Testing Formal Dialectic

Simon Wells and Chris Reed

Division of Applied Computing, University of Dundee, Dundee, UK, DD1 4HN
{swells, chris}@computing.dundee.ac.uk

Abstract. Systems of argumentation or 'computational dialectic' are emerging as a powerful means of structuring inter-agent communication in multi-agent systems. Individual systems of computational dialectic have been suggested and implemented to tackle specific problems but no comprehensive and comparative assessment has been made of such systems. This paper introduces Scenario$_{GC0}$, a framework for the implementation and testing of a wide range of computational dialectic systems. Scenario$_{GC0}$ has a range of benefits for both theoretical and practical work in computational dialectics, including: a means to test arbitrary dialectic systems using a unified knowledge base; a means to determine standard metrics by which dialectic systems can be measured and compared; enabling a body of example dialogue to be assembled for each dialectic system to demonstrate their qualities.

1 Introduction

Certain organisms are used in biological sciences research as models against which to measure new theories. One such organism is *Drosophila Melanogaster* a type of fruit fly [2] which is considered particularly important because much is known about this organism. The body of knowledge about *drosophila* is used as a base line against which to test new theories without extra investment in setting up an experimental structure. New results are thus compared and contrasted against the large body of experimental data already collected. Herbert Simon [20], and later John McCarthy [14] refer metaphorically to the game of Chess as a "drosophila" for AI. In a similar vein McCarthy also proposes the missionaries and cannibals problem as a drosophila for problems in logical AI [15]. The suggestion is that certain classes of problems, puzzles and games can be used to quantify progress in the field of AI overall and to demonstrate individual theories within the field.

This paper presents a drosophila for computational dialectics which we call Scenario$_{GC0}$ together with an implementation framework. The framework enables formal dialectic systems to be rapidly implemented and example dialogues to be produced. This process can be used to investigate the properties of dialectic systems. The results of such an investigation can in turn be used to inform the research, construction and implementation of computational dialectic systems.

Metrics can be identified for dialectic systems and measurements made of the output from running dialectic systems against Scenario$_{GC0}$. This allows the

behavior of each system to be examined and quantitative measurements to be derived for those behaviors. The behavior of dialectic systems can thus be measured, compared and evaluated. This fulfills a need in the field of agent communication for a means to evaluate systems of computational dialectic. It also takes the first steps towards building a corpus of example dialogue for each system, generated by an actual implementation of that system. This process enables the identification of the circumstances in which a particular formal dialectic system is most appropriately deployed. It can be used to demonstrate the benefits and efficacy of that formal dialectic system to potential implementors.

2 Problem

Formal dialectic systems were proposed in [7] as practical means to model the interactions between participants in a dialogue in order to examine the situations in which logical fallacies occur. Formal dialectic systems are two-player, turn taking games. The players use their turn to make moves according to the rules of the system. Formal dialectic systems specify the kinds of things that can be said by a participant in a dialogue and when those things can or cannot be said. They don't however make any provision for the propositional content of what is said but concern themselves with the speech acts that are uttered during each players move. Many dialectic systems have been proposed including but not limited to, H [7], DC [10], DL3 [6], PPD [22], R [19] as well as related systems such as the Toulmin Dialogue Game [4], the case study games [11], the eightfold model [12] and variations on existing dialogue games [1]. In addition argumentation, particularly through dialogue games and formal dialectic systems have been proposed as means to structure argumentative dialogue between agents in a MAS [16].

No substantial attempt has been made to establish which system is best for a given application. Many systems have been proposed and many more are possible yet there has been no structured way to approach the specification and implementation of formal dialectic. There has also been no structured approach to establishing the grounds upon which a comparison of systems might be built. Computational testing incorporating the unified specification and implementation of formal dialectic and the production of empirical data from running those systems under known conditions is required. This enables the properties of individual systems of formal dialectic to be determined such that those circumstances to which a given system might be best applied can be established. Because formal dialectic systems make use of but do not provide a formulation for the propositional content of a players moves, the framework that implements a game for testing purposes must supply data that can function as the content of the moves made during a dialogue. Additionally this data must have some provision for argument structures so that the dialogues generated are reminiscent of real-world interactions.

