semantic Web components which can be pipelined [1], these interfaces are offered as web services. DGEP offers a number of web service interactions:

- `/available` returns a list of protocols corresponding to the DGDL+ files that are available within the system. For example, this service currently returns a list of nine games: `{"dgdl": ["RPD", "CB", "mm", "IMPACT", "TDG", "DC", "Prakken", "PPD", "Lorenzen"]}`
- `/dialogue/new/<PROTOCOL>` is used to initiate a dialogue according to a given protocol. When this occurs, DGEP communicates with the Argument Web to create a new NodeSet, which will later be used to store the nodes created in the course of this particular dialogue. DGEP also makes an entry in its own internal database at this stage, recording the protocol being used, the AIFdb NodeSetID and a dialogueID, that will be used for all future interactions in this dialogue.
- `/dialogue/<DIALOGUEID>/roles` returns a list of the roles available in the dialogue. In this case DGEP's internal database is also consulted to determine any roles which may already have been filled.
- `/dialogue/<DIALOGUEID>/join/<ROLE>` allows for a new user joining a dialogue. The client application specifies the dialogueID for the dialogue that they are joining, and the role which they wish to have within the dialogue. DGEP again stores this information in its local database and generates a participantID that can then be used to retrieve the user's available move list as well as to make a move.
- `/dialogue/<DIALOGUEID>/moves` and `/dialogue/<DIALOGUEID>/moves/<PARTICIPANTID>` return available legal moves globally or for a specific participant.
- `/dialogue/<DIALOGUEID>/interaction/MOVEID` allows a participant to make a specifically identified legal move.
- `/dialogue/<DIALOGUEID>/transcript` and `/dialogue/<DIALOGUEID>/status` can be used by an application to get a list of moves carried out so far and the current status of the dialogue (w.r.t. participant roles, termination and so on).

These web service interactions allow a variety of interfaces to be connected to the execution engine. For example, Arvina[6] [1] allows users to play dialogue games against agents and each other in an instant-messaging environment. Another interaction style has recently been proposed in [4], where users can respond to online texts according to the chosen dialogue protocol, directly posting these responses on their blog.

6. Conclusions

In this paper, we have discussed the Dialogue Game Execution Platform (DGEP), a general approach to specifying, implementing and executing argumentative dialogue games. We have extended the specification language DGDL [16] into DGDL+. In addition to adding the necessary elements for illocutionary relations, the major change is that DGDL simpliciter defines pre- and postconditions for each `interaction`. This means that whenever a game is executed, at each new turn the execution engine will have to check all `interactions` in order to construct the *legal moves* list. DGDL+ checks requirements for the next possible move directly after the current move is played, which makes for a much more efficient execution process.

[6]`arg.dundee.ac.uk/arvina`

A wide variety of dialogue games can be specified in DGDL+; specifications exist for CB [14], [10]'s persuasion game, [9]'s games for inquiry, persuasion and information seeking, [15]'s game for value-based reasoning and a bespoke persuasion game developed for testing the Arvina interface. Further converting the multitude of dialogue games specified in DGDL [16] to the updated DGDL+ language is a straightforward task.

In addition to DGDL+, there are a number of other dialogue specification languages which are aimed at implementation. [8] present a simple extension for argumentation to the FIPA Agent Coordination Language, and [12] discusses the Lightweight Coordination Calculus, a high-level language for expressing coordination protocols in general. Though the general-purpose nature of FIPA ACL and LCC is a strength from a theoretical perspective, it forces one to build from scratch all the machinery used by the protocol, such as commitment stores and argument structures. As a result, though FIPA ACL and LCC provide an interesting overarching framework they are too lightweight for practical specification of dialogue games that use argumentation.

DGEP uses AIF, the language of the Argument Web, to express the dialogue histories generated by the execution of dialogue games. These dialogue histories capture not just the locutions that have been uttered, but also the underlying argument structures and the way in which they were updated by the locutions in the dialogue. This allows for a more fine-grained analysis of dialogical phenomena such as the way in which inference can be anchored in dialogue [11] and the ways in which the (credibility) of the participants of a dialogue can be attacked [6]. These dialogue histories can also serve as a denotational semantics for the DGDL+ language [8, 13]. Furthermore, by explicitly interpreting DGEP dialogue updates as applications of generic dialogue templates specified in AIF, we can define how these dialogue templates serve as the operational semantics for the language DGDL+ [13]. Thus, we have made a first step towards a theory that allows us to consider the properties of dialogue games not just on a formal case-by-case approach but rather in the empirical class-of-systems approach that is popular in verification research.

