

## DYNAMIC PLUGGING OF BUSINESS PROCESSES IN CROSS-ORGANIZATIONAL WORKFLOW

LOTFI BOUZGUENDA

*MIR@CL/ISIMS, Institut Supérieur d'Informatique et Multimedia,  
Route de Tunis, km 10, BP 242, 3021 Sakeit Ezzit, Sfax, Tunisia  
Lotfi.Bouzuenda@isimsf.rnu.tn*

RAFIK BOUAZIZ

*MIR@CL/Faculté de Sciences Economiques et Gestion,  
Route de l'aéroport, km 4, BP 1088, 3018, Sfax, Tunisia  
Rafik.Bouaziz@fsegs.rnu.tn*

ERIC ANDONOFF

*IRIT/UT1, 2 rue du Doyen Gabriel Marty  
31042 Toulouse Cédex, France  
Eric.Andonoff@univ-tlse1.fr*

This paper shows how it is possible to use agents and ontologies to deal with dynamic coordination of business processes in Cross-Organizational Workflow (COW). The aim of COW is to support the cooperation between distributed and heterogeneous business processes running in different autonomous organizations to reach a common goal, corresponding to a value-added service. In this paper, we consider a particular case of COW called dynamic COW in which the different partners (organizations) involved in the COW are not necessarily known before its execution either because they are unknown at design-time or no more available at run-time. Consequently, coordination in dynamic COW raises two specific problems, which are: (i) partners finding able to realize an COW service (i.e. a service implementing a business process involved in an COW), and (ii) negotiation of COW services between partners. This paper first defines an agent-based architecture to support COW service execution. Then, it presents two specific agent-based mediation infrastructures introduced to deal with partners finding and negotiation between them and shows, for each of these infrastructures, how they use ontologies for automatically discovering COW services and choosing protocols to negotiate COW services.

*Keywords:* Cross-Organizational Workflow; Coordination; Agent; Ontology.

### 1. Introduction

The aim of Cross-Organizational Workflow (COW) is to support the cooperation among distributed and heterogeneous business processes running in different autonomous organizations. The different organizations involved in an COW need to put resources and skills in common, and coordinate their respective business processes in order to reach a common goal, corresponding to a value-added service. Thus, COW is a key technology for helping participating organizations to face the emergence of the open and dynamic worldwide economy [van der Aalst (1999)].

A fundamental problem for COW is the coordination of the different processes. By coordination, we mean all the work needed to put all these processes together in order to provide the global common goal in an efficient manner. This coordination problem has been deeply investigated in a static context [van der Aalst (1999)], [Chebbi *et al.* (2006)] but it remains open in a dynamic (loose [Divitini *et al.* (2001)]) context for which the different partners (organizations) of the COW are not necessarily known before its execution either because they are unknown at design-time or no more available at run-time. So, the question is: *how to proceed if an organization responsible for the execution of a COW service, i.e. a service implementing a business process involved in the COW, is unknown or no more available at run-time?* Figure 1 illustrates this situation. In this example, Org1 is responsible for the execution of an COW (described using a Petri-net based formalism) and two COW services, each representing a part of the COW, are delegated to external organizations involved in it: Org2 is responsible for the execution of the first COW service while it is necessary to find an organization able to implement the second COW service.

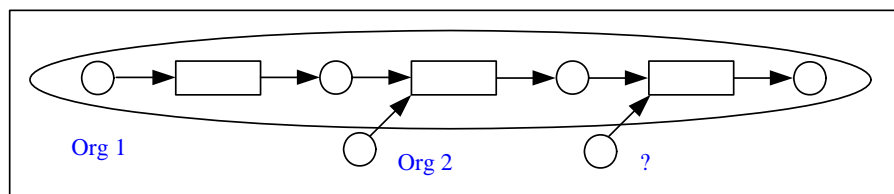


Fig. 1. Dynamic Coordination in COW.

In this paper, we have addressed the problem of coordination in dynamic COW. This problem raises several specific sub-problems, which are: (i) partners finding able to realize COW services, (ii) negotiation of COW services between partners, (iii) contract specification between partners, and (iv), distributed execution of COW services. More precisely, in this work, we have focused on the two first coordination problems. Partners finding consists in selecting one or several provider organizations able to execute an COW service, which is needed for a requester organization. After the partners finding step, a requester organization is connected to different provider organizations. But connecting is not enough to definitively choose the partner that is going to execute the COW service: a negotiation step, in terms of due time, price, visibility of the service evolution, and way of executing it, is necessary to evaluate and select the most suitable provider organization. Then it is possible to select the organization that is going to realize the requested COW service. However, we also advocate a contract specification step in order to formalize the interaction between the two partners: the requester and the provider. Finally the last step supports the execution of the COW service. Of course, it ensures that the rules defined in the specified contract are satisfied. In this paper we focus on the two first coordination problems i.e. partners finding and negotiation.

The paper also relies on the exploitation of agent and ontologies approaches, viewed as enabling technologies to deal with these coordination problems.

On the one hand, the agent approach provides natural abstractions to deal with distribution, autonomy and heterogeneity, which are inherent to dynamic COW. As illustrated in figure 2, each COW service may be seen as an autonomous agent having the mission to coordinate with other agents, each one representing the different COW services involved in the COW and acting on behalf of its organization. Adopting this view, a dynamic COW is no longer a set of coordinated business processes but a set of coordinated agents.

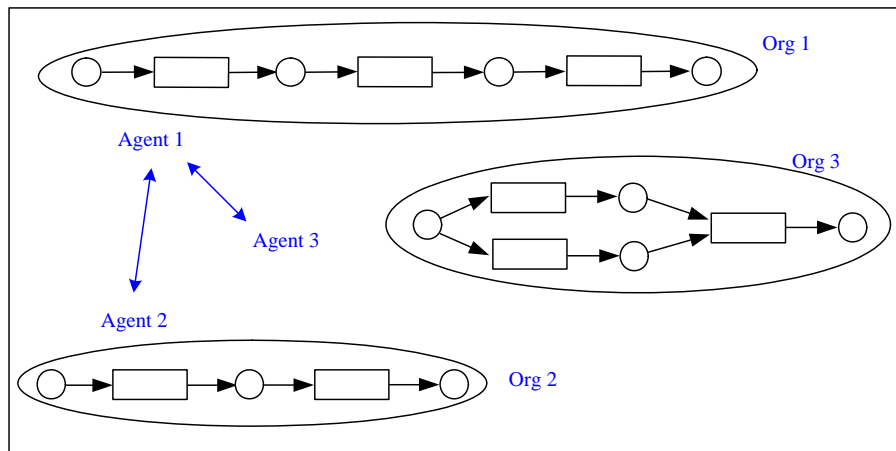


Fig. 2. Agent-based Approach for Coordination in Dynamic COW.

Moreover, using this approach, we also inherit numerous concrete solutions to deal with dynamic coordination in multi-agent systems [Jenning *et al.* (2001)], [Wooldridge (2002)], [Ferber *et al.* (2003)]: middleware components such as matchmakers or brokers, sophisticated interaction protocols supporting negotiation between agents, and finally organizational concepts, such as groups, roles or commitments, which are useful to structure and rule agent coordination at a macro level.

On the other hand, we use ontologies in order to deal with semantic heterogeneity problems, which obviously occur in dynamic COW where involved organizations are autonomous and involved business processes are distributed and heterogeneous. Indeed, ontologies are used in various domains of computer science such as knowledge engineering, information systems, multi-agent systems or semantic Web. Today, ontologies have taken an important place because they allow the sharing of knowledge to solve semantic heterogeneity problems that exist in distributed applications of these domains. For instance, ontologies are used in Information Retrieval to support the indexing mechanism [Hernandez (2007)], in Semantic Web for the composition of web services [Medjahed *et al.* (2003)] or in multi-agent systems to support automatic negotiation between agents [Tamma *et al.* (2005)]. In this paper, we claim that ontologies also play an important role in dynamic COW: in fact, they contribute to make automatic coordination in such a context possible.