A testing framework should implement a scenario that enables the following: (1) facilitate structured extension and enhancement, (2) enable the automatic

generation of arguments with a clear basis for those arguments in the structure of the scenario, (3) produce results that are easily analysed, compared and verified. Further, in a multi-agent system context, a scenario needs to provide a basis for at least the following behaviour: (1) goals that agents can pursue, (2) state changing actions which individual agents can perform. Agents can thus engage in dialectic based inter-agent communication to influence other agents to perform actions in pursuit of goal satisfaction.

3 Scenario

The four colour problem [5]asks whether any map can be coloured using only four colours such that no two neighbouring regions share the same colour. In graph theoretic terms each region of a map may be considered to be a vertice in a graph. It may then be asked whether, given a connected planar graph, only four colours are required to assign each vertice a colour such that no neighbouring vertices share the same colour. For non-planar graphs it may be asked whether the graph can be coloured using n colours such that no neighbouring vertices share the same colour. These type of problems are generally referred to as graph-colouring problems. Scenario$_0$ uses graph colouring problems to provide a basis for an agent society.

The first graph colouring scenario, Scenario$_{GC0}$, is conceived as a testing domain for computational dialectic systems that provides a social context for initiating argumentative dialogue and a knowledge domain to provide a basis for argumentative discourse between agents in a MAS. These properties are leveraged to provide automated, iterative and comparative testing of computational dialectic systems.

The aim is not to provide a solution to a graph colouring problem but to generate test data for computational dialectic systems using a MAS based characterisation of the problem as the basis for argument generation.

3.1 Scenario$_{GC0}$

This scenario is presented as a starting point for examining various types of dialogue including but not limited to information-seeking, persuasion and negotiation. The elements of a core scenario, called Scenario$_{GC0}$, for the graph-colouring problem domain are presented as follows;

Scenario Specific Properties Each agent in the MAS possesses a colour status. Colours are selected from a fixed pool of colours available to that instance of the scenario. As an initial starting point the pool of colours is fixed at four, red, yellow, green and blue which is reminiscent of the four colour problem. Each agent maintains relationships with a set of other agents in the MAS which are its neighbours. Relationships are defined as an edge joining two vertices. In those cases where two vertices are joined by exactly one edge those vertices are called neighbours. Neighbours are only ever connected directly by one edge although

there may be higher order relationships connecting two vertices through other vertices. Relationships are set during system startup and are fixed throughout the duration of the MAS.

Agent Knowledge At start-up an agent knows only its own colour and a set of neighbouring agents. An agent must therefore research other agents in the MAS in order to increase its knowledge base. To achieve this the agent must engage in dialogue with its neighbours in order to find out the colour properties and relationships of the other agents and identify any conflicts. Conflicts occur when neighbouring agents possess the same colour property. The kind of arguments that an agent can muster and the persuasiveness of those arguments is tied very closely to the knowledge that an agent has. Matters are further complicated because knowledge is uncertain. Whilst an agent can only ever be in a single colour state at any given time that colour state is mutable, it changes with time as the agent determines that another colour state is more suitable. The other agents in the system do not necessarily know that a particular agent has changed colour and might attempt to use information that is no longer accurate. As a result the agents must reason with uncertain and dynamic information in order to achieve their goals.

Conflicts Conflicts occur when two neighbouring agents share the same colour and is defined as a graph in which two vertices connected by one edge are the same colour. When a conflict occurs it is necessary to resolve the situation. Individual agents have at their disposal the capability to communicate with other agents in order to facillitate a resolution but they have no power to directly influence another agent other than through argumentative dialogue.

Goals Agents in the MAS maintain goals which pertain directly to the scenario. An agent initially has a single goal, to resolve all conflicts with its neighbours.

Actions An agent can elect to change its own colour at will should the colour change not bring it into conflict with any of its neighbours. Where there is conflict between agents those agents may elect to change their individual colours in order to remove that conflict. If an agent can change to a free colour, defined as a colour that none of its neighbours currently possesses, then that is the course of action an agent should take. If there are no free colours then an agent might have to change to a colour already possessed by another neighbour even though this will bring it into conflict with that neighbour. In this case the colour change, and resulting movement from a conflict with one neighbour to a conflict with a second neighbours depends upon the argumentation process that has occurred and the agents own internal reasoning.

