A MODEL FOR COORDINATING NEGOTIATIONS AMONG VIRTUAL ENTERPRISES

Adina CRETAN[1]
PhD Candidate, Economic Informatics Department,
University of Economics, Bucharest, Romania

E-mail: badina20@yahoo.com

Abstract: *This paper presents a model for coordinating negotiation processes in concurrent inter-organizational alliances. The IT system preserves the autonomy of organizations grouped in an alliance while enabling concurrency of their activities, flexibility of their negotiations and dynamic evolution of their environment. The purpose of this work is to offer support for small and medium enterprises which cannot or do not want to fulfill a big contract alone. This approach is illustrated by a sample scenario where partners are printshops grouped in an alliance to better accomplish customers' requests.*

Key words: *negotiation; virtual enterprises; multi-agent systems; coordination services*

1. Introduction

The advent of the Internet has led to the development of various forms of virtual enterprises which try to exploit the facilities of the network to achieve higher level collaboration goals.

We try to provide support to the collaborations within an alliance and we propose negotiation as a fundamental mechanism for these collaborations.

In this paper we present how organizations participate and control the status of the negotiations and how the negotiation processes are managed. We propose a model for coordinating different negotiations that may occur in a virtual alliance.

We consider a scenario of collaborations within an alliance of distributed autonomous printshops. The alliance is a dynamic entity where new printshops may join or leave [1][2]. A printshop manager interested in joining an alliance fills in an adhesion contract with information on his printshop competencies and preferences. If the alliance committee accepts it as a new partner, the new member commits to respecting the rules of the alliance and the adhesion contract and introduces itself to the other partners. Each printshop autonomously manages its contracts and schedules. When a print request reaches a printshop, the manager analyses it to understand if it can be accepted, taking into account

job schedules and resources availability. If the manager accepts the print request, he may decide to perform the job locally or to partially outsource it, given the printshop resource availability and technical capabilities. If the manager decides to outsource a job, he starts a negotiation within the alliance with selected participants. The manager may split the job into slots, notifying the partners about the outsourcing requests for the different slots. If the negotiation results in an agreement, a contract is settled between the outsourcer and the insourcer printshops, which defines an inter-organizational workflow enacting the business process fulfilling the outsourced jobs and a set of obligation relations among participants [8].

The printshops alliance scenario shows a typical example of the e-alliances: virtual alliances where partner organizations may *a priori* be in competition with each other, but may want to cooperate in order to be globally more responsive to market demand.

E-Alliance main goal is to provide a software support for inter-organizational alliances enabling management of an alliance's life-cycle and collaborative activities among alliance partners [1]. E-Alliance should flexibly support negotiation processes in the alliance respecting the autonomy of the partners.

The main objective of this work is to propose a model for coordinating negotiations in a dynamical system with autonomous agents where each agent is in charge of its own negotiations (Section 4). In Section 5 we present an example to use this model in describing a particular coordination service, namely Block. Finally, Section 6 concludes the paper.

2. E-Alliance infrastructure

The e-Alliance infrastructure is organized in three layers as shown in Figure 1:

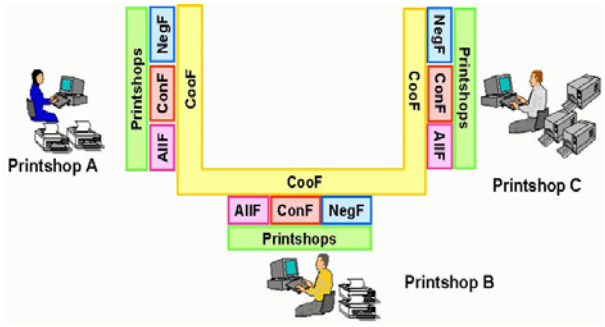

Figure 1. E-Alliance software infrastructure

A first layer is dedicated for different applications according to the specific domain in interaction with the Manager of each organization of the alliance (e.g., the printing domain). A second layer is dedicated to support the insourcing/outsourcing job within an alliance and comprises three facilities: AllF (alliance life-cycle management), ConF (contract management) and NegF (negotiation). The third, middleware and coordination layer (CooF) offers generic mechanisms to support negotiations in a distributed environment [5]. It is a global layer common to the different sites in which the partners of the alliance operate, while the two other layers are duplicated on the different sites.

Figure 2 shows the architecture of the NegF agent:

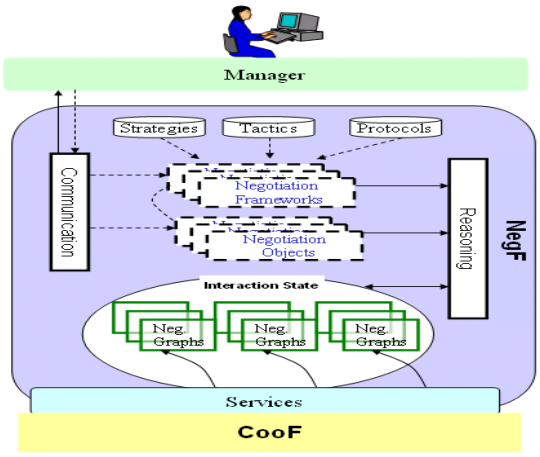

Figure 2. The architecture of the negotiation system

A NegF agent assists its printshop manager at a global level (negotiations with different participants on different jobs) and at a specific level (negotiation on the same job with different participants) by coordinating itself with the NegF of the other partners through the CooF, coordination and negotiation middleware.

