# Agent-based approach to model parallel and concurrent negotiations

**Adina Crețan[1*], Carlos Coutinho[2], Ben Bratu[3], Ricardo Jardim-Goncalves[4]**

[1]"Nicolae Titulescu" University, e-mail: badina20@yahoo.com, 185 Calea Văcărești, District 4, Bucharest, 040051, Romania
[2]Caixa Magica Software, e-mail: c.coutinho@campus.fct.unl.pt, Rua Soeiro Pereira Gomes, Lote 1-4 B, 1600-196, Lisboa, Portugal
[3]Extreme Computing R&D BULL, e-mail: ben.bratu@bull.net, Rue Jean Jaurès B.P.68, 78340 Les Clayes-sous-Bois, France
[4]CTS, Departamento de Engenharia Electrotecnica, Faculdade de Ciencias e Tecnologia, Universidade Nova de Lisboa, e-mail: rg@uninova.pt, Lisboa, UNINOVA, Portugal

## Abstract

*Each organization has limited resources and in order to better accomplish a higher external demand, the managers are forced to outsource parts of their contracts even to concurrent organizations. In this concurrent environment each organization wants to preserve its decision autonomy and to disclose as little as possible from its business information. To model this interaction our approach is to define a framework for describing and managing parallel and concurrent negotiations among independent organizations acting in the same industrial market. The complexity of our negotiation model is done by the dynamic environment in which multi-attribute and multi-participant negotiations are racing over the same set of resources. We are using the metaphor Interaction Abstract Machines (IAMs) to model the parallelism and the non-deterministic aspects of our negotiation process.*

***Keywords: Negotiation model, intelligent agents, computational method, dynamic environment, constraint satisfaction programming.***

## 1. Introduction

The advent of the Internet and more recently the cloud-computing trend have led to the development of various forms of virtual collaboration in which the organizations are trying to exploit the facilities of the network to achieve higher utilization of their resources. We try to provide support to these collaboration activities and we propose negotiation as a fundamental mechanism for such 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 framework for coordinating parallel negotiations occurring in a business-to-business interaction.

We consider a scenario of distributed autonomous business organizations (e.g., carpentry workshops). Each organization autonomously manages its contracts, schedules and resources. When a new task request reaches an organization, the manager analyses its acceptance taking into account the current schedule and the resources availability. After the manager accepts the new job task, he may decide to perform it locally or to partially outsource it. If the manager decides to outsource a job, he starts a negotiation within the collaborative infrastructure 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 organization, which defines an inter-organizational workflow enacting the business process fulfilling the outsourced jobs and a set of obligation relations among participants [1]. Each 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. This collaborative infrastructure should flexibly support negotiation processes respecting the autonomy of the partners. The main objective of this paper is to propose a framework for modelling parallel negotiations in a dynamical system with autonomous organizations. In Section 2, we describe a formal interaction model to manage multiple concurrent negotiations. Section 3 presents an example for modelling the negotiation process by using the metaphor Interaction Abstract Machines (IAMs). In section 4, we briefly present the architecture of the negotiation

[*] Corresponding author: badina20@yahoo.com

system in which the interactions take place and describe the coordination services that manage different negotiations which may take place simultaneously [2]. Also, we present how the proposed model can be used for describing a particular coordination negotiation service. Finally, Section 5 concludes this paper.

## 2. Building the Negotiation Model

In this section we propose a formal model to settle and to manage the coordination rules of one or more negotiations, which can take place in parallel. First, we define the basic concepts underlying the model and then, by using the metaphor of Interaction Abstract Machines (IAMs), we describe the negotiation model. We introduce the Program Formula to define the methods used to manage the parallel evolution of multiple negotiations.

### 2.1. Fundamental 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:

A *Negotiation Model* is defined as a quintuple M = <T, P, N, R, O> where:

- T denotes *the time of the system*, assumed to be discrete, linear, and uniform [3];
- P denotes *the set of participants* in the negotiation framework. The participants may be involved in one or many negotiations;
- N denotes *the set of negotiations* that take place within the negotiation framework;
- R denotes *the set of policies of coordination* of the negotiations that take place within the negotiation framework;
- 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 $Sq = \{si \mid i \in \mathbb{N}\}$ denote the set of *negotiation sequences*, such that $\forall si, sj \in Sq$, $i \neq j$ implies $si \neq sj$. A *negotiation sequence* $si \in Sq$ such that $si \in N(t)$ 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.

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).

### 2.2. Metaphor Interaction Abstract Machines (IAMs)

The metaphor Interaction Abstract Machines (IAMs) will be used to facilitate modeling of the evolution of a *multi-attribute, multi-participant, multi-phase negotiation*. In IAMs, a system consists of different *entities* and each entity is characterized by a state that is represented as a set of *resources* [4]. 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:

- *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;
- *communication:* insertion of the copies of the pre-defined resources in all entities that are present in the system at that time instance.

### 2.3. 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., *pjsj*).
- *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. 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)*).

Handling and creating negotiation proposals is performed through *computational* particles. Using the notion of *raw computation* introduced by IAMs, we assume that each atom contains implicit in his state particles that processes various negotiating proposals in terms of mathematical operations or of strings manipulations. For example, the particle *{construct (I1, Content, I)}:* given the previous instantiation of the negotiated attributes (*I1*) and new values of the attributes of the proposal made in *Content*, the *construct* particle builds the new instance of the attributes (*I*).

## 3. Modelling 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.

As we have seen, a characteristic of negotiation is its multi-node image, which allows parallel development of several phases of negotiation. A possibility to continue a negotiation is to create a new phase of negotiation from an existing one. In this regard, the Figure 1 presents the possible evolutions of a ph0 phase of negotiation described by the *atom (s,ph0)*.