Conflict Resolution On discovering a conflict between itself and a neighbouring agent, an agent can make use of the formal-dialectic system at its disposal to bring about a resolution of that conflict. The formal dialectic system might offer

various means of conflict resolution involving aspects of, for example, persuasion, negotiation or deliberation. The process of an actual dialogue in terms of the moves that can be made at any given time, the requirements for successful completion of those moves and the effects of a successful move are bound up in the specification of that formal dialectic system as proposed in [23].

Arguments The agents use arguments to support and justify the actions performed in the system. Formal dialectic systems generally make use of arguments as the content of moves. In this case the arguments are simply propositions that are used to support other propositions and in so doing are used to make a case for the performance or non-performance of some action. Provided that an agent has sufficient knowledge and that the current state of the system is such that an argument can be produced, then whenever a move requires production of some argument the agent should be able to furnish such an argument from its knowledge store.

4 Implementation

A framework to support dialectic testing has been constructed using the Java language and the Jackdaw Agent Framework[9] through the Jackdaw University Development Environment (JUDE). Jackdaw is a lightweight, flexible, industrial-strength agent platform that uses a modular approach to agent development. This enables domain specific functionality to be encapsulated into a module which can be dynamically loaded into a Jackdaw agent at runtime. A Jackdaw module has been implemented that facilitates dialogue between agents in a Jackdaw MAS. The module implements the graph-colouring scenario to enables the automated testing of formal dialectic. The module, named the dialogue manager, is comprised of several data stores and processing components. The data stores include the protocol store, commitment store, dialogue store, template store and knowledge store. The processing components which manipulate the contents of the data stores include an argument manager to facilitate the production of arguments, a protocol manager to govern the process of engaging in argumentative dialogue, and a reasoning component which is embodied in the dialogue manager to facilitate overall control and goal-directed behavior within the module. An overview of the system is shown in 1.

4.1 Protocol Store

The types of communicative acts that an agent can make during a dialogue are regulated by the formal dialectic system in force for that dialogue. The formal dialectic system is stored in a specification format [23] that enables the specification and implementation of arbitrary Hamblin-type formal dialectic systems[7]. Formal dialectic systems have traditionally been specified through lists of locution, commitment, structural and completion rules. Instead, because all a player can do is make moves, moves are made central to the specification of a system

