IADIS International Journal on Computer Science and Information Systems
Vol. 3, No. 2, pp. 110-125
ISSN: 1646-3692

AUTOMATING WORKFLOWS USING DIALECTICAL ARGUMENTATION

Visara Urovi* , Stefano Bromuri*, Jarred McGinnis*, Kostas Stathis* and Andrea Omicini**
**Royal Holloway University of London, McCrea Building, Egham, Surrey, TW200EX, United Kingdom*
***Alma Mater Studiorum–Università di Bologna, Via Venezia 52, 47023 Cesena (FC), Italy*

email: visara@cs.rhul.ac.uk

ABSTRACT

This paper presents a multi-agent framework based on argumentative agent technology for the automation of the workflow selection and execution. In this framework, workflow selection is coordinated by agent interactions governed by the rules of a dialogue game whose purpose is to evaluate the workflow's properties via argumentation. Once a workflow is selected using this process, the workflow is executed by dynamically configuring workflow engines to coordinate the participating agents' workflow activities. We illustrate the overall framework with an example of workflow composition that allows an agent to book an appropriate ticket and rent a car.

KEYWORDS

Workflow Management, Agent Dialogues, Argumentation, Tucson, Prosocs**.**

1. INTRODUCTION

The complexity of business processes is increasing along with the complexity of the computational systems that are designed to handle them. The design and development of such systems mandates for new methodologies, technologies and tools, but firstly it requires high-level metaphors to model and organize them. Multiagent Systems (MAS) and the related abstractions and technologies are today a promising approach for automating the solution to complex computational problems. In the context of MAS, one of the most effective approaches is to interpret complex computational problems as social / organizational issues, and re-cast them in terms of autonomous agents collaborating within a social framework to achieve individual as well as global goals.

In this paper, we focus on automated workflow management in the context of service-oriented architectures for virtual enterprise interoperability. By adopting MAS as the reference paradigm, we interpret workflow selection as a social problem involving workflow participants represented as agents. Thus, dialogue and argumentation among individual agents become essential tools: participants of a workflow have to talk and discuss in order to select toward the most effective workflow configuration. The use of MAS infrastructure allows the dynamic configuration of workflow engines [20]. The ability to discuss the workflow using deliberative dialogue games [11] and argumentation [1,3] is a promising approach for workflow participants. In this paper we present just such a MAS framework based on argumentative agent technology for the automation of the workflow selection and execution, where the workflow engines are dynamically configured according to the execution needs.

The reminder of this paper is organized as follows. Section 2 describes the general approach and architecture of our framework. Section 3 describes in details the implementation of the deliberative dialogue and of the workflow engines, and Section 4 presents a case study. Finally, future works and conclusions are presented in Section 5.

2. APPROACH

In our approach we utilize the Agents and Artifacts meta-model [19], where artifacts are computationally reactive entities aimed at the agent use, supporting agent social activity and their coordination. As a Workflow Management Systems (WfMS), coordination artifacts [17,18] naturally play the role of workflow execution engines [23]. By assigning activities of the workflow to the agents, workflow execution is coordinated by the artifact and allow dynamic configuration. The technologies and models used for our proposed approach are the following: the TuCSoN infrastructure [16] to provide tuple centers as coordination artifacts and workflow execution engines, the PROSOCS agent platform [26] and dialogue games [12, 1] to coordinate the interactions of the argumentative agents.

In developing this approach, we have made no assumptions about participating agent's decision making strategies or algorithms. Instead one of our goals is to allow heterogeneous populations of agents to share, understand and utilize a coordination mechanism regardless of internal design.

2.1 Workflows

The term workflow refers to the specification of a work procedure or a business process in a set of atomic activities and relations between them. These rules capture the expected flow of work coordinating participants and the activities they need to perform. A participant can be a human user, a software agent playing a specific role, a device, or a program. The workflow specification defines how activities are linked together by identifying a logic of execution between them. As shown in Fig. 1, workflow activities are executed using transition patterns:

- **Sequential**: Two or more activities are executed one after another.
- **Parallel**: Two or more tasks are executed concurrently. Concurrent activities identify two conditions: **and-split** between activities which allow them to be concurrent activities or **and-join** which synchronize parallel flows.

- **Loop**: The execution of a set of activities a certain number of times.
- **Conditional**: Choice of an activity to be executed from a set of alternatives. Conditional activities identify two conditions: **xor-join** none of the alternative branches is executed in parallel, **xor-split** based on a condition only one branch is chosen.