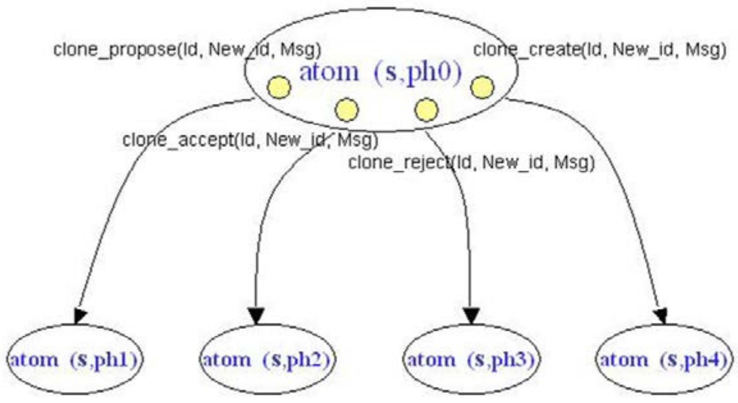
clone_propose(Id, New_id, Msg)
atom (s,ph0)
clone_create(Id, New_id, Msg)
clone_accept(Id, New_id, Msg)
clone_reject(Id, New_id, Msg)
atom (s,ph1)
atom (s,ph2)
atom (s,ph3)
atom (s,ph4)

**Fig. 1.** Evolution of negotiation process by cloning an atom

In accordance with the aspects of negotiation for which changes are made, three new negotiation phases are possible:

- evolution of negotiated attributes and / or of their value from *atom(s,ph0)* to *atom(s,ph1):* a participant sends a new proposal thus achieving either the *contraction* of the negotiation attributes, or their *extension*, by the introduction of new attributes to negotiate;
- evolution of the negotiation status perceived by one of the sequences sharing the new negotiation phase: one of the participants accepts - *atom(s,ph2)* - or refuses a proposal - *atom(s,ph3)*;
- evolution of participants and of dependences among negotiations by the evolution of the number of sequences sharing the same negotiation phase: a sequence can invite a new sequence to share a new phase of negotiation *atom(s,ph4).*

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*), made 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 has sent a message of acceptance of an older proposal (*clone_accept*).

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 acceptance message 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 refusal message 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 for sharing 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 further generate the occurrence of a new representation particle for the new sequence participating in the negotiation.

These methods are described in a generic way. Thus, new particles may be added depending on how the current sequence builds negotiation graphs.

By these methods of the event particles, the duplication of an atom has been modeled, in which new message particles are introduced (Figure 2). In the new atom, the representation particles for the current negotiation phase remain identical with those of the first atom.

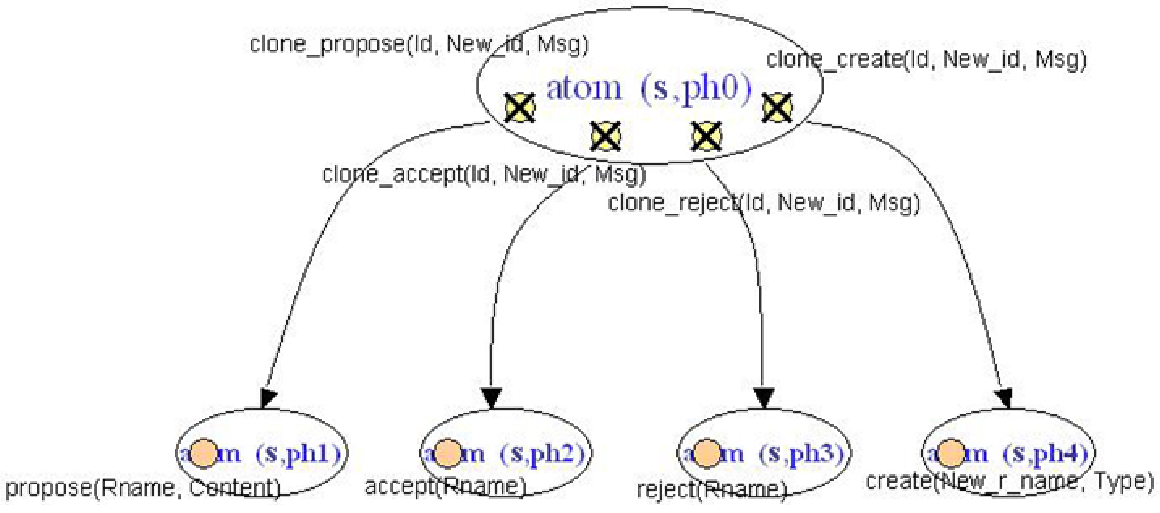

**Fig. 2**. Evolution of negotiation process by transformation of an atom state

According to our approach, the evolution of the negotiation process takes place by changing or creating a new negotiation phase. This phase can evolve according to: *i)* status; *ii)* attributes negotiated; *iii)* number of sequences participant in the negotiation, as in the following:

- The sequence of the statutes is the following: *i)* the sequence *s* finds itself in the *initiated* state at the creation of a first atom and of a first phase of negotiation; *ii)* the sequence s switches to the *undefined* state at the moment of emission or reception of a message; *iii)* if a participant accepts or declines a proposal, the s associated sequence may pass to a *success* or *failure* statute.
- Referring to the negotiated attributes (*Issues*), the different messages contribute to the evolution of the multitude of attributes and their values.
- The introduction of a new sequence in the current negotiation is modeled by inserting a new particle of representation on the current negotiation phase, which models the instant image of a new sequence on the current phase of negotiation.

In order to model these evolutions at the level of a negotiation phase, the message-type particles described above have been defined. The *message* particles participate in the transformation methods, which change the negotiation phase of an atom by replacing the representation particles of the negotiation sequences involved in the creation or in the reception of the exchanged messages.

In the following, we propose the basic forms of the *transformation* methods. Depending on the particular constraints of the negotiation, other transformation methods and other particles can be defined for modeling the foreseen constraints.

- The *transformation* method associated to a *propose(Rname, Content)* particle 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*. Further, 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 atom changes the state by consuming the *propose()* particle as well as two representation particles to create new representation particles that describe the new proposal received.

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

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

The atom changes the state by consuming the *accept()* particle and two representation particles to create the new representation particles whose status is changed in *success*.

- The transformation method associated to a *reject(Rname)* particle. This method is similar to the *accept(Rname)* particle, except for the fact that the evolution of the negotiation phase is achieved by changing the status of the representation particles concerned from initiated or *undefined* into *failure* :

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

The atom changes the state by consuming the *accept()* particle and two representation particles to create the new representation particles which have changed the status into *failure*.

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

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

As this sequence is just invited in the negotiation, its status is *initiated* and its set of the negotiated attributes is the empty set.

Thus, the negotiation phases and the evolution of these phases have been described using representation particles, event particles and message particles.

Given the fact that IAMs metaphor achieves a non-deterministic execution of the methods, we have introduced the *control* particles (see section 3.3) in order to counter this disadvantage and achieve a coherent execution of a negotiation process.

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 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. We assume that all the negotiation participants use the same language and ontology.

Next section presents an example of modeling a negotiation composed of a set of negotiation sequences.

### 3.1. Example - modeling the negotiation process using the IAMs metaphor

In this example, a simple negotiation scenario will be presented, whose negotiation process corresponds to an exchange of proposals leading to an agreement.