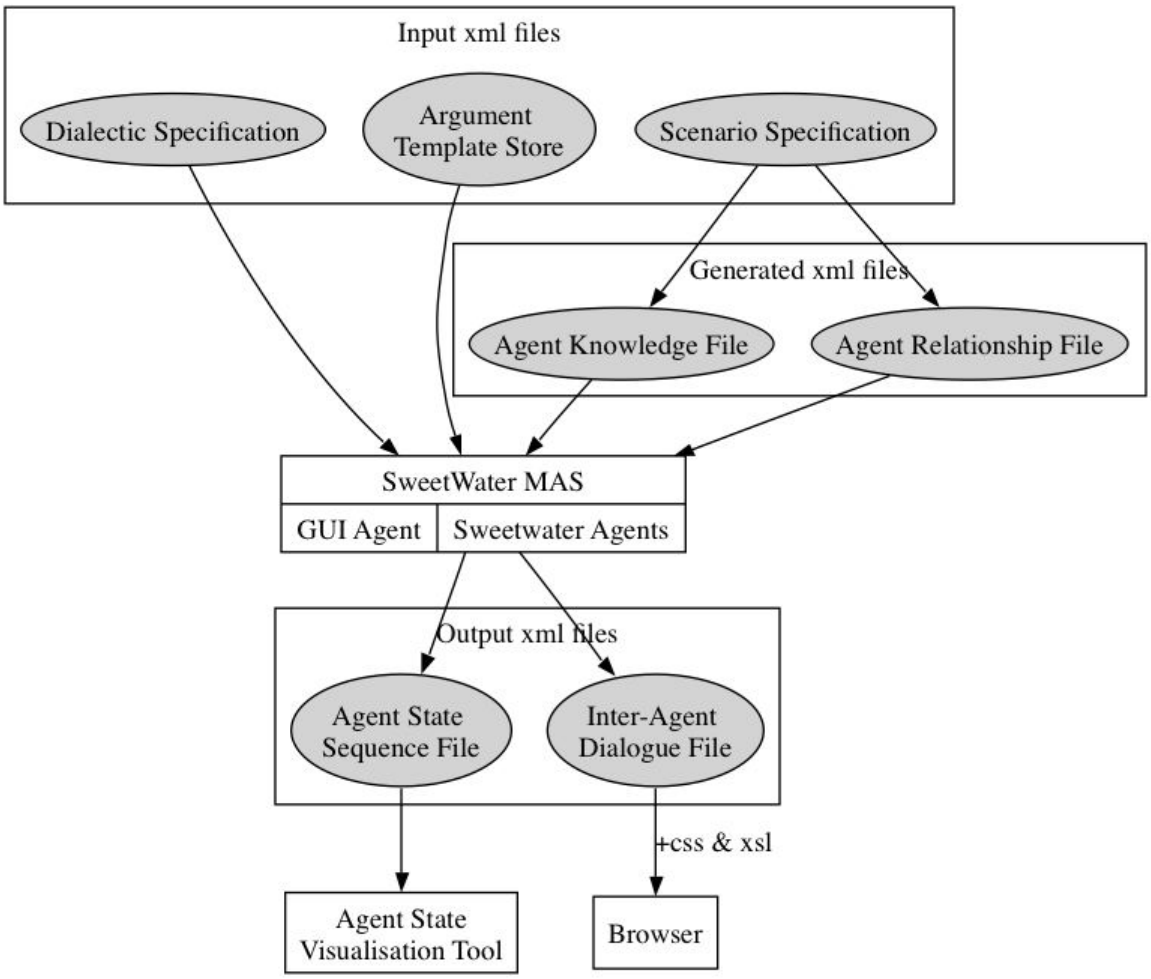

Fig. 1. An overview of the testing framework

and a consideration is made of the effects of making the move and the requirements for doing so. A formal-dialectic system is thus treated merely as a set of moves which the players can make during their turn. Each move is specified in terms of a set of requirements for the move to be legal and the effects of making the move. This has the benefit of enabling systems to be written in a compact format that is both human and machine readable. The range of parameters for move requirements and effects thus far identified enable the following systems to be implemented, H [7], DC [10], DL3 [6], PPD [22], R [19], as well as a myriad of variations on each individual system. Each set of moves that comprise a formal dialectic system is stored in an xml file. Agents can load systems dynamically at runtime to enable the rapid development, implementation and evaluation of new systems.

4.2 Dialogue Store

The dialogue store maintains the transcripts of each dialogue that an agent engages in. This is required for two reasons. Firstly, to fulfill the purposes of a testing framework, It is necessary that not only are results produced by the system but that the process of achieving those results is both clearly represented and easily comprehended. Secondly, the rules of many formal dialectic systems rely upon being able to verify earlier dialogical events and the process of producing arguments can be simplified if a store of arguments that have already been used is maintained. Thus the dialogue store maintains responsibility not only for enabling an agent to maintain some memory of earlier dialogical events but also for the production of transcripts of each dialogue for use in the analysis and verification process.

4.3 Commitment Store

The commitment store maintains a record of the commitments of the participants during a dialogue. In addition to storing the current commitments of participants, in accordance with the current formal dialectic rules, the commitment store enables earlier commitments which might have since been retracted to be examined. This enables rules to be formulated that govern whether a move is admissible based upon whether it has ever been committed to in the past.

4.4 Protocol Manager

The protocol manager utilises the protocol, dialogue and commitment stores in order to govern the process of engaging in argumentative dialogue. This process involves determining the current set of legal moves that can be made dependent upon the moves allowed by the system, earlier moves in the current dialogue and the state of the player's commitment stores.