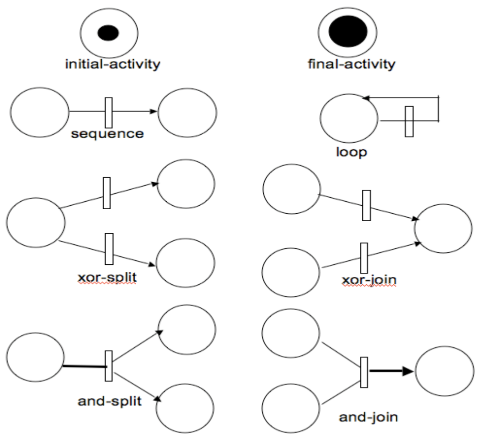

Figure 1. Workflow primitives

Using these primitives, arbitrary complex workflows are created defining the control flow between atomic activities in a workflow engine.

We define workflow engines using the coordination mechanisms of TuCSoN tuple centers and ReSpecT's reactive rules. The use of TuCSoN coordination mechanisms to coordinate agent's activities has been suggested in [20], where Omicini et al. show that TuCSoN's tuple centers can be deployed to work as workflow engines to coordinate agent's activities.

We extend this approach by identifying a set of generic operators for executing arbitrarily complex workflows. We also show how to link workflows that are executed in different tuple centers, thus ensuring modularity of workflow execution. The resulting logic-based approach is used to develop software agents that select and execute workflows, thus enabling for a distributed, automated and dynamic Workflow Management System (WfMS).

2.2 Architecture

Our agent-based architecture is based on a layered approach where every layer defines agent roles and infrastructural services to provide agents with what they need to achieve their goal. Four main roles are identified in our system:

– **Workflow Composition Agent (WFCA)**: The Workflow Composition Agent implements services for composing workflows according to its goals and plans. It specifies the needed services to the Workflow Selection Agents, building up workflow properties corresponding to these services.

– **Workflow Selection Agent (WFSA)**: The Workflow Selection Agent implements services for argumentation based on dialogues games. It has the ability to argue with other agents about known
workflows proposed as suitable for the service that the WFCA is searching.
– **User Agent (UA)**: The User Agent is the user representative, acting as intermediary between the user and
the system. It is able to use the services as composed by the WFCA, starting the workflow execution.
– Workflow Execution Agents (WFEA): The Workflow Execution Agent has the responsibility to execute workflow activities selected by WFSA agents.
These four agents are then distributed in three layers as shown in figure 2.

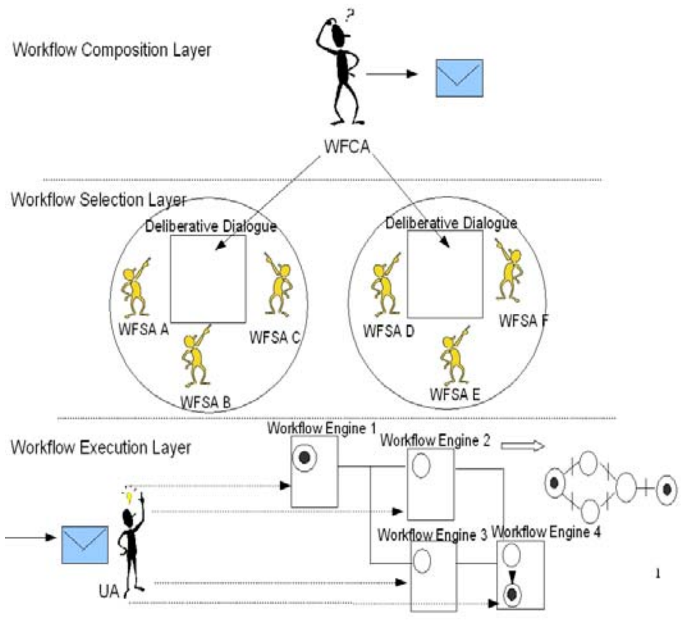

Figure 2. The layered MAS architecture

The first layer is the Workflow Composition Layer (WFCL). In this layer agents have internal goals and and generate plans to achieve these goals. The agents in the WFCL ask agents in the second layer, the **Workflow Selection Layer (WFSL)**, to find workflows with certain characteristics. We assume that WFCA has the ability to reason about which services are needed to achieve its goal. It is able to provide to the WFSL a workflow description scheme, which has a mapping with the workflow description file, indicating constrains over the workflow to be selected. A service may require the ordered execution of two or more workflows; the kind of ordering may vary and is often referred to as sequential, and-split, and-join, xor-split, and xor-join [29]. By providing the WFSL with a workflow scheme it constrains the structure of a particular workflow selected to fit in the general service composition. For example, the WFCA needs a service to buy a plane ticket and rent a car in the destination place. It constrains the WFSL to consider only car rental services offering the service in the destination place of the plane ticket company (this example will be explained with more detail in section 4).

