Chapter 9

ProtOCL: Specifying Dialogue Games Using UML and OCL

Tangming Yuan[1] *and Simon Wells*[2]

[1] University of York, York, United Kingdom, tommy.yuan@york.ac.uk
[2] Edinburgh Napier University, Edinburgh, United Kingdom, s.wells@napier.ac.uk

Abstract. Dialogue games are becoming increasingly popular tools for Human-Computer Dialogue and Agent Communication. However, whilst there is an increasing body of theoretical underpinning that demonstrates the value and utility of dialogue games, and also a range of novel implementations within specific problem domains, there remain very few tools to support the deployment of dialogue games based solutions within new problem domains. This paper introduces a new approach, called ProtOCL, to the specification of dialogue games. This approach adopts Unified Modeling Language (UML) and the Object Constraint Language (OCL) and enables the rapid movement from specification to deployment and execution. The dialogue game, DE, is used as an exemplar and is described using OCL to yield DE-OCL. Code generation is subsequently used to move from the DE-OCL description to executable code. This approach goes beyond existing description languages and their supporting tools by (1) using a description language that is familiar to a far larger user group, and, (2) enabling code-generation using languages and technologies that are current industry standards.

1. Introduction

Dialectics is a branch of philosophy that seeks to build models for "fair and reasonable" dialogue (Walton & Krabbe, 1995). A common approach within dialectics is to construct dialogue games such as those of Hamblin (Hamblin, 1970), Walton and Krabbe (Walton & Krabbe, 1995), Walton (Walton, 1998), and Mackenzie (Mackenzie, 1990). A dialogue game can be seen as a prescriptive set of rules, regulating the participants as they make moves in a dialogue. These rules legislate as to the permissible sequences of moves, and also as to the effect of moves on the participants' "commitment stores", a record of the player's positions with respect to the statements made thus far. Such dialogue games have received much

recent interest from people working in Human Computer Dialogue and in Artificial Intelligence, for example Bench-Capon and Dunne (Bench-Capon & Dunne, 2007), Reed and Grasso (Reed & Grasso, 2007), Rahwan and McBurney (Rahwan & McBurney, 2007), and Yuan et al.(Yuan, Moore, Reed, Ravenscroft, & Maudet, 2011).

A number of computerised dialectical systems have been designed. Grasso et al. (Grasso, Cawsey, & Jones, 2000) for example, outline a system designed to change the attitudes of its users in the domain of health promotion. Vreeswijk (Vreeswijk, 1995) has designed "IACAS", an interactive argumentation system enabling disputes between a user and the computer. Yuan et al. (Yuan, Svansson, Moore, & Grierson, 2007) have applied the argument game from Wooldridge (Wooldridge, 2002) and the abstract argumentation system of Dung (Dung, 1995) to the construction of a computational argument game called "Argumento". Argumento enables human-agent, agent-agent and human-agent to ex- change both abstract and concrete arguments (Yuan & Schulze, 2008) and has been adopted as the core for an arguing agents competition (Yuan, Schulze, Devereux, & Reed, 2008; Wells, Lozinski, & Pham, 2008). Ravenscroft and Pilkington (Ravenscroft & Pilkington, 2000) used a dialogue game framework to facilitate a "structured and constrained dialectic" which in turn aided the student in enhancing explanatory domain models in ways that led to conceptual development concerning the physics of motion. The framework has been implemented in a prototype system "CoLLeGE" (Computer based Lab for Language Games in Education). Empirical studies have shown the effectiveness of the dialogue game framework (Ravenscroft, 2000; Ravenscroft & Matheson, 2002). Mackenzies dialectical system named 'DC' (Mackenzie, 1979) has been used as the basis for developing a further system named 'DE' (Yuan, Moore, & Grierson, 2003) which has been used as the underlying model for a human-computer debating system (Yuan, 2004; Yuan, Moore, & Grierson, 2007, 2008). Recently, dialogue games have also been used to structure interaction between humans and intelligent agents in mixed initiative environment as demonstrated in the MultiAgent Argument Logic and Opinion (MAgtALO) systems (Reed & Wells, 2007) and between intelligent agents within a multi-agent system (Kalofonos et al., 2006). The systems we have outlined above face a distinct formal representation problem that is the representation of the structure of the protocol that governs the dialogue game as it unfolds (Yuan et al., 2011). We may, for example, build a system that is to use the DE model to argue about capital punishment, but how are we to store the rules of DE?