The contribution of this paper is threefold. First, the paper defines an agent-based architecture to support COW service execution. Then, the paper presents two specific agent-based mediation infrastructures introduced to deal with partners finding and negotiation between them. These infrastructures also include specific ontologies for automatically discovering COW services and choosing protocols to negotiate COW services. This paper also presents these ontologies and shows how the two mediation infrastructures use them to deal with partners finding and negotiation between partners.

The remainder of this paper is organized as follows. Section 2 presents the agent-based architecture we have proposed for COW service execution. More precisely, it shows how we have modified the WfMC (Workflow Management Coalition) reference architecture in order to communicate with the mediation infrastructures supporting partners finding and negotiation between them. Sections 3 and 4 are respectively devoted to these infrastructures. These sections first present the corresponding infrastructures, then describe their dynamics using AUML sequence diagrams and finally show how these infrastructures use the ontologies, on the one hand, to support the publication of COW service offers and requests and the comparison between these offers and requests and, on the other hand, to choose a type of negotiation protocol during the negotiation step. Section 5 briefly presents the MachFlow and NegoFlow prototypes, which implement these mediation infrastructures and their associated ontologies. Finally, section 6 compares our contribution to related works and concludes the paper.

## **2. An Agent-based Architecture for COW Service Execution**

This section presents the agent-based architecture we propose for COW service execution. It introduces the Workflow Management Coalition (WfMC) reference architecture, which is the starting point of our proposition, and explains why and how we revisit it with agents in order to deal with partners finding and negotiation between them.

### ***2.1. Insufficiency of the Reference Architecture***

The starting point of this proposition is the reference architecture of a Workflow Management System (WfMS) defined by the Workflow Management Coalition [Workflow Management Coalition (1994)].

This reference architecture is shown in figure 3. It is defined by giving the role of its software components and by specifying how they interact. The main component of this architecture is the Workflow Enactment Service (WES) that manages the execution of workflow processes, and that interacts, with workflow definition, execution and monitoring components, and, also with external WES. The five interfaces supporting the communication among the different components are:

- Interface 1 with Process Definition Tools,
- Interface 2 with Workflow Client Applications,
- Interface 3 with Invoked Applications,
- Interface 4 with others WESs,
- Interface 5 with Administration and Monitoring Tools.

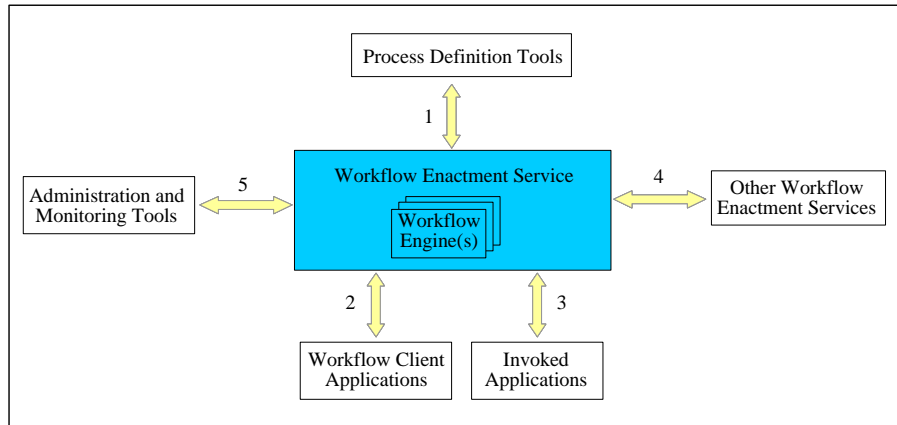


Fig. 3. WfMC's Reference Architecture.

Although this reference architecture was created in 1995, it still provides a relevant architecture for discussing workflow management systems today. Moreover, it is relevant to be compliant with it in order to ensure the adaptability of our solution.

Unfortunately, despite its advantages, it is insufficient in a dynamic COW context due to two main reasons. First, the WES must execute not only workflow process instances but it also should drive different concurrent activities such as process instance execution, partners finding, negotiation between partners, and cooperation with other WES by offering and requesting services. As it is defined in the reference architecture, the WES must not to support such concurrent activities. Second, interfaces 3 and 4, which support interactions with external applications, do not address partners finding and negotiation between partners. Indeed, interface 3 is designed for invoking applications without human intervention while interface 4 permits to automated workflow processes to interact with other automated workflow processes.

## 2.2. Revisiting the Reference Architecture

Consequently we have revisited this WFMS reference architecture, and more precisely the WES of this architecture, using the agent technology and introducing a new interface and two specific components to deal with partners finding and negotiation between partners. Indeed, it is necessary to revisit the WES in order to make it able to drive different concurrent activities. Moreover, instead of modifying an existing interface, and more precisely interface 4, we propose to define a new interface dedicated to partners finding and negotiation between partners. Thus, our proposition remains compliant with the WfMC architecture since the existing interfaces (1 to 5) are not modified.

Several reasons have led us to revisit this architecture using the agent technology. First, as defended in the introduction, this technology is convenient in the dynamic COW context since it provides natural abstractions to deal with distribution, heterogeneity and autonomy which are inherent to this context; thus each organization involved in a

dynamic COW may be seen as an autonomous agent having the mission to coordinate with other workflow agents, and acting on behalf of its organization. So, adopting this agent view, we benefit from all the work around coordination in multi-agent systems [Jennings *et al.* (2005)], which will be useful to deal with dynamic COW coordination problems: middleware components, sophisticated interactions protocols supporting negotiation between agents... Second, as defended in [Buhler and Vidal (2005)], this technology ensures a bigger flexibility and adaptability to the business processes involved in a dynamic COW. The agents implementing them can easily adapt at run-time to their specific requirements. For instance in case of a partner failure, it is then necessary to modify the COW process in order to find a new partner able to ensure the COW service the failed partner had to execute.

Figure 4 presents the agent-based architecture we propose for COW service execution. This architecture revisits the WES architecture (for which the WfMC imposes no constraints) using agents. It includes:

- Several workflow agents, each one implementing a business process instance,
- An agent manager which controls and monitors the running of workflow agents,
- A connection server which interacts with mediation infrastructures specialized for partners finding and negotiation between partners,
- A new interface, interface 6 supporting the communication between the connection server and the mediation infrastructures.

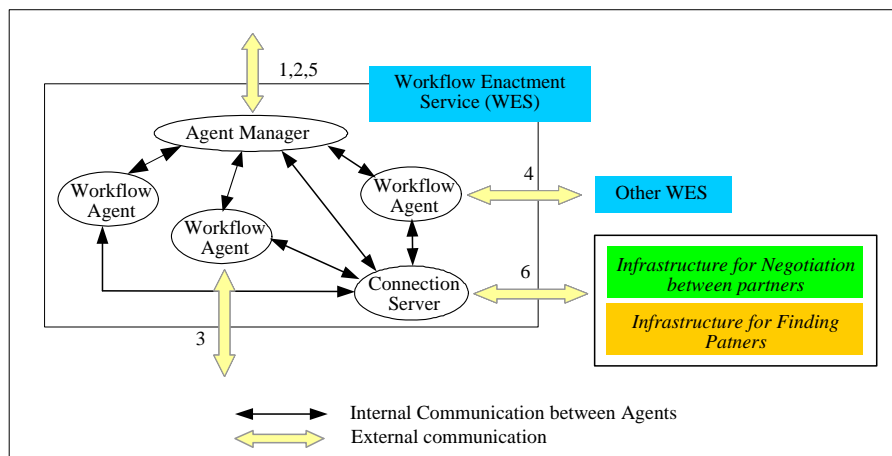


Fig. 4. Agent-based Architecture of COW Service Execution.

