Coordination and Access Control in Open Distributed Agent Systems: The TuCSoN Approach

Marco Cremonini[1], Andrea Omicini[1], and Franco Zambonelli[2]

[1] LIA – DEIS – Università di Bologna Viale Risorgimento 2
40126 Bologna, Italy
{mcremonini, aomicini}@deis.unibo.it

[2] DSI – Università di Modena e Reggio Emilia Via Campi 213/b
41100 Modena, Italy
franco.zambonelli@unimo.it

Abstract. Coordination and access control are related issues in open distributed agent systems, being both concerned with governing interaction between agents and resources. In particular, while coordination deals with enabling interaction and making it fruitful, access control is meant to control interaction to make it harmless. We argue that this twofold facet has to be supported by a system in a uniform and decentralised manner. To this end, we describe how the application of the TuCSoN tuple-based coordination model over a hierarchical topology is well-suited in this context. On the one hand, policies can be enforced by means of a single mechanism based on tuples and can be scoped to manage access to groups of distributed resources. On the other hand, agents can interact along a hierarchical infrastructure by applying a standard tuple-based communication template. This makes TuCSoN a single coherent framework for the design and development of Internet-based multiagent systems, which takes coordination as the basis for dealing with network topology and access control in a uniform way.

1 Introduction

Open distributed multiagent systems (*MAS*s, henceforth) have gained sheer interest due to their suitability to the Internet scenario. Their best property is to cope well with the *unpredictability* and the *dynamics* of the environment. The lack of a global state of the Internet can be addressed by exploiting the agent autonomy and flexibility. In addition, the openness and the spatial distribution of systems make the ability to deal with *heterogeneous* environments and *decentralised* forms of control an issue. In particular, systems are likely to be composed of several heterogeneous subsystems managed by *independent authorities*, where heterogeneity refers both to their design and implementation [9, 16].

In such a context, the ability to coordinate the agents [1] coupled with the possibility to control the operations they perform is important, so we argue that *coordination* and *access control* should be regarded as tightly connected issues [3, 4, 14].

A. Porto and G.-C. Roman (Eds.): COORDINATION 2000, LNCS 1906, pp. 99-114, 2000.

Furthermore, the openness and the distribution of the environment make traditional centralised solutions no longer effective. Architectures with a central coordinator, which supervises to the whole activity of a system, are not applicable in many contexts. In open distributed systems is convenient to employ components in charge of coordinating tasks and active entities, but they should be decentralised and managed by local authorities. Consequently, the relationships among subsystems, local authorities and decentralised coordinators should be explicitly represented in the design of a system infrastructure. This leads to design system topologies for modelling the distribution of the components and their interconnection. Both the coordination and the access control issue should be addressed in such a decentralised, unpredictable and dynamic environment.

The aim of this paper is to present how the TuCSoN system faces to and manages such a complexity of open distributed agent systems. The essential components of TuCSoN are:

- a coordination model based on multiple programmable tuple spaces that mediates all communications among active entities;
- a distributed infrastructure, modelling a system, upon which tuple spaces are deployed;
- the integration of mechanisms for the access control within the tuple-based environment and their application to the hierarchical infrastructure.

In particular, the paper shows that implementing access control mechanisms in TuCSoN is worthwhile and may provide for a relevant benefit: agents could be supported along their interaction without a static knowledge of the environment.

In Section 2 we discuss the relationship between coordination and access control. Section 3 describes the TuCSoN coordination model and its application to a hierarchical infrastructure. Section 4 presents how access control techniques have been applied and fully integrated with the coordination model. In Section 5, the case study considered throughout the paper is presented in its entirety. Finally, in Section 6, some related systems are analysed and conclusions drawn.

2 Coordination and Access Control

In general, coordination technology for the Internet is typically concerned with enabling interaction and making it fruitful, while access control technology is typically meant to bound interaction to make it harmless. Then, access control could be regarded as the conceptual security counterpart of coordination. This makes coordination and access control two strictly related topics, and their combination allows to fully characterising the interaction between agents and resources in an Internet-based environment.

In addition, both coordination and access control policies (i.e., sets of rules) have to be defined and enforced. *Enforcement of policies*, that is agents being compelled to act according to preset policies, should occur in a decentralised manner, with a single and uniform mechanism, and be applied with different granularity, i.e., to individual agents or to entire multiagent systems.

Decentralisation is necessary because agent systems need to be scalable, possibly spread over a wide area network and have no single point of failure.

A *single and uniform mechanism* for the enforcement of policies is required for managing the heterogeneity of the overall system. In an organisation, many different MASs could be deployed, each one with a specific policy to be supported.

Finally, also the *granularity* of access control policies should be considered: whether rights are granted to single agents or to entire MASs. New MASs could be dynamically added in the organisation, and new agents could be dynamically created within each MAS. Thus, it is often convenient to be able to define policies first in term of MASs, so as to bound the interaction space of a system within the organisation, and then in term of agents belonging to MASs.