To date, computational dialectic systems have approached the problems by expressing dialogue models informally using plain or structured English and then hard-wiring them into the program structure by the developer of the system. Hard-wiring means that the game rules cannot be easily modified unless re-coded and the entire system rebuilt. This makes reuse impossible as the game rules cannot be formally specified, saved and subsequently interfaced by other systems. A formal means of specifying dialogue games is therefore needed. In this paper we report on an approach to the specification of dialogue games that we have named ProtOCL. This approaches uses the Unified Modeling Language (UML) to describe a generic

dialogue game consisting of the common core elements found across a range of dialogue games, and the Object Constraint Language (OCL) to express specific rules as UML annotations that enable the generic game to be made specific to a particular dialogue game.

The remainder of this paper is organised as follows. Section 2 provides literature reviews of different methods for specifying agent dialogue protocols. Section 3 argues and demonstrates the case of using of UML and OCL as an approach to specify dialogue games. Section 4 discusses how a dialogue game framework can be generated and interfaced by the dialogue game engine and agents. Section 5 concludes the paper and point out our intended future work.

2. Methods for Specifying Dialogue Protocols

This section reviews some of the methods from the literature that have been used to specify dialogue protocols. These generally fall into the following categories: (i) natural language, (ii) formal logical notation, (iii) diagrammatic, and, (iv) domain specific language (DSL).

Natural languages descriptions, such as the game DE, a simplified version of which is illustrated as follows:

Move Types

Assertions: The content of an assertion is a statement P, Q, etc. or the truth-functional compounds of statements: “Not P”, “If P then Q”, “P and Q”.

Questions: The question of the statement P is “Is it the case that P?”

Challenges: The challenge of the statement P is “Why P?”

Withdrawals: The withdrawal of the statement P is “no commitment P”.

Resolution demands: The resolution demand of the statement P is “resolve whether P”.

Dialogue Rules

R_{FORM} : Participants may make one of the permitted types of move in turn.

$R_{REPSTAT}$: Mutual commitment can only be asserted when a question or challenge is responded.

R_{QUEST}: The question P can be answered only by P, “Not P” or “no commitment P”.

R_{CHALL}: “Why P?” has to be responded to by either a withdrawal of P, a statement that the challenger accepts, or a resolution demands of the previous commitments of the challenger which immediately imply P.

$R_{RESOLVE}$: A resolution demand can be made only in situations that the other party of the dialogue has committed in an immediate inconsistent conjunction of statements, or he withdraws or challenges an immediate consequent of previous commitments.

$R_{RESOLUTION}$: A resolution demand has to be responded by either the withdrawal of the offending
conjuncts or confirmation of the disputed consequent.
$R_{LEGALCHALL}$: "Why P?" cannot be used unless P has been explicitly stated by the dialogue partner.

Commitment Rules

Initial commitment, CR_0: The initial commitment of each participant is null.
Withdrawals, CR_W: After the withdrawal of P, the statement P is not included in the move makers store.
Statements, CR_S: After a statement P, unless the preceding event was a challenge, P is included in the move makers store.
Defence, CR_{YS}: After a statement P, if the preceding event was Why Q?, P and If P then Q are included in the move makers store.
Challenges, CR_Y: A challenge of P results in P being removed from the store of the move maker if it is there.

Termination Rules

1. The game will be ended when a participant accepts another participants view.

Such descriptions are both popular and plentiful in the literature, are generally well organized but have drawbacks, most importantly from the computation perspective, failing to lend themselves to either immediate execution or automated evaluation. In a natural language description, the rules of the game are generally grouped into a limited number of categories that define the types of available move (locution rules), how the moves interact with each other (structural rules), how playing the moves affect the commitments of the players (commitment rules), and the circumstances under which the game comes to an end (termination rules). A strength of the natural language approach is that the resulting descriptions are expressive and are, to a degree, easily understood by developers. However natural language descriptions of game rules can lead to problems with ambiguity. This aspect is compounded when the aim is for computational use of the game as natural language specifications are generally not machine-readable so automated testing, deployment, and execution become a difficult problem.