In the proposed scenario, we consider a carpentry workshop of the p1 participant. The participant p1 decides to outsource a job (the assembling for 10K LM at a cost less than 2€/LM, within less than 5 days) to another carpentry workshop of a p2 participant.

The negotiation N that occurs between p1 and p2 is a bilateral negotiation.

It is described by two sequences: N(t) = {s1, s2} with s1∈sequences(t,p1), s2∈sequences(t,p2) and role(t,p1,N)=initiator, role(t,p2,N)=guest.

The scenario modeled subsequently takes place in three distinct steps:

- **step1** : after a first proposal made by participant p1 to participant p2, the participant p2 decides to send a new proposal;
- **step2** : the participants decide to agree on the second proposal;
- **step3** : the agreement is established and the negotiation stops.

Modeling takes place as it follows:

**Step 1**

Figure 3.a) shows the *s1* , *view(s1)= (p1,N,R1)* negotiation sequence with an a1 atom corresponding to a negotiation phase described by two representative particles, a local one and an external one, and two control particles, (*enable* and *name(a1)*). This atom can be considered as being the proposal made by p1 to p2 at the beginning of the negotiation.

Going on with the scenario, there is assumed that within the *a1* atom of the *s1* negotiation sequence, the *clone_propose(a1, a2, cost=18K delay=3)* event was introduced in order to announce a new proposal.

By using the *Propose* method, the *s1* negotiation sequence will contain two atoms (see Figure 3.b)): a1 atom that has changed by consuming the *clone_propose* particle and an a2 new atom, which is the a1 clone (representation particles not involved in the method remain unchanged in the two atoms). The expression of this method is:

name(Id) @ enable @ clone_propose(Id, New_ id, Msg)<>- (enable @ name(Id)) & ( freeze @ name(New_ id) @ propose(Rname, Content))

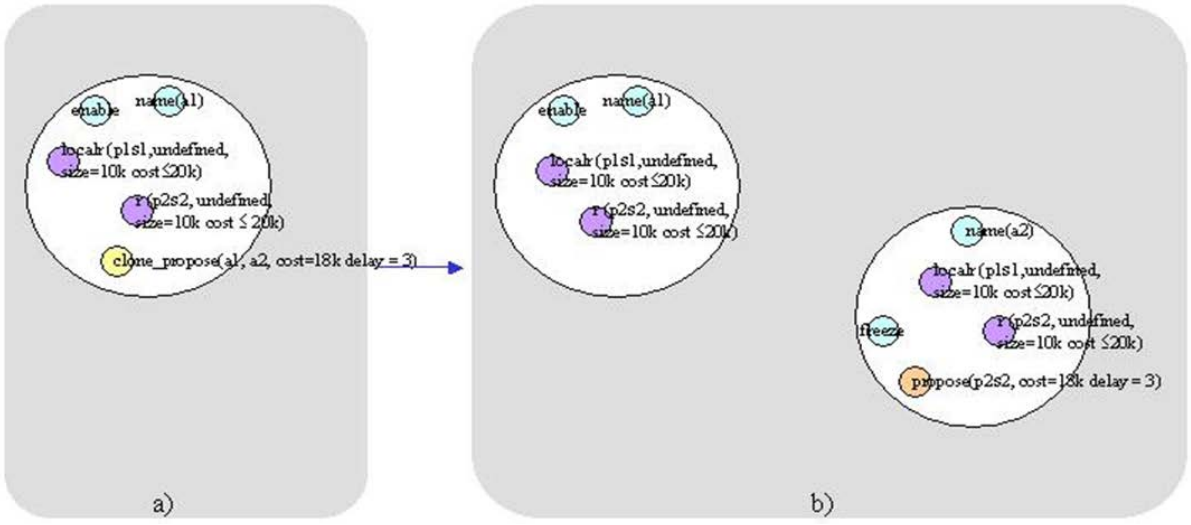
enable
name(a1)
localr(p1s1,undefined, size=10k cost ≤20k)
r(p2s2, undefined, size=10k cost ≤ 20k)
clone_propose(a1, a2, cost=18k delay = 3)
a)
enable
name(a1)
localr(p1s1,undefined, size=10k cost ≤20k)
r(p2s2, undefined, size=10k cost ≤20k)
name(a2)
localr(p1s1,undefined, size=10k cost ≤20k)
freeze
r(p2s2, undefined, size=10k cost ≤20k)
propose(p2s2, cost=18k delay = 3)
b)

**Fig. 3.** a) – Proposal of p1 to p2 contained in a1atom

b) – Transformation of a1 atom in a2

From now on, the negotiation is described through the two negotiation atoms, in which the negotiation process can evolve independently.

The following methods will help us to model the fact that the (*name*, *freeze, propose*) new particles introduced in the a2 atom will make that the negotiation phase attached to this atom have a different evolution in comparison to that of a1 atom.

For the *propose* particle, the following method will be used:

freeze @ localr(Rname1, St1, I1) @ extr(Rname2, St2, I1) @ propose(Rname2, Content)<>- enable@ localr(Rname1, undefined, I) @ extr(Rname2, undefined, I)

This method exchanges the representation particles, which preserve the old values of the attributes, with the new representation particles that describe the new proposals received.

Thus, to move from an *extr(p2s2, undefined, size=10K cost≤20K)* representation particle, and from a *propose(p2s2, cost=18K delay=3)* message particle to an *extr(p2s2, undefined, size=10K cost=18K delay=3)* representation particle, the existence of a computational-type particle called *construct(I1,Content,I)* has been supposed, which calculates this transformation (see Figure 3.c)).

The expression of this method is:

freeze @ localr(Rname1, St1, I1) @ extr(Rname2, St2, I1) @ propose(Rname2, Content) @ {construct(I1,Content,I)} <>- enable @ localr(Rname1, undefined, I) @ extr(Rname2, undefined, I)

