Agent Coordination Contexts: Experiments in TuCSoN

Alessandro Ricci Andrea Omicini

***Abstract*— The notion of agent coordination context has been introduced as a means to model and shape the space of agent interaction and communication, and the agent presence in a situated environment. This paper describes experiments in modelling and developing the notion of agent coordination context within the TuCSoN model for agent coordination, and discusses the related benefits in terms of the engineering of multiagent system organisation and integration with service-oriented infrastructures.**

I. Introduction

The notion of *agent coordination context* is introduced in [27] as a means to model and shape the space of agent interaction and communication, generalising upon the concept of *context-depended coordination* [5]. From the theoretical perspective, agent coordination contexts both enable agents to model the environment where they interact and communicate, and provide a framework to express how the environment affects the interpretation of agent communication acts. Dually, from an engineering perspective, agent coordination contexts enable agents to perceive the space where they act and interact, reason about the effect of their actions and communications, and possibly affect their environment to achieve their goals; also, agent coordination contexts allow engineers to encapsulate rules for governing applications built as agent systems, mediate the interactions amongst agents and environment, and possibly affect them so as to change global application behaviour incrementally and dynamically.

In this work we describe how the notion of agent coordination context is modelled in the TuCSoN coordination model, discussing the benefits in terms of organisation and coordination of multiagent system (MAS). In particular, we consider the benefits in terms of both modelling aspects of MAS organisation, and modelling issues related to *coordination as service* concept [35] – in the context of service-oriented environments [22]. For this exploration, we considered requirements coming from state-of-the-art research work about role models, in particular supporting dynamic issues, in the context of AOSE [4], [20], [21], [17], in the context of Computer Supported Cooperative Work (CSCW) [33], [14], [7], [23], and access control models developed to manage security in distributed systems [15], [1], [19], with particular attention for Role Base Access Control (RBAC) models [30], [24] and systems providing policies to rules and promote organisation dynamics [34], [10]. These references have been exploited to understand the basic requirements about dynamism and security model that must be considered by models supporting organisation, integrated with coordination; moreover, they have been exploited to define the first basic ontology, concerning the coordination context description, in particular the role management part. From this perspective, this work can be considered a natural following of a previous work where the relationships between coordination and the issue of security and topology have been explored, and the TuCSoN model extended accordingly [9].

A. Omicini and A. Ricci are with DEIS, Università degli Studi di Bologna, via Rasi e Spinelli 176, 47023 Cesena (FC), Italy – email: {aomicini,aricci}@deis.unibo.it

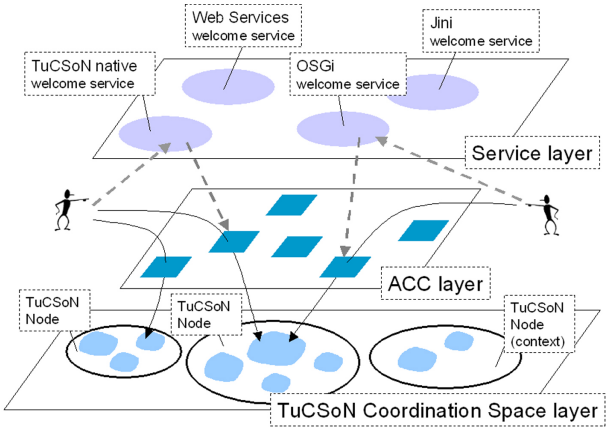

Fig. 1. TuCSoN *Overview* – Agents access the TuCSoN coordination contexts (services) – located at the coordination space layer – through TuCSoN Agent Coordination Contexts – located at the ACC layer. The ACCs are negotiated (gray dotted thick line) with the Welcome Services, located at the service management layer.

II. Agent Coordination Context in TuCSoN

In order to introduce the concept of agent coordination context, let's recall here some basic elements of the TuCSoN model for agent coordination. Coordination in TuCSoN is provided as a *service* [35] to agents by TuCSoN nodes through *tuple centres* [28] hosted by the nodes. Each TuCSoN node defines a *coordination context* where the coordination services are situated; a coordination context can be conceived as an open set of services enacted through tuple centres, organisational rules (roles, policies, ...) and dynamics that concern a TuCSoN node, its static and dynamic context. The collection of all the TuCSoN coordination contexts provided by the TuCSoN nodes constitute the TuCSoN *coordination space*. So, in the following when we speak about a coordination context, the scope is a node. In order to access the services, an agent must *enter* the coordination context provided by the node, and this is where the agent coordination context notion comes into play. An overall conceptual picture of TuCSoN space and

layers is provided in Fig. 1, described in more details in Section III, where the TuCSoN service architecture is considered.