Formal specifications use notations from mathematical or formal logics to represent the semantics of dialogue rules in a precise way. Notable examples of this approach are to be found in (Bodenstaff, Prakken, & Vreeswijk, 2006), (Prakken, 2005) (Brewka, 2001) and (Artikis, Sergot, & Pitt, 2007). Recent work has attempted to create more generic description formats that retain the rigorousness of the formal approach whilst providing a range of descriptive features that are closer to the problem domain, for example in (Wells & Reed, 2004), the typical moves of Hamblin-type dialectical games are characterised in terms of a limited number of states and updates. The rules of a complete game are then expressed by assembling

collections of moves in terms of pre- and post- conditions using a set-theoretic formal notation. For example the following set theoretic specification for the Hamblin-type game illustrates some of the pre-condition checks on commitment store content:

$C \in CSn$	Commitment C is currently in commitment Store *CS*
$C \notin CSn$	Commitment C is not currently in commitment Store *CS*

The following specification illustrates some of the post-conditions for commitment store alterations:

$CS_{n+1} = CS_n \cup \{C\}$	Commitment C is added to Commitment Store CS
$CS_{n+1} = CS_n \cap \{C\}$	Commitment C is removed from Commitment Store CS

The following example then uses the expressions from for the commitment store checks and updates to complete the pre- and post-conditions for the statement and withdrawal moves of Hamblin's game, H:

Statement(S_x) Pre: $\emptyset$
Post: $CP_{n+1} = CP_n \cup \{S_x\} \wedge CO_{n+1} = CO_n \cup \{S_x\}$

Withdrawal(S_x) Pre: $\emptyset$
Post: $CP_{n+1} = CP_n \setminus \{S_x\}$

Whilst this approach improves the specificity of the rules and leads to a reduced chance for ambiguity, this approach is difficult to communicate to developers who do not possess the necessary mathematical background required to understand the notation (cf. Sommerville, 2011).

A Domain specific language (DSL) provides an intermediate position between the natural-language and formal approaches. An aim of the DSL approach is to re-use the established language of the domain problem, e.g. language used by people working with dialectical games, so that developers have an intuitive understanding of the expressions in the language, but to confine the expressions to those that are legal according to a formal grammar. Thus protocols are expressed in a way that is executable, assuming that adequate tooling is created to support the language, and immediately comprehensible to those versed in the language of the problem domain. An example of this kind of approach can be found in the Dialogue Game Description Language (DGDL) (Wells & Reed, 2012) a DSL that is founded on an Extended Backus-Naur Form (EBNF) grammar[1] to support the description of

[1] https://github.com/siwells/DGDL/tree/master/grammar

syntactically correct and verifiable dialectical games. The language at the current stage of development, however, needs software tool support particularly in terms of user-facing (design) tools and execution "engines". The following is an example of a DGDL game description name "Simple":

```
Simple{
   {turns,magnitude:single,ordering:strict}
   {players,min:2,max:2}
   {player,id:Player1}
   {player,id:Player2}
   {store,id:CStore,owner:Player1}
   {store,id:CStore,owner:Player2}
   {Assert,{p},"I assert that",
      {store(add, {p}, CStore, Speaker),store(add, {p}, CStore, Listener)}
   }
}
```

In this example game a turn structure, two named players, and a commitment store for each player are defined. A single assert move is then defined which incurs commitment in both players commitment stores when it is played. This game is for purely illustrative purposes and is indicative of the features and descriptive character of DGDL descriptions.

There have been a variety of approaches to the diagrammatic description of dialogue protocols. For example, in the Toulmin Dialogue Game (TDG) (Bench-Capon, 1998) a state diagram is used to regulate the order of moves and assignment of roles within a TDG dialogue. Finite State Machines (FSMs) have long been used to define network protocols and have been widely used to model, analyse, and prototype distributed systems (Shen, Norrie, & Barthes, 2001). FSMs have also been used to describe conversation policies in multi-agent systems (Bradshaw, Dutfield, Benoit, & Woolley, 1997). UML sequence diagrams also provide a way to diagrammatically depict dialogue protocols. For example, Agent UML (AUML) (Odell, Bauer, & Parunak, 2001) extends the unified modeling language (UML) to model intelligent software agents and related agent-based systems. FIPA adopted this approach to specify agent communication protocols such as the Subscribe Interaction Protocol[2], which enables an agent to subscribe to messages from another agent with respect to a specific referenced object. The state machine and sequence diagram approach may be more suitable for simple communication protocols (Norman, Carbogim, Krabbe, & Walton, 2004) as they visualize the actually occurred sequence of communications. It is not clear how certain constraints and rules, for example the DE rule $R_{REPSTAT}$: Mutual commitment may not be asserted unless to answer a question or a challenge can be represented on the diagram.