Before discussing in more detail the TuCSoN's coordination and access control models, which are the subjects of next two sections, we first introduce an underlying assumption that holds throughout this paper. The pre-condition for each system in order to establish what an entity is allowed to do is to recognise whom the interacting entity is. This leads to the *authentication* issue: each execution environment holding protected resources should determine which agent is requiring the execution of an action before allowing it to be done [11]. A coordination framework should then endorse an agent naming scheme and support an authentication protocol, univocally mapping agents onto names. Coherently, a coordination model should embody a suitable notion of *agent identity*, by allowing any communication operation to be associated to an agent identifier. This way, given that policies have to be also defined in terms of MASs and not only in terms of single agents, a broader notion of identity should be supported, which enables a MAS to be denoted as a whole. TuCSoN defines a global agent naming scheme by combining a *MAS identity* and an *agent individual identity*, where the latter is determined within the scope of the former. Hence, when we refer to agent identities, we intend composite objects having the form MAS_{ID}:$Agent_{ID}$, where the first is a global name referring a specific MAS and the second is the relative name for the single agent.

Throughout this paper, we always assume that whether an agent interacts with an execution environment and access a tuple space, its global name, composed by a MAS and an individual identity, has been already verified by the system.

3 TuCSoN Coordination Model

In TuCSoN [13], agents interact through a multiplicity of independent coordination media, called *tuple centres*, spread over Internet nodes. Agents exchange tuples by means of standard Linda primitives [2, 6, 7].

Each tuple centre is associated to a node and has a unique name: in particular, a tuple centre can be denoted either by its full Internet (*absolute*) name or by its local (*relative*) name. By means of the absolute name *tc*@*node*, tuple centre *tc* provided by the Internet node *node* is referred to from everywhere in the Internet, and by means of its relative name *tc* in the context of node *node*. The general form for any admissible

TuCSoN communication operation performed by an agent is *tc@node?op(tuple)* asking tuple centre *tc* of node *node* to perform operation *op* using *tuple*.

TuCSoN tuple centres are tuple spaces enhanced with the notion of *behaviour specification*: the behaviour in response to communication events of every tuple centre can be defined according to the system requirements as the observable state transition following a communication event. Correspondingly, the definition of a new behaviour for a tuple centre basically amounts to specifying a new state transition in response to a standard communication event. This is achieved by allowing any communication event to be associated to specific computations, called *reactions*. In particular, a *specification tuple*, stated by *reaction(Op,R)*, associates the event generated by an incoming communication operation `Op` to the reaction `R`.

A reaction is defined as a sequence of *reaction goals*, which may access the properties of the communication event triggering the reaction, perform simple term operations, and manipulate tuples in the tuple centre. Each reaction is executed with a transactional semantics: a successful reaction can atomically modify the tuple centre state, while a failed reaction yields no result at all. In particular, operations on the tuple space (*out_r, in_r, rd_r, no_r*) work similarly to communication operations, and can trigger further reactions in a chain. Reaction goals are executed sequentially and a chain of reactions is either a *successful* or a *failed* atomic operation depending on whether or not all its reactions succeed [5].

Then, from the agent's viewpoint, the result of the invocation of a communication primitive is the sum of the effects of the primitive itself and of all the reactions it has triggered, perceived altogether as a single-step transition of the tuple centre state. This makes it possible to uncouple the agent's view of the tuple centre, viewed as a standard tuple space, from the tuple centre actual state, and to connect them so as to embed the laws of coordination and access control.

3.1 TuCSoN Tree-Like Infrastructure

The interaction space provided by a TuCSoN system relies on a multiplicity of distributed tuple centres. This way, TuCSoN shares the advantages of models based on multiple tuple spaces and goes beyond, since different coordination media can encapsulate different coordination and access control laws. This leads to introduce the relationship between a coordination model and its distributed topology, and to discuss two main problems: *(i)* how the space where agents live is modelled (*network modelling*), and *(ii)* how the knowledge about the structure of that space is made available to the agents (*network knowledge*).

The problem of network modelling is particularly evident when dealing with intrinsically structured domains, as Internet-based organisations frequently are. In fact, Internet nodes are often grouped in clusters, subject to highly coordinated management policies and possibly protected by firewalls. Moreover, large clusters can be further characterised by the presence of enclosed sub-clusters, often arranged as a (logical or physical) hierarchical structure of protected organisational domains. Different enclosed clusters provide protected domains of shared resources and are largely independent from the rest of the organisation in the definition of policies concerning

their resources. For example, most academic environments or medium- and large-sized companies have infrastructures that suit this hierarchical model.

As far as network knowledge is concerned, it is, at least, unrealistic to assume that agents could have a complete knowledge of the whole network topology, as well as of resources availability. In fact, Internet-based domains are typically dynamic and unpredictable, due to their complex structure where many decentralised authorities are present and no central repository of information is available. Therefore, knowledge about the environment should be acquired dynamically and incrementally by agents through their interaction. This actually affects the coordination protocol, since part of the agent interaction concerns the acquisition of information about topology, and makes network knowledge a coordination-related issue.

