GES 2007 German e-Science

Available online at http://www.ges2007.de

Automated Workflows using Dialectical Argumentation

Jarred McGinnis[1], Stefano Bromuri[1], Visara Urovi[2] and Kostas Stathis[1]

[1] Royal Holloway College, University of London, McCrea Building , Egham, Surrey TW20 0EX United Kingdom

[2] Second Faculty of Engineering, University of Bologna Via Rasi e Spinelli 176 47023 Cesena (FC)

email: jarred@cs.rhul.ac.uk

phone: +44 (0)17 84 44 34 36

Abstract

This paper presents a framework for dynamic workflow creation and execution developed as part of ARGUGRID, a collaborative project that seeks to provide a new model for programming the Grid at a semantic, knowledge-based level of abstraction through the use of argumentative agent technology. In this framework, workflow selection is coordinated by agent interactions based upon a dialogue game that allows agents to argue about workflows and their properties.

1 Introduction

The misleading schism between research on the Grid and multiagent systems is quickly dissipating as researchers on both sides recognise that Grid infrastructures can be made more flexible and dynamic through the use of agent technologies [4]. Conversely, the agency community see the Grid paradigm useful for grounding their work in a practical environment. Motivated by these observations, the ARGUGRID project [1] seeks to provide a new model for programming the Grid by using argumentative agent technology. Agents in ARGUGRID enact the reasoning and decision making processes and negotiation required for dynamic composition of Grid resources and services into executable workflows, using argumentative agents to support grid service providers and requestors.

As part of our contribution to the ARGUGRID project we aim at creating dynamic and run-time execution of workflows using argumentative agents. Such agents use the notion of *dialogue games* [5] to reason about the workflows available, decide amongst themselves the most satisfactory for a certain goal, and then execute that workflow by using the same mechanism to coordinate their interaction. This is a especially useful approach for scenarios in which it is impossible or impractical to know the workflow needed before execution time. The burden no longer lays upon the engineer to foresee all possible workflows, nor do we rely on one agent to have every possible solution either.

The following section, section 2, briefly describes dialogue games and their use as a communication model for the selection of workflows by autonomous

agents. Section 3 describes the framework which coordinates the agents as they make moves in the dialogue game in order to deliberate upon which workflow is to be executed. This same framework can also be used to coordinate the execution of the selected workflow. A practical example and scenario are given in section 5. The technologies used for this approach is described in section 4 and concluding remarks are given in section 6.

2 Workflow selection via dialogue games

Our framework builds upon previous work on service-oriented agent architectures [2] using argumentation to model communication of propositions, which allows agents to more quickly find solutions or identify problems. In addition to communicating propositions, our framework also supports the reasoning and justification for those propositions, in order to make easier for participants to find common ground which is acceptable to all. However, unlike [2], we employ dialogue games to provide rules for the commencement, termination and continuance of an agent dialogue to increase its reliability and the certainty that progress is made towards a conclusion.

This approach is flexible in that it can support different dialogue types to be formulated as different dialogue games. In [10] a set of six atomic dialogue types are identified: information seeking, inquiry, persuasion, negotiation, deliberation and eristic dialogues. Dialogue types are determined by the shared and private goals of the participants. This typology has been influential in the development communicative models for multiagent systems [8].

This framework implements the deliberation game described in [5]. However the system is not dependent on any one dialogue game type or specific rules. Deliberation games are traditionally seen as a discussion of a course of action and we model workflow as such. In this model agents have access to a library of workflows which are labeled with properties (metainformation about workflows such as a computational cost, time to execute, and other useful details). By using these properties, agents will thus be in a position to justify or reject a workflow being proposed during a dialogue.

The dialogue game has several components which create the basis for a semantically rich model of communication. This is achieved by formalising the dialogue game at various levels. Firstly the logical language which the domain (i.e. workflows) is discussed can be expressed as a number of sentence types such as : *actions*, *goals*, *constraints*, and *facts*. Locutions which contain propositions of the type *Actions* are propositions for workflows. *Goals* specify top level properties of the proposed workflows. *Constraints* are meant to restrict the set of potential workflows by limiting them upon some grounds such as time for execution or requiring the use or a particular service or provider. *Facts* allow agents to communicate propositions about the state of the world or affairs that may have an indirect effect on selection of the workflow.