In summary, natural language specifications are not machine readable and are subject to ambiguity, formal approaches are generally not user- and developer-

[2] http://www.fipa.org/specs/fipa00035/

friendly, diagrammatic approaches are suitable for more simple protocols but must be underpinned by some formal representation to enable them to be executable without additional work, and DSLs currently have insufficient tooling to support wide-scale popular adoption.

3. Specifying Dialogue Games Using UML

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems. The latest version, UML 2.0, supports 13 types of diagrams including class, sequence, and state diagrams and one language; the object constraint language (OCL). OCL is used to describe rules that apply to UML models and adds vital information to the model that cannot be otherwise depicted using diagrammatic means. OCL is formally defined, readable and writeable by both humans and a range of software tools, and is easy to use (O.M.G., 2012). This goes a long way towards satisfying the need for a developer friendly formal language for describing dialogue games.

To use UML to represent the rules of a dialogue game, a model that captures general properties of a dialogue game, such as that depicted in the UML class diagram shown in Figure 1, is constructed. The class diagram captures common terms in the dialogue game domain such as the: game, player, dialogue history, commitment store, turn, move, move content, proposition and inference. Each dialogue game has a thesis and two players, a proponent of the thesis and an opponent. Each game also contains a dialogue history that records the moves made by the players on a turn-by-turn basis. The size of the dialogue history is the total number of turns made by both players. Each turn has a unique number and a player may make one or more moves in one turn. Each move contains a move type and a move content, which could be a proposition or an inference. Each inference contains a set of data and a conclusion. The negative value of a proposition or inference can be retrieved on request. Each player has a commitment store that record the statements made or accepted during dialogue. The commitment stores are publicly inspectable so a player can also view their opponent's store. The internal structure of a commitment store can be flexible depending on individual games, e.g. to maintain separate lists of propositions or inferences. A proposition or an inference can be checked against a commitment store to see whether it is supported by others or by itself. The latter is useful for banning circular arguments in dialectical systems. While the class diagram specifies the generic terms used by dialogue games, OCL is required to annotate the class diagram in order to provide a full specification of a dialogue game. The description of DE as presented in 1 is used to demonstrate this. For example, the DE move types rule can be specified as

```
--Player makes a legal move
context Player::makeMove():Move
   --Permitted move types:
```

```
post: Set{'Assertion', 'Question', 'Challenge', 'Resolve', 'Withdrawal'}
   ->includes(result.getType())
```

The rule is specified as a post condition within the context of player make- Move operation. context, post and result are OCL keywords and includes is an OCL operation that applies to a set.

The DE dialogue rule RFORM can be specified as

```
--RFORM: Participants may make one of the permitted types of move in
turn.
context Turn
   inv: move->size()=1
context DialogueGame
   inv: self.proponent.turn->forAll(getNumber()/2=1) and self.opponent.turn
      ->forAll(getNumber()/2=0)
```

The rule is specified jointly within the context of Turn and DialogueGame class as two invariants: the first is that the set of moves associated with each turn is exactly one and the second is that the turn numbers for the proponent are odd numbers and for the opponent are even numbers given that the proponent always starts a game. 'inv', 'self', and 'and' are all OCL keywords and size is an OCL operation that applies to a set.

The DE commitment rule CR0 can be specified as

```
--Initial commitment, CR0: The initial commitment of each participant is
null.
context DialogueGame::start():String
post: proponent.store.content->isEmpty() and
     opponent.store.content->isEmpty()
```

The rule is specified within the context of dialogue game start operation as post conditions. isEmpty is an OCL operation that applies to a set.

The DE termination rule can be specified as

```
--Termination Rules: The game will be ended when a participant accepts the
other participant's view.
context DialogueGame::end():String
   pre:proponent.store.content->includes(thesis.getNegation()) or
      opponent.store.content->includes(thesis)
--Playing
context DialogueGame::play():String
  pre: proponent.store.content->excludes(thesis.getNegation())and
     opponent.store.content->excludes(thesis)
```

The precondition for a dialogue game to end is that one party's store contains the opponent's thesis. Otherwise, the game is in the playing state.