In TuCSoN, the *gateway* locality abstraction has been specifically introduced to enable the modelling of an Internet-based agent system as a hierarchical infrastructure. A *gateway* is a node augmented with the ability of authenticating (although we do not address this issue in the present paper) and of authorising agents to access some other nodes and their tuple centres. By connecting gateways in a tree-like topology, the definition of nested protection domains is natural: for each possible sub-tree, the root gateway may hold the policy for managing the access to the corresponding children. Ordinary nodes are the leaves of the tree, while intermediate nodes could act also as gateways. Fig. 1 provides an example for a case study. This way, the collection of the Internet nodes hosting a MAS can be conveniently represented. A hierarchical structure is well-suited in many real-world cases and the root of the hierarchy, that is the most external gateway, may work as a bridge with the Internet so as to permit the connection among different systems and infrastructures.

Considering the coordination model, each node and each gateway of a TuCSoN infrastructure implements its own set of tuple centres. In particular, one or more *application* tuple centres can be defined to coordinate and authorise the agent access to the local resources of a node. Each *gateway*, in addition, is supposed to provide for a unique *default* tuple centre, which represents the standard communication media that agents access to get information about other available tuple centres of nodes and gateways.

3.2 The Journal Review Case Study: A First Glance

As our case study, we consider a reviewing process of a scientific journal, as depicted in Fig. 1: the hierarchical topology represents the relationships among the *Publisher*, the *Editors*, the *Area Editors* and the *Reviewers* agents, showed as ovals. The system is configured with several distributed tuple centres, drawn as boxes: each member of the system exchanges papers and reviews through tuple centres called *papers*. Differently from Reviewers, the Publisher, Editors and Area Editors also act as gateways, thus exploiting a *default* tuple centre to exchange information with agents. Relationships among the components of the reviewing process are represented with arrows. The Authors are the external entities interacting with the Publisher gateway to submit papers and receive reviews. The Publisher receives submitted papers from Authors, allocates them to the Editors and should be enabled to manage both papers and re-

views (i.e., check the status of a paper under review, collect reviews, re-allocate papers, etc.). Editors act as the Publisher with respect to their corresponding Area Editors, although they can not interfere with each other or with Publisher's actions. Area Editors may work for different Editors (see Area Editor2 in Fig. 1), allocate papers to Reviewers and collect reviews. Reviewers may receive papers from different Area Editors. Finally, consider agent identities, which have the form MAS_{ID}:$Agent_{ID}$. We assume to distinguish between two systems (MAS_{ID}), one managing the first issue of the journal (*issue1*) and another managing the second issue (*issue2*). The Publisher deals with both issues, as Area Editor2 that can receive papers from both editors. Editor1 and Area Editor1 deals with the first issue only, while Editor2 and Area Editor3 deals only with the second. Reviewers can deal with both issues.

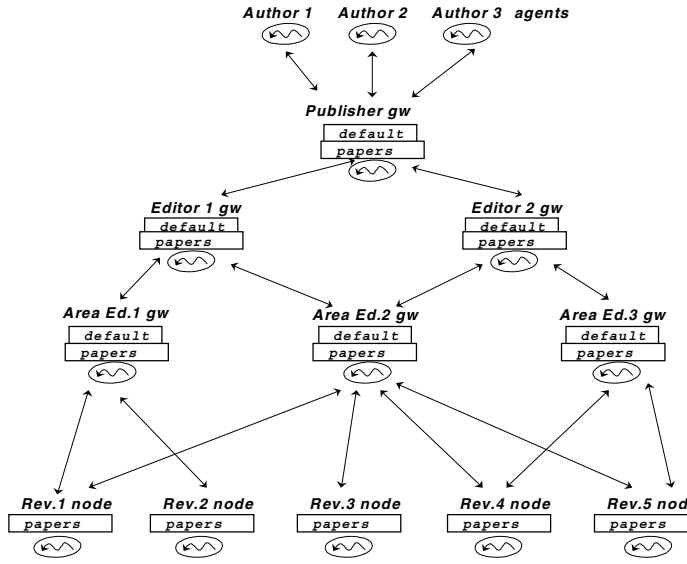

Fig. 1. Topology for a scientific journal reviewing process

4 Protection in TuCSoN

As far as agent interaction through tuple centres is concerned, the control of agent access and authorisation can be achieved only by handling *all* the communication events performed in tuple centres. Each communication event should be carried out according to *access control policies*. In TuCSoN, tuple centres could be made visible or invisible to agents under the control of a gateway and the access to the tuples can be controlled for each agent by tuple centres reactions. The programmability of reactions in response to each communication event allows serving differently any operation performed in a tuple centre, according to the rights granted to agents.

4.1 Access Control Matrix

In TuCSoN, the different behaviours of gateways and nodes are exploited in order to control agent access to tuple centres.