In WFSL, agents use deliberative dialogues to select the workflow required from the WFCA. The WFSA is

an agent capable of interacting with other agents playing dialogue games and proposing workflows which satisfies the WFCA requirements. This type of agent argues using deliberative dialogues in order to share workflow knowledge, to find the best workflows given the workflow schema, to identify hidden dependencies or to build trust concepts over workflows e.g: an agent may express a preference between two workflows because one is more trusted.

The third layer is the **Workflow Execution Layer (WfEL)** and it executes the workflow previously selected and composed. In the framework, workflow engines are provided to coordinate workflow activities and allow run time linking with other workflow engines. The UA starts the required service by initiating the workflow execution and configuring the workflow engines. The linking conditions and the input structure of the workflow engine are provided by the WFCA to the UA. The workflow is designed as a set tasks distributed to WFEA-s which, by coordinating together, realizes a social goal.

3. WORKFLOW SELECTION LAYER

The particular deliberative dialogue utilized by the WFSA is described in [11] and implemented in [12]. The dialogue defined by McBurney is composed by a set of stages (*Open, Inform, Propose, Consider, Revise, Recommend, Confirm* or *Close*) in which it is possible to express a set of locutions (*open_dialogue, enter_dialogue, withdraw_dialogue, propose, assert, prefer, ask_justify, move* and *retract*) according to the current stage of the dialogue game.

The locutions allow agents to argument by expressing statements and the kind of proposition they are stating (e.g. question, action, goal, fact). The agents play the game in respect to the dialogue rules which states what moves (locutions) are possible to perform at certain stage.

Their moves determine the new dialogue state. The figure 3 represents the dialogue as a push down automata. Although it does not express all the possible path between stages or all the possible locutions that make the dialogue changing its state, it is a general picture of how the dialogue changes when two or more agents argue by using the locutions.

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 *Recommended* for execution. Afterwards, during the *Conform* 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.

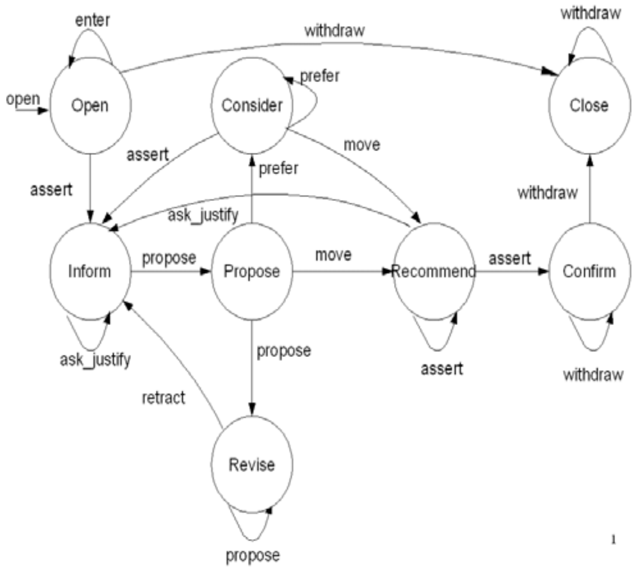

Figure 3. The Dialogue Game

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.

4. WORKFLOW EXECUTION LAYER

The agents in our system are developed using the PROSOCS platform presented in [26]. This platform is integrated with coordination artifacts which can be conceived as persistent entities specialized to provide services in Multi Agent Systems [17, 24] and used to model services for the social activities of agents. By encapsulating services inside coordination artifacts, we allow agents to abstract away from how the service is implemented. Both PROSOCS and the coordination artifacts that we use are implemented on top of TuCSoN [16], a coordination infrastructure which provides the reification of the coordination artifact concept. Coordination is based on the tuple center model, empowered with the ability to determine its behavior in response to communication events according to the specific coordination needs. Agents access tuple centers associatively by using simple communication operations such as assert (out), blocking reading (rd), blocking retract (in), retract (inp), and reading (rdp). The communication language between agents is tuple based [6]. The behavior of a tuple center can be modeled addressing the application needs by defining a set of specification tuples expressed in the ReSpecT language [15], which define how a tuple center should react to incoming communication events. A ReSpecT program takes the form of a set of reactions:

reaction (Event, [Guard], Body))

where ***Event*** is a tuple centre event, ***Guard*** is an optional sequence of predicates on the event properties, ***Body*** is a set of operations typically inspecting and changing the content of the tuple set, by inserting, retrieving or reading associatively tuples.