Regarding the Workflow Agents, the idea is to implement each business process (stored in a Business Process Database –not visualized in figure 4-) instance as a software process, and to encapsulate this process within an agent. Such a Workflow Agent includes a workflow engine that, as and when the business process instance progresses, reads the business process definition and triggers the action(s) to be done according to its

current state. This Workflow Agent supports interface 3 with the applications that are to be used to perform pieces of work associated to process' tasks.

The Agent Manager controls and monitors the running of Workflow Agents:

- Upon a request for a new instance of a business process, the Agent Manager creates a new instance of the corresponding Workflow Agent type, initializes its parameters according to the context, and launches its workflow engine,
- It ensures the persistency of Workflow Agents that execute long-term business processes in which task performances are interleaved with periods of inactivity,
- It coordinates Workflow Agents in their use of the local shared resources,
- It assumes interfaces 1, 2 and 5 of the WfMC's reference architecture.

In a dynamic COW context, workflow agents need to find external workflow agents running in other organizations and able to contribute to the achievement of their goal. Connecting them not only requires finding, negotiation and contracting capacities but also maintaining knowledge about resources of the environment. The role of the Connection Server is to manage this knowledge (stored in a Knowledge database –not visualized in figure 4-) and to help agents to connect to the partners they need. To do this, the connection server interacts with the mediation infrastructures using a new interface, Interface 6. For instance, this interface supports the communication between a connection server of a WES and a mediator agent of the infrastructures (e.g. a matchmaker, a moderator), but also between two connection servers of two different WESs.

### **3. The Mediation Infrastructure for Partners Finding**

This section first presents the infrastructure we propose to deal with partners finding and also specifies, through an AUML sequence diagram, the dynamics of the partners finding process. Then, it shows how we use a domain ontology for offer and request workflow service publication, but also in order to implement intelligent comparisons between offered and requested workflow services.

#### **3.1. Infrastructure for Partners Finding**

In the loose COW context, requesters workflow agents need to find providers workflow agents running in other organizations and able to contribute to the achievement of their goal. Finding them requires, in addition to the Connection Server and Interface 6 of the WES, the specification of a mediation infrastructure. This infrastructure is described in figure 5 below.

This infrastructure includes: (i) a Matchmaker Agent, which compares offers and requests of COW services, (ii) a database, which stores the COW services offered by provider organizations, and (iii) a database, which stores different domain ontologies. In fact, this last database is both shared by the agent-based architecture introduced for COW service execution (i.e. the WES) and the infrastructure.

More precisely, the role of the Matchmaker is to connect requesters workflow agents to providers workflow agents according to the following protocol: (a) a provider workflow agent advertises the proposed COW service to the matchmaker, (b) the

matchmaker stores the service, (c) a requester workflow agent asks the matchmaker whether it knows providers offering a desired COW service, and finally (d) the matchmaker matches the request against the stored services and returns the result as a set of COW service providers.

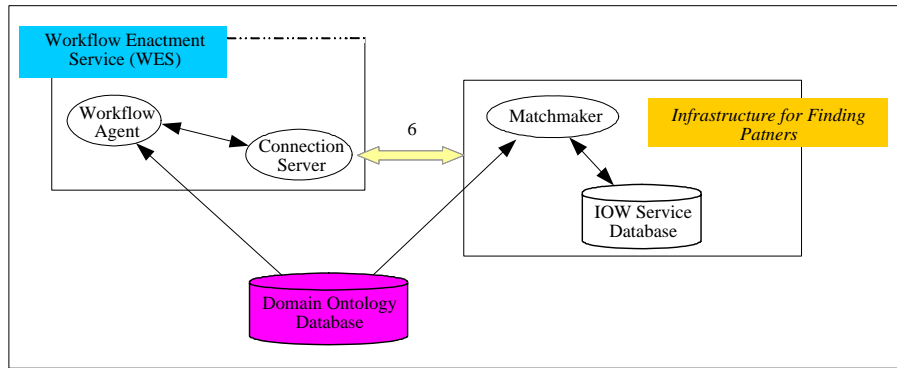


Fig. 5. Mediation Infrastructure for Partners Finding.

The AUML sequence diagram below illustrates the dynamics of the partners finding process.

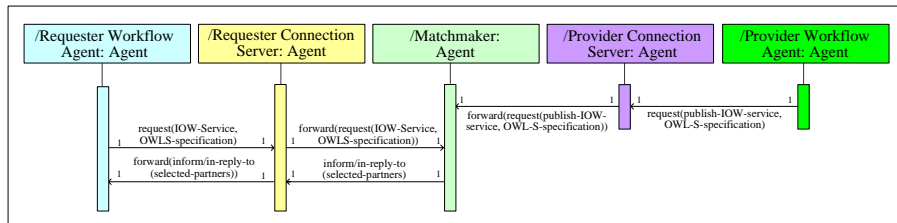


Fig. 6. AUML Sequence Diagram for Partners Finding.

This sequence diagram illustrates messages exchanged between agents belonging to both the agent-based architecture for COW service execution and the mediation infrastructure introduced to deal with partners finding. The involved agents are providers and requesters workflow agents, connection servers and the matchmaker. Provider workflow agents, through their connection server, publish OWL-S specifications of offered COW services to the Matchmaker agent: a request message is send by a provider workflow agent and forwarded to the Matchmaker agent. Requester workflow agents solicit the Matchmaker to find partners able to realize a requested COW service: a request message is send by a requester workflow agent through its connection server to the Matchmaker. In reply, the Matchmaker selects a set of possible partners and informs the requester of the selected partners (inform/in-reply-to message).

### 3.2. Using a Domain Ontology

This section first explains why using a domain ontology during the partners finding step and then illustrates its use through an example.

#### 3.2.1. Why Using a Domain Ontology

In our proposition, a domain ontology is used to define the vocabulary shared by the partners involved in a dynamic COW. More precisely, this domain ontology eases the definition of COW services' offers and requests, but also facilitates the automatic discovering of services since it improves the partners finding process which makes the correspondence between requests and offers of services. Indeed, using this ontology, it is possible to implement intelligent comparison mechanisms (not only a simple equality) between requests and offers including for instance subsumption... [Klusch *et al.* (2006)].

First, the domain ontology is at the basis of COW service definition: it gathers the common vocabulary shared and accepted by all the partners involved in the COW and this vocabulary is used to define the provided and requested workflow services. In our proposition, domain ontologies are defined using Protégé-3.1.1 and OWL (Ontology Web Language) and COW services are defined by using together two complementary formalisms that are PNOs (Petri Nets with Objects) and OWL-S (Ontology Web Language for Services): PNOs permit the formal and graphical specification of COW services, their simulation and validation, while OWL-S permits COW service publication to the Matchmaker. We have defended the combined used of PNOs and OWL-S in [Andonoff *et al.* (2005)] for COW services specification and publication and we also have provided rules and algorithms to derive PNO specifications onto OWL-S service specifications\*. More precisely, OWL-S's specifications of COW services are given in terms of ServiceProfile and ServiceProcess classes. The ServiceProfile class provides all the necessary information for a service to be found and possibly selected while the ServiceProcess class defines the structure of the service using atomic and composite processes. Atomic processes correspond to operations that the service can directly execute; they have no sub-processes. Composite processes are collections of processes coordinated by control constructs including sequence, loops, conditionals and concurrency. The ServiceProfile class is used for comparison of COW services while the ServiceProcess class is used during a negotiation since the process itself (i.e. the way of implementing an COW service) is considered during this step.