Each negotiation is organized in three main steps: initialization; refinement of the job under negotiation; and closing. The initialization step allows to define what has to be negotiated (*Negotiation Object*) and how (*Negotiation Framework*). A selection of negotiation participants can be made using history on passed negotiation, available locally or provided by the AllF [4]. In the refinement step, participants exchange proposals on the negotiation object trying to satisfy their constraints [6]. The manager may participate in the definition and evolution of negotiation frameworks and objects. Decisions are taken by the manager, assisted by his NegF agent. Decision functions operate in the "Reasoning" box (Fig.2). For each negotiation, a NegF manages one or more negotiation objects, one framework and the negotiation status.

A manager can specify some global parameters: duration; maximum number of messages to be exchanged; maximum number of candidates to be considered in the negotiation and involved in the contract; tactics; protocols for the NegF interactions with the manager and with the other NegFs [7]. Differing from [11], where tactics are defined for managing the negotiation, here tactics define constraints on the negotiation process. For example, a tactic may state that a job has to be outsourced as a block, another one that it has to be split in several slots. Executing a tactic corresponds to activating a combination of *services*, implemented above the CooF, producing a coordinated modification of alternatives within the current negotiation. Each service manages a local view of the global negotiation, translating negotiation decisions to modifications on the set of the visible alternatives on the job under negotiation using primitives of the CooF.

3. Coordination services

In order to handle the complex types of negotiation scenarios, we propose six different services:

- *Outsrc* (resp. *Insrc*), for outsourcing (resp. insourcing) jobs by exchanging proposals among participants known from the beginning [9];

- *Block* service for assuring that a task is entirely subcontracted by the single partner;
- *Split* service manages the propagation of constraints among several slots, negotiated in parallel and issued from the split of a single job;
- *Broker*: a service automating the process of selection of possible partners to start the negotiation;
- *SwapIn/SwapOut* services implement a coordination mechanism between two ongoing negotiations in order to find and manage a possible exchange between their two tasks;
- *Transport* service implements a coordination mechanism between two ongoing negotiations in order to find and synchronize on the common transport of both tasks.

These services are able to evaluate the received proposals and, further, if these are valid, the services will be able to reply with new proposals constructed based on their particular coordination constraints.

In the following sections we build the model for coordinating negotiations in a dynamical system with autonomous agents and, further, we describe the Block service based on this model.

4. Building the model

4.1. Basic concepts

In this setup, at a *local* level, the model requires a formal description of the rules of coordination that manage the behavior of the agent in a negotiation; at a *global* level, the model must provide a global coordination of all negotiations of an agent.

The fundamentals of the negotiation model are given by the following basic concepts:

An *alliance* is defined as a quintuple $\mathcal{A} = <\mathcal{T}, \mathcal{P}, \mathcal{N}, \mathcal{R}, O>$ where:

1. $\mathcal{T}$ denotes the *time of the system*, assumed to be discrete, linear, and uniform;
2. $\mathcal{P}$ denotes the *set of participants* in the alliance. The participants may be involved in one or many negotiations within the alliance;
3. $\mathcal{N}$ denotes the *set of negotiations* that take place within the alliance;
4. $\mathcal{R}$ denotes *the set of policies of coordination* of the negotiations that take place within the alliance;
5. O denotes the common ontology that consists of the set of definitions of the attributes that are used in a negotiation.

A *negotiation* is described at a time instance through a set of negotiation sequences.

Let $Sq = \{s_i \mid i \in \mathbb{N}\}$ denote the *set of negotiation sequences*, such that $\forall s_i, s_j \in Sq$, $i \neq j$ implies $s_i \neq s_j$. A *negotiation sequence* $s_i \in Sq$ such that $s_i \in N(t)$ is a succession of negotiation graphs that describe the negotiation N from the moment of its initiation and up to the time instance *t*. The negotiation graph created at a given time instance is an oriented graph in which the nodes describe the negotiation phases that are present at that time instance (i.e., the negotiation proposals sent up to that moment in terms of status and of attributes negotiated) and the edges express the precedence relationship between the negotiation phases [10].

The *negotiation phase* (*ph*) indicates a particular stage of the negotiation under consideration.

The *Status* is the possible state of a negotiation. This state takes one of the following values (*Status*∈{ *initiated, undefined, success, failure*}):

- *initiated* – the negotiation, described in a sequence, has just been initiated;
- *undefined* – the negotiation process for the sequence under consideration is ongoing;
- *success* – in the negotiation process, modeled through the sequence under consideration, an agreement has been reached;
- *failure* – the negotiation process, modeled through the sequence under consideration, resulted in a denial.