Agents can enter the node coordination context by *negotiating* a TuCSoN *Agent Coordination Context* (ACC) with the node. The ACC is the means that allows the agent to exploit the coordination services provided by a node; in other word, the agent can act upon and perceive the coordination context through the actions provided by ACC, it can participate to multiple coordination contexts simultaneously, by means of different agent coordination contexts. On the one side, the coordination context concept does not concern a specific agent; on the other side, the agent coordination context concept is specifically tailored to individual agents, and their relationship with the coordination context which they entered. The ACC provide the set of the possible actions and observations that the agent can perform in the context: as for the *control room* metaphor described in [27], it establishes the admissible agent inputs and the admissible agent outputs. The idea (and related modelling and engineering discipline) is to use the ACC to embed and enforce *subject-centered* rules, i.e. rules constraining agent actions by virtue of its own position (role) inside the organisation, its own actions in the specific coordination context related to that role, but also as the means to model and provide the quality it negotiated for the coordination. Accordingly, the ACC concept is strictly related to the notion of *Controller* that is provided the Law Governed Interaction model [25]. Instead, as already described extensively in other papers [28], [12], the tuple centres are used to embed and enforce coordination laws toward the fulfilment of social tasks, the prescription of social norms, so rules and constraints that are more *society/organisation* centered. Accordingly, in this context, they will be used to model rules concerning the entrance and exit of an agent from a coordination context, and organisational/security policies that can be found in policy-driven middleware [33], [10] So, ACC and tuple centres together provide different and complementary means for modelling and shaping agent interaction space within the context of an agent society/organisation: ACCs focus on the individuals, tuple centres focus on societies. In this way, we recall the dualism between security and coordination issues described in [9], and we generalise it toward the organisation and coordination aspects.

The ACC enables agent participation to the context in specific *role*: each coordination context is characterised by a dynamic set of roles, as *context policy constructs* – extending the notion of role as a security construct [31], [30] toward the organisation / coordination extent [38], [26], [21], [33]. The same role can be found in different coordination contexts (so different nodes), with different properties. An agent can assume different roles also in the same coordination context, by participating with different ACCs. Following [31], a role is meant to:

- group subjects who may act in a role;
- group the operations/actions comprising what may be done in the role, and what obligations are charged upon the role; in our case the actions are based on the basic coordination primitives `in,out,rd,inp,rdp`, the extended ones `set_spec, get_spec`;
- group the objects/targets that may be acted upon, that in our case are tuples and tuple centres;

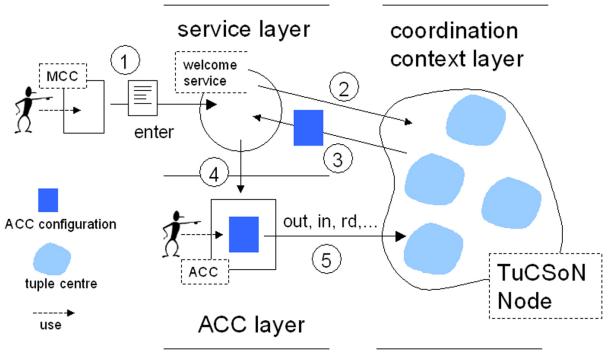

Fig. 2. *ACC Negotiation and Usage* – The negotiation of an agent coordination context involves (1) the agent issuing to a node Welcome Service the request to enter, providing the description of the ACC in terms of role and qualities of the coordination service; (2,3) the evaluation of the request with the creation of the specific ACC configuration, and (4) the setup of the ACC with such a configuration in the ACC layer, exploited then by the agent to access coordination context (5).

A role is characterised by a logic name, which must be unique inside the coordination context; the role name is specified in the agent coordination context description provided by the agent during the negotiation stage. Also *environmental/implicit role* are supported, i.e. role not explicitly identified by a specific name but from dynamic/organisation properties; In this case, in the negotiation stage the agent does not specify directly the a role name in the agent context description, and identifies it by providing some logic facts describing it.

A. Agent Coordination Context Negotiation

As already pointed out, the negotiation stage is engaged by an agent with a specific TuCSoN node, providing the context in which the agent wants to participate. More specifically, the agent issues the enter request to some TuCSoN *welcome service*, representing the TuCSoN coordination service in the context of a *service-oriented infrastructure* (see Fig. 1), as will be explained better in Section III. The agent issues the enter request through a TuCSoN Meta Agent Coordination Context (MCC), whose interface is described in details later on; the enter request must include the description of the agent coordination context characterising its participation inside the node context (point 1, Fig. 2). Such a description is provided as a set of first logic facts. The ontology concerning ACC description can include information about agent role inside the context (such as `role(RoleName)`), and, more generally, information that define agent presence inside the coordination context and the qualities of the coordination service. The agent request is evaluated (point 2) according the current (dynamic) context characterising the TuCSoN node, its security and organisational rules. If the requested is accepted, the specific ACC configuration is built (point 3), and the ACC for the agent is setup in the ACC layer (point 4). After that, the agent can exploit the released ACC to act inside the coordination context (point 5), according to the constraints, rules, possibilities and qualities provided by its agent coordination context.