4.5 Knowledge Store

The knowledge store contains the agents beliefs about agent properties and relationships. This knowledge is represented in an xml file which enables agent knowledge to be organised in a top-down fashion. This facilitates the structured expansion of the concepts that an agent can store as enhanced scenarios are implemented whilst maintaining a strong correspondence between an agents knowledge and the scenario. The knowledge store is essentially a frame-based implementation of knowledge representation. The following tags, once instantiated, are sufficient to record all the information that an agent needs to know to operate successfully in Scenario$_{GC0}$:
⟨*name*⟩is a given agent's unique identifier for some other agent in the MAS.
⟨*colour*⟩is the status of the colour property for the agent indicated by the name tag.
⟨*neighbour_of*⟩records an agent's neighbour. Each neighbour of each agent is recorded using this tag.
The knowledge store provides an interface that enables other components within the agent, such as the argument manager, to retrieve and make use of information. The core of this interface are the getConcept and checkCondition methods. The getConcept method is used to retrieve a proposition from the knowledge store and the check condition method is used to verify that the store contains a particular concept. An agent's knowledge is expanded by adding new tagged data to the store or by specifying methods that process the existing information to extract new concepts. For example the number of conflicts that agent$_1$ has is not stored explicitly but can be calculated by counting the number of neighbours of agent$_1$ who have the same colour state as agent$_1$.

4.6 Template Store

The template store contains *argument templates*. Argument templates can be characterised as semi-instantiated argumentation schemes. Argumentation schemes are traditionally used to capture stereotypical patterns of reasoning and have been used in argument analysis [17, 21] and argument generation [18, 3]. Argument templates are less abstract than argumentation schemes. Templates specify the form of possible arguments within a given knowledge domain whereas schemes are concerned with the form of arguments regardless of the actual content of the arguments. Agents construct arguments by completing a template from the template store with propositional content garnered from the knowledge store to produce an argument instantiation. A template consists of a number of components; a conclusion, a set of one or more premises and a warrant relating the premises to the conclusion. Each component in a template may take one of three forms; static, dynamic or conditional. Static components are expressed in the form of propositions which do not change with respect to the agent's knowledge store. Dynamic components relate to concepts which can be extracted from the knowledge store, the exact values of which vary over time as the agent learns

of changes in its situation. Conditional components specify knowledge store concepts which must have particular values. Each component also has a name which corresponds to a concept in the knowledge base. When completing an argument template the getConcept method of the knowledge store is called with the type and name as parameters. This approach enables agents to construct arguments using the dynamic information that they have gathered during interactions with other agents in the system whilst maintaining strict control over the structure of said arguments. The aim here is not to implement a comprehensive model of argument generation but to enable dynamic arguments to be generated in a very controlled manner, to produce known inputs for the process of testing dialectic systems.

Argument templates are specified in an xml document that allows the following tags:
⟨*template*⟩is the name for this template
⟨*scheme*⟩is the scheme associated with this template
⟨*conc type="type" name="name"*⟩is the conclusion of the argument
⟨*prem type="type" name="name"*⟩is a premise in this argument.
⟨*warrant type="type" name="name"*⟩links the premises to the conclusion.

The Templates Scenario$_{GC0}$ implements several argument templates that build upon the concepts of reducing agent conflicts and increasing the stability of areas of the MAS with respect to agent colour states. For a group of agents of depth d centered around a particular agent, the stability of that group can be measured as the sum of those agents conflicts. The lower the number of conflicts the higher the stability of the agents. The intuition is that agents in more stable areas should make a colour change even if it leads to another conflict if this colour change will increase the stability of the other agent in the less stable area. The templates attempt to capture a path of reasoning from an agent's basic knowledge of colour states and relationships through to a course of action that is based upon that knowledge. As more templates are added to the store agents are able to engage in more varied dialogical behaviours and construct a wider range of arguments. An Araucaria analysis diagram of arguments produced from the templates is shown in diagram 2. The following template extract provides an argument for an agent making a colour change based on the fact that they have a conflict and that one agent has more conflicts than the other;
⟨*conc type="static" name="colour_change"*⟩You should change colour⟨*/conc*⟩
⟨*prem type="conditional" name="conflict_check" /*⟩
⟨*prem type="conditional" name="conflict_comparison" /*⟩
The name parameters for each of the premises are passed into the knowledge store and a proposition is returned for each premise which can be expressed as required by the agent as the content of a move. Typically the proposition returned for the conflict_check parameter would "We are in conflict" and that for the conflict_comparison parameter might be "You have more conflicts than me".