Issues is the set of attributes with associated values that describe the proposals made in a negotiation phase.

Snapshot is the set of combinations between a negotiation aspect (*Status*) and the information that is negotiated (*Issues*).

The functions *status* and *issues* return, respectively, the state (status) of a negotiation instance and the set of the attributes negotiated (issues) within a negotiation instance.

A *coordination policy* is the set of coordination rules that are used to establish various relationships between negotiations regarding the information that may be distributed among many participants and many sequences (global rules) or that may be recovered as a whole in the same sequence (local rules).

4.2. Metaphor Interaction Abstract Machines (IAM)

The metaphor Interaction Abstract Machines (IAM) will be used to facilitate modeling of the time evolution of a *multi-attribute, multi-participant, multi-phase* negotiation [2]. In IAMs, a system consists of different *entities* and each entity is characterized by a state that is represented as a multiset of *resources* [3]. It may evolve according to different laws of the following form, also called "*methods*":

A1@…@An <>- B1@…@Bm

A method is executed if the state of the entity contains all resources from the left side (called the "*head*") and, in this case, the entity may perform a transition to a new state where the old resources (*A1,…,An*) are replaced by the resources (*B1,…,Bm*) on the right side (called the "*body*"). All other resources of the entity that do not participate in the execution of the method are present in the new state.

The operators used in a method are:

- the operator @ assembles together resources that are present in the same state of an entity;
- the operator <>- indicates the transition to a new state of an entity;
- the operator & is used in the body of a method to connect several sets of resources;
- the symbol T is used to indicate an empty body.

In IAMs, an entity has the following characteristics:

- if there are two methods whose heads consist of two sets of distinct resources, then the methods may be executed in parallel;

- if two methods share common resources, then a single method may be executed and the selection procedure is made in a non-deterministic manner.

In IAMs, the methods may model four types of transition that may occur to an entity: *transformation*, *cloning*, *destruction* and *communication*.

Through the methods of type *transformation* the state of an entity is simply transformed in a new state. If the state of the entity contains all the resources of the head of a transformation method, the entity performs a transition to a new state where the head resources are replaced by the body resources of the method.

Through the methods of type *cloning* an entity is cloned in a finite number of entities that have the same state. If the state of the entity contains all the resources of a head of a cloning method and if the body of the method contains several sets of distinct resources, then the entity is cloned several times, as determined by the number of distinct sets, and each of the resulting clones suffers a transformation by replacing the head of the method with the corresponding body.

In the case of a *destruction* of the state, the entity disappears. If the state of the entity contains all the resources of the head of a transformation method and if the body of the method is the resource T, then the entity disappears.

In IAMs, the *communication* among various entities is of type broadcasting and it is represented by the symbol "^". This symbol is used to the heads of the methods to predefine the resources involved in the broadcasting. These resources are inserted in the current entity and broadcasted to all the entities existent in the system, with the exception of the current entity. This mechanism of communication thus executes two synchronous operations:

i) transformation: if all resources that are not pre-defined at the head of the method enter in collision, then the pre-defined resources are inserted in the entity and are immediately consumed through the application of the method;
ii) communication: insertion of the copies of the pre-defined resources in all entities that are present in the system at that time instance.

4.3. Program Formula

In a multi-entity system, the metaphor IAMs allows the modeling and control of the autonomous evolution process for each entity in the system. Each entity may change its state independently of others, using its own resources and the methods of its computational space. This approach allows us to model in parallel the evolution of multiple negotiation phases. By using the metaphor IAMs, the evolution of the negotiation phases, associated to the nodes of a negotiation sequence, will be managed through different methods that are put together in a *Program Formula* (*PF*).

Program Formula of a negotiation sequence *s* - *PF(s)* – represents the set of the methods used to manage the evolution of the sequence *s*. In our negotiation model, a negotiation phase is connected to the set of snapshots of the negotiation status and of the instants of the attributes negotiated that are present in a node of the negotiation graph. In this way, to specify not only the information regarding the negotiation state and the attributes values but also the actions that will contribute to the evolution of the negotiation, we model the nodes of a graph of the negotiation sequence as sets of particles, called *negotiation atoms*. Therefore, a negotiation atom, denoted *atom(s,ph),* is a set of resources, called *particles*, that describe the negotiation state in terms of the negotiation sequence *s* for

the negotiation stage *ph*. We defined in this way five types of particles: *representation* particles, *event* particles, *message* particles, *control* particles, and *computational* particles.

In our negotiation model, a negotiation sequence keeps, in the nodes of the graphs, sets of snapshots, images that a participant has about the negotiation status and about the attributes that are negotiated in the current sequence as well as in all other sequences for which there is a distribution of information. This information is modeled within the negotiation process as representation particles that are described by three parameters: *Name, S*, and *I*

- *Name* is defined by concatenation of the identifiers of the participants with the sequence under consideration (e.g., $p_i s_j$).
- *S* takes values in the set *Status*= {*initiated, undefined, success, failure*}. This value corresponds to the value returned by the function *status()*.
- *I* takes values in the set *Issues* of the negotiated attributes with the associated values (e.g., *I*= {*size =1k, cost =9.5k, delay =5*}). This value corresponds to the value returned by the function *issues()*.