A recent dialogue execution framework similar to DGEP is [5]'s Framework for Dialogical Argumentation (FDA). In FDA each protocol is specified as a number of *action rules*, similar to DGDL+'s `rules` and `interactions`, where an action rule has an antecedent that refers to the current state of the dialogue and a head that specifies the possible actions that can be undertaken on the next state of the dialogue. Whilst FDA allows for the definition of various protocol-specific predicates, certain other dialogue mechanics (such as private and public states and turn taking functions) are 'hard-coded' into the framework, which makes FDA somewhat less flexible.

FDA mainly focuses on the automatic generation of execution traces for the investigation of theoretical properties of simple protocols and leaves elements like dialogue participants, argument structures and human-agent dialogues untouched. DGEP, with its accessible web services and connection to the Argument Web, aims to get argumentative dialogue out of the lab and into practical use. Software agents can use the 1500+ arguments for playing dialogue games that in turn further update the Argument Web. In this way, the Argument Web provides a large knowledge base for realistic multi-agent dialogues, a knowledge base that is continually updated by exactly the mixed-initiative dialogue it supports. Second, the compatibility between DGEP and AIF means that any user interface or agent that uses the DGEP API is directly compatible with the wider Semantic Web, of which the Argument Web is also a part.

References

[1] F. Bex, J. Lawrence, M. Snaith, and C. Reed. Implementing the argument web. *Communications of the ACM*, 56(10):66–73, 2013.

[2] F. Bex, S. Modgil, H. Prakken, and C. Reed. On logical reifications of the argument interchange format. *Journal of Logic and Computation*, 23(5), 2013.

[3] F. Bex and C. Reed. Dialogue templates for automatic argument processing. In *Proceedings of COMMA 2012*, pages 266–377, 2012.

[4] F. Bex, M. Snaith, J. Lawrence, and C. Reed. Argublogging: An application for the argument web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2014. to appear.

[5] E. Black and A. Hunter. Executable logic for dialogical argumentation. In *Proceedings of ECAI*, pages 15–20, 2012.

[6] K. Budzynska and C. Reed. The structure of ad hominem dialogues. In *Proceedings of COMMA 2012*, pages 410–421, 2012.

[7] C. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316, 2006.

[8] P. Mcburney and S. Parsons. Dialogue Games for Agent Argumentation. In I. Rahwan and G. Simari, editors, *Argumentation in Artificial Intelligence*, chapter 22, pages 261–280. Springer, 2009.

[9] S. Parsons, P. McBurney, and M. Wooldridge. The mechanics of some formal inter-agent dialogues. In *Advances in Agent Communication*, pages 329–348. 2004.

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

[11] C. Reed, S. Wells, K. Budzynska, and J. Devereux. Building arguments with argumentation : the role of illocutionary force in computational models of argument. In *Proceedings of COMMA 2010*, 2010.

[12] D. Robertson. A lightweight coordination calculus for agent systems. In *Declarative agent languages and technologies II*, pages 183–197. Springer, 2005.

[13] R. M. Van Eijk, F. S. De Boer, W. Van Der Hoek, and J.-J. C. Meyer. A verification framework for agent communication. *Autonomous Agents and Multi-Agent Systems*, 6(2):185–219, 2003.

[14] D. Walton. *Logical dialogue-games and fallacies*. University Press of America, Inc., Lanham, MD., 1984.

[15] M. Wardeh, A. Wyner, K. Atkinson, and T. J. M. Bench-Capon. Argumentation based tools for policy-making. In *Proceedings of ICAIL 2013*, pages 249–250, 2013.

[16] S. Wells and C. Reed. A domain specific language for describing diverse systems of dialogue. *Journal of Applied Logic*, 10(4):309–329, 2012.

[17] T. Yuan, D. Moore, C. Reed, A. Ravenscroft, and N. Maudet. Informal logic dialogue games in human-computer dialogue. *Knowledge Engineering Review*, 26(2):159–174, 2011.