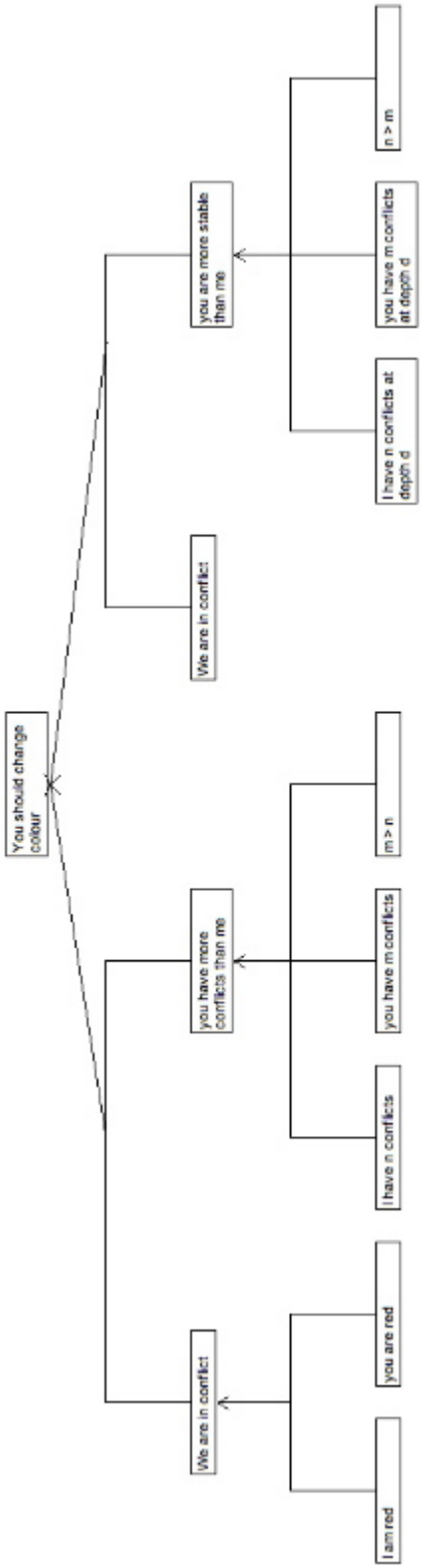

Fig. 2. Instantiations of the arguments specified for the scenario GC_0

Template Chaining The use of argument templates to provide arguments from the knowledge store suitable for use during a dialogue relies upon the use of *template chaining*. Template chaining involves following the path between templates as required to provide support for a given position. Working from a conclusion, an agent can determine the premises that lend support to that conclusion. The agent can then get supporting data for each premise by treating each premise as the conclusion to a further argument which is in turn supported by other premises. The agent thus finds templates which provide support for each premise and *chains* through the template store instantiating arguments until the premises are based on data which is self-evident from the MAS such as agent colours or relationships. Thus an agent can backwards chain through the templates from the conclusion of an argument to infer the basis for the argument. Likewise an agent can use its observations of agent states and relationships to determine a course of action to follow in pursuit of its goals by forward chaining through the templates.

4.7 The Dialogue Manager

The dialogue manager itself implements the basic reasoning required by the module and coordinates the abilities of the protocol and argument managers. The basic reasoning process that the dialogue manager follows is to find out what options it has at each juncture, e.g. which moves can legally be made. This is achieved by interacting with the protocol manager. The dialogue manager must then determine which moves can be fully instantiated with propositional content from the agents knowledge store, this process is mediated by the argument manager. If an argument is required then the agent utilises the scheme and knowledge stores in an attempt to construct a fully instantiated argument.

5 Metrics For Computational Dialectics

The testing of systems of computational dialectic leads to greater benefits than merely refining the rules of existing games and producing examples. Metrics can be identified by which each distinct dialectic system can be distinguished and categorised. Metrics fall into two broad categories, *inspection metrics*, those metrics which can be measured through external examination of the rules of a system, and *process metrics*, those metrics which are most easily measured through application of the system and subsequent examination of the results. Many inpection metrics are identified in [13] which sets out 13 desirable qualities of argumentative dialogue systems. These include; statement of purpose, diversity of purposes, inclusiveness, transparency, fairness, clarity of theory, separation of syntax and semantics, rule-consistency, encouragement towards resolution, discouragement of disruption, change of position, and system and computational simplicity.

A number of process metrics have been identified that can be applied to the dialogues produced using a computational dialectic system in order to gain insights into that system. The following list is representative but not exhaustive;

5.1 Simplicity of Representation

Communicative Act How complex are the communicative acts required by the system?
System How complex is the computational representation of the rules of the system?