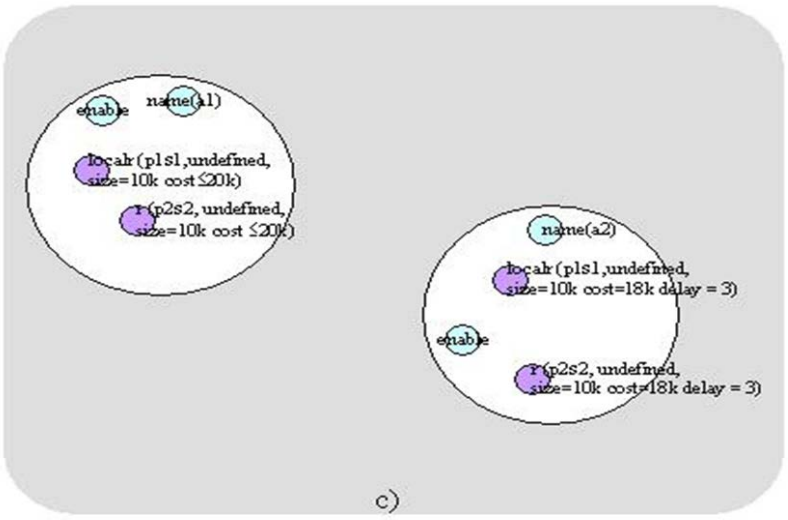
enable
name(a1)
localr (p1s1,undefined, size=10k cost ≤20k)
r (p2s2, undefined, size=10k cost ≤20k)
name(a2)
localr (p1s1,undefined, size=10k cost=18k delay = 3)
enable
r (p2s2, undefined, size=10k cost=18k delay = 3)
c)

**Fig. 3.** c) Evolution of a1 and a2 atoms

**Step 2**

Further, we assume that p1 participant examines the two proposals and decides to accept the second proposal. Acceptance is modeled as follows:

i) the atom containing the proposals to be accepted is duplicated by *Accept* method associated to an event *clone_accept()* particle;

name(Id) @ enable @ clone_accept(Id, New_ Id, Msg) <>- (enable @ name(Id)) & ( freeze @ name(New_ Id) @ accept(Rname))

ii) the clone atom evolves by consuming the *accept()* message particle;

freeze @ localr(Rname1, St1, I1) @ extr(Rname, St2, I1) @ accept(Rname) <>- localr(Rname1, success, I1) @ extr(Rname, success, I1)

Thus, in Figure 3.d) the s1 negotiation sequence of a p1 participant is composed of three negotiation phases. The third a3 atom contains a negotiation phase in which *success* is the negotiation status.

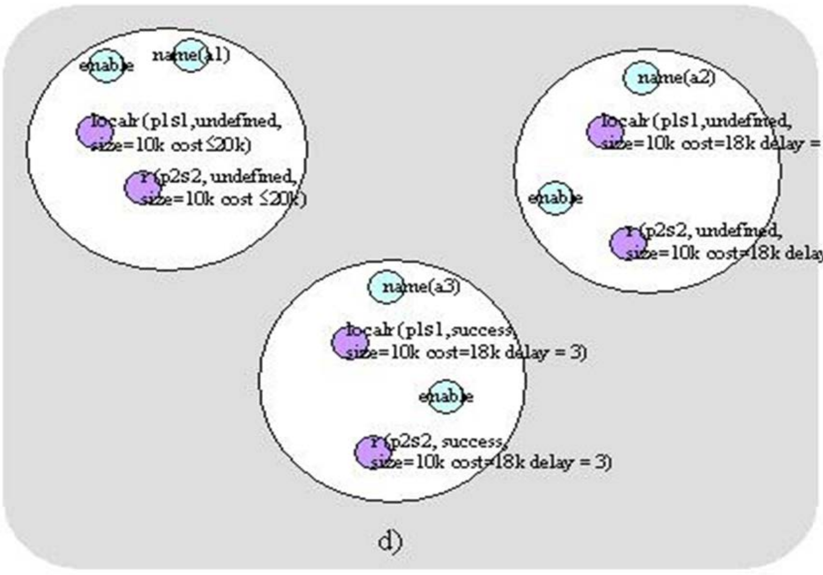
name(a1)
enable
localr(p1s1,undefined, size=10k cost≤20k)
r(p2s2, undefined, size=10k cost ≤20k)
name(a2)
localr(p1s1,undefined, size=10k cost=18k delay = 3)
enable
r(p2s2, undefined, size=10k cost=18k delay = 3)
name(a3)
localr(p1s1,success, size=10k cost=18k delay = 3)
enable
r(p2s2, success, size=10k cost=18k delay = 3)
d)

**Fig. 3.** d) Negotiation phases of the s1 sequence of a p1 participant

**Step 3**

The a3 atom can thus be perceived as the image of a negotiation phase on which the two participants have agreed. The existence of this agreement has to imply the fact that the negotiation has to come to an end.

In Figure 3.e), a3 atom changes its state and, at the same time, communicates to the other atoms to stop.

The expression of this method is:

localr(Rname1, success, I1) @ extr(Rname2, success, I1) @ ^stop(I1) <>- ready(I1)

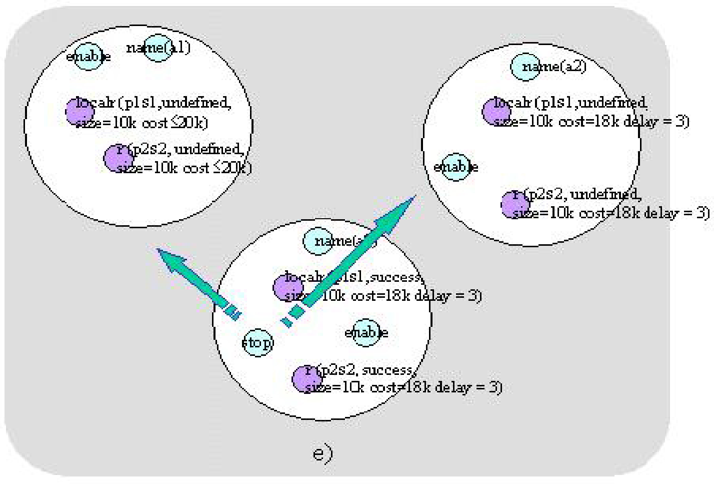
name(a1)
enable
localr(p1s1,undefined, size=10k cost≤20k)
r(p2s2, undefined, size=10k cost ≤20k)
name(a2)
localr(p1s1,undefined, size=10k cost=18k delay = 3)
enable
r(p2s2, undefined, size=10k cost=18k delay = 3)
name(a3)
localr(p1s1,success, size=10k cost=18k delay = 3)
stop
enable
r(p2s2, success, size=10k cost=18k delay = 3)
e)

**Fig. 3.** e) Transformation of a3 atom state

The method stop <>- #t makes that, at a certain moment, the atoms containing *stop* particles be dissolved (see Figure 3. f.), the only active atom being the one containing the agreement (see Figure 3.g)).