4. Automatic Generation of Dialogue Game Framework

Given a description of a dialogue game, such as the description of DE as presented in Section 1, and a UML diagram of a generic dialogue game as previously depicted in Figure 1, the UML diagram can be annotated using an OCL specification file. An example of such a file can be found in the protocol_de_ocl description file[3], which presents the OCL expressions used to describe DE. Suitable tools can subsequently process this description file. The Object Constraint Language Environment (OCLE)[4] is one example, to generate executable code. The output of the code generation stage is executable Java code. This process essentially yields the core of a dialogue game engine via code generation, for example, providing checks against the dialogue rules when a particular operation, such as makeMove, is invoked, and generating an error message if a player breaks the game rules.

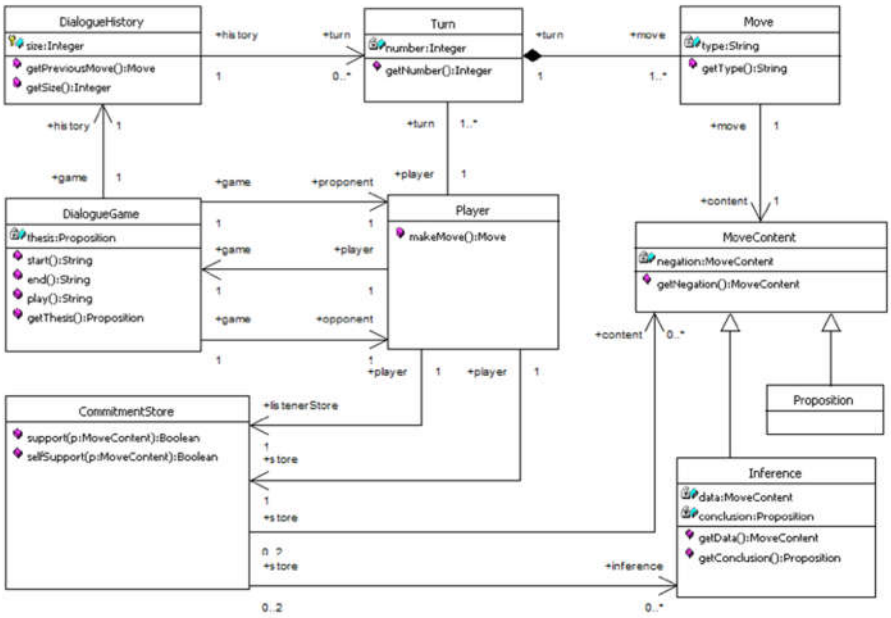

Figure 1. A generic model for dialogue games expressed using the UML class diagram notation. This model captures the general core elements of dialogue games and provides the basis for an API for generated code.

[3]This is available from the ProtOCL Git repository located at the following URL: https://github.com/siwells/ProtOCL

[4] http://lci.cs.ubbcluj.ro/ocle/

There are a number of advantages to using code generation to pro- duce the core of the game engine. Primarily, it reduces the opportunity for the game designer to introduce errors. Additionally code-generation reduces the time required to implement and deploy new game engines and streamlines the effort required, by automating much of the implementation process. As a result effort can be focused on the design of the game rather than implementation details.

Because the game engine Application Programming Interface (API) is based upon the classes generated from the UML class diagram, which is static, it is straightforward to build new, dialogue aware tools against it, for example, using the engine to provide dialogue game managements for intelligent agents. This relationship is illustrated in Figure 2. Additionally, so long as the class model API remains unchanged, different games can be generated by OCL tools and then played by the agents via the game engine.

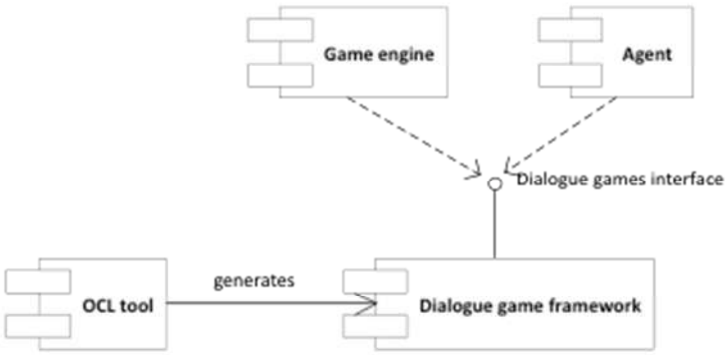

Figure 2. An overview of the ProtOCL dialogue system API from the code generation perspective. The OCL tool generates a dialogue game framework that exposes a common interface. This interface is then exploited by software within specific problem domains.