Second, the Domain ontology is also at the basis of COW service comparisons. More precisely, the Matchmaker first selects the Domain ontology of the requested COW service and then extracts classes and relationships between classes from this ontology in order to represent them as a tree. Thus, each node of this tree corresponds to a class of the ontology and each link (oriented) corresponds to a sub-class relationship between two nodes: the direction of the arrow indicates the sub-class. In our context, this extraction is

\* This paper does not detail rules and algorithms supporting the automatic derivation from PNO specifications onto OWL-S ones. The interested reader can consult [Andonoff *et al.* (2005)] for that.

possible since the vocabulary defined in the ontology is described in a hierarchical way. Section 3.2.2 illustrates this extraction through an example.

Thus, using this tree, it is possible to deduce if a concept is more general or not than another one. We say that a concept C1 subsume a concept C2 if the corresponding node of C1 is linked to the corresponding node of C2 by a sub-class relationship. Doing so, it is possible to implement flexible comparison between offers and requests: the objective is to be able to select offers (provided COW services) which does not exactly match with requests (requested COW services) but are close one to others. So, we have implemented three comparison modes [Klusch *et al.* (2006)]: the Exact mode, the PlugIn mode and the Subsume mode. The Exact mode selects an offer if it corresponds exactly to the request (request = offer), the Plug-In mode returns an offer if it subsumes a request (request < offer) and the Subsume mode returns an offer if it is included in a request (request > offer). The Plug-In and Subsume modes use the domain ontology and the matching process considers in the comparison all the elements defined in the Input and Output clauses of the OWL-S ServiceProfile of the compared offers and requests. The comparison algorithm which is used for both Plug-In and Subsume comparisons uses the following Include function.

```

Function Include (E1: string, E2: string): boolean
-- returns true if E1 subsumes E2.
-- E1 is one of the elements of the Input or Output clauses of an offer,
-- and E2 is one of the elements of the Input or Output clauses of a request.
-- O is the Ontology represented as a tree.
-- We also have the following functions:
-- Father(E) : gives E's father in O
-- Root(O) : gives the root of O
Variables
  CurrentNode: Node -- Current Node of O
  Ancestors: SetOfNodes -- E2's ancestors
Begin
  Ancestors ← ∅
  If E2 = Root(O) Then
    Ancestors ← ∅
  Else
    CurrentNode ← Father(E2)
    Ancestors ← Father(E2)
    While (CurrentNode <> Root(O)) Do
      CurrentNode ← Father(CurrentNode)
      Ancestors ← Ancestors + CurrentNode
    End While
  End If
  Include (E1 ∈ Ancestors)
End

```

3.2.2. Example

To illustrate that, let us consider the “Reviewing papers” case study introduced in [Bouzguenda (2006)] which considers a review process based on volunteer reviewers, as it is the case for the ACM Symposium on Applied Computing, Special Track on Coordination for instance. In this case, the review process is deployed in the context of a loose COW since it involves actors (PC Chair, reviewers...) from different organizations (universities, laboratories, industries), without special agreements between them, and the reviewers involved and their number are not pre-defined but dynamically recruited. Once a reviewer is found, the PC Chair negotiates the workflow service (i.e. reviewing papers) that the reviewer provides according to different criteria (due time, price, visibility of the service evolution and way of executing the service) in order to select him or not.

To implement this case study, we have first specified the “Conference” ontology, which defined the underlying vocabulary. This ontology has been specified using Protégé-3.3.1 and derived in OWL using a specific Plug-In. We do not present below the complete specification of this ontology but figure 7 gives its Protégé screenshot while figure 8 only gives its partial OWL specification and its corresponding tree.

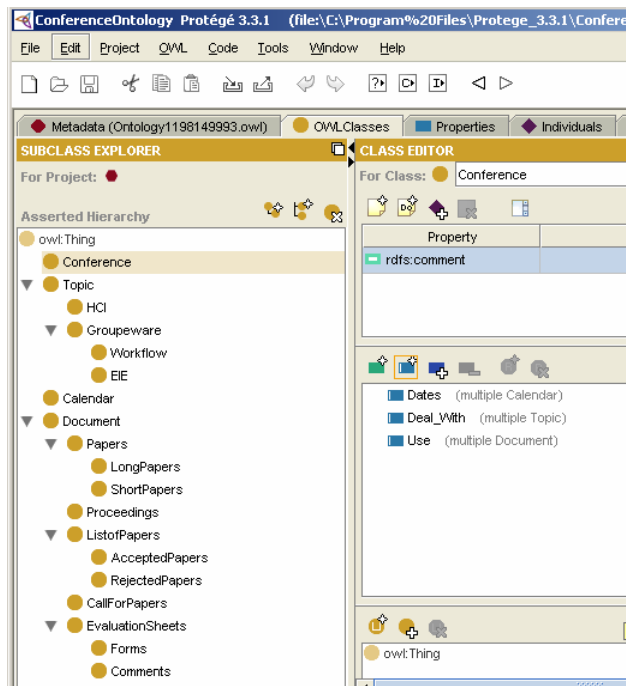


Fig. 7. Conference Domain Ontology.

The AXML sequence diagram below illustrates the dynamics of the partners finding process.

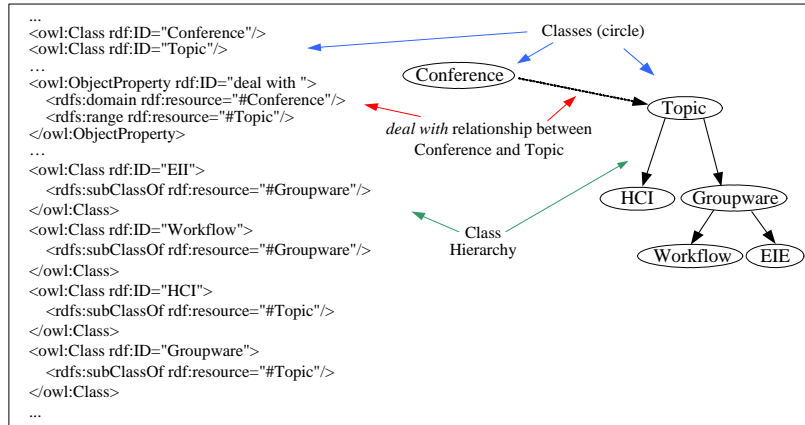


Fig. 8. Extract of the OWL Specification of the Conference Domain Ontology.

The Conference ontology is used for publication of provided and requested COW services to the Matchmaker. Figures 9 and 10 give two examples of COW services published in OWL-S. More precisely these figures give the OWL-S service profile class which will be later used for the comparison of requested and provided COW services. Figure 9 presents an offered COW service while figure 10 presents a requested one.

```

<rdf:RDF "">
  % general information of the service such as rdf files along with its non functional
  % attributes serviceName, textDescription, contactInformation, serviceParameter,
  % qualityRating and serviceCategory
  <owl:Ontology rdf:about="">
    % rdf files describing the ontologies of the service, profile, process and domain
    ...
  </owl:Ontology>
  <profileHierarchy: ReviewingPapers rdf:ID="Profile_ReviewingPapersConference">
    <service:presentedBy rdf:resource ""/>
    <profile:has_process rdf:resource ""/>
    <profile:serviceName> ReviewingPapersConference </profile:serviceName>
    <profile:textDescription> ... </profile:textDescription>
    <profile:contactInformation> ... </profile:contactInformation>
    <profile:serviceParameter> ... </profile:serviceParameter>
    <profile:qualityRating> ... </profile:qualityRating>
    <profile:serviceCategory> ... </profile:serviceCategory>
    % Input and output data and corresponding precondition and effects
    <profile:hasInput rdf:resource="URL Process#Paper_IN"/>
    <profile:hasInput rdf:resource="URL Process#ReviewSheet_IN"/>
    <profile:hasInput rdf:resource="URL Process#Groupware_IN"/>
    <profile:hasOutput rdf:resource="URL Process#FullReviewSheet_Out"/>
    <profile:hasPrecondition rdf:resource="URL Process#PaperWrittenEnglish"/>
    <profile:hasPrecondition rdf:resource="URL Process#EmptyReviewSheet"/>
    <profile:hasEffect rdf:resource="URL Process#PaperReviewed"/>
  </profileHierarchy : ReviewingPapers >
</rdf:RDF>