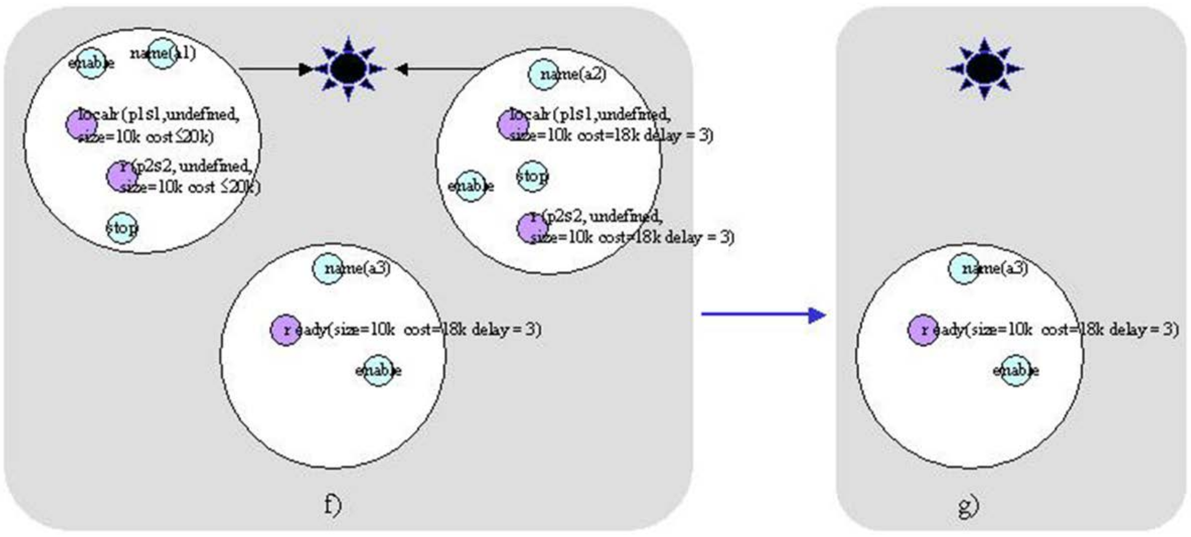
enable
name(a1)
localr(p1s1,undefined, size=10k cost ≤20k)
r(p2s2, undefined, size=10k cost ≤20k)
stop
name(a2)
localr(p1s1,undefined, size=10k cost=18k delay = 3)
enable
stop
r(p2s2, undefined, size=10k cost=18k delay = 3)
name(a3)
ready(size=10k cost=18k delay = 3)
enable
f)
name(a3)
ready(size=10k cost=18k delay = 3)
enable
g)

**Fig. 3**. f) Dissolution of a1 and a2 atoms containing stop particle

g) The active atom: a3 atom containing the agreement

Thus, in this section, the negotiation process by using the IAMs metaphor has been defined, and the negotiation composed of a set of negotiation sequences has been modeled. Each of these sequences was represented by a set of negotiation atoms, which, at their turns, each of them administrates independently a private negotiation phase as well as the set of the snapshots associated to the phase.

A snapshot describes the status of the negotiation and the set of the negotiated attributes. Also, a negotiation is characterized by the role of participants and the policy of coordination attached to it. These policies can be described by using different methods. The proposed model of the negotiation process allows describing the evolution of a negotiation by a parallel development of negotiation atoms associated to methods modeling the coordination policy.

In the next sections, the coordination model will be defined by using this describing model of the negotiation process. Our objective is to achieve a correspondence between the coordination policy that a sequence has to satisfy (static model), and the set of methods modeling the evolution in time of a negotiation sequence (dynamic model).

Given that each negotiation is composed of a set of negotiation sequences, the coordination process of one or more negotiations will be structured into modules (services) that correspond to the sequences involved in the negotiation.

In the next section, we will briefly describe the architecture of the negotiation system in which the interactions take place.

## 4. The Negotiation Infrastructure

The Negotiation infrastructure offers generic mechanisms to support negotiations in a distributed environment.

Figure 4 shows the architecture of the negotiation system:

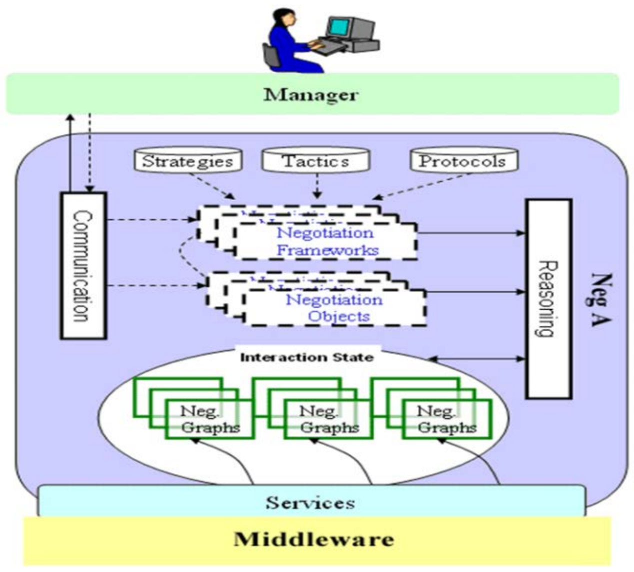

**Fig. 4**. The architecture of the negotiation system.

A Negotiation Agent (NegA) assists its 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 NegA of the other partners through the coordination and negotiation middleware [5].

Each negotiation is organized in three main steps: initialization; refinement of the job under negotiation; and closing [6]. The initialization step allows to define what has to be negotiated (Negotiation Object) and how (Negotiation Framework) [7]. A selection of negotiation participants can be made using history on passed negotiation, available locally or provided by the negotiation infrastructure [8]. In the refinement step, participants exchange proposals on the negotiation object trying to satisfy their constraints [9]. The manager participates in the initial definition of negotiation frameworks and objects [10]. During the negotiation the decisions are taken by the manager, assisted by his NegA - decision functions operate in the "Reasoning" box (Fig.4). For each negotiation, a NegA manages one or more negotiation objects, one framework and the negotiation status represented as several graph structures. 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 NegA interactions with the manager and with the other NegAs.

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 producing a coordinated modification of alternatives within the current negotiation [12]. 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 negotiation middleware [13].

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

- *Outsrc:* the main service for a participant who initiates a negotiation for outsourcing jobs by exchanging proposals among participants known from the beginning;
- *Insrc:* the main service for a guest participant who is invited in a negotiation for insourcing jobs;
- *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 for 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.