4.1 Modeling Dialogues Using ReSpecT

In particular, the deliberative dialogue defines a set of constructive rules to shape the social deliberative dialogue activity. Using ReSpecT it is possible to define a set of rules which capture how the dialogue state changes when a locution is made. The dialogue protocol can be described as a set of rules stating:

```
Deliberative Dialogue  Protocol:
        reaction(out(Locution1),
                when Conditions1 then Stage1).
        reaction(out(Locution2),
                when Conditions2 then Stage2).
        ...
        reaction(out(Locutionj),
                when Conditionsj then Stagej).
```

These rules describe the effects on the dialogue state and commitments due to the utterances of locutions by

the agents . Every locution made by the agents is maintained in the locution history. The transition of stages is also recorded. C*onditionsj* expresses how the dialogue progresses from a state to another according to the current stage and admissible moves. Table 1 shows one of the dialogue rules. The rule states that if any of the participants makes a *propose* locution and the dialogue has been in the *Open* stage, then the new dialogue state is *Inform* stage. The other rules for the deliberation dialogue game are defined similarly. For example, the dialogue cannot start from *Consider* stage if the participants have not firstly opened, shared information and exchanged proposal for actions.

Table 1. Example of a Dialogue rule in ReSPecT

OPEN TO INFORM
reaction(out(propose(AID,ID,Type,Q)), (in(propose(AID,ID,Type,Q)), X is propose(AID,ID,Type,Q), in(dialogue_history(ID,GQ,Sh,Lh)), isearch(open,Sh), append(Sh,[inform],NewSh), append(Lh,[X],NewLh), out(dialogue_history(ID,GQ,NewSh,NewLh)), out(dialogue_changed(ID)))) .

Once the dialogue goes through *Open, Inform, Propose* agents are free to access *Revise, Consider, Recommend, Confirm* and *Close* stage. If necessary, the agent may return to one of the three initial stages.

The Close stage concludes the workflow selection. The dialogue's progression from one stage to another depends on the history of the stages and on the history of the locutions. We use commitment stores [9, 30] to track locutions that create obligations between agents. This allows the agent to reason about the expectations it has about others and what others expect of it. When agents communicate a locution that commits an agent, the ReSpecT rules update the Commitment Store. The locutions that creates commitments are the *assert* locution when its type is *question, goal, constraint, perspective, fact* or *evaluation*, the *move* locution and the *retract* locution. In the retract locution, the agent indicates an *assert* or *move* or *prefer* locution to be removed from the commitment store. Although the locution will be removed, the trace of the locutions retracted will remain in the locution history. The agents can inspect the Commitment Store and they will have all a coherent view of each participant's obligations**.**

4.2 Modeling WFMS Using ReSpecT

The deliberative dialogues rules model a protocol for the agents' communication, and coordinates agents during their workflow selection. Once the selection occurs the workflow should be executed in the specified order. This section explains how coordination artifacts can be specialized as workflow engines to provide coordination of workflow activities.

It is possible to model two levels of workflow coordination rules. The first level specifies the coordination of a single workflow, seen as an atomic service (e.g. booking a ticket). As described in [23], the ReSpecT rules embedded in tuple centers are an alternative way to model workflow engines which coordinates workflow activities. The coordination artifact orchestrates the activities of the workflow by telling agents for which activities they are responsible.

As shown in Table 2 the workflow procedure starts with a start(I) event whose reactions generate the first activity by writing in the tuple space the next activity denoted by next(A1). Once an activity Ai is completed, the event completed(Ai) must happen, which in turn activates new reaction rules. The above specification requires that the implementation of workflow execution must take care of the synchronization of activities and shared variables (e.g. in and join) or mutual exclusion (e.g. in xor split).

Table 2. Execution of workflow A1A2. . .An(I, O). The terms A1,A2,. . .An denote workflow activities whose execution produce output O when supplied with input I.

Activity Relation	**Event**	**ReSpecT Expression Rules**
Initial	start(I)	when Conditions then next(A1)
Final	completed(An)	when Conditions then result(O)
Loop	completed(Ai)	when Conditions then next(Ai)
	completed(Ai)	when not Conditions then next(Ai+1)

Sequential	completed(Ai)	when Conditions then next(Ai+1)
And_join	completed(Ai)	when contains(synchronize([Ai,1, . . . ,Ai,j]), Ai) and Conditions then update(Completed)
	completed(Ai)	when Completed is [Ai,1, . . . ,Ai,j] then next(Ai+1)
Xor_join	completed(Ai,1)	when Conditions then next(Ai+1)
	completed(Ai,2)	when Conditions then next(Ai+1)
	...	....
	completed(Ai,j)	when Conditions then next(Ai+1)
And_split	completed(Ai)	when Conditions then next(Ai,1) next(Ai,2) . . . next(Ai,j)
Xor_split	completed(Ai)	when Conditions1 then next(Ai,1)
	completed(Ai)	when Conditions2 then next(Ai,2)
	...	...
	completed(Ai)	when Conditionsj then next(Ai,j)

The second level of workflow coordination rules enables the linkability between distributed workflow engines. These rules provide coordination between worklfow procedures that enable the linkability between distributed WfEs to support workflow composition. It is possible to express linking conditions between workflow engines by configuring them with precondition and postcondition rules. The idea is to manage the workflow engines by providing them with local vision about how they are related (e.g. sequential, xor_join, and_join) with other workflow engines.