Firstly, nodes are the natural location for the definition and the enforcement of authorisation policies, because resources are located in nodes, tuple centres mediate all interactions, and each node might be independently managed. Considering that a node may implement several independent tuple centres, the global security policy supported by a node can be formalised by an *access matrix* in terms of both agent identities, tuple centres and permissions [10].

$$\mathsf{ACM}[i,j] ::= \mathsf{P}_{i,j}\ , \forall\ i \in \mathsf{ID},\ \forall\ j \in \mathsf{TC} \qquad (\mathbf{1})$$

where

- ID is a set of different *agent identifiers*.
- TC is the set of different *tuple centres* implemented by that node.
- $\mathsf{P}_{i,j}$ is the access permission granted to the *i*-th identity by the *j*-th tuple centre of the given node.

Since ACM is sparse, in practice, it is never stored in its tabular form. Instead, two implementations have been traditionally derived [10].
From the *rows* of ACM, a policy could be represented as a collection of *capabilities*:

$$\mathsf{CAP}[i] ::= \langle j, \mathsf{P}_{i,j} \rangle\ , \forall\ j \in \mathsf{TC} \mid i \in \mathsf{ID} \qquad (\mathbf{2})$$

where ID , P and TC are defined as in (1).

From the *columns* of ACM, instead, *access control lists* have been defined as list of pairs:

$$\mathsf{ACL}[j] ::= \langle i, \mathsf{P}_{i,j} \rangle, \forall\ i \in \mathsf{ID}' \mid j \in \mathsf{TC} \qquad (\mathbf{3})$$

where ID' is the set of agent identifiers defined in the policy of a given tuple centre. P and TC are defined as in (1).

Let us show an example of these access control mechanisms applied to our case study. Consider, for instance, the Area Editor1. He should permit Editor1 to insert papers and withdraw reviews concerning the first issue of the journal, and the Publisher to do every desired action. In terms of capabilities and access control lists, the following could be defined:

CAP[issue1:editor1] ::= <{default@areaed1, *insert papers*},
{default@areaed1, *withdraw reviews*}>
CAP[publisher] ::= <default@areaed1, *everything* >

ACL[default@areaed1] ::= <{issue1:editor1, *insert papers*},
{issue1:editor1, *withdraw reviews*},
{publisher, *everything*}>

The two mechanisms are managed differently: capabilities are meant to be physically held by interacting entities, i.e., Editor1 and the Publisher, while the ACL is meant to be stored by the resource holder, i.e., Area Editor1.

For sake of completeness, however, if the administrator of a tuple centre needs to allow *anonymous accesses* (i.e. agents having unknown identities), the *anonymous* pseudo-identity must be defined in the access control policy. In TuCSoN tuple centres, agents with an identifier not listed in the policy succeed in matching against the *anonymous* one, if specified, thus gaining the permission assigned to it.

In the following, we focus on the case of ACLs, although the analysis could be easily generalised to the case of capabilities as well.

4.2 Enhanced Access Control Matrix

In the previous subsection, we have described the most traditional mechanisms for protecting shared resources, and their integration with the TuCSoN coordination model. Conversely, when the distribution of the system is considered, some particularities arise and traditional protection models need to be enhanced. This, in TuCSoN, impacts on the hierarchy of gateways.

On the one hand, a gateway is queried by all agents willing to access the tuple centres implemented by the nodes of its associated domain, thus it is well-suited to enforce access control policies on the behalf of those nodes. Furthermore, the hierarchical topology of the infrastructure lets gateways control both the accesses to nodes' tuple centres and the ability of agents to further interact with sub-gateways. This permits gateways to decide which agents could be allowed to explore different subtrees of an infrastructure.

On the other hand, for the characteristics of open distributed systems, local administrators have the authority over their resources, thus policies still have to be locally defined in nodes, not in gateways.

Then, to make gateways able to enforce access control policies, the *default* tuple centre implemented by each gateway must hold a combination of the access control policies defined by tuple centres located in domain's nodes and sub-gateways. More precisely, nodes and sub-gateways must partially *delegate* their authority to an upper gateway. *Delegation*, in this context, is an action performed by nodes and gateways, whose effect is to invest another gateway with the onus of matching agents identities against their access control policies and thus granting some permissions to agents. The authority is only partially delegated since gateways can not modify the delegated policies on their own.

In practice, the result of delegation actions is that the delegated gateway receives from some nodes and/or sub-gateways, a number of tuples representing entries of their ACM[i,j] (1). The combination of those policies in the default tuple centre can be formalised with the *3-dimensions Access Control Matrix.*

$$\mathrm{ACM}[i,j,k] ::= P_{i,j,k}, \forall\, i \in \mathrm{ID}, \forall\, j \in \mathrm{TC}, \forall\, k \in \mathrm{N} \qquad (4)$$

where

- ID is the set of different *identifiers* stated by all the access control policies delegated to the gateway.