## 4.1. Coordination Services

According to our approach, the coordination problems managing the constraints among several negotiations can be divided into two distinct classes of services:

- Coordination services in a closed environment: services that build their images on the negotiation in progress and manage the coordination constraints according to information extracted only from their current negotiation graph (e.g., Outsrc, Insrc, Block, Split);
- Coordination services in an open environment: services that also build their images on the negotiation in progress, but they manage the coordination constraints according to available information in data structures representing certain characteristics of other negotiations currently ongoing into the system (e.g., Broker, Transport, SwapIn/SwapOut).

Following the descriptions of these services, we can state that unlike the services in a closed environment that manage the coordination constraints of a single negotiation at a time, the services in an open environment allow the coordination of constraints among several different negotiations in parallel. Thus, the introduction of coordination services level allows the management of all simultaneous negotiations in which a network partner can be involved. This level has two main functions:

- Mediation the transition between the negotiation image at the Negotiation Agent level and the image at the Middleware level;
- Allowing the implementation of various types of appropriate behavior in particular cases of negotiation.
- Consequently, each service is considered to correspond to a particular negotiation type.

Next section presents in detail the coordination services in a closed environment.

### 4.1.1. Outsrc Service

The *Outsrc* service is the main service of a negotiation. The automatic negotiation process is initiated by creating an instance of this service starting from the initial negotiation object. Further, this service must build the negotiation graph by following the negotiation requirements (i.e., assessment and creation of proposals and coordination rules). The service meets these requirements by manipulating the Xplore primitives [14].

Besides these functionalities, the *Outsrc* service has to interpret and check the negotiation constraints, which are set up in the following two data structures : *Negotiation Object and Negotiation Framework*.

The information provided by the structure of the Negotiation Object on the possible values of the attributes to be negotiated allow easily the *Outsrc* service to check whether the proposals received concern the attributes negotiated in the current negotiation and if they are associated to the values of the intervals specified.

For example, assuming that the Negotiation Object requires that the price should be (*cost <= 10k*), the *Outsrc* service can stop the continuation of the negotiation in the phases associated to the white nodes where the proposals are outside the interval.

Also, by using the *partner* coordination attribute, the *Outsrc* service can make known to the other services the participants imposed by the Negotiation Object or whether other services instantiate this attribute. In this regard, the *Outsrc* service can easily check if the associated value confirms the constraints imposed by the Manager.

At middleware level, the *Outsrc* service has also the function of administrating the transactional aspect of the negotiation. This service is seen like a *coordinator* and has the role to conclude an agreement among the service instances participating in the same negotiation.

Another *Outsrc* service functionality is to interpret and execute the tactics specified in the Negotiation Framework structure by connecting a combination of different instances of the other services.

Thus, the *Outsrc* service as well as the *Insrc* service described below are those connecting the aspects specified at the Negotiation Agent level and their implementation at the coordination services level.

### 4.1.2. Insrc Service

The *Insrc* service manages the negotiation from the organization side deciding to accept a task proposed in the collaborative networked environment, with some functionalities similar to those of the *Outsrc* service.

The differences come from the fact that this service does not have a complete picture on the negotiation and that, at the beginning of the negotiation, it has no information about what is negotiated or about the constraints of its Manager.

Therefore, looking to the differences, we can say at first that the image of the *Insrc* service on the negotiation graph is limited to the data referring only to its direct negotiation with the *Outsrc* service or with another service negotiating for the organization having initiated the negotiation.

Secondly, unlike the *Outsrc* service, which, from the beginning, has constraints specified by the Manager within the data structures of the *Negotiation Object* and the *Negotiation Framework*, the *Insrc* service has a close interaction with its own Manager on the new aspects required in the negotiation.

Thus, depending on attributes required by the negotiation initiator the *Insrc* service is able to progressively build the data structures describing the Manager's preferences on the negotiation object and on the negotiation process.

### 4.1.3. Block Service

The *Block* service is used in the negotiations where the task must be executed in its totality by a single participant of the negotiation process. Its main role is to mediate the negotiation among the enterprise that initiated the negotiation and all other enterprises that are invited to the current negotiation. The mediation is performed with the goal of establishing a contract regarding the execution of the whole task by a single participant. In this way, this service is set to manage the constraint of not splitting the subcontracted task in different slots.

An example of the interactions of the Block service is provided by the Figure 5.

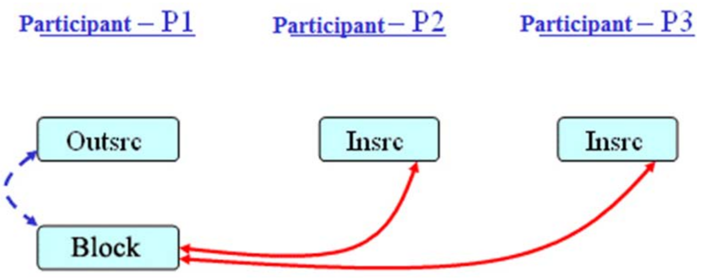

**Fig. 5**. The interactions of the Block service

It is considered a negotiation among an initiator carpentry workshop (participant P1) and other two guest carpentry workshops (participants P2 and P3). In this case, the negotiation begins by the initialisation of the *Outsrc* service of the participant P1, which invites the *Block* service to the negotiation. Subsequently, *Block* connects to the *Insrc* service of each participant (P2 and P3) and will coordinate simultaneously the bilateral negotiations with them.

As soon as all services are connected, the interaction process among the participants may begin. During this process, the Negotiation Agent of each carpentry workshop involved in the negotiation starts generating and exchanging proposals and counterproposals for the task at hand. The negotiation ends when the participant P1 reaches an agreement with one of the partners (e.g., participant P2) regarding the set of attributes that describe the task being negotiated. At the same time, participant P1 ends the negotiation with P3, this coordination being provided by the *Block* service. 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).

### 4.1.4. Split Service

The *Split* service is connected in negotiations in which the possibility of fragmentation of the task in two parts for two different contractors is provided. Its main function is to coordinate the connections between the complete descriptions of the task accomplished by an instance of an *Outsrc* service and the proposals concerning both sides of the task accomplished in two instances of the *Insrc* service.

As soon as the *Split* service is connected, it becomes an intermediary between the *Outsrc* service and the two *Insrc* services. The *Split* service raises some constraints. The constraints on the negotiated task are established by the *Outsrc* service. When one of the *Insrc* services makes a proposal, the *Split* service is able to build the proposal for the second *Insrc* service, so that the task proposed by the *Outsrc* service can be entirely subcontracted. The *Split* service is invited in the negotiation by the *Outsrc* service.

In Figure 6, the implementation of the negotiation for the initiator carpentry workshop (participant P1) is achieved through *Outsrc* and *Split* services. For each of the invited carpentry workshops, the implementation of the negotiation is realized through one of the *Insrc* services.