B. Agent Coordination Context Structure

As shown in Fig. 2, the TuCSoN Meta Agent Coordination Context is deployed to request an ACC and enter the coordina-

tion context. In particular, the model of the MCC account for the following logic interface:

```
MCC Interface ::=
    enter|enter(Node)|enter(Node, Description)
    current(ContextId)|select(ContextId)
```

Node is the identifier of the TuCSoN node providing the context which the agent aims at entering. *ContextId* is an identifier of the agent coordination context released to the agent: it is provided by the system and it can be used by the agent for distinguish locally different coordination contexts which it is participating simultaneously. *Description* is a text providing a Prolog (first logic) theory. The `enter` commands are used to enter a coordination context:

- `enter` – used by an agent to enter the default coordination context and get an agent coordination context, according to its identity. For instance, it could be the local node where the agent is executing, or the nearest node that is available to an agent in the case of nomadic computing: the agent has not to know this details, it simply asks to enter the coordination context;
- `enter(`*Node*`)` – used by an agent to enter the coordination context provided by a specific TuCSoN node;
- `enter(`*Node*`,`*Description*`)` – used by an agent to enter the coordination context provided by a specific TuCSoN node, describing the properties and features of the agent coordination context which defines agent participation;

According to the organisational rules enforced by the specific TuCSoN nodes – whose model is described in the following sections – the entrance can be allowed – providing the agent with an ACC with a specific configuration – or forbidden.

Since the agent can participate to multiple contexts at the same time, the system provides a way to identify and select the agent coordination context previously retrieved:

- `current(`*ContextId*`)` – gets an identifier of current agent coordination context; it is provided by the system and it can be used while the agent is inside the related coordination context, for selecting it or exiting it. After leaving it, it is no more valid;
- `select(`*ContextId*`)` – selects current coordination context target of agent actions, by means of the ACC identified by *ContextId*. The operations succeeds only if *ContextId* refers to a valid agent coordination context, that is the agent is yet inside the coordination context;

It is worth noting that the node name/address can not be directly used as identifier, since an agent can participate to the same node (context) with different ACCs – related, for instance, with different roles.

Once entered a context, the agent can act and perceive according to the coordination actions provided/ruled by its ACC *configuration* [27]; the model of the actions provided by the ACC – which are the analogous of the commands of the *control room* [27]) – accounts for:

```
ACC Interface ::=
    CoordinationOp |
    Tid ? CoordinationOp |
    Tid @ Node ? CoordinationOp |
    exit
```

```
CoordinationOp ::=
    out(T)|in(TT)|rd(TT)|inp(TT)|rdp(TT)|
    set_spec(S)|get_spec(S)
```

The set {`out(`*T*`)`, `in(`*TT*`)`, `rd(`*TT*`)`, `inp(`*TT*`)`, `rdp(`*TT*`)`, `set_spec(`*S*`)`, `get_spec(`*S*`)`} is the set of the classic Linda coordination primitives, extended with the primitives to manage tuple centre behaviour, setting and retrieving its behaviour specification; according to the rules and constraints of the specific agent coordination context, the execution of these actions with specific argument can be allowed or forbidden, and in this last case the agent request for executing the action would fail, reporting an *operation not allowed* exception. A target tuple centre can be either specified or not:

- *CoordinationOp* – without specifying a target tuple centre, the `default` tuple centre of the node (context) is considered implicitly as the target;
- *Tid ? CoordinationOp* – in this case the target of the action is the tuple centre identified by the *Tid* identifier; this identifier must be any valid logic term.

By means of the *Tid @ Node ? CoordinationOp* command it is possible to execute an action on a different context than the current one, provided by the node *Node*, according to the global space perception supported by TuCSoN along with the local one; if the agent has not yet an ACC to act inside that context, the entrance is negotiated as in the `enter(`*Node*`)` case. Finally, the ACC provides the `exit` commands to leave current coordination context. This could be important when the total time of agent participation to the context matters, for instance when computing the cost of the coordination services.

C. The Organisational Model

Contexts and roles description, including inter-role relationships and role admission/activation policies used in the negotiation stage to accept/build or refuse agent request, are declaratively described by tuples in the `config` tuple centre of each TuCSoN node. The coordination laws that define the behaviour of the `config` tuple centre – expressed in the ReSpecT language [1] – coordinate agent entering requests delegated to the welcome services, by either allowing requests to be satisfied and building the specific ACC configuration, or rejecting them, according to the the organisational rules and security constraints that occur (in that moment, dynamically) as tuples in the tuple set. In this way, we exploit a general purpose coordination virtual machine – the `config` tuple centre – to realise a general purpose *organisation/security virtual machine*, similarly to what can be found in some policy-driven and CSCW middleware [34], [6], [23], [7]. As will be in details in the following, the `config` tuple centre content is updated dynamically, according to the organisation and environment dynamics, tracing agents currently playing roles, dynamic information from environment and events – reified as tuples – that could be significative for defining organistational and security rules.