In order to regiment the discussion between the agents and guide the dialogue game towards some conclusion. The rules of the dialogue game are organised into

the stages of: **Open, Inform, Propose, Consider, Revise, Recommend, Confirm** and **Close**. The dialogue begins at the **Open** stage where agents register the interest in the solution to a governing question (i.e. a problem to be solved by the selection of a workflow). In the **Inform** stage, agents establish their positions, biases, facts and constraints. The proposed workflows are passed during the **Propose** phase. The agents can specify preferences between the workflows proposed in the **Consider** stage. The **Revise** stage allows agents to modify any proposals or preferences they have made. At a certain point a proposed workflow is **Recommend**ed for execution. Afterwards, during the **Confirm** stage, a poll is taken amongst the players of the dialogue to find if there is a consensus on a workflow to be executed. Once that consensus is reached the dialogue moves to the **Close** stage.

The changes of stages reflect moves made by the players and the moves available are dictated by the current stage. Moves are made by agents communicating locutions. Locutions for our deliberation game are as follows: open_dialogue, enter_dialogue, withdraw_dialogue, propose, assert, prefer, ask_justify, move, reject and retract.

The **Open** stage begins once some agent has sent an open_dialogue locution and at least one other agent has shown their interest in playing the dialogue game by communicating enter_dialogue. This stage occurs once in a dialogue. The agents can now progress to the **Inform** stage by the utterances of propose, assert, retract and ask_justify locutions for the proposition types of *goal*, *constraint* and *facts*. In order to progress to the **Propose** stage on of the agents must communicate a propose with the proposition of type *action* and the content of the locution will be a workflow for execution. The agents are now able to move to the **Consider** stage by utterances of prefer and ask_justify. Activities in the stage can be followed by the **Revise** stage, which allows the revision of proposed workflows. This is achieved by the assumption that the agent will only suggest one workflow at a time (i.e. its best workflow by whatever private and public criteria that has been established). If the agent wishes to propose a revised workflow, it can by the communication of a propose with the content being the new workflow. The dialogue game reactive rules will update the commitment store to reflect the revised proposed workflow for the agent. The game moves to the **Recommend** stage by the utterance of the move locution with its content being the workflow to be executed. The other players of the dialogue came can either accept the workflow with an assert or reject it with a reject. Once all players have sf asserted their acceptance of the workflow as part of the **Confirm** stage, the game progresses to the Close stage by the participants utterance of the withdraw_dialogue locution.

Certain locutions carry with them obligations. These are expressed in publicly verifiable commitments that disambiguates the requirement for the agents' interaction and makes the semantics of the conversation and the subject of that conversation clearer. For this dialogue game, the reactive game rules on the communication medium update the commitment store upon the utterance of assert, retract, reject, and prefer. The details of the semantics for each of the locutions

are similar to those of [5].

3 Framework for workflow selection and execution

This section describes the framework in abstract terms leaving the technical details to a later section. Figure 1 shows our framework. Firstly the agents (not limited to the two shown) communicate through a reactive medium by the sending and receiving locutions acting as moves in the dialogue game. Also, agents in the framework can read the dialogue state from the communication medium enabling to determine the next appropriate move to make. This medium reacts to the moves made by the agents by updating the state of the dialogue according to the rules defining the dialogue game thus coordinating the activities of the participating agents. In addition the reactive dialogue game rules ensure that the commitments made or satisfied during the dialogue game are reflected by the commitment store. This public and independent maintenance of the commitments helps to ensure accountability and trust in the framework.

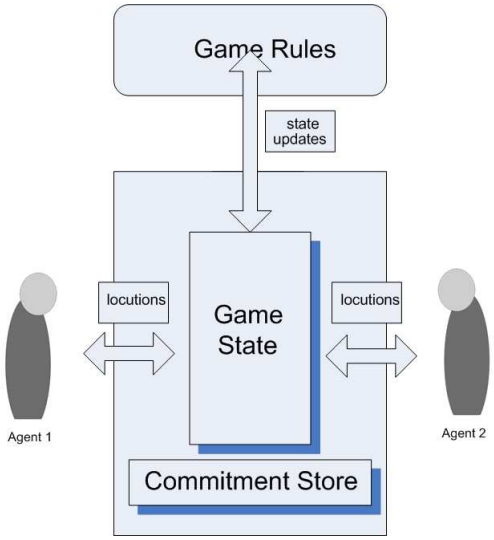

Figure 1: A dialogue game played between two agents.