```

Fig. 9. Provided COW service.

```

<rdf:RDF "">
% general information of the service such as rdf files along with its non functional
% attributes serviceName, textDescription, contactInformation, serviceParameter,
% qualityRating and serviceCategory
<owl:Ontology rdf:about="">
% rdf files describing the ontologies of the service, profile, process and domain
...
</owl:Ontology>
<profileHierarchy: ReviewingPapers rdf:ID="Profile_ReviewingPapersConference">
  <service:presentedBy rdf:resource ""/>
  <profile:has_process rdf:resource ""/>
  <profile:serviceName> ReviewingPapersConference </profile:serviceName>
  <profile:textDescription> ... </profile:textDescription>
  <profile:contactInformation> ... </profile:contactInformation>
  <profile:serviceParameter> ... </profile:serviceParameter>
  <profile:qualityRating> ... </profile:qualityRating>
  <profile:serviceCategory> ... </profile:serviceCategory>
  % Input and output data and corresponding precondition and effects
  <profile:hasInput rdf:resource="URL Process#ShortPaper_IN"/>
  <profile:hasInput rdf:resource="URL Process#ReviewSheet_IN"/>
  <profile:hasInput rdf:resource="URL Process#Workflow_IN"/>
  <profile:hasOutput rdf:resource="URL Process#FullReviewSheet_Out"/>
  <profile:hasPrecondition rdf:resource="URL Process#PaperWrittenEnglish"/>
  <profile:hasPrecondition rdf:resource="URL Process#EmptyReviewSheet"/>
  <profile:hasEffect rdf:resource="URL Process#PaperReviewed"/>
</profileHierarchy : ReviewingPapers >
</rdf:RDF>

```

Fig. 10. Requested COW service.

The domain ontology is also used to compare COW service offers and requests. More precisely, the ontology permits Plug-In and Subsume comparisons in order to indicate if an offer is greater or lower than a request. In the previous example, the provided COW service (cf. figure 9) is greater than the requested one (cf. figure 10) since *Groupware* is more general than *Workflow* in the Conference ontology. Consequently, the organization providing it is selected by the Matchmaker and sent to the requester organization looking for help to realize its requested COW service.

#### 4. The Mediation Infrastructure for Negotiation between Partners

This section first describes the mediation infrastructure we propose to deal with negotiation between partners. It then presents the Negotiation Ontology and finally shows how we use it in order to facilitate the dynamic choice of a protocol for a given negotiation.

##### 4.1. Mediation Infrastructure

After the partners finding step, a COW service requester is connected to different COW service providers. But connecting is not enough to definitively choose the partner that is going to execute the service: a negotiation step is necessary to evaluate and select the most suitable COW service provider. So, in addition to the Connection Server and

Interface 6 of the agent-based architecture introduced for COW service execution, the specification of a new mediation infrastructure is necessary. This infrastructure is described in figure 11.

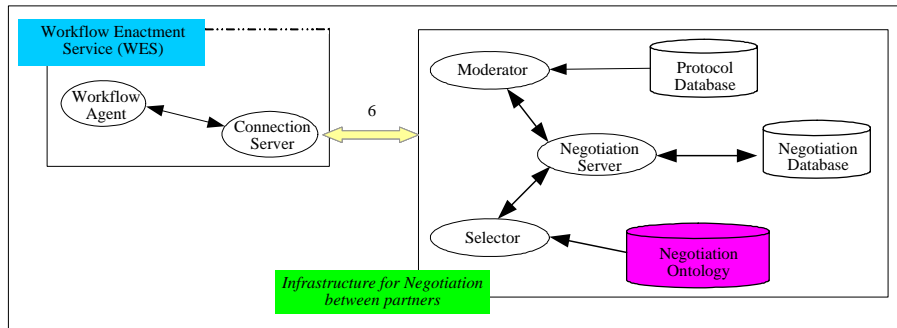


Fig. 11. Mediation Infrastructure for Negotiation between Partners.

This infrastructure includes (i) a Negotiation Server which provides information about current negotiations and notably negotiation acts submitted by the participants, (ii) a Moderator which ensures that negotiation acts are consistent with the underlying negotiation protocol, (iii) a Negotiation database which stores information about negotiations such as the identity of moderators, the identity of involved Connection Servers, the underlying protocols..., (iv) a Negotiation ontology which stores negotiation protocols which are convenient in our dynamic COW context, and finally (v) a Selector which helps requester Workflow Agents in choosing a negotiation protocol according to a certain number of criteria (such as number of providers Workflow Agents, duration of negotiation, universe of negotiation which can be open or closed, termination of the negotiation, type of partner who can be competitive or cooperative...).

Of course, the architecture integrates as many Moderators as the number of negotiations opened by different requester Workflow Agents. Each Moderator implements the negotiation protocol of its corresponding negotiation. Doing so, requester and provider Workflow Agents do not integrate a negotiation protocol and only focus on their specific activity.

#### 4.2. Negotiation Ontology

The aim of the Negotiation ontology is to describe interaction protocols which are convenient to the dynamic COW context in order to permit the different requester Workflow Agents to dynamically select at run-time a given negotiation protocol during the opening of a negotiation step.

Thus, as suggested in [Andonoff and Bouzguenda (2005)], before defining this ontology, we have first examined the following interaction protocols, stemming from Multi-Agent System area, and which are suitable to negotiation in dynamic COW: Heuristic, Argumentation, Contract-Net, Auction and more particularly English auction, Dutch auction, private first well price and private second well price [Jennings *et al.*

(2001)]. We have also used the following properties to feature these protocols: number of partners involved in negotiation, behavior of these partners (are they competitive or cooperative), number of negotiation rounds and number of attributes to negotiate. The obtained result is described in table 1 below.

Table 1. Features of Negotiation Protocols.

	<b>Behavior</b>	<b>Number of Partners</b>	<b>Number of Rounds</b>	<b>Number of Attributes</b>
Heuristic	competitive	several	several	several
Argumentation	competitive	several	several	several
Contract-Net	cooperative	several	one	several
English Auction	competitive	several	several	one
Dutch Auction	competitive	several	several	one
First Best Price	competitive	several	one	one
Second Best Price	competitive	several	one	one

According to table 1, we have defined a Negotiation ontology which both specifies the hierarchical organization of protocols and the features of these protocols. More precisely, we have defined a set of classes representing these types of protocols along with their hierarchical organization. Moreover, the properties of these classes correspond to the four properties featuring the protocols (behavior, number of partners, number of rounds and number of attributes), and the instances of these classes correspond to the types of protocols themselves: their values correspond to the values of their properties as indicated in table 1.

This Negotiation ontology, presented below, has been implemented using Protégé-3.3.1. We also have derived a corresponding OWL file, which will be used later for dynamic selection of protocols. Figures 12 and 13 presented below describe the classes, properties and instances of the ontology.