In this way, a representation particle of an atom, associated to a sequence *s* for a phase *ph*, is a snapshot of the sequence *s* for the phase *ph*. To provide a detailed description of the negotiation sequences involved in a negotiation phase, we define the following particles:

- *localr(Name, S, I)* : local representation particle. This particle holds the local snapshot of the current sequence;
- *extr(Name, S, I)* : external representation particle. This particle holds the external snapshot that describes the modality in which another sequence perceives the same negotiation phase;
- *firstr(Name, S, I)* : external negotiation particle. This particle holds the external snapshot associated to the sequence that generated the current sequence.

In this way, a new node of a negotiation sequence may be described through a set of representation particles that are part of the same atom.

The particles *event* specify the types of transitions used by IAMs in terms of the message types that are exchanged within a negotiation. A particle event is described by three parameters:

- *Id* identifies the atom to be cloned;
- *New_id* identifies the newly created atom;
- *Msg* contains the negotiation message with data that will contribute to the evolution of the negotiation in the newly created atom.

To facilitate the identification of both the cloning operation and of the direction in which the new negotiation atom will evolve, we propose four particles event: *clone_propose, clone_accept, clone_reject,* and *clone_create*. The particles clone_propose(Id, New_id, Msg), clone_accept(Id, New_id, Msg), and clone_reject(Id, New_id, Msg**)** are modeling an event that signals the existence of a new negotiation message of type *propose*, of type *accept*, and of type *reject*, respectively. The particle clone_create(Id, New_id, Msg) models an event that signals the existence of a new negotiation message that announces creation of a new sequence for the current negotiation.

The particles *message* model the messages sent to allow their processing in terms of their interpretations in a typical negotiation process. The particles *message* have the following parameters:

- *Rname* and *New_r_name* are identifiers of the sequence that generates the message and of a new sequence that is invited to negotiation, respectively.
- *Content* represents the content of the message which is a proposal regarding the negotiation task.
- *Type* represents an identifier of the new coordination policy that satisfies a certain pattern and that must be managed by the sequence invited to negotiation.

We propose four types of message particles: *propose, accept, reject,* and *create*. The particle *propose(Rname, Content)* signals the existence of a new proposal in the negotiation process, and the particles *accept(Rname)* and *reject(Rname)* signal the existence of an acceptance of a proposal and the existence of a denial of a proposal, respectively. The proposal to accept and, respectively, to deny was sent by a participant in the negotiation through the sequence *Rname*. The particle *create(New_r_name, Type)* signals the existence of a new sequence that is part of the current negotiation phase and that is identified by *New_r_name*.

To properly formulate a coherent execution of a negotiation process, we introduced the *control* particles. These particles have several functions in the computation space of a negotiation sequence:

- an identification function (e.g., *name(Id)*) that identifies the negotiation atoms by specifying an unique value to the parameter *Id* for each atom. This unique value allows only to the specified atom to consume various events introduced in the system that are addressed to this atom;
- a limitation function (e.g., *start(), enable(), freeze(), waiting())* that introduces the concept of control over the methods that may induce errors in negotiation. This type of particles limits the number of methods that may be executed in a given state. In this manner, we may establish a proper succession in the execution of certain methods. For example, we will use the particles *enable* and *freeze* to favor the methods to consume the events and to consume the messages, respectively. Through the aid of these two methods we will introduce a well-defined order in the negotiation process, first the creation of a negotiation atom and, second, the evolution of the negotiation phase in this newly created atom;
- a notification function (e.g., *stop(Accord), ready(Accord)*).

4.4. Description of the negotiation process

According to our approach regarding the negotiation, the participants to a negotiation may *propose* offers and each participant may decide in an autonomous manner to stop a negotiation either by *accepting* or by *rejecting* the offer received. Also, depending on its role in a negotiation, a participant may *invite* new participants to the negotiation. To model this type of negotiation, we will make use of the previously defined particles and we will propose the methods to manage the evolution of these particles.

Through the use of the metaphor IAMs, the evolutions of the negotiation phases correspond to the evolutions at the atoms level. The evolution may be regarded as a process consisting of two stages: a *cloning* operation of the atom existent in the initial stage and a *transformation* operation within the cloned atom to allow for the new negotiation phase.

The cloning operation is expressed by a set of methods involving the particles *event* and these methods are used to facilitate the evolution of the negotiation. We propose the

following methods associated to the particles event to model the cloning of an atom where new message particles are introduced:

- The method *Propose* is associated to the particle event *clone_propose(Id, New_id, Msg)* and models the introduction of a new proposal (*clone_propose*) by one of the participants to the negotiation. This method is expressed:
name(Id) @ enable @ clone_propose(Id, New_ id, Msg)<>- (enable @ name(Id)) & (freeze @ name(New_ id) @ propose(Rname, Content))

  - the atom identified by the particle *name(Id)* is cloned. The new proposal contained in the particle *propose(Rname, Content)* will be introduced in the new atom *name(New_id)*.