We express linkability between WfEs by assertions of the form: wfe link (Current, Previous, Next). Assertions of this kind state that when the Previous WfE has reached its final state the Current WfE must start, which after it finishes, the Next WfE should be informed to continue for the composite workflow to complete. Thus, we coordinate WfEs by providing them with a local view of how they are related with each other. We also expect that the parameters Previous and Next to be supplied as follows:

Previous ::= initial | sequence(WfE) | and join(WfEs) | xor join(Conditions, WfEs).

Next ::= final | sequence(WfE) | and split (WfEs) | xor split (Conditions, WfEs).

For instance, the figure 4 shows a set of 6 workflow engines (from WfE0 to WfE5), where there is an and_split between WfE1 and WfE2, WfE3, and an and_join between WfE2,WfE3 and WfE4 .

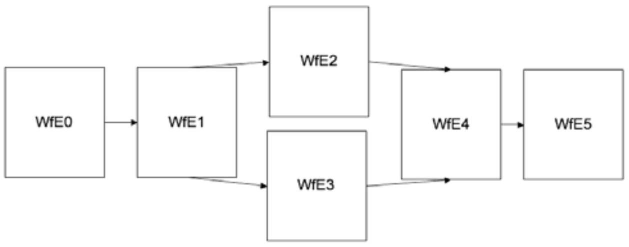

Figure 4. Workflow Engines Linking

In this case, the set of tuples to configure every workflow engines would be respectively:

WfE0: wfe_link(WfE0, initial, sequence(WfE1)).
WfE1: wfe_link(WfE1, sequence(WfE0), and_split([AddrWfE2,WfE3])).
WfE2: wfe_link(WfE2, sequence(WfE1), sequence(WfE4)).
WfE3: wfe_link(WfE3, sequence(WfE1), sequence(WfE4)).
WfE4: wfe_link(WfE4,and_join([WfE2,WfE3]), sequence(WfE5)).
WfE5: wfe_link(WfE5, sequence(WfE4), final).

And a set of similar rules as defined in table 2 are are defined to link the different WfE. These reactive rules make use of the ReSpecT operation out_tc defined to enable the communication between two tuple centers.(See [20] for more details).

5. CASE STUDY

To clarify some of the ideas discussed in the previous sections, we exemplify our approach by considering a scenario for the dynamic composition of workflows. In particular we will consider a simple example where a user requests, via the User Agent, to buy a plane ticket to a certain destination and to rent a car in this destination. The WFCA that receives the request has no suitable workflow in its own library to satisfy the user's goal, so it requests to the WFSL to find two workflows fulfilling the user's requirements. The requirements are expressed using a Workflow Description Format as in table 3 and their main purpose is to constrain and to help the WFSL during the argumentation. The figure shows the workflow description scheme provided to the WFSL for the flight ticket workflow selection, the workflow description scheme for the car rental workflow selection will be similar.

Table 3. The workflow description scheme

```
<?xml version=" 1 . 0 " encoding="ISO?8859?1" ?>
<Workflow Description>
<Description>plane ticket</Description>
<Name> </Name>
<Address> </Address>
<Input>  input (Name, Customer, Ticket, Seller,Carrier) </ Input>
```

```
<Output></Output>
<ServiceCost>  0    </ ServiceCost>
<Availability>   99 </ Availability>
<Reliability>     98 </ Reliability>
<Trust>            90  </Trust>
</Workflow Description>
```

5.1 Simple Workflow

We exemplify how we use the logic rules illustrated in section 2 with the workflow book plane ticket workflow procedure depicted in figure 5. In this example a user requests a plane ticket. This generates a query on the availability of the ticket. It notifies the user and terminates if it does not find results, otherwise it generates the booking of the ticket. Afterwards a payment is required, if the requester of the service authorize the payment, a payment request is generated, otherwise the booking is made invalid and the user is notified and the procedure is terminated. If the payment procedure terminates successfully a ticket is send to the user which is notified and the procedure terminated.