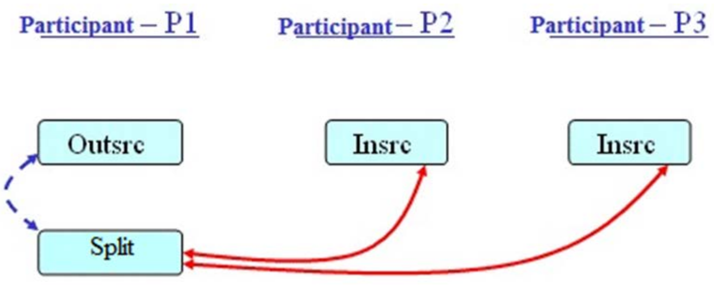

**Fig. 6.** The interactions of the Split service

Furthermore, as in the first scenario, the two *Insrc* services coordinate only the interactions with the *Outsrc* service of the P1 initiators. Thus, P2 and P3 workshops are not necessarily aware of their entry into the competition and the synchronization among their proposals is made by the *Split* service. The management of the constraints resolution is thus delegated to the *Split* service. This service implements generic mechanisms for constraints solution and ensures the propagation of these constraints.

The negotiation ends at the moment when the carpentry workshop P1 is satisfied with the proposals made by P2 and P3, the difference from the first negotiation residing in the two negotiations ending simultaneously by a single agreement, which will result in two contracts.

As a consequence, for a negotiation among these types of services in a closed environment, the coordination is achieved only through bilateral negotiations that compose the same negotiation. In the next section, we will model the Split service using the negotiation approach proposed in the previous section.

### 4.2. Example – Modeling the Split Service using Program Formula

It is assumed that for each partner involved in a negotiation there is a negotiation sequence. By using Program Formula, all methods that model the entire negotiation process that must be managed by a certain sequence are defined.

In the following, the behavior of the $s_{Split}$ negotiation sequence with $view(s_{Split}) = (p_1,N, R_{Split})$ will be modeled.

Assuming that the first atom of the negotiation sequence $s_{Split}$ contains initially the following particles: **name**(Id), **start**, **localr**(Rname1,initiated, ∅), the representation particle for *Split* sequence, **firstr**(Rname2,initiated, ∅), the representation particle for *Outsrc* sequence, **count**(X), the particle used to divide the task.

In this scheme we look for the split of the task into two parts, so X=2

For each **clone_create**(Id, New_Id, Msg) event a new atom is created (1.). The second method (2.) generates a new **extr**(Rname, initiated, ∅) representation particle, which represents the image that a sequence of a new participant has on the new negotiation phase. Considering that our aim is to divide the task in two parts, the atoms where we can start negotiation are those containing the **count**(X) particle. Thus, we can obtain two particles of external representation for two new partners.(3.).

(1.) **count**(X) @ **name**(Id) @ **start** @ **clone_create**(Id, New_Id, Msg) @ {X !=0} @{**string_create**(Msg, Rname, Type)} <>**-** (**start** @ **name**(Id) @ **count**(X)) & (**start** @ **freeze** @ **name**(New_Id) @ **create**(Rname, Type) @ **count**(X))
(2.) **count**(X) @ **freeze** @ **create**(Rname, type) <>**- extr**(Rname, initiated, ∅) @ **count**(--X)
(3.) **count**(0) @ start <>**- enable**

The methods from (4.) to (6.) deal with the exchange of proposals and guarantee dependencies among the proposals made on the execution of the two parts of the task. Method (4.) introduces a new proposal within the system through the **clone_propose** event. The negotiation will develop by taking into account this proposal only on the atoms containing this **enable** control particle.

The modeling of the **propose** particle is different depending on who made the proposals. If the initiator participant (5.) made the proposal, this will refer to the whole task and will be divided by the **construct_split** computational particle in order to build proposals for each invited participant. In case a participant invited in the negotiation makes the proposal, this will be handled through the **solve_split** computational particle in order to build a proposal for the other participant concerning the execution of the remaining part of the task (6).

(4.) **name**(Id)@**enable** @ **clone_propose**(Id, New_ Id, Msg) @{**string_propose**(Msg, Rname, Content)} <>- (**enable** @ **name**(Id)) & ( **freeze** @ **name**(New_ Id) @ **propose**(Rname, Content))
(5.) **freeze** @ **localr**(Rname1, St1, I1) @ **firstr**(Rname2, St2, I1) @ **extr**(Rname3, St3, I3)@ **extr**(Rname4, St4, I4)@ **propose**(Rname2, Content) @ {**construct_split**(I1,I3,I4Content,I5,I6,I7)}<>- **enable**@ **localr**(Rname1, undefined, I5) @ **firstr**(Rname2, undefined, I5)@**extr**(Rname3, undefined, I6)@ **extr**(Rname4, undefined, I7)
(6.) **freeze** @ **localr**(Rname1, undefined, I1) @ **firstr**(Rname2, undefined, I1) @ **extr**(Rname3, undefined, I3)@ **extr**(Rname4, undefined, I4)@ **propose**(Rname2, Content) @ {**solve_split**(I1,I3,I4Content,I5,I6,I7)}<>- **enable**@ **localr**(Rname1, undefined, I5) @ **firstr**(Rname2, undefined, I5)@**extr**(Rname3, undefined, I6)@ **extr**(Rname4, undefined, I7)

Methods (7)–(10) model the dependencies regarding the status. In all valid negotiation phases (i.e., those having the control particle **enable**) the negotiation partners may accept the current proposal – method (7). In the event that the two partners reach an agreement, in the newly created atom the representation particles for a single partner will have the status success (8 and 9). A new control particle – **waiting -** was introduced to preserve the negotiation atom only for events of type **clone_accept** (10).

Differing form *Block* services [15], in this example there are two ways of concluding an agreement. In the first case, all four representation particles must be in a *success* (11.) status. In the second case, one can conclude an agreement if one of the external representation particles finds itself in a *success* status, but the concluded agreement refers to the whole task (12.). In both cases, when an agreement is concluded, all other atoms are notified to stop negotiation (the **stop** particle is introduced by the mechanism of broadcasting through all atoms composing the $_{\text{SSplit}}$ negotiation sequence.)