[1] The ReSpecT source code defining the behaviour of `config` tuple centre is not reported here because of lack of space, but it is available with the TuCSoN open source distribution at the TuCSoN web site [11]

In the following we describe the basic ontology currenly used to define role properties and the organisation rules as logic tuples in the `config` tuple centre.

Role description – The templates involved are:

- `role(Name,Cardinality,Policy)`
- `default_role(Role)`
- `default_role(AgentId,Role)`

A tuple of template:

`role(Name,Cardinality,Policy)`

means that a role called `Name` is defined in the coordination context; `Cardinality` indicates the maximum number of agents that can play simultaneously the role; `Policy` described the constraints ruling the actions of agents playing the role; policies are defined as a list of facts and rules concerning the constraints and properties of the role, and are described in details later. Each coordination context (node) defines a default role, that is assumed by agents not specifying explicitly the role name in the description provided in agent coordination context negotiation; this is described by tuples of template:

`default_role(Role)`

Also, a specific default role for a particular agent can be specified. The template is:

`default_role(AgentId,Role)`

When an agent `AgentId` asks to enter the coordination context without specifying any role, the default role specific for it is considered. If this information is not present, then the default role for the context is considered. The default values enable effective *awareness* of the coordination service toward the agent entering the context, by setting up specific agent coordination context according to its identity, without requiring it to explicitly provide this information. The same agent could have different default roles in different coordination nodes, according to the information stored in the `config` tuple centres. Some basic roles are provided by the default: Guest, Inspector, Administrator. Finally, information about authentication requirement for agents assuming a specific role can be specified be means of tuples of template:

`authentication_required(Role,Type)`

If no tuples of this kind are present, the role `Role` does not require the authentication of the agent in order to be played; instead, if specified, it requires the identity of agents assuming the role `Role` to be authenticated according to the authentication type `Type`.

Role-User relationship – The templates involved are:

- `allowed_membership(Role,AgentId)`
- `forbidden_membership(Role,AgentId)`

A tuple of template:

`allowed_membership(Role,AgentId)`

means that an agent `AgentId` is allowed to be member of the role `Role`. A tuple of template:

`forbidden_membership(Role,AgentId)`

means that an agent `AgentId` is forbidded to be member of the role `Role`.

Role-Role relationships – The templates involved are:

- `role_not_compatible(RoleA,RoleB)`
- `role_excludes(RoleA,RoleB)`
- `role_requires(RoleA,RoleB)`

A tuple of template:

`role_not_compatible(RoleA,RoleB)`

means that the two roles can not be played at the same time by the same agent. Note that this is an equivalence relationship. A tuple of template:

`role_excludes(RoleA,RoleB)`

means that if an agent is playing the role `RoleA`, it cannot assume the role `RoleB`. Note that this relationship is not symmetric. A tuple of template:

`role_requires(RoleA,RoleB)`

means that in order to assume the role `RoleA`, an agent must be playing the role `RoleB`.

Context dynamics – Tuples of template:

`player(AgentId,Role)`

describe agents currently inside a coordination context, in particular that the agent `AgentId` is currently playing the role `Role`. Instead, tuples of template:

`tuple_centre(TupleCentreId)`

describe the tuple centres actually created (accessed at least once) in the TuCSoN node.

D. Organisational Rules

1) Ruling Agent Entrance: These rules are applied by the TuCSoN node during the negotiation of the agent coordination context, when entering the coordination context. Typically, in role-based models these rules are called *role admission constraints* and *role activation* policies. The request of an agent `AgentId` to enter the coordination context in the role `Role` is satisfied (and the agent coordination context released) if and only if three constraints are verified:

- the agent is allowed to assume that role. This constraint is satisfied if *(i)* agent identity can be authenticated according to the authentication type possibly required by the role and expressed by `authentication_required` tuples; *(ii)* directly `allowed_membership` is found, or `forbidden_membership` is not found;
- the pre-condition on roles applies (such as role cardinality). This rule if satisfied if number of current agents playing the role (a sub set of the `player` tuple set) is less than the cardinality specified in the `role` description tuple;
- the role assignment is compatible with roles currently played by the agent, according to the role relationships described the tuple centre. This rule is satisfied if *(i)* the new role `Rn` is compatible with all the roles specified for each `player(Id,R)` tuple, i.e. no tuple `role_not_compatible(Rn,R)` and `role_not_compatible(R,Rn)` are found, *(ii)* each role `R` currently played by the agent (tuples `player(Id,R)`) must not exclude the new role `Rn`, i.e. no `role_excludes(R,Rn)` tuples must be found, and *(iii)* each role `R` required by the new role `Rn` (found in tuples `role_requires(Rn,R)`), must be currently played by the agent, that is a tuple `player(Id,R)`) must be found.