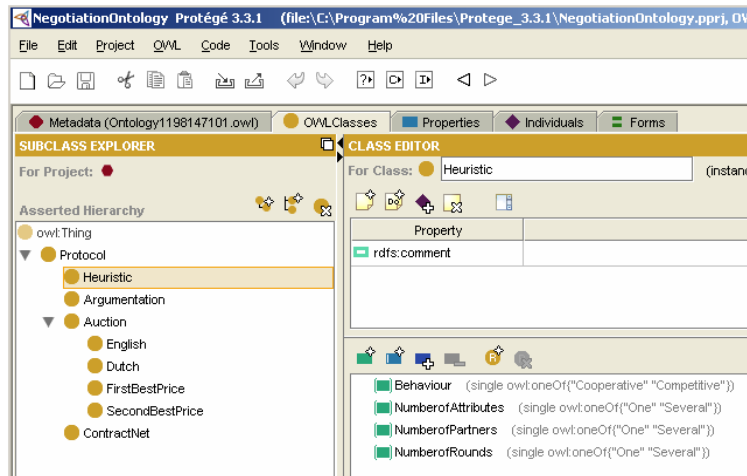


Fig. 12. Negotiation Ontology: Classes and Properties.

Figure 12 presents the hierarchy of negotiation protocols that are convenient to dynamic COW. It also shows the properties of these protocols. These properties are defined in the Protocol class and inherited in its subclasses. Figure 13 presents an instance of the Heuristic class. Its name is HeuristicProtocol and the values of its properties indicate (i) that a negotiation following a heuristic protocol involves several partners, (ii) that these partners are competitive, (iii) that a multi-attribute negotiation is possible and, (iv) that a multi-round negotiation is possible.

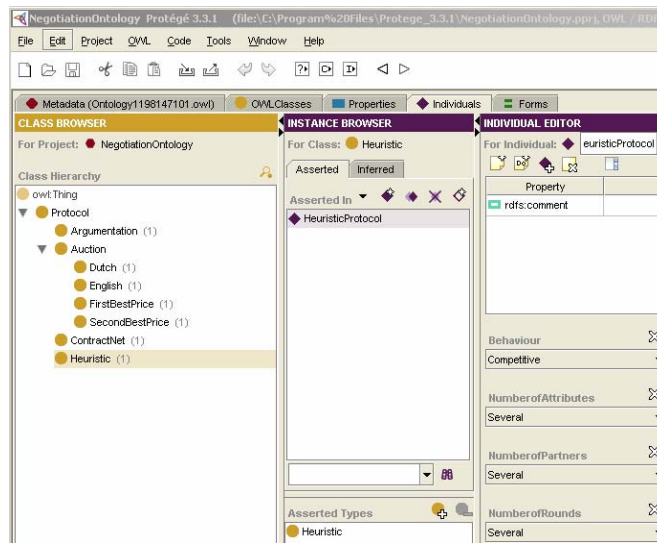


Fig. 13. Negotiation Ontology: Instances.

The OWL code obtained from this graphic specification is partially given below with the example of the Heuristic protocol (cf. figure 14). This OWL code will be used later for dynamic selection of protocols.

```

-- Specification of the Protocol class and its property NumberofPartners
<owl:Class rdf:ID="Protocol">
  <owl:DatatypeProperty rdf:ID="NumberofPartners">
    <rdf:type rdf:resource="..." />
    <rdfs:domain rdf:resource="#Protocol"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  ...
</owl:Class>

-- Specification of the Heuristic class as a sub-class of the Protocol class
<owl:Class rdf:ID="Heuristic">
  <rdfs:subClassOf rdf:resource="#Protocol"/>
</owl:Class>

-- Specification of the HeuristicProtocol instance of heuristic class
<Heuristic rdf:ID="HeuristicProtocol">
  <Behaviour rdf:datatype="...">Competitive</ Behaviour>
  <NumberofAttributes rdf:datatype="...">Several</ NumberofAttributes>
  < NumberofPartners rdf:datatype="...">Several</ NumberofPartners >
  < NumberofRounds rdf:datatype="...">Several</ NumberofRounds>
</Heuristic>

```

Fig. 14. Negotiation Ontology: Partial OWL specification.

### 4.3. Dynamic Selection of a Negotiation Protocol

The Negotiation ontology giving a classification of COW negotiation protocols, it is then possible to easily distinguish them and select them at run-time according to the context of a negotiation. As illustrated in the AUML (Agent UML) sequence diagram of figure 15, we proceed as follows to choose a negotiation protocol: (i) the requester Workflow Agent initiating the negotiation asks the Negotiation Server for the creation of a moderator implementing a negotiation protocol, (ii) in reply, the Negotiation Server asks the requester Workflow Agent for featuring the protocol it wishes by giving the values of the properties defined in the Negotiation ontology, (iii) then, the Negotiation Server delegates the selection of the negotiation protocol to the Selector agent sending it the values given by the requester Workflow Agent, (iv) the Selector agent find a negotiation protocol corresponding to these values and asks the Negotiation Server for the creation of a Moderator implementing it, and (v) finally, the Negotiation Server stores information about the open negotiation in the Negotiation database and informs the requester Workflow Agent of the creation of the negotiation.

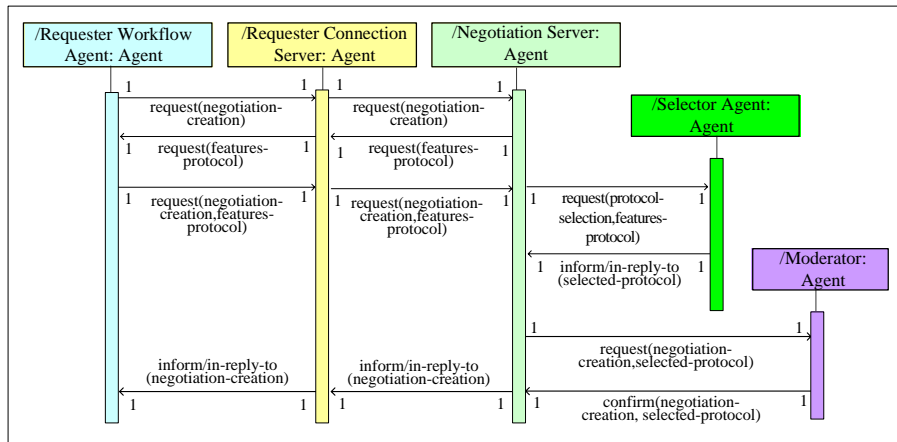


Fig. 15. Negotiation Ontology: Instances.

As indicated before, the Selector agent is responsible for protocol selection: it is able to select, from the Negotiation ontology OWL file, a (set of) negotiation protocol(s) according to criteria defined by the requester workflow agent initiating the negotiation. The selection algorithm of this Selector agent uses the FindProtocol function given below.

Function FindProtocol (Behav: string, NbAtt: string, NbPart: string, NbRound: string):

SetofProtocols

- Returns a set of protocols which check the requested workflow agent criteria
- Ps is the set of protocols which are suitable to the dynamic COW context
- P is a protocol of Ps and TheP is a set of selected protocols

## Variables

P: Protocol -- a protocol  
 Ps: SetOfProtocols -- set of negotiation protocols  
 TheP: SetOfProtocols -- a set of selected protocols

## Begin

```
TheP ← ∅
-- We select the set of negotiation protocols
Ps ← SetOfNegotiationProtocols()
-- What protocols check the criteria?
For Each P in Ps Do
  -- Compare indicates if the current protocol P corresponds to the criteria
  If Compare("Behaviour",Behav) and
    Compare("NumberofAttributes", NbAtt) and
    Compare("NumberofPartners ",NbPart) and
    Compare("NumberofRounds",NbRound) Then
    -- We add P in the set of selected protocols
    TheP ← TheP + P
  End If
End For
FindProtocol ← TheP
```

## End

We could have used OWL query languages such as nRql [Haarslev *et al.* (2004)] or OWL-QL [Fikes *et al.* (2003)] for this dynamic selection of negotiation protocols. But using such languages would have obviously set the use of servers able to execute these queries (for instance Racer for nRql queries). Doing that, the complexity of our coordination architecture would have unnecessarily increased. That is the reason why we have specified an algorithm implementing finding of negotiation protocols parsing an OWL file.