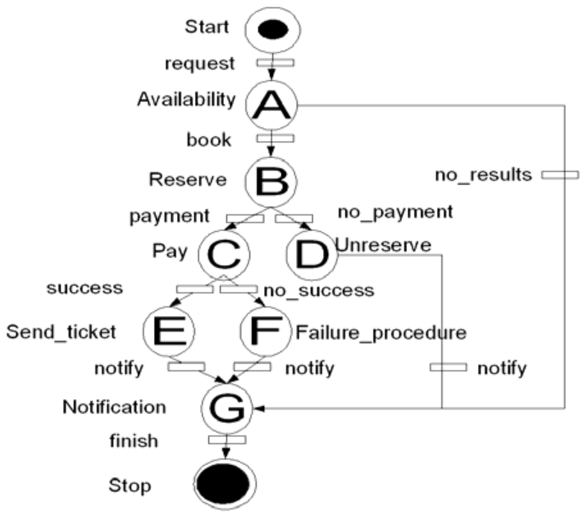

Figure 5. Book Plane Ticket Workflow Procedure

The following listing shows how we specify the logic rules for the start of this procedure and an and split procedure using ReSpecT rules. The procedure starts with a request(Input) made by a user agent or as part of the linking of the workflow engines.

```
START
reaction(out(start(request(Input))),(
          check_input(Input, Data),
          generate_id(Input,Id),
          out(wf(Id)),
          out(next(availability(ID,Data))) )).
AND_SPLIT
reaction(out(completed(availability(ID,Data))),(
          in(completed(availability(ID,Data))),
          out(next(reserve(ID,Data))),
          out(next(notification(ID,Data))) )).
```

5.2 Dynamic Selection of Workflows

As a result, two dialogues are opened in two diffent tuple centres. The table 4 shows a dialogue example between three agents: agentA, agentB and agentC. The agentA, perceiving the request for a plane ticket workflow selection, opens a dialogue with governing question *"plane_ticket"*. The other agents enter the dialogue by using an enter_dialogue locution. Once the dialogue is opened and some constrains are proposed, the agents propose two different workflows suitable to satisfy the requirements. At the end the three agents agree for the first workflow, which better satisfies the requirements of the WFCA.

Table 4. A dialogue example

1. open_dialogue (agentA, ID, plane_ticket)
2. enter_dialogue (agentB, ID, plane_ticket)
3. enter_dialogue (agentC, ID, plane_ticket)
4. propose (agentA, ID, constraint , service time)
5. propose (agentC, ID, constraint , cost)
6. propose (agentB, ID, action , buyticket1.txt)
7. propose (agentC, ID, action, buyticket2.txt)
8. ask_justify (agentA, ID, agentB, buyticket1.txt)
9. assert (agentB, ID, fact, Service Time=2)
10. prefer (agentC, ID, action , buyticket1.txt, buyticket2.txt)
11. move (agentA, ID, buyticket1.txt)
12. assert (agentB, ID, buyticket1.txt)
13. assert (agentC, ID, buyticket1.txt)
14. withdraw (agentA, ID, plane_ticket)
15. withdraw (agentB, ID, plane_ticket)
16. withdraw (agentC, ID, plane_ticket)

Once the workflows are selected, the workflow description files are delivered to the WFCA which confer

with the UA for their execution. The WFCA knows that the workflows should be executed sequentially and as a consequence it provides the UA with the instructions about how to set the workflows engines in order to execute the workflows. In other words, the WFCA, once the selection layer deliver to him the two workflow execution, sends the following message to the UA:

Message: user agent message[case to start(plane ticket, Pre, Post, Address1, Address2, Input), case to start(car rental, Pre, Post, Address2, Address2, input)]

The Pre and Post are the linking conditions as described in the sections above, Address1 is the address of the workflow (the address of plane ticket workflow engine in the first case and the address of car rental workflow engine in the second case) Address2 are the addresses of the activities indicated in the postcondition (here it is the address of the car rental workflow engine), Input is the input in order to start the new case.The plane ticket and car rental workflow coordinated by the two workflow engines is simplified in three activities to execute in parallel. When the plane ticket workflow starts three sequential tasks are generated for the WFEA-s (reservation, dispatch ticket and payment). When the execution of these activities conclude, the car rental workflow provides the agents with similar activities (reservation, dispatch the car receipt and payment).

6. RELATED WORK

Combining service-oriented computing and architectures with software agents is an active area of research for intelligent systems [21]. More specifically, current visions of web-services and agent computing predict important implications in the engineering of complex distributed systems [7] in general and GRID [5] and ubiquitous [8] computing in particular. A large part of this effort focuses on the service composition problem [22], where a computational logic approach is playing an important role, for example see McIlraith and colleagues [13,14], Baldoni et al [2], and Lomuscio et al [10].

The advantage of using dialectical argumentation, compared with approaches that do not, is that agents can provide supporting arguments for selecting a service, thus being in a position to provide reasons about why a particular service has to be selected instead of another.

The use of coordination artifacts for the definition of distributed workflow is not new: in [20] Omicini et al. propose a framework where distributed coordination artifacts coordinates the activity of a MAS to provide a distributed workflow management system. The difference between [20] and our approach is that we consider a layered architecture where workflows are dynamically composed using dialectical argumentation according to the preferences of a user.