Once the dialogue game concludes successfully and the participants have chosen a workflow to be executed. The agent migrate to a different communication medium which is loaded with the interaction rules, which are now the execution order of the processes of the chosen workflow. This process is illustrated by figure 2. Instead of trading speech acts, agents now inform the communication medium of the execution and completion of processes and the medium reacts

according to the workflow by initiating the next task. Agents act as proxies for services as shown in figure 3.

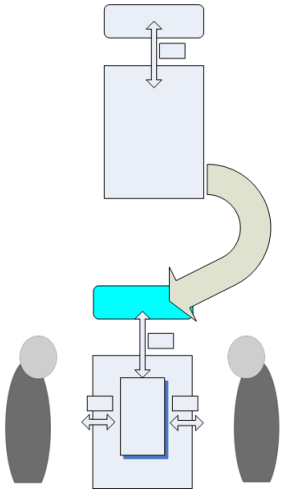

Figure 2: Agents move to another communication medium with the selected protocol

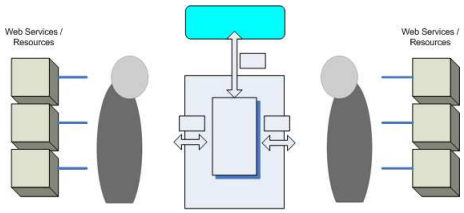

Figure 3: Agents are coordinated by the reactive communication medium to execute the protocol

4 Implementation

We are experimenting with the development of a multiagent system prototype implemented using the PROSOCS platform [9]. This platform allows the deployment of agents with logical capabilities that can discover other agents or objects such as game states and interact with each other by sending and receiving messages over a network. In the current implementation we focus on the simple

case of two agents that play the game over a network using the components shown in Fig. 1.

The underlying transport layer of PROSOCS platform is implemented using TuCSoN [3] and ReSpecT [6]. TuCSoN (Tuple Centres Spread over the Network) is a distributed systems infrastructure providing services for communication and coordination. TuCSoN relies on a blackboard model of communication which is useful for dealing with the asynchronous nature of agent interactions [7]. The TuCSoN tuple centres can be programmable using ReSpecT. Using ReSpecT it is possible to define a reactive tuple centre in order to achieve a complex interaction by activities such as propagating messages containing locutions or maintaining a shared object states such as those of dialogue games.

4.1 Agent Architecture

The agent architecture used is an improvement of the PROSOCS architecture presented in [9]. Every agent is defined as a set of Java components:

- Agent Mind: The agent mind is a wrapper around a Prolog theory that defines declaratively the agent behaviour. In this paper we utilized a simple Prolog mind capable to manage the locutions of the deliberative dialogue, in order to select a suitable workflow.
- Agent Body: The agent body is a composition of sensors and effectors, it is an interface between the mind and its sensors and effectors, receiving perceptions from the sensors linked to the environment and executing actions on the environment through the effectors.
- Agent Effectors: Effectors are proxies hiding the complexity to deal with the environment. The effectors execute actions defined as first order logic predicates (i.e. *speech_act(open_dialogue(Aid,Id,Q))*). According to the kind of environment this actions could have different effects. In our framework agents utilize effectors to send speech acts to a tuple centre.
- Agent Sensors: Like effectors, the sensors are proxies between the agent and the environment. They hide the complexity required to interface with an environment. They retrieve information from the environment and translate it in perceptions for the mind.

Figure 4 represents the agent architecture: the lifecycle stage of the agent (alive, frozen, dead) are controlled by the body within the body state.

4.2 ReSpecT Dialogue Game

The ReSpecT language was designed to program the behaviour of a TuCSoN tuple centre. It constrains and enables the interaction of heterogeneus agents. Therefore, it can be used to model the agent interaction and move the responsibility of maintaining the state of the interaction from the agents to the TuCSoN tuple centres. In the case of the deliberative dialogue we developed, the ReSpecT theory defines the working mechanism of a complex shared structure representing the dialogue state in form of tuples.