- TC is the set of different *tuple centres* from which the default tuple centre has been delegated.
- N is the set of *nodes* (nodes and sub-gateways) hosting the TC tuple centres.
- $P_{i,j,k}$ is the access permission granted to the *i*-th identifier by the *j*-th tuple centre of the *k*-th node of the matrix.

Again, in real contexts the access matrix tends to be highly sparse, being many $P_{i,j,k}$ possibly left unspecified. The reason is twofold: *(i)* agent identifiers relevant for a MAS occur only in policies defined by nodes interacting with that MAS; *(ii)* not all tuple centres are implemented by every node. Hence, in TuCSoN, an ACM is represented by a table listing only the elements that exhibit legal values of $P_{i,j,k}$. The semantic of an unspecified element of ACM - and then of an empty value of $P_{i,j,k}$ - is that no right should be granted.

Differently from the traditional case discussed in the previous subsection, in this one it is useful to maintain the access matrix as a global table held by gateways. This permits to inspect the matrix along all the three dimensions, making possible both the enforcement of access policies and the control of agent interaction along the hierarchy.

Let us show what discussed above with an example. Consider Area Editor2 gateway in Fig.1. Agents that interact with it want information and permissions to further interact with tuple centres of lower nodes, not with its local resources. Hence, it is in charge of enforcing access policies for its protection domain. In this case, its domain is composed by Reviewers' nodes Rev1, Rev3, Rev4 and Rev5. Agents interacting with him could be the ones of the Publisher and those of the two Editors. In the case of Publisher's agents, they must be allowed to further access whatever Reviewer's *papers* tuple centre they want among the ones of the domain. In the case of an Editor's agent, it must be allowed to access only reviews concerning its corresponding journal issue, not the ones of the other. An example of access control matrix of Area Editor2 might be:

ACM[issue1:editor1, papers, {rev1, rev4}] ::= <*withdraw reviews*>,
ACM[issue2:editor2, papers, {rev3, rev5}] ::= <*withdraw reviews*>,
ACM[publisher, papers, {rev1, rev3, rev4, rev5}] ::= <*everything*>

where Editor1, for example, receives, from the *default* tuple centre of Area Editor2, permissions to withdraw tuples from *papers* tuple centres of Reviewers Rev1 and Rev4. This way, Editor1's agents are unaware of — and have no rights on — both other Reviewers and other tuple centres apart the *papers* one.

4.3 Standard Tuple-Based Communication Template

As we have seen so far, in a TuCSoN infrastructure policies could be uniformly defined and enforced in a decentralised manner. We move now to consider how agents may interact with such an infrastructure. We first recall the two basic features: *(i)* TuCSoN tuple-based interaction has the standard form *tc@node?op(tuple)*, and *(ii)* all the default tuple centres of an infrastructure hold a standard name, e.g., *default*.

From this, a *standard tuple-based communication template* can be applied to the interaction of agents with *default* tuple centres:

```
default@gateway?in(standard domain tuple template)
```

where the standard domain tuple template is:

```
domain(Id, info(Tc, Node, Permission), InfoList))
```

and

- the `Id` parameter holds the actual value of the identifier of the interacting agent, which has, in general, a form like MAS_{ID}:$Agent_{ID}$;
- `info(Tc, Node, Permission)` is the template for the expected tuples carrying the information about the domain policy. Each `info` tuple carries an access permission (`Permission`) that is granted to the agent identifier `Id` by the tuple centre `Tc` of the node `Node`;
- `InfoList` is the variable that will hold the resulting list of `info` tuples by unifying with the `domain`/3 answer tuple produced by the reaction(s).

This way, all agents accessing a TuCSoN environment are able to interact with the hierarchy of gateways without any previous knowledge of its structure – a part the firstly accessed gateway, which is supposed to be statically known. Then they could explore the infrastructure in a controlled fashion, because they only rely on the information acquired dynamically and incrementally from gateways.

Therefore, a querying agent is enabled to specify which information and permissions it needs by setting in different manners the attributes of the standard tuple template. Let us show an example, in our case study, by supposing that Editor1 and Editor2 agents want to check the status of two papers that they have allocated to Area Editor2. The partial knowledge that agents own can be represented as: *(i)* both editors do not know who are the actual reviewers of their papers (the Area Editor2 itself might be as well) and *(ii)* Editor2 knows that papers, reviews and status information are stored in tuple centres called *papers*. Consequently, their queries for the default tuple centre of Area Editor2 might be:

From editor1

```
default@areaed2?in(domain(editor1, info(_, _, _),
InfoList))
```

From editor2

```
default@areaed2?in(domain(editor2, info(papers, _, _),
InfoList))

default@areaed2?in(domain(editor2, info(default, _, _),
InfoList))
```

In the former, Editor1 agent generically queries for all tuple centres, local or remote, application-based or *default*, that it is allowed to access, having no additional

information. Differently, in the latter, Editor2 agent can make a more specific search by querying only for *papers* tuple centres, but must also specify a second tuple for retrieving accessible sub-gateways. Tuples *out*'ed in response to the previous templates may be:

```
domain(editor1, info(_, _, _), [info(papers, rev1,
withdraw reviews), info(papers, rev3, withdraw re-
views)])

domain(editor2, info(papers, _, _), [info(papers, rev2,
withdraw reviews), info(papers, rev5, withdraw re-
views)])

domain(editor2, info(default, _, _), [])
```

the first matched by Editor1, while the others matched by Editor2.