2) Ruling Agent Actions: The ACC embodies the possibilities and constraints related to a role in a coordination context, in

particular by enforcing the constraints specified as `Policy` in the role which the ACC represents. In particular, current property set ontology for the `Policy` list accounts for:

`allowed_actions(ActionList)`
`forbidden_actions(ActionList)`

These templates define what actions the agent is allowed/forbidden to do. More specifically, an agent is allowed to execute a generic action *a* if *a* matches some action in the list given by `allowed_actions` tuple, or does not match any action in the list given by `forbidden_actions` tuple. An action can be in the form

`Tid ? op(T)`

for `op` in {`out`, `in`, `rd`, `rdp`, `inp`}, and

`Tid ? op_spec(Spec)`

for `op` in {`set_spec`, `get_spec`}.
Then

`validity_date(DateFrom,DateTo)`

defines the period in terms of date range in which the agent coordination context used by the agent is valid; `DateFrom` and `DateTo` must have the form

`date(Year,Month,Day,Hour,Minute)`

Finally

`validity_time(Time)`

defines the period of validity (in milliseconds) of the agent coordination context, starting from the moment in which the agent enters the context.

Summing up, according to the configuration currently supported, an agent action `Tid?op(T)` issued at the time `t` and date `date` (which are local to and provided by the ACC), is allowed iff

- the temporal validity of the ACC described by the policy still holds (`t` is less than the validity time specified and/or the `d` is comprised in the range of the validity dates);
- an action matching `Tid?op(T)` can be found in the allowed actions, or no action matching `Tid?op(T)` can be found in the forbidden actions.

E. Dynamic Inspectability and Modifiability of Organisational/Security Setting

Dynamic inspectability and adaptability of the organisational/security settings are fundamental properties of the model. First, the organisational and security settings defining the coordination context can be dynamically obsered by inspecting the tuple set content of the `config` tuple centre, either by means of tools such as the inspectors [13], or by reading/retrieving tuples through `rd` and `in` coordination primitives. This makes it possible for intelligent agents *(i)* to inspect the coordination context to understand what constraints and possibility they have, according to their possible role, *(ii)* analyse and reason about organisation dynamics and evolution.

Then, the organisational and security settings defining the coordination context can be changed dynamically by changing the logic tuples contained in the `config` tuple centre; this is a fundamental property for managing organisation and securities issues in open systems, when rules and knowledge must be changed adapted according to the organisation and environment dynamics, unpredictable situations and problems, as happens in the case of dynamic and environmental role based systems [14], [8].

Since the `config` tuple centre itself is part of the organisation and the ruled coordination context, it contains also the roles and rules for accessing to its dynamic state – both in terms of tuple set and specification set. For instance, in its default configuration, only agents belonging to the `Administrator` role can change/remove tuples defining roles and role rules. So, both in the cases of inspectability and modifiability, the agent action on/perception of the `config` tuple centre is ruled by agent coordination contexts.

Finally, the behaviour of the `config` tuple centre as organisation/security virtual machine can be extended by adding coordination laws (as ReSpecT reaction tuples), for instance, to trace the basic events occurring in the coordination contexts as reified tuples. Examples of basic events currently traced are:

`event(AgentId, enter(Role),When)`
`event(AgentId, exit(Role),When)`

Tuples of the first template represent the event concerning an agent `AgentId` entrance in current context in the role `Role`; conversely, the tuples of the second template represent the event concerning the exit of an agent `AgentId` – playing the role `Role` – from the coordination context.

F. Advanced Policies

The agent coordination context notion promotes the support of multiple models for role policy; in the case of TuCSoN, the description of policies according to different models are eventually supported as a list of facts and rules interpreted and enacted by the ACC. For instance, to support a basic form of history based constraints, it could be useful to specify allowed sequences of actions, and so, more generally, the *protocol* related to the role. Other useful constraints that we are evaluating (that in part can be conceived by a suitable description of the protocol, according previous discussion) could be:

- the *maximum number* of specific tuples that an agent is allowed to place in a specific tuple centres of the node which provides the coordination context;
- *temporal rules*: the maximum number of interactions per seconds that an agent is allowed to execute; the minimum latency that must present between two agent specific coordination operations.

Moreover, not only constraints, but also quality of service (QoS) issues could be interesting to include in the policy, for instance:

- minimum throughput of the coordination service;
- maximum latencies of the coordination service. For instance: maximum latency between two agent operation requests.

These aspects concerning advanced role policies are subject of undergoing explorations and development.