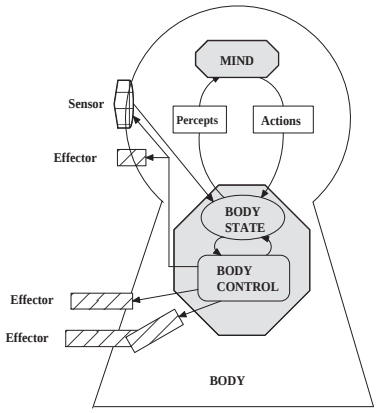

Figure 4: PROSOCS agent architecture

```
%     OPEN -> INFORM
reaction(out_r(propose(AID,ID,Type,Q)),(
     in_r(propose(AID,ID,Type,Q)),
          X is propose(AID,ID,Type,Q),
          inform_locution(X),
     in_r(dialogue_history(ID,GQ,Sh,Lh)),
     isearch(open,Sh),
     append(Sh,[inform],NewSh),
     append(Lh,[X],NewLh),
     out_r(dialogue_history(ID,GQ,NewSh,NewLh)),
          out_r(dialogue_changed(ID))
)).

%     OPEN -> INFORM
reaction(out_r(assert(AID,ID,Type,Q)),(
     in_r(assert(AID,ID,Type,Q)),
          X is assert(AID,ID,Type,Q),
          inform_locution(X),
     in_r(dialogue_history(ID,GQ,Sh,Lh)),
     isearch(open,Sh),
     append(Sh,[inform],NewSh),
     append(Lh,[X],NewLh),
     out_r(dialogue_history(ID,GQ,NewSh,NewLh)),
     %Commitment store update
     in_r(commitment_store(ID,Cs)),
     append(Cs,[assert(AID,ID,Type,Q)],Cs1),
          out_r(commitment_store(ID,Cs1)),
          out_r(dialogue_changed(ID))
)).
```

Table 1: Inform stage of the Dialogue Game Theory

Table 1 is an example of the Dialogue ReSpecT reaction rules. The two reactions state that the inform stage is accessible from the open stage with an assert or a propose performed by the agents. The ReSpecT language is an extension of Prolog, so it is possible to define predicates to check the preconditions' satisfaction. Likewise, in the two reactions considered, there is a constraint on the locutions types that must be in the set *goal, constraint, fact*. Utilizing similar reactions, we coded in ReSpecT the dialogue described in the previous section adapted from [5]. The result is that the agents decide in a deliberative way

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<WorkflowDescription>
    <Description> plane ticket </Description>
    <Name> buyticket.txt </Name>
    <Address> flights@britishairways.com </Address>
    <Input>new_order(CaseName,Customer,Ticket,Seller,Carrier)</Input>
    <Output>case_done(ID) </Output>
    <ServiceCost>2 </ServiceCost>
    <Avaiability>99 </Avaiability>
    <Reliability>98 </Relaiability>
    <Trust>90 </Trust>
    <ServiceTime>3 </ServiceTime>
</WorkflowProperties>
```

Table 2: The workflow description format

which is the best workflow to execute, according to the workflow characteristics, that are expressed in a XML format like in table 2.

In section 5 shows an example clarifying how this XML document is utilized to deliberate between workflows and decide the most suitable for the execution.

5 Example and scenario

We exemplify our approach by considering a scenario for the dynamic provision of web-services for booking flights. In this context, virtual organisations (VOs) of *broker*, *service-provider* and *personal-service* agents need to be formed. We assume that users initiate such VOs by expressing their preferences to personal service agents to book one(or more) ticket(s) on their behalf. We are interested in the specific interactions that require agents to to book tickets according to a workflow (steps such as 1.Dates, 2.Flights, 3.Price, 4.Passengers, 5.Payment, 6.Confirmation), reminiscent of how users buy such tickets on the web.

This section presents a scenario where the process of buying airplane ticket is found by using the dialogue game. In this scenario, the agent which receives the request for buying plane ticket service does not have a suitable workflow in its own library, it must confer with other agents by requesting the instantiation of a dialogue game described in secion 2. The dialogue takes place in a tuple center and it can involve more than two agents. We use tuple centers as the room where the participants can communicate with others using the dialogue. The agents which are participating in the dialogue can observe the dialogue and decide the next move to do. In our example three agents, agentA, agentB and agentC participate in the dialogue with the governing question *planeTicket*. The agentA perceiving the request for dialogue about plane ticket workflow, opens the dialogue with the same governing question. The other agents react by inserting an enter_dialogue locution in the tuple space.

The table 3 shows a trace of the dialogue game being played. The dialogue enters the **Inform** stage when agentA proposes as constraint the service time. The dialogue remains in **Inform** stage when agentB proposes as a constraint the cost. From **Inform** stage, the dialogue moves to the *propose* stage because agentB proposes as an action the buyticket1.xml workflow and it remains in the **Propose** stage when AgentC proposes another action. When the agentA, asks