In Table 1 the possible configuration of `info(Tc, Node, Permission)` are shown and explained.

Table 1. Main instances of the `info` tuple parameter

`info(_,_,_)`	*All possible* `info` *tuples are out'ed. This way, the agent has a complete knowledge about tuple centres, nodes and permission in that domain, according with the rights granted to its identity.*
`info(tc,_,_)`	*This way, the agent requests only those nodes that hold a certain* `tc` *tuple centre, if accessible.*
`info(_,node,_)`	*The agent requests only tuple centres hold by the node* `node`*, if accessible*
`info(_,_,perm)`	*The agent requests all the tuple centres of the domain where it could act with the access right* `perm`*.*

4.4 Delegated Access Control Policies

The application of our security model needs to be further elaborated when policies are propagated along a hierarchy of gateways, i.e., some nodes delegate policies to a gateway, this one, in turn, delegates another upper gateway and so on. This because, whether the process of delegating tuples that compose a policy is applied *as-is* also to access matrixes delegated from gateway to gateway, the system consistency would be hard to be kept. All distributed access matrixes should be *synchronised* when application tuple centres modify their local policies. Local modifications would be propagated to every gateway's *default* tuple centre that was previously delegated with the changed policy.

The time frame from one consistent state of the configuration of delegated access control policies to another one could vary according to the depth of the tree, the network latency, and the size of the access matrixes. We have focused in particular the latter aspect for improving the system scalability. The idea is that recording in a gateway the combination of the access matrixes held by all its sub-gateways is not

strictly necessary. Instead, a gateway should simply establish whether a certain agent could be permitted to interact with nodes and sub-gateways directly connected with it in the hierarchical model, and not be concerned with nodes and sub-gateways of lower levels. To this end, the knowledge of the complete sub-gateway's access matrix is not required. Hence, the complexity of policies delegated from gateway to gateway could be reduced by establishing that delegation along the gateway tree should proceed in the following way:

- when delegation is performed *from a node to a gateway*, the delegated policies are the ones effectively defined by application tuple centres as ACLs;
- when delegation is performed along the gateway tree, *from gateway to gateway*, the delegated policies should depend only from the multiagent system to which an agent belongs (MAS_{ID}) and not from its individual identifier ($Agent_{ID}$).
  Hence, delegation between gateways can be realised by delegating tuples as:

$$\langle \mathsf{ID}^*, \textit{default}, \mathsf{N}^*, \textit{explore} \rangle \qquad (\mathbf{5})$$

where

- ID* is the set of different *MAS identifiers* exhibited by the access matrixes of the N* sub-gateways. This parameter has the form MAS_{ID}:_.
- *default* is the standard name of all the tuple centres providing for the gateway facility.
- N* is the set of sub-gateways that delegate a policy.
- *explore* is a new access mode that we have introduced, which captures the fact that a gateway may allow agents simply to interact with a sub-gateways, being unaware of the exact sub-gateways' access matrixes.

As an example, consider the delegation from Editor1 and Editor2 to the Publisher. Its resulting access control matrix is:

ACM[issue1:_, default, editor1] ::= <*explore*>
ACM[issue2:_, default, editor2] ::= <*explore*>

which is meant to provide each agent interacting with the Publisher with the minimum set of information and rights needed to deal with the review process. In this case, for instance, agents related to the first issue of the journal are authorised to interact with Editor1, which holds its own ACM, built up by entries from Area Editors.

From the scalability viewpoint, this solution has some benefits: when some modifications occur in local policies of a node but no MASs is removed or created, the access matrix to be synchronised is only the one of the gateway delegated by that node. A distributed synchronisation along the hierarchy is required only when MASs are created or deleted from the system, because only in this case the ID* parameter of (5) should be changed.

5 The Journal Review Process Case Study

In this section, the Journal Review Process case study is drawn out for showing how it works in the TuCSoN framework, including the implications of the hierarchy and

the tuple-based interactions. To this end, first consider an outlook of the system's local and delegated policies, as depicted in Fig.2. Access matrixes have been determined following the rules defined in the previous section. For sake of simplicity, we have kept at the minimum the information presented, thus the example is not completely specified, even though its generalisation has no difficulties. The elements for which details are not provided (AreaEd3, Rev2, Rev4 and Rev5) are left unspecified.