III. Coordination as a Service

The notion of agent coordination concept allows issues about the *coordination as a service* [35] concept to be better modelled, making easier to conceive the explotation of TuCSoN coordination services in service-oriented computing contexts

[22]. In particular, ACC can be used to model explicitly QoS aspects and the context-awareness issues (in terms of both awareness of the agent with respect to services, and viceversa); for instance: non-functional aspects related to time (service duration, leasing, validity time, etc.), space (resource usage, such as CPU time, memory,etc.), quality of the communication (bandwidth, latency, etc.), and cost of interaction/communication. Security issues play an important role also in this case, and by means of the ACC can be faced at different abstraction levels:

- from the service point of view, in terms of both *(i)* the privileges/credentials that an agent must posses in order to exploit the service, when negotiating the ACC, and *(ii)* the actions allowed or forbidden for an agent exploiting a specific service, according its ACC;
- from the *meta*-service point of view, in terms of the way the service is exploited, in particular the security properties that characterise the medium enabling the interaction and the exchange of information (for instance, encryption).

So, the ACC abstraction becomes the key point to support the vision of TuCSoN as a *coordination provider*, providing coordination as service with existent service-oriented infrastructures, such as OSGi [29], SOLACE [16], Web Services [36], Jini [37], and the Grid [3]. The general idea is to enable the participation to the same TuCSoN coordination space through heterogeneous service-oriented infrastructures (see Fig. 1), upon which the TuCSoN coordination service has been registered accordingly, according to the environment in which agents are immersed.

Through the specific service discovery protocol of the hosting service-oriented architecture, the TuCSoN Agent Meta Coordination Context can be obtained to negotiate the ACC (as already seen in Fig. 2). In the Jini case for instance (Fig. 3), a proxy object acting as MCC is registered to a lookup service (point 1); after that, the proxy can then be retrieved by clients asking the lookup service for the TuCSoN MCC interface (point 2), and then used to negotiate ACCs with the Jini Welcome Service of the TuCSoN node (point 3). In the Web Service case (Fig. 4), the TuCSoN Meta Coordination Context is published as Web service by the TuCSoN node through a SOAP request to an UDDI Service (point 1). Thus, the registry information from UDDI about the TuCSoN service location can be find by an agent (point 2), and used to locate a WSDL document that details the interface semantics (related to the Agent Meta Coordination Context) for interacting with the TuCSoN Web (Welcome) Service (point 3). Then, with the WSLD document representing the MCC interface, the agent can contact the Welcome Service though a SOAP request (point 4) in order to negotiate the ACC, describing an `enter` command and related ACC properties. As a result of (a successful) negotiation, the ACC is published as a web service, and its WSLD document is released to the agent to bind and use its agent coordination context.

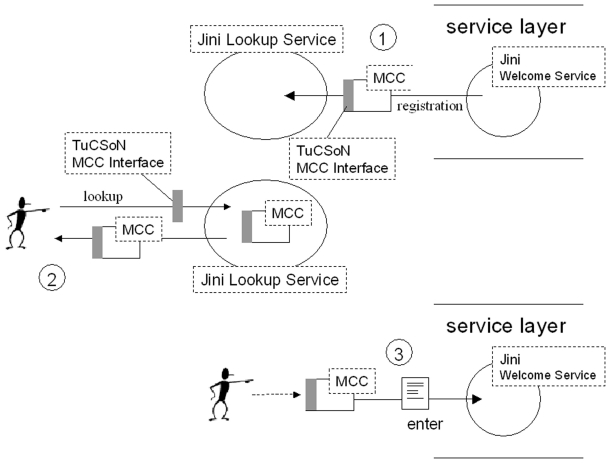

Fig. 3. Retrieval of the Agent Meta Coordination Context in the case Jini case.

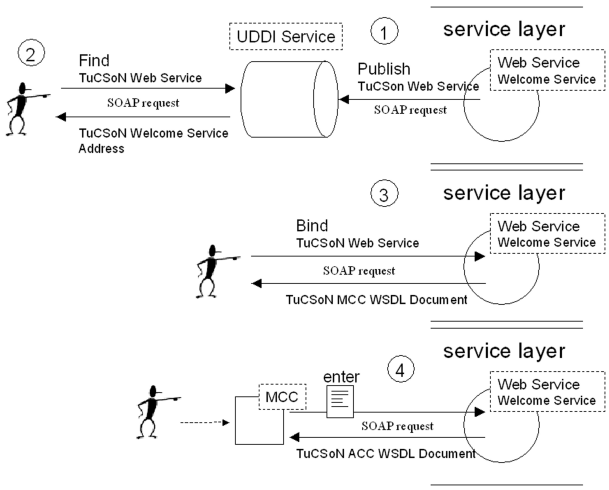

Fig. 4. Retrieval of the Agent Meta Coordination Context in Web Service case.