```
open_dialogue(agentA,ID,planeticket)
enter_dialogue(agentB,ID,planeticket)
enter_dialogue(agentC,ID,planeticket)
propose(agentA,ID,constraint,service time)
propose(agentC,ID,constraint,cost)
propose(agentB,ID,action,buyticket1.xml)
propose(agentC,ID,action,buyticket2.xml)
ask_justify(agentA,ID,agentB,buyticket1.xml)
assert(agentB,ID,fact,Service Time = 2)
prefer(agentC,ID,action,buyticket1.xml,buyticket2.xml)
move(agentA,ID,buyticket1.xml)
assert(agentB,ID,buyticket1.xml)
assert(agentC,ID,buyticket1.xml)
withdraw(agentA,ID,planeticket)
withdraw(agentB,ID,planeticket)
withdraw(agentC,ID,planeticket)
```

Table 3: Trace of the Example Dialogue

the agentB to justify his action, the dialogue state changes to the **Consider** state. The reply of agentB, changes the dialogue in **Inform** stage. After that, agentC sends a prefer locution to express his preference for the buyticket1.xml workflow as the best workflow. This locution changes the dialogue state from **Inform** to **Consider** stage. The move locution made by agentA changes the dialogue to the **Recommend** stage, then all the agents agree and the dialogue is begins its closing. The dialogue changes from **Recommend** into **Confirm** stage when agentC makes his assert locution and it changes from **Confirm** to **Closed** when agentB withdraws from the dialogue.

Once the dialogue is closed, the service to buy plane tickets has been decided amongst the agents. The workflow execution consists in providing the input to the workflow engine which will perform the service. By defining ReSpecT rules rules outside agents, it is possible to govern their interaction in a predictable way. In this case, the workflow management system consists in workflow engines to manage activities which are complied to by the agents. The workflow coordinated by the workflow engine consists of three tasks to execute in sequence. Once the workflow execution starts, the first task for the agents is to *buy the ticket*, when it is concluded the agent which performed it should inform the workflow engine that the task assigned to him has been successful. Then the workflow engine generates the next task which is *dispatch the ticket* and then *pay ticket*. When this activity is finished, the workflow case is considered completed.

6 Future work

We have presented a framework for dynamic workflow creation and execution developed as part of ARGUGRID, a collaborative project that seeks to provide a new model for programming the Grid at a semantic, knowledge-based level of abstraction through the use of argumentative agent technology. In our framework, workflow selection is coordinated by agent interactions based upon a dialogue game that allows agents to argue about workflows and their properties. A significant feature of our framework is the use of dialogue games to provide (a) a high-level coordination mechanism for workflow execution and (b) a specification tool for testing the conformance of the interaction according to the rules of the communication protocols used by the agents in the system. In this way we avoid the need for a tangled mix of interpreters and translation tools in order to interoperate the resulting multiagent system with the language of a workflow execution engine. We have also developed a prototype, which has provided us with an immediate and satisfactory testbed for our approach. Our next step is to use dialogue games to support workflow execution in Virtual Organisations for the Grid. The detailed consideration of the issues involved in such a task we plan to report in our future work.

References

1. The ArguGRID Project, http://www.argugrid.eu.
2. V. Curcin, M. Ghanem, Y. Guo, K. Stathis, and F. Toni. Building next generation Service-Oriented Architectures using Argumentation Agents. In A. Polze and R. Kowalczyk, editors, *3rd International Conference on Grid Service Engineering and Management*, pages 249 – 263, Germany, Sep 2006.
3. DEIS. *TuCSoN Guide*. University of Engineering of Bologna, Italy.
4. I. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. In *Proc. 3rd Int. Conf. on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2004.
5. P. McBurney, D. Hitchcock, and S. Parsons. The eightfold way of deliberation dialogue. *International Journal of Intelligent Systems*, 22(1):95–132, Jan 2007.
6. A. Omicini and E. Denti. Formal respect. *Electronic Notes in Theoretical Computer Science*, 48, 2001.
7. A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3), 2001.
8. P. M. S. Parsons, N. Maudet and I. Rahwan, editors. *Proceedings of the Third International Workshop on Argumentation in Multiagent Systems*, volume 4049 of *Lecture Notes in Artificial Intelligence*. Spriner-Verlag, 2006.
9. K. Stathis, A. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In J. Müller and P. Petta, editors, *Proceedings of the Fourth International Symposium "From Agent Theory to Agent Implementation"*, Vienna, Austria, April 13-16 2004.
10. D. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY press, Albany, NY, USA, 1995.