Firstly, consider identities: we have supposed that two systems exist for managing different issues of the journal, marked with *issue1* and *issue2* identifiers. This would make clear how different systems could co-exist and be separately managed in TuCSoN. For example, in Rev3 the access control policy states that: *(i)* agents belonging to the *issue2* system and holding the *areaed2* identifier could access the local *papers* tuple centre *out'*ing papers and *in*'ing reviews; *(ii)* the belonging to the system that manage the *issue2* permits agents to read the status of a paper to be reviewed; and *(iii)* any other agent is rejected. Policies defined by other nodes follows the same schema.

Secondly, consider the delegation performed by the Rev1's and the Rev3's *papers* tuple centres towards Area Editor2's *default* tuple centre. The access matrix is the combination of all Rev1's and the Rev3's access control policies.

Finally, consider what happens when delegation proceeds through the hierarchy. The access matrix of the Publisher gateway is build up only by considering journal issues' identifiers and its policy depends only from the number of different MASs.

Given the above configuration of policies, agents can interact with the hierarchy by using the communication template described in Subsection 4.3 to explore the whole infrastructure. In this way, they are unaware of the physical structure of the environment and are granted with the permissions corresponding to their behaviour. For instance, assume to let Author's agents check their paper's status. To this extent, Author's agents can interact with the Publisher's gateway, they can be assigned with a journal issue identifiers and can be informed about the corresponding editor. In its turn, the editor may advise them who is the area editor to contact, which, finally, can provide author's agents with the required information.

6. Conclusions and Related Work

In this paper, we have discussed the relation between coordination and access control in open, distributed agent systems and argued that the two issues are tightly connected. We have then presented how these issues have been integrated in the design of the TuCSoN framework and how the resulting features have been exploited for supporting a well-known case study in the area of tuple-based coordination. As far as the relationship between coordination and access control is concerned, we highlight how this aspect still lacks an extensive analysis, and this underestimation might slow-down the adoption of coordination models in open environments. The aim of this work, hence, has been to approach this important facet, presenting a solution that is applicable to a variety of application contexts. Further works will be devoted to improve and extend the integration of access control and coordination. For instance,

other models for the access control seems well-suited for TuCSoN, the role-based one in particular [14].

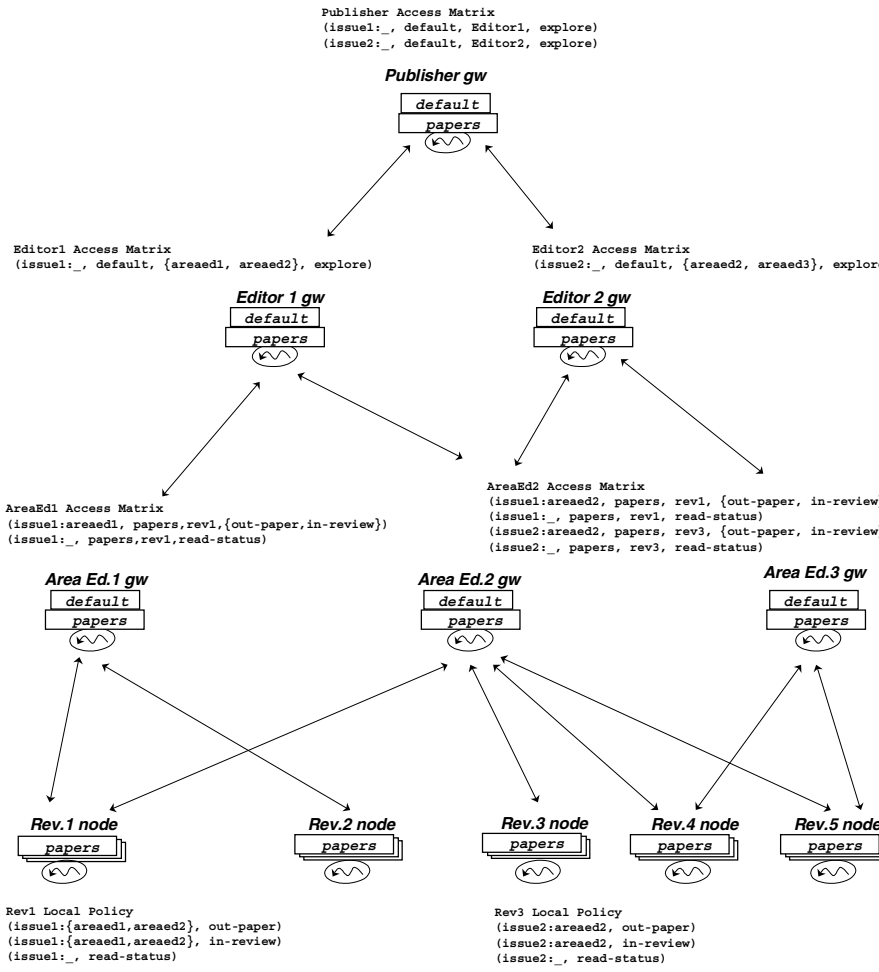

Fig. 2. Policies in the Journal Review Process case study

However, other works have addressed issues similar to the ones of this paper, Law-Governed Interaction, SecOS and ActorSpace in particular.