A similar approach to the one of coordination artifacts is the one based on games proposed by Stathis in [25,27,28], where agents play a complex logical interaction protocol on top of an umpire agent that keeps the state of the game between two participants. The added value of our approach with respect to [25, 27,28] is that we have a distributed approach where we can link multiple societies of argumentative agents playing complex protocols in order to fulfill a workflow proposed by a workflow composition agent. In the second place our approach based on coordination artifacts allows us to have a society of agents playing a collaborative argumentation dialogue, so in a certain way our approach tries to generalise the approach presented in [25, 27, 28] by introducing the concept of coordination artifact.

From the infrastructure point of view, our approach is similar to the one proposed by Electronic Institutions [4], where the activities inside a multiagent environment are seen in terms of multiple, concurrent dialogical activities. For every activity the interaction between a group of agents happens inside scenes, that allows well-defined communication protocols. Multiple connected scenes defines a performative structure in which the norms define the commitments, obligations and rights of the participant agents as well as defining the rules of transition between one scene to another.

In our approach the scenes can be seen as the societies of agents belonging to the Workflow Selection Layer, and the performative structure can be seen as the workflow proposed by the workflow composition layer, while the norms and protocols are defined by the respect theories within every society of the Workflow Selection Layer. The added value of our approach with respect of EI is the fact that our infrastructure allows agents to play collaborative dialogues using dialectical argumentation. At the same time, the outcome of the deliberative dialogues can be composed at the Workflow Composition Layer and instruct the Workflow Execution Layer to execute the overall workflow in a dynamic coreography created by linking the coordination artifacts of the Workflow Execution Layer.

7. CONCLUSIONS

We have presented a multi-agent systems framework based on argumentative agent technology for the automation of the workflow selection and execution. In this framework, workflow selection has been coordinated by agent interactions governed by the rules of a dialogue game whose purpose has been to evaluate the workflow's properties via argumentation. When a workflow has been selected using this process, the workflow has been executed by dynamically configuring workflow engines that in turn coordinate the participating agents' workflow activities. We have further exemplified our approach by showing how the framework can be instantiated for a concrete example application implemented using the TuCSoN infrastructure and its associated ReSpecT language. The use case described is simplified in order to explain the concepts our approach. However, the framework is designed to execute arbitrarily complex workflows. This includes issues such as conflict amongst agents, which would be handled during the dialogue game. If the overall system required a more competitive agent system (rather than the collaborative one described), it would only require a different protocol such as a negotiation protocol. The resulting execution of the workflow would remain the same.

Future work will consider the incorporation of standardized workflow languages. This will allow the workflow selectors can incorporate trust policies related to feedbacks received from the other system which use this workflows. More specifically, the user agent could propose modifications on the workflow description according to a feedback received from the user. Finally, another future development regards the possibility to consider different kind of dialogues in addition to the deliberative one, according to the kind of workflow the agent are trying to compose. In particular, agents could perform a persuasive dialogue when a conflict of point of view exists between them.

ACKNOWLEDGMENTS

The work was supported by the EU IST-6 ArguGRID project.