- The method *Accept* is associated to the event particle *clone_accept(Id, New_id, Msg)* and models the case when one of the participants sent a message indicating acceptance (*clone_accept*) of an older proposal. This method is expressed:
name(Id) @ enable @ clone_accept(Id, New_ Id, Msg) <>- (enable @ name(Id)) & (freeze @ name(New_ Id) @ accept(Rname))

  - the atom identified by the *name(Id)* is cloned. The message to accept that is contained in the particle *accept(Rname)* will be introduced in the new atom *name (New_id)* .

- The method *Reject* is associated to the event particle *clone_reject(Id, New_id, Msg)* and models the denial of an older proposal (*clone_reject*) made by one of the participants. This method is expressed:
name(Id) @ enable @ clone_reject(Id, New_ Id, Msg) <>- (enable @ name(Id)) & (freeze @ name(New_ Id) @ reject(Rname))

  - the atom identified by the particle *name(Id)* is cloned. The message of denial contained in the particle *reject(Rname)* will be introduced in the new atom *name(New_ id)*.

- The method *Create* is associated to the event particle *clone_create(Id, New_id, Msg)*. This method models the invitation of a new sequence (*clone_create*) made by one of the participants toward the distribution of the newly created negotiation phase. This method is expressed:
name(Id) @ enable @ clone_create(Id, New_Id, Msg) @ <>- (enable @name(Id)) & (freeze @ name(New_Id) @ create(Rname, Type))

  - the atom identified by the particle *name(Id)* is cloned and a particle *create(Rname, Type)* is introduced in the new atom *name(New_ id)* that will subsequently generate a new representation particle for the new sequence that is participating to the negotiation.

The particles message participate to transformation methods that change the negotiation phase of an atom by replacing the representation particles of the negotiation

sequences involved in the generation or in the receiving of the messages that are exchanged. Next we propose the following transformation methods:

- The transformation method associated to the particle *propose(Rname, Content)* contributes to the local evolution of a negotiation phase regarding the status and the attributes negotiated. This evolution takes place by replacing, in the existing atom, all representation particles that are involved (depending on the method) with the new particles that have the status changed to *undefined*, and the set of the negotiated attributes (Issues) contains the new proposal expressed in the *Content* of the message particle.

freeze @ localr(Rname1, S1, I1) @ extr(Rname, S2, I1) @ propose(Rname, Content)<>- enable@ localr(Rname1, undefined, I) @ extr(Rname, undefined, I)

- The transformation method associated to a particle *accept(Rname)* leads to the local evolution of a negotiation phase regarding the status. The evolution is made by replacing, in the existing atom, the representation particles involved with the new particles whose status has been changed from *initiated* or *undefined* to *success* :

freeze @ localr(Rname1, S1, I1) @ extr(Rname, S2, I1) @ accept(Rname) <>- localr(Rname1, success, I1) @ extr(Rname, success, I1)

- The transformation method associated to a particle *reject(Rname)*. This is similar to the method of particle *accept(Rname)*, the distinction being that the evolution of the negotiation phase is made through modifying the status of the representation particles involved from *initiated* or *undefined* to *failure*:

freeze @ localr(Rname1, S1, I1) @ extr(Rname, S2, I1) @ reject(Rname) <>- localr(Rname1, fail, I1) @ extr(Rname, fail, I1)

- The transformation method associated to a particle *create(New_r_name, Type)* contributes to the evolution of a negotiation phase regarding the number of the sequences that participate to this negotiation phase. This evolution is made by introducing in the corresponding atom a new representation particle:

freeze @ create(Rname, type) <>- extr(Rname, init, ∅) @ enable

As soon as this sequence is invited to the negotiation, its status is *initiated* and its set of the negotiated attributes is the empty set.

The evolution of all negotiation atoms and the negotiation phases take place in parallel. To model the coordination of the execution of the negotiation process perceived within a sequence, we used the communication mechanism among the existing negotiations. This type of particles that are part of the communication process among different negotiation atoms communicate to all negotiation atoms a certain result.

In the negotiation processes, the messages hold meta-information regarding the content of the messages that describe the proposals in terms of the value of different attributes of the negotiation object. To handle the negotiation proposals we use the concept of "*raw*" *computation* introduced within IAMs. We assume that each of the atoms implicitly contains in its state particles for processing different proposals in terms of mathematical operations or of strings manipulations.

In the next section, we will present, as an example, the description of the Block service using the model proposed above.

5. Description of the Block service

The service Block is used in the negotiations where the task must be executed in its totality by a single partner of the negotiation process. Its main role is to mediate the negotiation between the printshop that initiated the negotiation and all other printshops that are invited to the current negotiation. The mediation is made with the goal of establishing a contract regarding the execution of the whole task by a single participant.