Law-Governed Interactions (LGI) [12] is a coordination mechanism that allows an open group of distributed active entities (agents, processes) to interact with each other, under an explicitly specified policy – called the *law* of the group. A peculiar feature of LGI is that for each member of a group, a *controller* is defined. A controller is a specific system component that holds the law governing an agent (i.e., the security and coordination policy) and its control state. In this way a controller totally mediates each event involving the associated entity and has complete knowledge and

authority over it. This is a different design choice with respect to TuCSoN, which, in turn, embeds laws into the behaviour specification of tuple centres and does not need to associate external components to agents or spaces. In the context of LGI, various access control policies have been modelled, but differently from TuCSoN, issues concerning network modelling, partial knowledge and delegation of policies are not considered.

The *SecOS* model [15] is a different example of integration between a Linda-like coordination model and security-related issues. In this case, the problem been tackled is to protect the access to tuples and tuple's attributes by means of cryptographic mechanisms. Its basic idea is to associate cryptographic keys with attributes in tuples. An agent needs to furnish a matching key before it can access the corresponding tuple's object. What is more interesting to highlight about SecOS is that the use of cryptography in the access to tuple's data is in some way complementary with respect to the subject of this paper. It represents in fact another possible direction for future developments of TuCSoN.

ActorSpace [8] provides an underlying platform for agent systems that enables to control the access to, and the management of resources, but does not adopt coordination models based on tuples. ActorSpace explicitly models the location of agents on particular hosts and the amount of computational resources that an agent is allowed to consume. Resource interaction is mediated by means of proxies, which are components of the agent's behaviour specifically customised to interact with a certain type of resource. Differently from ActorSpace, in TuCSoN agents interact in a standard way with resources and have no need to embed components (i.e. proxies) specifically tailored for the different resource types. This better management of the heterogeneity of the interaction space is one of the main benefits derived from the adoption of a tuple-based coordination model. Moreover, also issues related to the structure of the environment are not explicitly addressed in ActorSpace.

References

[1] Cabri, G., Leonardi, L., Zambonelli, F., Mobile-Agent Coordination Models for Internet Applications. IEEE Computer, 33(2): 52-58, Feb.2000.

[2] Carriero, N. and Gelernter, D., Linda in context. Communications of the ACM, 32(4): 444-458, 1989.

[3] Ciancarini, P., Tolksdorf, R., Vitali, F., Rossi, D., Knoche, A., Coordinating Multiagent Applications on the WWW: A Reference Architecture. IEEE Transactions on Software Engineering, 24(5): 362-375, May 1998.

[4] Cremonini, M., Omicini, A., Zambonelli, F., Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In Proc. of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAMAAW'99), LNAI no. 1647, pages 77-88, Springer-Verlag, 1999.

[5] Denti, E., Natali, A., Omicini, A., On the expressive power of a language for programming coordination media. In Proceedings of the 1998 ACM Symposium on Applied Computing (SAC'98), pages 169-177, Atlanta (GA), 1998.

[6] Gelernter, D., Generative Communications in Linda. ACM Transactions on Programming Languages and Systems, 7(1), January 1985.
[7] Gelernter, D., Carriero, N., Coordination languages and their significance, Communications of the ACM, 35(2): 97-107, Feb. 1992.
[8] Jamali, N., Thati, P., Agha, G. A., An Actor-based Architecture for Customising and Controlling Agent Ensembles. IEEE Intelligent Systems, Special Issue on Intelligent Agents, 38-44, March-April 1999.
[9] Jennings, N. R., Sycara, K., and Wooldridge, M., A Roadmap of Agent Research and Development. International Journal of Autonomous Agents and Multi-Agent Systems 1(1): 7-38, 1998.
[10] Lampson, B., Protection. ACM Operating Systems Review, 8(1), 1974.
[11] Lampson, B., Abadi, M., Burrows M., Wobber, E. P., Authentication in distributed systems: Theory and Practice. ACM Transactions on Computer Systems, 10(4): 265-310, November 1992.
[12] Minsky, N., Ungureanu, V. Unified support for heterogeneous security policies in distributed systems. In Proc. of the 7th USENIX Security Symposium, San Antonio, Tx, USA, 1998.
[13] Omicini, A. and Zambonelli, F., Coordination for Internet Application Development. Journal of Autonomous Agents and Multi-Agent Systems, Special Issue: Coordination Mechanisms for Web Agents, Kluwer Academic Press, 2(3), September 1999.
[14] Sandhu, R. (editor), ACM Transactions on Information and System Security, 2(1):1-135, 1999.
[15] Vitek, J., Bryce, C., Oriol, M., Coordinating Agents with Secure Spaces, In Proceedings of Coordination '99, Amsterdam, The Nederlands, May 1999.
[16] Weiss, G., Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press, June 1999.