REFERENCES

1. Amgoud, L, Maudet, N and Parsons S, 2000, "Modeling Dialogues Using Argumentation", *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, p 31.
2. Baldoni, M, Baroglio, C, Martelli, A, and Patti,V, 2004, "Reasoning About Interaction Protocols for Web Service Composition", *Electr. Notes Theor. Comput. Sci.* 105, pp. 21–36.
3. Curcin, V, Ghanem, M, Guo, Y, Stathis, K and Toni, F, 2006, "Building next generation Service-Oriented Architectures using Argumentation Agents", *3rd International Conference on Grid Service Engineering and Management*, Germany, pp 249-263.
4. Esteva, M, Rodriguez-Aguilar, J. A, Sierra, M, Garcia, P and Arcos. J.L, 2001, "On the formal specifications of electronic institutions", *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, Springer-Verlag, London, UK, pp 126–147.
5. Foster, I. T, Jennings, N. R, and Kesselman, C, 2004, "Brain Meets Brawn: Why Grid and Agents Need Each Other", *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004),* New York, NY, USA, pp. 8–15.
6. Gelernter, D, 1985, "Generative communication in Linda", *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112.
7. Ghanem, M., N. Azam, M, Boniface, and Ferris, J, 2006, "Grid-enabled workflows for industrial product design", *Proc. of the 2nd IEEE International Conference on e-Science and Grid Computing (e-Science'06).*
8. Huhns, M. N, Singh, M. P, Burstein, M. H, Decker, K. S, Durfee, E. H, Finin, T. W, Gasser, L, Goradia, H. J, Jennings, N. R, Lakkaraju, K, Nakashima, H, Parunak, H. V. D, Rosenschein, J. S, Ruvinsky, A, Sukthankar, G, Swarup, S, Sycara, K.P, Tambe, M, Wagner, T, and Gutierrez, R. L. Z, 2005, "Research Directions for Service-Oriented Multiagent Systems", *IEEE Internet Computing* 9(6), p.p 65–70.
9. Levesque, H.J, Cohen, P.R and Nunes, J. H.T, 1990, "On acting together", *In Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90),* Boston, MA, pp. 94-99.
10. Lomuscio, A, Qu, H, Sergot, M. J, and Solanki, M, 2007, 'Verifying Temporal and Epistemic Properties of Web Service Compositions'. In: B. J. Krämer, K.-J. Lin, and P. Narasimhan (eds.): *Service-Oriented Computing - ICSOC 2007, Fifth International Conference,* Vol. 4749 of Lecture Notes in Computer Science. Vienna, Austria, pp. 456–461.
11. McBurney, P, Hitchcock, D and Parsons, S, 2007, "The eightfold way of deliberation dialogue", *International Journal of Intelligent Systems*, 22(1), pp. 95–132.
12. McGinnis, J, Bromuri, S, Urovi, V, Stathis, K, 2007, "Automated Workflows Using Dialectical Argumentation", *German e-Science Conference,* Germany, to appear.
13. McIlraith, S. A. and Son,T. C, 2002, "Adapting Golog for Composition of Semantic Web Services", D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.A. Williams (eds.): *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02).* Toulouse,France, pp. 482–496.
14. McIlraith, S. A, Son, T. C, and Zeng, H, 2001, 'Semantic Web Services'. IEEE Intelligent Systems 16(2), 46–53.

15. Omicini, A and Denti, E, 2001, "Formal ReSpecT", *Electronic Notes in Theoretical Computer Science*, 48.
16. Omicini, A and Denti, E, 2001, "From tuple spaces to tuple centres",*Science of Computer Programming*, vol. 40, n.2.
17. Omicini, A, Ricci, A, Viroli, M, Castelfranchi, C, Tummolini, L, 2004, "Coordination artifacts: Environment-based coordination for intelligent agents", *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004),* New York, USA, Volume 1, pp. 286-293
18. Omicini, A, Ricci, A and Viroli. M, 2006, "Coordination artifacts as first-class abstractions for MAS engineering: State of the research", *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications,* volume 3914 of LNAI, pp 71–90.
19. Omicini, A, Ricci, A, and Zaghini, N, 2008, "Artifacts in the A&A Meta-Model for Multi-Agent Systems", *Autonomous Agents and Multi-Agent Systems 17(3)*, pp. 432-456.
20. Omicini, A, Ricci, A and Zaghini, N, 2006, "Distributed Workflow upon Linkable Coordination Artifacts", *Lecture Notes in Computer Science : Coordination Models and Languages*, pp. 228-246.
21. Payne, T. R, 2008, "Web Services from an Agent Perspective", *IEEE Intelligent Systems* 23(2), pp. 12–14.
22. Rao, J. and Su,X, 2005, "A Survey of Automated Web Service Composition Methods", *Semantic Web Services and Web Process Composition*, Vol. LNCS 3387/2005. Springer, pp. 43–54.
23. Ricci, A, Omicini, A, Denti, E, 2001, "Agent Coordination Infrastructures for Virtual Enterprises and Workflow Management", *Cooperative Information Agent V*, Modena, Italy, 2182, p 235.
24. Ricci, A, Viroli, M and Omicini, A, 2005, "Environment-Based Coordination Throught Coordination Artifacts", *Book Series Lecture Notes in Computer Science*, Volume 3374, pp. 190-214.
25. Stathis, K, 2000, "A game-based architecture for developing interactive components in computational logic", *Journal of Functional and Logic Programming*,(5).
26. Stathis, K, Kakas A, C, Lu, W, Demetriou, N, Endriss, U and Bracciali, A, 2004, "PROSOCS: a platform for programming software agents in computational logic", *Proceedings of the Fourth International Symposium "From Agent Theory to Agent Implementation"*, Vienna, Austria, pp 523-528.
27. Stathis, K, Lekeas, G, Kloukinas, C, 2006, "Competence checking for the global e-service society using games", *ESAW*, pp. 384–400.
28. Stathis, K,Sergot, M. J, 1996, "Games as a metaphor for interactive systems", *in: BCS HCI*, pp. 19–33.
29.Van der Aalst, W, 1998, "The application of petri nets to workflow management",*The Journal of Circuits*, p 21–66.
30. Walton, D and Krabbe, E, 1995, "*Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*", Suny, USA