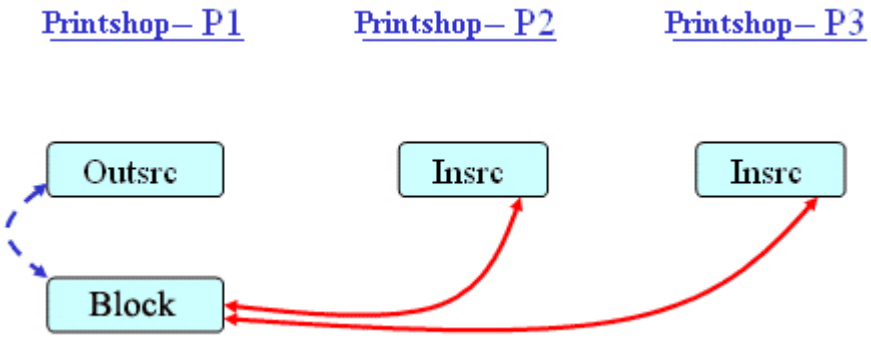

Figure 3**. Interactions of the Block Service**

In this way, a service, called "Block", is set to manage the constraint to not split the subcontracted task in different slots. In figure 3, we presented the services involved in a negotiation among a printshop P1 and other two printshops P2 and P3. In this case the negotiation begins by the initialization of Outsrc that invites to the negotiation the service Block. Subsequently, Block connects the services Insrc of each partner and will coordinate bilateral negotiations with the two partners P2 and P3, simultaneously. As soon as all services are connected, the interaction process between the participants may begin. During this process, the NegF's of each printshop involved in the negotiations begin to generate and exchange proposals and counter-proposals regarding the task at hand.

The negotiation ends when the printshop P1 reaches an agreement with one of the partners (e.g., P2) regarding the set of attributes that describe the task being negotiated. At the same time, P1 ends the negotiation with P3, this coordination being provided by the service Block. It should be noticed that the negotiation may also end without reaching an agreement (e.g., a time limit set for the negotiation has passed or the two partners P2 and P3 are no longer interested in the negotiation).

The following actions will be modeled:

- initialization of the negotiation: the sequence *s*1 invites sequences *s*2 and *s*3 to the negotiation;
- processing of the negotiation proposals;
- processing of the messages to accept a proposal ;
- implementation of the constraint to accept the task as a whole (i.e., the constraint regarding the size of the task contracted);
- processing of the messages to reject a proposal;
- end of the negotiation.

We assume that for each partner involved in the negotiation N there is a negotiation sequence. By using *Program Formula* we define all methods that model the entire negotiation process that must be managed by a certain sequence (e.g., sequence *s*1). We have identified this behavior using a particular negotiation sequence named s_{Block}.

Assume that the first atom of the negotiation sequence s_{Block} initially contains the following particles: **name**(Id), **start**, **localr**(Rname1,initiated,∅) – the representation particle of the sequence Block, **firstr**(Rname2,initiated, ∅) – the representation particle of the sequence Outsrc, **test_size**(value) – the particle that contains the size of the task fixed by the participant p_1.

Assuming that the sequence s_{Block} sees all the bilateral negotiations, this sequence will participate in the negotiation from the beginning, thus from the moment when invitations are transmitted to the potential participants. For each event **clone_create**(Id, New_Id, Msg) a new atom is generated - method (1.). The particle {**string_create**(Msg, Rname, Type)} will call a local computational particle to determine in the variable Msg the values of the parameters of the particle **create**(Rname, Type). The second method (2.) generates a new representation particle **extr(**Rname,initiated,∅) that represents the image of the new sequence over this new negotiation phase. The control particle **start,** that is only present in the first atom, limits the introduction of events **clone_create** to the first atom. In this way, all of the cloned atoms, starting with the first atom, will be negotiation phases distribute between the participant p1 and only one other participant.

(1.) **name**(Id) @ **start** @ **clone_create**(Id, New_Id, Msg) @ {**string_create**(Msg, Rname, Type)} <>- (**start** @ **name**(Id)) & (**freeze** @ **name**(New_Id) @ **create**(Rname, Type))

(2.) **freeze** @ **create**(Rname, type) <>- **extr(**Rname, initiated, ∅) @ **enable**

The following two methods manage the dependence of the size of the task negotiated, according to the principle Block service. Next, starting from the newly created atoms, we will continue the negotiation with the newly received proposals - method (3.). The proposal will result in the evolution of the new negotiation phase, only if the constraint imposed on the size of the task is satisfied - the computational particle {**substring**(value,Content,true)} verifies this constraint – method (4.).

