EXPERIENCES IN AUTOMATED WORKFLOWS USING DIALECTICAL ARGUMENTATION

Visara Urovi* , Stefano Bromuri**, Jarred McGinnis**, Kostas Stathis** and Andrea Omicini*
*Second Faculty of Engineering. University of Bologna, Via Venezia 52, 47023 Cesena(FC), Italy
**Royal Holloway University of London, McCrea Building, Egham, Surrey, TW200EX, United Kingdom
email: visara@cs.rhul.ac.uk
phone: +44 (0) 1784443696

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.

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 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 the most appealing 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 [11]. The ability to discuss the workflow using deliberative dialogue games [5] and argumentation [2] is a promising approach for workflow participant 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.

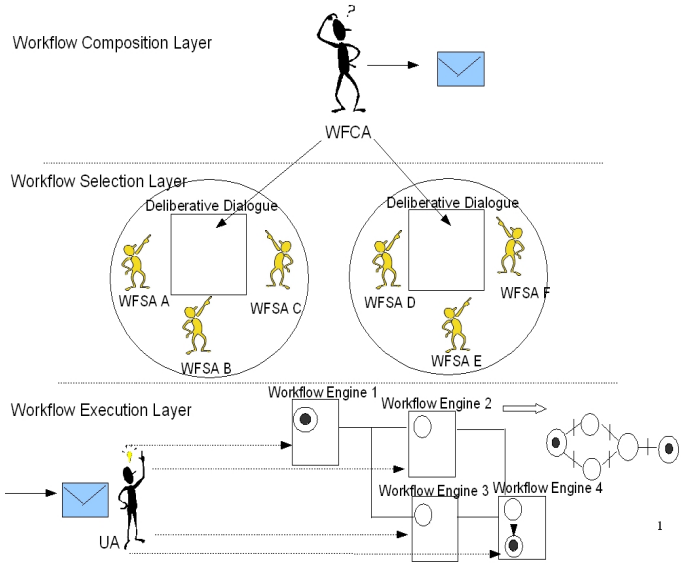
2. APPROACH

In our approach we utilize the Agents and Artifacts meta-model[10] , 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 naturally play the role of workflow execution engines [12]. 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 [8] to provide tuple centers as coordination artifacts and workflow execution engines, the PROSOCS agents platform [14], and dialogue games [15,6] 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 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 1.

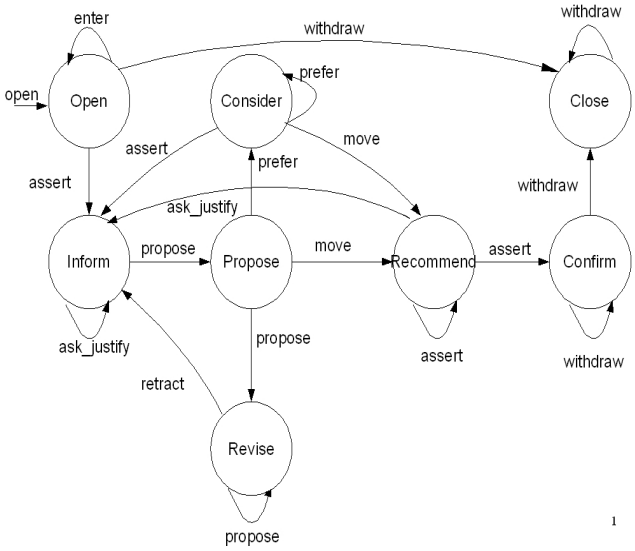
Figure 1. The layered MAS architecture



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 [15]. 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.

Figure 2. The Dialogue Game



The particular deliberative dialogue utilized by the WFSA is described in [5] and implemented in [6]. 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 2 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 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. MODELING THE WORKFLOW SELECTION LAYER AND THE WORKFLOW EXECUTION LAYER

The agents in our system are developed using the PROSOCS platform presented in [14]. This platform is integrated with coordination artifacts which can be conceived as persistent entities specialized to provide services in Multi Agent Systems [9,13] and used to model services for the social activities of agents. By encapsulating services inside coordination artifacts, we allow agents to abstract from how the service is implemented. Both PROSOCS and the coordination artifacts that we use are implemented in TuCSoN [8], 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 [3]. 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 [7], 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, (Body))

where ***Event*** is a communication event and ***Body*** is a set of primitives which create the possibility to inspect and change the content of the tuple set, by inserting, retrieving or reading associatively tuples.

3.1 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_r (propose (AID, ID , Type ,Q)) ,
X is propose (AID, ID , Type ,Q) ,
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)))) .