IV. BENEFITS

The agent coordination context notion makes it possible to suitably model both static and dynamic issues of agent organisations, in terms of role management, dynamic security model, integrated uniformly with coordination model. Modelling a MAS organisation implies giving explicit account for its structure, pattern and rules [38]. As requirement, organisations must be dynamics, flexible, and context-awareness. The TuCSoN ACC model explicitly provides a role-based organisational model, getting the benefits largely described in different contexts: MAS [21], [4], [20], [17], CSCW [33], distributed systems [24]. In particular, Role Based Access Model (RBAC) have been extensively studied and used to effectively model security aspects in distributed systems [1], [30]. The joint effort of the ACC model and tuple centre coordination model makes it possible to support explicitly organisational rules and constraints, involving the specification of polices concerning multiple roles, team, and contextual information, for which RBAC models have been found to be not fully adequate [18], [32], [8]. From this perspective, the TuCSoN approach provides features found in policy-driven middleware [6], [34], [24], providing means for encapsulation of organisational rules as policies, their dynamic management, and their deployment for enacting dynamic security models. In the context of agent-oriented software engineering, this property makes TuCSoN a suitable model for supporting smoothly the transition to the development and deployment stages from the design stages carried on

with methodologies explicitly focusing on the organisational and societal issues [26], [38].

A. Separation of Duty

The models support various degree of *separation of duty* [31], [33], a security principle used to formulate multi-agent control policies, requiring that two or more different agents be responsible for the completion of a task or set of related tasks. Typically separation of duties are enforced by constraints on role membership, on role activation and on role use. In our model, the constraints on role membership and on role activation are enforced during the negotiation stage, while the constraints on role use are enforced by the ACC used by the agent. More specifically, the simplest form of separation of duty is the *static* one, which accounts for strongly exclusion of agents performing a specific role, and, consequently, exclusion relationship among roles. Typically, static separation of duty is realised by constraints on role membership; In our model, this is provided by specifying organisational rules such as `allowed_membership`, `forbidden_membership` and by the `role_not_compatible` relationships. A more complex separation of duty strategy is the *dynamic* one (also called *weak* separation of duty), considered more suitable than the static to support the actual functioning of dynamic/complex organisation [31]. Instead of focusing on role membership, it concerns role activations, expressing user-role conflict according to current user's roles and role relationships. The simplest form of dynamic separation of duty makes it possible to specify roles with common members, but with users that cannot assume both roles at the same time; in our model this constraints can be expressed through `role_not_compatible`, `role_excludes` and `role_requires` relationships. Constraints on role use are generally adopted to provide *operational*, *object-based* and *history-based* separation of duty, which are the most complex form of dynamic separation of duties; generally speaking they account for constraining agent actions according to dynamic evaluation of the action contexts, involving shared objects/targets and interaction histories. Currently these constraints can be supported in the TuCSoN model not directly by the ACC, but as coordinations laws embedded in tuple centres. The advanced policies sketched in Subsection II-F provide the expressivity required for the purpose: accordingly, while tuple centres are conceptually better suited for enforcing the constraints related to shared objects/targets (tuples in our case), ACC, along with the advanced policy models is suitable to manage constraints related to individual agents interaction history, as required by operational and history-based separation of duty.

B. Dynamic Security Model

The notion of agent coordination context makes it possible to explicitly model the authentication and authorisation issues pointed out in [9], in particular:

- the negotiation of the ACC with a welcome service of a node can require the agent not only to provide but also to prove its identity, according to the agent coordination context requested. Standard authentication strategies, based on public key infrastructures, are exploited: if the agent coordination context requested by the agent need some form of authentication (for instance, because of the role the agent aims at playing), then the agent must provide a certificate proving its identity along with the enter request (through the service provided by the TuCSoN Meta Agent Coordination Context); the request is then encrypted with the private key of the target welcome service.
- the authorisation issues discussed in [9] are directly modelled by the ACC role policies: altogether, the policies enforced by ACC and the coordination policies embedded in tuple centres define the effect of the communication primitives used by agent;

[9] clearly remarks the dualism and relationship between coordination and security, in terms of modelling and shaping (constraining) the agent interaction space; here we extend this point of view, remarking the dualism between coordination and organisation, understanding the ACC (and related role model) as a tool to model not only security issues, but also (and mostly) the organisation of multiagent systems

C. Organisation Inspection and Analysis

The model provides direct support for observation and analysis of organisation rules and dynamics: the `config` tuple centre can be inspected by intelligent agents, and the organisation state and behaviour read retrieved in terms of logic tuples, thus promoting agent reasoning. This properties is especially important when considering open computational societies of agents [2]. Key characteristics of such societies are agent heterogeneity, possibly conflicting individual goals, limited trust and high probability of non-conformance to specifications. E-markets are a good examples of application domains where agents form such open computational societies in order to achieve their goals. Consequently, it is very important that the activity of such societies could be inspected, eventually using the observed information to reason about their current organisation and coordination state; This capability turns out to be fundamental in order to both make predictions about society evolution, and to change the coordination laws and the organisational rules in order to lead the society behaviours toward the desired patterns.