(3.) **name**(Id)@**enable** @ **clone_propose**(Id, New_ Id, Msg) @{**string_propose**(Msg, Rname, Content)} <>- (**enable** @ **name**(Id)) & (**freeze** @ **name**(New_ Id) @ **propose**(Rname, Content))

(4.) **freeze** @ **test_size** (value) @ **localr**(Rname1, S1, I1) @ **firstr**(Rname2, S2, I1) @ **extr(**Rname3, S3, I1) @ **propose**(Rname, Content) @ {**substring**(value,Content, true)}@ {**construct**(I1,Content,I)}<>- **enable**@ **localr**(Rname1, undefined, I) @ **firstr**(Rname2, undefined, I) @ **extr(**Rname3, undefined, I)

The methods (5.) to (9.) model the dependencies regarding the status. In all the negotiation phases that are valid (those having the control particle **enable**), the negotiation partners may accept the current proposal - method (5.). In the event that the two partners will reach an agreement, in the newly created atom, the representation particles for a single partner will be in the status *success* (6. and 7.). We introduce a new control particle (**waiting**) to preserve the negotiation atom only for events of type **clone_accept** or **clone_reject** (8. and 10.). An agreement is reached only in the event that all three

representation particles are in the status *success* – method (9). In this situation, all other negotiation atoms are instructed to stop the negotiation (the particle **stop** is introduced through the broadcasting mechanism in all atoms of the negotiation sequence s_{Block}).

(5.) **name**(Id) @ **enable** @ **clone_accept**(Id, New_ Id, Msg) @ { **string_accept**(Msg, Rname)}<>- (**enable** @ **name**(Id)) & (**freeze** @ **name**(New_ Id) @ **accept**(Rname))

(6.) **freeze** @ **localr**(Rname1, S1, I) @ **firstr**(Rname2, S2, I) @ **accept**(Rname2)<>- **localr**(Rname1, success, I) @ **firstr**(Rname2, success, I) @ **waiting**

(7.) **freeze** @ **extr(**Rname3, S3, I) @ **accept**(Rname3)<>- **extr(**Rname3, success, I) @ **waiting**

(8.) **name**(Id) @ **waiting** @ **clone_accept**(Id, New_ Id, Msg) @ { **string_accept**(Msg, Rname)}<>- **name**(Id) @ **freeze** @ **accept**(Rname)

(9.) **localr**(Rname1, success, I) @ **firstr**(Rname2, success, I) @ **extr(**Rname3, success, I) @ ^**stop** <>- **ready**(I)

Next we present in methods (10.) and (11.) the event of type **clone_reject** and, in method (12.), the message of type **reject**.

Method (10.) models the transformation of the event **clone_reject** in a negotiation atom where one of the participants has made a proposal of type accept in the current negotiation phase (the particle **waiting** is created only in a method processing a message of type accept – method 7.). The method (11.) models the transformation of the event **clone_reject** into a negotiation atom where, temporary, the proposal made is neither accepted nor rejected. The two methods introduce the message particle **reject** in the current negotiation atom. In processing the message **reject**, we choose to preserve the negotiation phase associated to the current atom by modifying all statuses to *failure* and preventing all possible evolutions of the atom through the use of all control particles – method (12.).

Method (13.) is a rule that completely destroys all the negotiation atoms visible through the sequence s_{Block}, except the negotiation atom that contains the agreement regarding the negotiation (the particle **stop** has been created through this atom – method 9.).

(10.) **name**(Id) @ **waiting** @ **clone_reject**(Id, New_ Id, Msg) @ { **string_reject**(Msg, Rname)}<>- **freeze** @ **name**(New_ Id) @ **reject**(Rname)

(11.) **name**(Id) @ **enable** @ **clone_reject**(Id, New_ Id, Msg) @ { **string_reject**(Msg, Rname)}<>- **freeze** @ **name**(New_ Id) @ **reject**(Rname)

(12.) **freeze** @ **localr**(Rname1, S1, I) @ **firstr**(Rname2, S2, I) @ **extr(**Rname3, S3, I) @ **reject**(Rname)<>- **localr**(Rname1, failure, I) @ **firstr**(Rname2, failure, I) @ **extr(**Rname3, failure, I)

(13.) **stop** <>- **#t**

6. Conclusions

This paper aims at modeling the negotiation process at least at three levels (middleware, multi-agent and human).

We propose a decentralized multi-issue negotiation model in which a set of agents can conduct several one-to-one conversations in a concurrent manner according to the coordination services.

This kind of alliance is typical of virtual enterprises, e-business, and e-commerce networks.