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 [4] 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**.**

3.2 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

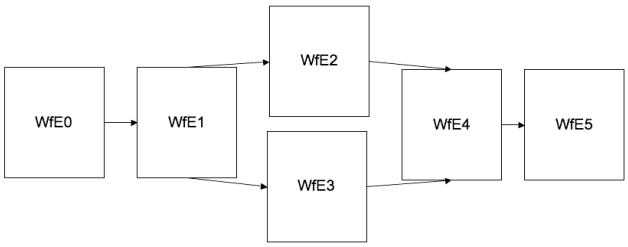
order. This section explains how coordination artifacts can be specialized as workflow engines to provide coordination of workflow activities.
It is possible to model two levels of workflow coordination rules. The first level specifies the coordination of a single workflow, seen as an atomic service (e.g. booking a ticket). As described in [12], 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. The workflow defines a set of activities and relations between them. In particular, we distinguish between initial, final, loop, sequence, and_join, xor_join, and_split, xor_split relations between activities. Table 2 demonstrates our translation from the theoretical activity relations of workflows to their implementation as ReSpecT expression rules.

Table 2. The table express how relationships between workflow activities can be captured by using the ReSpecT rules

Activity Relations	**ReSpecT Expression Rules**
Initial - The first activity of the workflow.	**reaction(Input,** when **Preconditions** then **next(A0)**)
Final - The last activity of the workflow.	**reaction(completed(An-1),** when **Conditions** then **Output**)
Loop - The execution of an activity certain number of times while the Conditions hold.	**reaction(completed(A),** when **Conditions** then **next(A)**) **reaction(completed(A),** when **not Conditions** then **next(B)**)
Sequential - An activity B is sequential to the activity A when B is executed after A.	**reaction(completed(A),** when **Conditions** then **next(B)**)
And_join - synchronize two or more parallel flows. E.g: A, B and C must be synchronized in D, when A, B or C are completed the conditions change accordingly. D starts when all of the 3 activities are completed.	**reaction(completed(X),** when **contains(synchronize([A1,A2,..., An]), X)** **and Conditions** then **update(ListCompleted)**) **reaction(completed(X),** when **ListCompleted is [A1,A2,...,An]** then **next(D)**)
Xor_join - none of the alternative branches is executed in parallel. A, B or C are completed, according to the conditions only one of them activates D.	**reaction(completed(A),** when **Conditionsi** then **next(D))** **reaction(completed(B),** when **Conditionsj** then **next(D)**) **reaction(completed(C),** when **Conditionsk** then **next(D)**)
And_split - two or more concurrent activities are executed in parallel.	**reaction(completed(A1),** when **Conditions** then **next(A2)** **next(A3)** **...** **next(An)**)
Xor_split - The next activity is chosen based on which condition becomes true . Mutual exclusion between conditions has to be provided.	**reaction(completed(A1),** when **Conditionsi** then **next(A2))** **reaction(completed(A1),** when **Conditionsj** then **next(A3)**) **...** **reaction(completed(A),** when **Conditionsk** then **next(An)**)

The second level of workflow coordination rules enables the linkability between distributed workflow engines. These rules promote the possibility to perform workflow composition by building up a more complex one. 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.
For instance, the figure 3 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 3. Workflow Engines Linking



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

WfE0: *linking_conditions(WfE0, initial activity, sequential(WfE1))*
WfE1: *linking_conditions(WfE1, sequential(WfE0),and_split([WfE2,WfE3]))*
WfE2: *linking_conditions(WfE2, sequential(WfE1), sequential(WfE4))*
WfE3: *linking_conditions(WfE3, sequential(WfE1), sequential(WfE4))*
WfE4: *linking_condition(WfE4,and_join([WfE2,WfE3]), sequential(WfE5))*
WfE5: *linking conditions(WfE5, sequential(WfE4), final activity)*

4. 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>
```

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).

5. CONCLUSION

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.

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. 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.

3.Gelernter, D, 1985, "Generative communication in Linda", *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112.

4. 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.

5. McBurney, P, Hitchcock, D and Parsons, S, 2007, "The eightfold way of deliberation dialogue", *International Journal of Intelligent Systems*, 22(1), pp. 95–132.

6. McGinnis, J, Bromuri, S, Urovi, V, Stathis, K, 2007, "Automated Workflows Using Dialectical Argumentation", *German e-Science Conference*, Germany, to appear.

7. Omicini, A and Denti, E, 2001, "Formal respect", *Electronic Notes in Theoretical Computer Science*, 48.

8. Omicini, A and Denti, E, 2001, "From tuple spaces to tuple centres",*Science of Computer Programming*, vol. 40, n.2.

9. 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

10. 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.

11. 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.

12. 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.

13. 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.

14. 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.

15.Van der Aalst, W, 1998, "The application of petri nets to workflow management",*The Journal of Circuits*, p 21–66.

16. Walton, D and Krabbe, E, 1995, "*Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*", Suny, USA