5.2 Efficiency of Process

Communication Size and number of messages
Computational How much computational power does the system require?
Optimality How optimal are dialogue results?
Completion Do all dialogues complete? Are deadlocks avoided? Does the system specify completion conditions?
Repetition Is repetition of utterances minimised?

5.3 Flexibility

Data Requirements How complete does data have to be for a dialogue to reach a satisfactory completion

5.4 Expressiveness

Evolution Can participants effect a change of position during a dialogue?
Range Which profiles of dialogue [8] are permitted or prohibited by the system?
Symmetry Are the same moves available to each participant? Is this always the case?

5.5 Representativeness

Realism How representative of real world arguments are the resultant dialogues? This is not a prescription for realism but an objective measurement and comparison.

5.6 Stability

Reproducibility Given the same general inputs, does the system converge to the same set of results?
Predictability Can the results of a dialogue be predicted?

The analysis of dialogues generated by a given system can be used to make direct measurements of *some* metrics, for example, aspects of the *efficiency of process[communication]* metric can be examined directly. The average number and size of communicative acts per dialogue can be measured and compared between systems. This is sufficient to enable a quantitative comparison to be

made between systems on the basis of communicative efficiency. Other metrics rely upon further knowledge against which to compare results, for example, the *expressiveness[range]* metric relies upon knowledge of the range of possible dialogue profiles. Actual measurements of inspection and process metrics can thus be used to categorise individual dialectic systems. This will enable developers to select dialectic systems for use on the basis of their measured attributes and performance.

6 Results

6.1 An Example DC Dialogue

Agent1 and Agent2 are neighbours who are in the same colour state, hence they are in conflict. Upon discovering this Agent1 initiates a dialogue to resolve the conflict. Because of the colour distribution of their neighbours neither agent has any free colours hence any colour change will result in conflict.

$\mathbf{Agent}_1$	Statement("Agent2 should change colour")
$\mathbf{Agent}_2$	Challenge("Agent2 should change colour")
$\mathbf{Agent}_1$	Defense("Agent2 is more stable than Agent1")
$\mathbf{Agent}_2$	Challenge("Agent2 is more stable than Agent1")
$\mathbf{Agent}_1$	Defense("Agent2 has less conflicts than Agent1")
$\mathbf{Agent}_2$	Challenge("Agent2 has less conflicts than Agent1")
$\mathbf{Agent}_1$	Defense("Agent2 has 2 conflicts $\wedge$ Agent1 has 3 conflicts")
$\mathbf{Agent}_2$	Statement("Agent2 should change colour")

6.2 An Example H Dialogue

$\mathbf{Agent}_1$	Statement("Agent2 should change colour")
$\mathbf{Agent}_2$	Challenge("Agent2 should change colour")
$\mathbf{Agent}_1$	Support("Agent2 is more stable than Agent1")
$\mathbf{Agent}_2$	Challenge("Agent2 is more stable than Agent1")
$\mathbf{Agent}_1$	Support("Agent2 has less conflicts than Agent1")
$\mathbf{Agent}_2$	Challenge("Agent2 has less conflicts than Agent1")
$\mathbf{Agent}_1$	Support("Agent2 has 1 conflict $\wedge$ Agent1 has 3 conflicts")
$\mathbf{Agent}_2$	Statement("Agent2 should change colour")

The examples demonstrate the application of the implementation of Scenario_{GC0} to conflict resolution using the dialectic systems DC and H. The implementation enables agents to engage in information seeking dialogues to discover information about their neighbours and to resolve conflicts when they are discovered through persuasion type dialogues.

7 Conclusions

Scenario$_{GC0}$ and the associated implementation framework are a good way to test systems of formal dialectic through computational implementation. As a baseline it provides a simple, structured body of domain knowledge which forms the content of communicative acts within argumentative dialogue. Further the content of communicative acts is tied closely to the structure of the system and the information inherent in that structure. The basic scenario fulfills the needs of a system for automated production of simple dialogue for testing purposes. The simple nature of generated dialogues can be made more complex through the addition of new argumentation templates to the template store, through the addition of new parameters to the system structure and hence the knowledge of each agent, and through the specification and addition to the protocol store of new systems of formal dialectic. The architecture enables a wider field of experimentation than solely testing models of computational dialectic through the replacement of any or all components so a different model of knowledge might easily be incorporated or different model to govern communications. This is in addition to the flexibility of xml based input data providing a simple, efficient and flexible upgrade path for argument templates, agent knowledge and dialectic systems.