References

1. Alloui, I., Andreoli, J.M., Boissier, O., Bratu, M.B., Castellani, S. and Megzari, K. **E-Alliance: a Software Infrastructure for Inter-Organizational Alliances**, In "9th ISPE International Conference on Concurrent Engineering: Research and Applications (CE2002)", Cranfield University (UK), 2002
2. Andreoli, J.M **Object Orientation with Parallelism and Persistence**, chapter Coordination as negotiation transactions, Kluwer Academic Publishers, 1996
3. Andreoli, J.M. **Coordination in LO**, in "Coordination Programming: Mechanisms, Models and Semantics" Imperial College Press, 1996
4. Andreoli, J.M., Castellani, S. and Munier, M. **AllianceNet: Information Sharing, Negotiation and Decision-Making for Distributed Organizations**, in "Proc. of EcWeb", Greenwhich, U.K., 2000
5. Andreoli, J.M. and Castellani, S. **Towards a Flexible Middleware Negotiation Facility for Distributed Components**, in "Proc. of DEXA <<E-Negotiation>> workshop", Munich, Germany, 2001
6. Bui, V. and Kowalczyk, R. **On constraint-based reasoning in e-negotiation agents**, in "AMEC III", LNAI, 2003, pp. 31-46
7. Carron, T., Proton, H. and Boissier, O. **A Temporal Agent Communication Language for Dynamic Multi-Agents Systems**, in "Proc. of 9th MAAMAW", LNAI 1647, 1999
8. Cretan, A. **Virtual Alliances among Autonomous Organizations**, in "Digital Economy – The sixth International Conference on Economic Informatics", Bucharest, Romania, 2003, pp. 725-731
9. Cretan, A., **Intelligent Solutions for Virtual Negotiations**, in "Information & Knowledge Age – The seventh International Conference on Informatics in Economy", Bucharest, Romania, 2005, pp. 441-446
10. Cretan, A., **Negotiation processes within inter-organizational alliances,** in "Economic Informatics Journal", Vol. XII/No. 3 (47), 2008
11. Faratin, P. **Automated service negotiation between autonomous computational agent**, Ph.D. Thesis, Department of Electronic Engineering Queen Mary & West-field College, 2000
12. Keeny, R. and Raiffa, H. **Decisions with Multiple Objectives: Preferences and Value Tradeoffs**, JohnWilley & Sons, 1976
13. Smith, R. and Davis, R. **Framework for cooperation in distributed problem solving**, IEEE Transactions on Systems, Man and Cybernetics, SMC-11, 1981
14. Sycara, K. **Problem restructuring in negotiation**, Management Science, 37(10), 1991

[1] Adina Cretan graduated Polytechnic University of Craiova, Faculty of Electromechanical Servo systems in 1994. She also follows Master of Business Administration courses within Academy of Economic Studies Bucharest and CNAM Paris.
Currently she is working for Absolute IT Solutions, leading company in providing software solutions for several major US companies, as Senior Consultant Software Engineer.
Main capabilities and skills: advanced programming in Java, PHP, ASP, project management and team leading experience.
Her research interests include multi-agent systems, electronic commerce, use and design of formal methods, coordination systems, communication protocols. She has published six articles in these fields.
She is also interesting in the area of algebra and number theory and she is co-author of two articles.

[2] Codification of references:

[1]	Alloui, I., Andreoli, J.M., Boissier, O., Bratu, M.B., Castellani, S. and Megzari, K. **E-Alliance: a Software Infrastructure for Inter-Organizational Alliances**, In "9th ISPE International Conference on Concurrent Engineering: Research and Applications (CE2002)", Cranfield University (UK), 2002
[2]	Andreoli, J.M **Object Orientation with Parallelism and Persistence**, chapter Coordination as negotiation transactions, Kluwer Academic Publishers, 1996
[3]	Andreoli, J.M. **Coordination in LO**, in "Coordination Programming: Mechanisms, Models and Semantics" Imperial College Press, 1996
[4]	Andreoli, J.M., Castellani, S. and Munier, M. **AllianceNet: Information Sharing, Negotiation and Decision-Making for Distributed Organizations**, in "Proc. of EcWeb", Greenwhich, U.K., 2000
[5]	Andreoli, J.M. and Castellani, S. **Towards a Flexible Middleware Negotiation Facility for Distributed Components**, in "Proc. of DEXA <<E-Negotiation>> workshop", Munich, Germany, 2001
[6]	Bui, V. and Kowalczyk, R. **On constraint-based reasoning in e-negotiation agents**, in "AMEC III", LNAI, 2003, pp. 31-46
[7]	Carron, T., Proton, H. and Boissier, O. **A Temporal Agent Communication Language for Dynamic Multi-Agents Systems**, in "Proc. of 9th MAAMAW", LNAI 1647, 1999
[8]	Cretan, A. **Virtual Alliances among Autonomous Organizations**, in "Digital Economy – The sixth International Conference on Economic Informatics", Bucharest, Romania, 2003, pp. 725-731
[9]	Cretan, A., **Intelligent Solutions for Virtual Negotiations**, in "Information & Knowledge Age – The seventh International Conference on Informatics in Economy", Bucharest, Romania, 2005, pp. 441-446
[10]	Cretan, A., **Negotiation processes within inter-organizational alliances,** in "Economic Informatics Journal", Vol. XII/No. 3 (47), 2008
[11]	Faratin, P. **Automated service negotiation between autonomous computational agent**, Ph.D. Thesis, Department of Electronic Engineering Queen Mary & West-field College, 2000
[12]	Keeny, R. and Raiffa, H. **Decisions with Multiple Objectives: Preferences and Value Tradeoffs**, JohnWilley & Sons, 1976
[13]	Smith, R. and Davis, R. **Framework for cooperation in distributed problem solving**, IEEE Transactions on Systems, Man and Cybernetics, SMC-11, 1981
[14]	Sycara, K. **Problem restructuring in negotiation**, Management Science, 37(10), 1991