5. Conclusions & Future Work

Formal specification of dialogue games in a developer-friendly manner is attractive in terms of developing and testing dialogue games. OCL as part of the UML is a formal language to describe complex business rules and thus providing a tool framework that is more familiar to existing developers. The system we have introduced, ProtOCL, demonstrates how to use OCL to specify dialogue games via a generic UML class model that contains the terms and languages that are persistent to the dialectical system domain. Particularly, the naturally expressed dialogue games rules can be translated into OCL invariants and pre- and post- conditions. The

expressive power of UML/OCL to the dialogue games has so far been demonstrated by the representation of the DE game using UML/OCL. A dialogue game framework can be generated via the existing OCL tools to generate the dialogue framework. The game engine and intelligent agents can then interface with this framework. Using this approach it would be convenient for the dialogue game designers to modify the game rules they are designing and test their games via the game engine and agents on the fly. It is anticipated that the proposed work represents a step forward in the implementation of dialogue games and dissemination of this approach within the wider software engineering context.

We plan to experiment with the generic class model enhancing it to support a wider variety of dialogue games to enable further refinements taking place. A dialectical system test-bed (the game engine and agents) can then be constructed for the game developers to test the dialogue games and dialogue strategies they have developed. One area of future work is to connect the current approach, using UML and OCL specify a dialogue game using human-oriented, graphical tools, with previous work on the Dialogue Game Description Language (DGDL) which enables games to be formally defined and syntactically verified. This opens the door to bi-directional movement of dialogue game specifications between a system that aims for increased human utility, and a system that aims for verifiability and formal correctness.

By bringing both approaches together we believe that a usable dialogue game definition and execution system can be assembled that enables software developers to build systems using the tools that they are already familiar with, and whose outputs can be evaluated and checked to ensure that they conform with developer expectations. This work contributes not just to the design and implementation of new games but also enables developers to immediately utilise dialogue games using industry standard software engineering tools and techniques.

References

Artikis, A., Sergot, M. J., & Pitt, J. (2007). An executable specification of a formal argumentation protocol. *Artificial Intelligence*, 171(10–15), 776– 804.

Bench-Capon, T. J. M. (1998). Specification and implementation of Toulmin dialogue game. In *Proceedings of jurix 98* (p. 5-20).

Bench-Capon, T. J. M., & Dunne, P. E. (2007). Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10–15), 619–641.

Bodenstaff, L., Prakken, H., & Vreeswijk, G. (2006). On formalising dialogue systems for argumentation in the event calculus. In Proceedings of the eleventh international workshop on nonmonotonic reasoning (pp. 374–382).

Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1997). KAoS: Toward an industrial-strength generic agent architectiure. In *Software agents*. In (pp. 375–418). MIT Press, Cambridge, MA, USA.

Brewka, G. (2001). Dynamic argument systems: A formal model of argumentation processes based on situation calculus. *Logic and Computation*, 11(257–282), 619–641.

Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming and n-person games. *Artificial Intelligence*, 77, 321-357.

Grasso, F., Cawsey, A., & Jones, R. (2000). Dialectical argumentation to solve conflicts in advice giving: a case study in the promotion of healthy nutrition. *International Journal of Human-Computer Studies* , 53 (6), 10771115.

Hamblin, C. L. (1970). *Fallacies*. Methuen and Co. Ltd.

Kalofonos, D., Karunatillake, N., Jennings, N. R., Norman, T. J., Reed, C., & Wells, S. (2006). Building agents that plan and argue in a social context. In P. E. Dunne & T. J. M. Bench-Capon (Eds.), *Computational models of argument* (pp. 15–26). IOS Press.

Mackenzie, J. D. (1979). Question begging in non-cumulative systems. *Journal Of Philosophical Logic*, 8, 117-133.

Mackenzie, J. D. (1990). Four dialogue systems. *Studia Logica*, 49, 567-583.