Scenario$_{GC0}$ enables the comparative testing of arbitrary systems of formal dialectic using a standardised knowledge base so that differences in results stem from differences in the dialectic system. It has enabled steps to be taken towards a system of standardised metrics for computational dialectics. Finally it facilitates the automated construction of a corpus of computationally generated dialogue which can be used to compare and contrast the performance of different systems of dialectic. Ongoing work with Scenario$_{GC0}$ involves the testing and comparison of a wide range of formal dialectic systems and dialogue games, the identification of further metrics according to which systems of computational dialectics might be classified, the implementation of additional enhanced scenarios and new argumentation templates.

8 Acknowledgements

This research is funded by EPSRC under the Information Exchange project. Gratitude is expressed to Calico Jack Ltd. for their JackDaw agent framework and JUDE development environment.

References

1. L. Amgoud, N. Maudet, and S. Parsons. Modelling dialogues using argumentation. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, 2000.
2. M. Ashburner. *Drosophila: A Laboratory Handbook*. Cold Spring Harbor Laboratory Press, 1989.

3. K. Atkinson, T. Bench-Capon, and P. McBurney. Justifying practical reasoning. In I. Rahwan, P. Moraitis, and C. Reed, editors, *First International Workshop on Argumentation in Multi-Agent Systems*, 2004.
4. T. J. M. Bench-Capon. Specification and implementation of toulmin dialogue game. In *Proceedings of JURIX 98*, pages 5–20, 2001.
5. A. Cayley. Open problem. *Proceedings of the London Mathematical Society*, 9:148, 1878.
6. R. A. Girle. Commands in dialogue logic. *Practical Reasoning: International Conference on Formal and Applied Practical Reasoning, Springer Lecture Notes in AI*, 1996.
7. C. L. Hamblin. *Fallacies*. Methuen and Co. Ltd., 1970.
8. E. C. W. Krabbe. Profiles of dialogue. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, editors, *JFAK - Essays Dedicated to Johan van Benthem on the occasion of his 50th birthday*. Amsterdam University Press, 1999.
9. Calico Jack Ltd. http://www.calicojack.co.uk, 2005.
10. J. D. Mackenzie. Question begging in non-cumulative systems. *Journal Of Philosophical Logic*, 8:117–133, 1979.
11. N Maudet and F. Evrard. A generic framework for dialogue game implementation. In *Proceedings of the Second Workshop on Formal Semantics and Pragmatics of Dialog*, 1998.
12. P. McBurney, D. Hitchcock, and S. Parsons. The eightfold way of deliberation dialogue. *International Journal of Intelligent Systems*, 2002.
13. P. McBurney, S. Parsons, and M. Wooldridge. Desiderata for agent argumentation protocols. *Proceedings of the First AAMAS*, pages 402–409, 2002.
14. J. McCarthy. Ai as sport. *Science*, 276(5318):1518–1519, 1997.
15. J. McCarthy. Elaboration tolerance, 1998.
16. S. Parsons and N. R. Jennings. Negotiation through argumentation. In *Proceedings of ICMAS'96*, pages 267–274, 1996.
17. C. Reed and G. Rowe. Araucaria: "software for puzzles in argument diagramming and xml. Technical report, University Of Dundee, 2001.
18. C. Reed and D. Walton. Towards a formal and implemented model of argumentation schemes in agent communication. In I. Rahwan, P. Moraitis, and C. Reed, editors, *First International Workshop on Argumentation in Multi-Agent Systems*, 2004.
19. N. Rescher. *Dialectics*. State University of New York Press, Albany., 1977.
20. H. Simon and J. Schaeffer. The game of chess, 1992.
21. D. Walton. *Argumentation Schemes for Presumptive Reasoning*. Lawrence Erlbaum Associates, 1996.
22. D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue*. SUNY series in Logic and Language. State University of New York Press, 1995.
23. S. Wells and C. Reed. Formal dialectic specification. In I. Rahwan, P. Moraitis, and C. Reed, editors, *First International Workshop on Argumentation in Multi-Agent Systems*, 2004.