## 5. Implementation

We have implemented a prototype called MatchFlow which validates our mediation infrastructure for partners finding [Bouzguenda (2006)]. MatchFlow connects requester Workflow Agents looking for a workflow service to one or several provider Workflow Agents able to implement the requested service. As explained in [Andonoff *et al.* (2005)], offers and requests are specified using the Petri-Net with Objects (PNO) formalism and stored by a Matchmaker in the OWL-S format: the PNO formalism is used to design, analyze, simulate, check and validate workflow services which are then automatically derived into OWL-S specifications to be published through the Matchmaker. In the current version of MatchFlow, the Matchmaker compares the offers and requests service profiles: it establishes intelligent comparisons (exact, plug in, subsume) based on a domain ontology.

MatchFlow has been implemented with Madkit platform [Gutknecht and Ferber (2000)], which permits the development of distributed applications using multi-agent principles.

We have implemented the “Reviewing papers” case study using MatchFlow and also have evaluated the Matchmaker performances in terms of result quality and processing time. The interested reader can consult [Bouzguenda (2006)] for more information about this evaluation.

Figure 16 below gives an overview of the MatchFlow prototype. This figure shows a set of coordinated agents during a partners finding step: the Matchmaker (window 1), a requester Connection Server (window 2) and its corresponding requester Workflow Agent (window 3), a provider Connection server (window 4) and its corresponding provider agent Manager (window 5), the OWL-S specification of a requested workflow service (window 6) and its corresponding PNO tree (window 7). Let us briefly highlight these two last windows. We first define a (requested or provided) COW Service using the PNO formalism. We thus obtain a formal specification of the service: its simulation and validation is then possible. Second, we automatically derive its corresponding OWL-S specification for its publication to the Matchmaker. More precisely, the PNO specification of a workflow service is represented as a tree, called a PNO tree. This PNO tree is the starting point for the automatic derivation of both OWL-S Service Profile and Service Process of considered COW Services [Andonoff *et al.* (2005)].

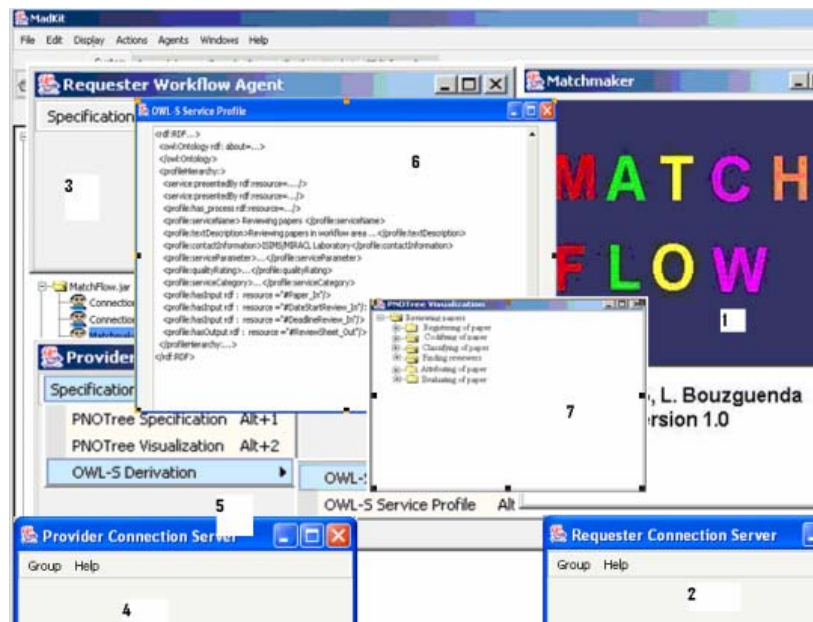


Fig. 16. Overview of the MatchFlow Prototype.

Second, we currently implement a prototype called NegoFlow whose aim is to support our mediation infrastructure for negotiation between partners. More precisely, we have specified the Negotiation ontology using Protégé-3.3.1 and derived in OWL using a specific Plug-In of Protégé. We also have implemented using Madkit platform some agents of the negotiation infrastructure: the Negotiation Server, the Connection Server, the Selector and some requester and provider Workflow Agents. At the present time, we focus on the implementation of a specific negotiation protocol: the heuristic one. Each moderator implementing this type of negotiation protocol is in fact an agent playing the behavior underlying

Figure 17 and 18 give an overview of the current state of the NegoFlow prototype. More precisely, figure 17 shows a set of coordinated agents during a negotiation step: a requester Workflow agent (window 1) and its Connexion Server (window 2), a provider Connexion Server (window 3) and its corresponding Agent Manager (window 4), the Negotiation Server (window 5) and the Selector (window 6). This figure also shows how the Selector helps to select a negotiation protocol (window 7). This window is detailed in figure 18. This latter shows how the Selector agent permits the requester Workflow Agent to specify the features of its negotiation protocol (behaviour, number of partners, number of rounds and number of attributes). Moreover, it returns one or several negotiation protocols, which correspond, to the specified features. Then, the requester Workflow Agent selects one of the returned protocols and indicates its choice, via its Connexion Server, to the Negotiation Server. Finally, this later creates a negotiation implementing the chosen protocol.

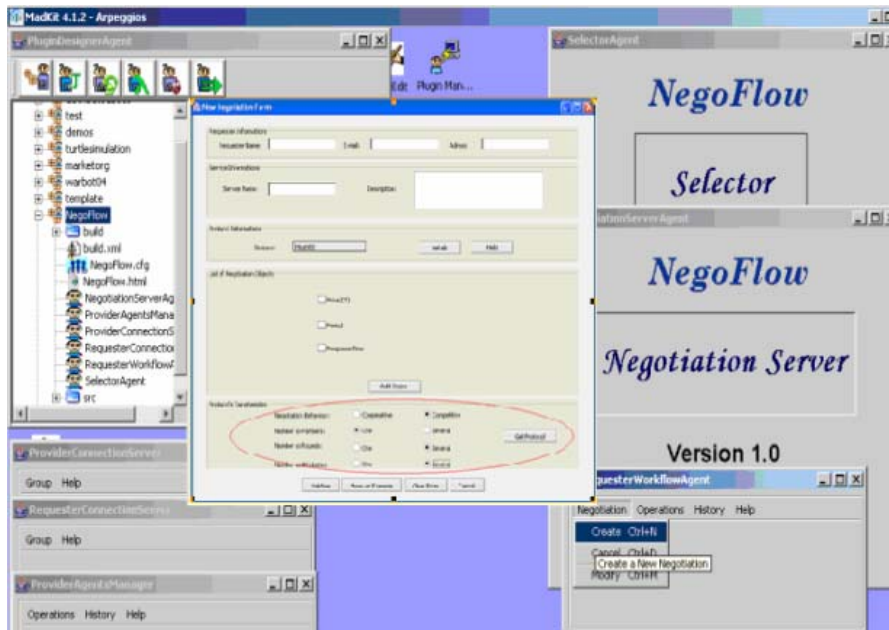


Fig. 17. Overview of the NegoFlow Prototype.

Fig. 18. NegoFlow: Specification of Protocol Features.