Norman, T. J., Carbogim, D. V., Krabbe, E. C. W., & Walton, D. (2004). Argument and Multiagent Systems. In *Argumentation machines: New frontiers in argument and computation*. Kluwer.

Odell, J., Bauer, B., & Parunak, H. V. D. (2001). Representing agent interaction protocols in UML. In 22nd international conference on software engineering (isce), berlin (p. 121-140).

O. M. G. (2012). Object constraint language (OCL) 2.3.1 specification, ISO/IEC 19507.

Prakken, H. (2005). Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation*, 15, 1009–1040.

Rahwan, I., & McBurney, P. (2007). Argumentation technology: introduction to the special issue. *IEEE Intelligent Systems*, 22(6), 21–23.

Ravenscroft, A. (2000). Designing argumentation for conceptual development. *Computers and Education*, 34, 241–255.

Ravenscroft, A., & Matheson, P. (2002). Developing and evaluating dialogue games for collaborative e-learning. *Journal of Computer Assisted Learning*, 18, 93–101.

Ravenscroft, A., & Pilkington, R. M. (2000). Investigation by design: developing dialogue models to support reasoning and conceptual change. *International Journal of Artificial Intelligence in Education*, 11, 273–298.

Reed, C., & Grasso, F. (2007). Recent advances in computational models of argument. International Journal of Intelligent Systems, 22(1), 1–15. Reed, C., & Wells, S. (2007). Using dialogical argument as an interface to complex debates. IEEE Intelligent Systems Journal: Special Issue on Argumentation Technology, 22(6), 60–65.

Shen, W., Norrie, D. H., & Barthes, J. (2001). *Multi-agent systems for concurrent intelligent design and manufacturing*. CRC Press.

Vreeswijk, G. A. W. (1995). IACAS: An implementation of Chisholm's principles of knowledge. In *proceedings of the 2nd Dutch/German workshop on non-monotonic reasoning*, Utrecht (p. 225234).

Walton, D. N. (1998). *The new dialectic*. University of Toronto Press.

Walton, D. N., & Krabbe, E. C. W. (1995). *Commitment in dialogue*. State University of New York Press.

Wells, S., Lozinski, P., & Pham, N. M. (2008). Towards an arguing agents competition: Architectural considerations. In Proceedings of 8th CMNA (computational models of natural argument) workshop, *European conference on artificial intelligence* (ECAI), university of Patras, Greece.

Wells, S., & Reed, C. (2004). Formal dialectic specification. In I. Rahwan, P. Moraitis, & C. Reed (Eds.), *First international workshop on argumentation in multi-agent systems*.

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

Wooldridge, M. (2002). *An introduction to multiagent systems*. John Wiley & Sons, Ltd.

Yuan, T. (2004). *Human computer debate, a computational dialectics approach* (Unpublished doctoral dissertation). Leeds Metropolitan University.

Yuan, T., Moore, D., & Grierson, A. (2003). A conversational agent system as a test-bed to study the philosophical model DC. In *Proceedings of the 3rd workshop on computational models of natural argument* (CMNA'03).

Yuan, T., Moore, D., & Grierson, A. (2007). A human computer debating system and its dialogue strategies. *International Journal of Intelligent Systems*, 22(1), 133–156.

Yuan, T., Moore, D., & Grierson, A. (2008). A human-computer dialogue system for educational debate, a computational dialectics approach. *International Journal of Artificial Intelligence in Education*, 18(1), 3–26.

Yuan, T., Moore, D., Reed, C., Ravenscroft, A., & Maudet, N. (2011). Informal logic dialogue games in human-computer dialogue. *Knowledge Engineering Review*, 26(3), 159–174.

Yuan, T., & Schulze, J. (2008). Arg!draw: an argument graphs drawing tool. In *The second international conference on computational models of argument* (COMMA), Toulouse, France. (pp. 62–68).

Yuan, T., Schulze, J., Devereux, J., & Reed, C. (2008). Towards an arguing agents competition: Building on Argumento. In *Proceedings of 8th CMNA (computational models of natural argument) workshop*, European conference on artificial intelligence (ECAI), university of Patras, Greece.

Yuan, T., Svansson, V., Moore, D., & Grierson, A. (2007). A computer game for abstract argumentation. In *proceedings of IJCAI'2007 workshop on computational models of natural argument*, Hyderabad, India. (pp. 62–68).