(7.) **name**(Id) @ **enable** @ **clone_accept**(Id, New_ Id, Msg) @ { **string_accept**(Msg, Rname)}<>- (**enable** @ **name**(Id)) & ( **freeze** @ **name**(New_ Id) @ **accept**(Rname))
(8.) **freeze** @ **localr**(Rname1, St1, I) @ **firstr**(Rname2, St2, I) @ **accept**(Rname2)<>**- localr**(Rname1, success, I) @ **firstr**(Rname2, success, I) @ **waiting**
(9.) **freeze** @ **extr**(Rname3, St3, I) @ **accept**(Rname3)<>**- extr**(Rname3, success, I) @ **waiting**
(10.) **name**(Id) @ **waiting** @ **clone_accept**(Id, New_ Id, Msg) @ { **string_accept**(Msg, Rname)}<>**- name**(Id) @ **freeze** @ **accept**(Rname)
(11.) **localr**(Rname1, success, I) @ **firstr**(Rname2, success, I) @ **extr**(Rname3, success, I3) @ **extr**(Rname4, success, I4) @^**stop** <>**- ready**(I)
(12.) **localr**(Rname1, success, I) @ **firstr**(Rname2, success, I) @ **extr**(Rname3, success, I) @^**stop** <>**- ready**(I)

By (13.) and (14.) methods the **clone_reject** event is processed. Thus, two cases are modeled: firstly, an atom is created following the acceptance of a proposal (atom created by method 9) and, secondly, an atom is created following a new proposal (atom created by method 4).

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

The processing of the **reject** type message is done according to the partner who decides to decline a proposal. If one of the invited participants decides to stop the negotiation in the current atom, this will be visible in the atom managed by the $_{\text{SSplit}}$ sequence only at the level of the representation particle of the participant who sent the rejection.(15.). If the initiator participant sent the refusal, the negotiation atom is completely dissolved (16.). Similarly, the negotiation stops in the atoms where the two participants are invited in a *failure* (17.) status and in the atoms containing the **stop** (18.) particle.

(15.) **freeze** @ **extr**(Rname3, St3, I) @ **reject**(Rname3) <>**-** **extr**(Rname3, failure, I)
(16.) **freeze** @ **localr**(Rname1, St1, I) @ **firstr**(Rname2, St2, I) @ **reject**(Rname2) <>**-** #t
(17.) **extr**(Rname3, failure, I3) @ **extr**(Rname4, failure, I4) <>**-** #**t**

stop <>**-** #t

## 5. Final Considerations

This paper proposes a framework for modeling and managing parallel and concurrent negotiations. The business-to-business interaction context in which our negotiations take place forces us to model the unexpected and the dynamic aspects of this environment. An organization may participate in several parallel negotiations. Each negotiation may end with the acceptance of a contract that will automatically reduce the available resources and it will modify the context for the remaining negotiations. We have modeled this dynamic evolution of the context using IAMs metaphor that allows us to limit the acceptance of a negotiation to the available set of resources.

In the current work we have described in our negotiation framework only the interactions with the goal to outsource or insource a task. A negotiation process may end with a contract and in that case the supply schedule management and the well going of the contracted task are both parts of the outsourcing process. In the sequence of our research we will complete out framework with the contract management process and a possible renegotiation mechanism.

## References

[1] Duan L., Dogru M., Ozen U., Beck J., *"A negotiation framework for linked combinatorial optimization problems"*, Autonomous Agents and Multi-Agent Systems,Volume 25 (1), pp. 158-182, Springer Netherlands, 2012.

[2] Cretan, A., Coutinho, C., Bratu, B., and Jardim-Goncalves, R., *"A Framework for Sustainable Interoperability of Negotiation Processes"*, The 14th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'12), 14(1): 1258-1263, 2012, Elsevier, DOI: 10.3182/20120523-3-RO-2023.00240

[3] Emerson, E.A.,*"Time of time"*, Proceedings of the 17th intl. conf. on Static analysis (SAS'10), Springer-Verlag, 2010.

[4] 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.

[5] Cretan, A., Coutinho, C., Bratu, B., and Jardim-Goncalves, R., *"NEGOSEIO: A Framework for Negotiations toward Sustainable Enterprise Interoperability"*, IFAC Journal Annual Reviews in Control, Vol. 36, Issues 2, pp. 291-299, 2012.

[6] Hu, J., Deng, L., *"An Association Rule-Based Bilateral Multi-Issue Negotiation Model"*, in Computational Intelligence and Design (ISCID), Fourth International Symposium, Volume 2, pp. 234 – 237, 2011.

[7] Luo, X., Miao, C., Jennings, N., He, M., Shen, Z. and Zhang, M.,*"KEMNAD: A Knowledge Engineering Methodology for Negotiating Agent Development"*, Computational Intelligence ., 28 (1), 51-105, 2012 .

[8] Georgantas, N., Rahaman, M., Ameziani, H., Pathak, A., and Issarny, V.,*"A Coordination Middleware for Orchestrating Heterogeneous Distributed Systems"*, Grid and Pervasive Computing - GPC , pp. 221-232, 2011.

[9] Coutinho, C., Cretan, A., and Jardim-Goncalves, R.,*"Cloud-based negotiation for sustainable Enterprise Interoperability"*, The 18th International ICE Conference on Engineering, Technology and Innovation, Munich, Germany, 18-20 June, 2012 (ICE'12).

[10] Jardim-Goncalves, R., Sarraipa, J., Agostinho, C., and Panetto, H., *"Knowledge Framework for Intelligent Manufacturing Systems"*, Journal of Intelligent Manufacturing, Vol. 22, Issue 5, pp. 725–735, 2011.

[11] Faratin, P., *"Automated service negotiation between autonomous computational agent"*, Ph.D. Thesis, Department of Electronic Engineering Queen Mary & West-field College, 2000.

[12] Oliveira, A. I., and Camarinha-Matos, L. M., *"Electronic Negotiation Support Environment in Collaborative Networks"*, Advances in Information and Communication Technology 372/2012, 21–32, 2012.

[13] Sycara, K., Dai, T.,*"Agent Reasoning in Negotiation"*, Advances in Group Decision and Negotiation, Volume 4, Handbook of Group Decision and Negotiation, Part 4, pp. 437-451, 2010.

[14] Kadar, M., Cretan, A., Muntean, M., and Jardim-Goncalves, R., *"A Multi-Agent Based Negotiation System for Re-establishing Enterprise Interoperability in Collaborative Networked Environments"*, in the 15th International Conference on Computer Modelling and Simulation (UKSim 2013), Cambridge, United Kingdom, pp. 190-196, 10-12 April 2013.

[15] Coutinho, C., Cretan, A., and Jardim-Goncalves, R., *"Sustainable Interoperability on Space Mission Feasibility Studies"*, Computers in Industry, 2013 (In Press).