## 6. Discussion and Conclusion

This paper has dealt with coordination of business processes involved in a dynamic COW and contributes to make automatic coordination in such a context possible. More precisely, this paper has presented an agent-based architecture for COW service execution and two mediation infrastructures introduced to deal with partners finding and negotiation between partners. It has also shown how these infrastructures use respectively:

- a Domain ontology which helps a Matchmaker to connect a requester organization looking for an COW service to one or several providers organizations able to execute this requested service.
- a Negotiation ontology which both allows the description and the dynamic selection of protocols which are convenient in a dynamic COW context.

Related works have been proposed to deal with coordination in dynamic COW [Zeng *et al.* (2001)], [Buhler and Vidal (2005)], [Blake and Gomaa (2004)], [Aberg *et al.* (2005)], [Savarimuthu *et al.* (2005)]. Every works adopt an agent-based approach: indeed agents are considered as a suitable mean to ensure flexibility to business processes involved in an COW, and also as a suitable mean to deal with partners finding and negotiation between partners. Except for [Zeng *et al.* (2001)], these works also use Web services to design and implement business processes involved in dynamic COW: UDDI

service discovery registries support service finding but without taking into account Domain ontologies. Consequently, these works, unlike ours, do not propose a solution to both solve semantic heterogeneity problems that obviously occur in the dynamic COW context, and improve the comparison mechanism between requested and offered COW services.

Moreover, these works only consider the finding partner problem. But finding partner is not enough to definitively choose the partner that is going to execute the COW service: a negotiation step, in terms of due time, price, visibility of the service evolution and way of executing it (i.e. its process), is necessary to evaluate and select the best provider.

Thus, our negotiation infrastructure and ontology contribute to deal with this coordination problem. Moreover, the approach we adopt, i.e. representing negotiation protocols as specific components, called moderators, separated from the COW processes themselves, and using an ontology to describe the suitable negotiation protocols in the dynamic COW context, is interesting for several reasons:

- Negotiation protocols (Moderators) are autonomous and reusable components that may be specified and verified independently from any specific workflow agent (implementing an COW service).
- Requester and provider Workflow Agents only focus on their own business process since what is related to negotiation is left to Moderators; thus, our approach imposes only few constraints to requester and provider workflow agents: they are loosely coupled.
- The Negotiation ontology both permits the description and the dynamic selection of negotiation protocols; consequently it constitutes a step forward to reach the objective of automated coordination in dynamic COW.

Regarding future works, we have planed to revisit the two implementations using JADE [Bellifemine *et al.* (2007)]. Indeed, this recent multi-agent development environment is very interesting in our context since JADE provides, using WADE [], the possibility to implement agents integrating engines for execution of business processes. In addition to the heuristic negotiation protocol, we also have planed to implement Moderators playing protocols such as English Auction, Argumentation, Dutch Auction... Finally, we have planed to put together the two prototypes, MatchFlow and NegoFlow, in order to provide a single environment for dynamic COW execution able to deal with partners finding and negotiation between partners. Later, we will address the contract specification between partners problem introduced in section 1.

## References

- van der Aalst, W. (1999): *Inter-Organizational Workflows: An Approach Based on Message Sequence Charts and Petri Nets*. Journal on Systems Analysis, Modeling and Simulation, 34(3), pp. 335-367.
- Aberg, C.; Lambrix, C.; Shahmehri, N. (2005): *An Agent-Based Framework for Integrating Workflows and Web Services*, 14th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE'05, Linköping, Sweden, pp. 27-32.

- Andonoff, E.; Bouzguenda, L.; Hanachi, C. (2005): *Specifying Workflow Web Services for Finding Partners in the context of Loose Inter-Organizational Workflow*. 3rd International Conference on Business Process Management, BPM'05, Nancy, France, pp. 120-136.
- Andonoff, E.; Bouzguenda, L. (2005): *Agent-Based Negotiation between Partners in Loose Inter-Organizational Workflow*. 5th International Conference on Intelligent Agent Technology, IAT'05, Compiègne, France, pp. 619-625.
- Bellifemine, FL; Caire, G.; Greenwood, D. (2007): *Developing Multi-Agent Systems with JADE*. Wiley Editions.
- Blake, B.; Gomaa, H. (2004): *Agent-Oriented Compositional Approaches to Services-based Cross Organizational Workflow*. Journal on Decision Support Systems, 40(1), pp. 176-181.
- Bouzguenda, L. (2006): *Coordination Multi Agents pour le Workflow Inter Organisationnel Lâche*. PHD Dissertation, Université de Toulouse 1.
- Buhler, P.; Vidal, J. (2005): *Towards Adaptive Workflow Enactment Using Multi Agent Systems*. Journal on Information Technology and Management, 6(1), pp. 61-87.
- Chebbi, I.; Dustdar, S.; Tata. S. (2006): *The View based-approach to Dynamic Inter-Organizational Workflow Cooperation*. International Journal on Data and Knowledge Engineering, 56(2), pp.139-173.
- Divitini, M.; Hanachi, C.; Sibertin-Blanc, C. (2001): *Inter-Organizational Workflows for Enterprise Coordination*. In: A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf (eds): *Coordination of Internet Agents*, Springer-Verlag, pp. 46-77.
- Ferber, J.; Gutknecht, O.; Fabien, M. (2003): *From Agents to Organizations: an Organizational View of Multi-Agent Systems*. 4th International Workshop on Agent-Oriented Software Engineering, Melbourne, Australia, pp. 214-230.
- Fikes, R.; Hayes, P.; Horrocks, I. (2003): *OWL-QL - A Language for Deductive Query Answering on the Semantic Web*. Knowledge Systems Laboratory, Stanford University, Stanford, California, USA.
- Gutknecht, O.; Ferber, J. (2000): *MadKit: a Generic Multi-Agent Platform*. 4th International Conference on Autonomous Agents, Barcelona, Spain, pp. 78-79.
- Haarslev, V.; Moeller, R.; Wessel, M. (2004): *Querying the Semantic Web with Racer and nRQL*. 3rd International Workshop on Applications of Description Logic, ADL'04, Ulm, Germany.
- Hernandez, N. (2007): *Modeling Context through Domain Ontologies*. Journal of Information Retrieval, Special Issue on Contextual Information Retrieval, 10(2), pp. 143-172.
- Jenning, N.; Faratin, P.; Lomuscio, A.; Parsons, S.; Sierra, C.; Wooldridge, M. (2001): *Automated Negotiation: Prospects, Methods and Challenge*. Journal on Group Decision and Negotiation, 10(2), pp. 199-215.
- Klusch, M.; Fries, B.; Sycara, K. (2006): *Automated Semantic Web Service Discovery with OWLS-MX*. 5th International Conference on Autonomous Agents and Multi-Agents Systems, AAMAS'06, Hakodate, Japan, pp. 915-922.
- Medjahed, B.; Bouguettaya, A.; Elmagarmid, A. (2003): *Composing Web Services on the Semantic Web*. Journal on Very Large DataBases, 12( 4), pp. 333-351.
- Savarimuthu, T.; Purvis, M.; Purvis, MK.; Cranefield, S. (2005): *Integrating Web Services with Agent Based Workflow Management System*, 5th International Conference on Web Intelligence, WI'05, Compiègne, France, pp. 471-474.
- Tamma, V.; Phelps, S.; Dickinson, I.; Wooldridge, M. (2005): *Ontologies for Supporting Negotiation in E-Commerce*. Journal on Engineering Applications of Artificial Intelligence, 18(2), pp. 223-236.
- Wooldridge, M. (2002): *An Introduction to Multi-Agent Systems*. Wiley Editions.
- Workflow Management Coalition (1994): *The Workflow Reference Model*. Technical Report WfMC-TC-1003.

Zeng, L.; Ngu, A.; Benatallah, B.; O'Dell, M. (2001): *An Agent-Based Approach for Supporting Cross-Enterprise Workflows*. 12th Australian Database Conference, Bond, Australia, pp. 123-130.