V. Conclusions and Future Works

By these first experiments, modelling and developing the notion of agent coordination context in TuCSoN model and infrastructure makes it possible to face some important issues related to MAS organisation and integration with service-oriented infrastructures as first class aspects. Nevertheless, we plan to evaluate and test current role model/ontology within complex real world scenarios previously considered without the agent coordination context framework, related to Virtual Enterprises and workflow management systems, pervasive computing (smart home and health care, in particular), and CSCW environments. In particular, this exploration should drive the enhancement of the basic agent coordination context with the advanced policies sketched in Subsection II-F; Also, we aim at

developing a formal theory related to agent coordination contexts and the role model, which – along with formal foundation of the coordination as a service notion [35], and more specifically of tuple centre model and of the ReSpecT language semantics – would enhance the possibilities of analysis and reasoning sketched in Subsection IV-C.

REFERENCES

[1] G.-J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):207–226, 2000.

[2] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1053–1061. ACM Press, 2002.

[3] F. Berman, G. Fox, and T. Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, Dec. 2002.

[4] G. Cabri. Role-based infrastructures for agents. In *Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2001)*. IEEE, 2001.

[5] G. Cabri, L. Leonardi, and F. Zambonelli. Context-dependency in Internet-agent coordination. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents World*, volume 1972 of *LNAI*, pages 51–63. Springer-Verlag, Dec. 2000.

[6] A. Corradi, N. Dulay, R. Montanari, and C. Stefanelli. Policy-driven management of agent systems. In M. Sloman, J. Lobo, and E. Lupu, editors, *Policies for Distributed Systems and Networks, International Workshop, POLICY 2001 Bristol, UK, January 29-31, 2001, Proceedings*, volume 1995 of *Lecture Notes in Computer Science*, pages 214–229. Springer-Verlag, 2001.

[7] M. Cortes. A coordination language for building collaborative applications. *International Journal of Computer Supported Cooperative Work (CSCW)*, 9(1):5–31, 2000.

[8] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies*, pages 10–20. ACM Press, 2001.

[9] M. Cremonini, A. Omicini, and F. Zambonelli. Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In F. J. Garijo and M. Boman, editors, *Multi-Agent Systems Engineering*, volume 1647 of *LNAI*, pages 77–88. Springer-Verlag, 1999. 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Valencia (E), 30 June – 2 July 1999, Proceedings.

[10] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995:18–39, 2001.

[11] DEIS Università di Bologna/Cesena. TuCSoN home page. `http://lia.deis.unibo.it/tucson/`.

[12] E. Denti, A. Natali, and A. Omicini. Programmable coordination media. In D. Garlan and D. Le Métayer, editors, *Coordination Languages and Models – Proceedings of the 2nd International Conference (COORDINATION'97)*, volume 1282 of *LNCS*, pages 274–288, Berlin (D), 1–3 Sept. 1997. Springer-Verlag.

[13] E. Denti, A. Omicini, and A. Ricci. Coordination tools for the development of agent-based systems. *Applied Artificial Intelligence*, 16(9), Oct. 2002. Special Issue "From Agent Theory to Agent Implementation".

[14] W. K. Edwards. Policies and roles in collaborative applications. In *Proceedings of the ACM 1996 conference on Computer supported cooperative work*, pages 11–20. ACM Press, 1996.

[15] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[16] M. Fredriksson and R. Gustavsson. Theory and practice of behavior in open computational systems. In R. Trappl, editor, *Cybernetics and Systems 2002*, Vienna, Austria, 2002. Austrian Society for Cybernetic Studies. 16th European Meeting on Cybernetics and System Research (EMCSR 2002), 2–5 Apr. 2002, Vienna, Austria, Proceedings.

[17] L. Gasser. Mas infrastructure: Definitions, needs, and prospects. In T. Wagner and O. Rana, editors, *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, volume 1887 of *LNAI*. Springer-Verlag, 2001.

[18] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies*, pages 21–27. ACM Press, 2001.

[19] J. B. D. Joshi, W. G. Aref, A. Ghafoor, and E. H. Spafford. Security models for web-based applications. *Communications of the ACM*, 44(2):38–44, 2001.

[20] A. Karageorgos, S. Thompson, and N. Mehandjiev. Semi-automatic design of agent organisations. In *Proceedings of the 17th symposium on Proceedings of the 2002 ACM symposium on applied computing*, pages 306–313. ACM Press, 2002.

[21] E. A. Kendall. Role modelling for agent systems analysis, design and implementation. *IEEE Concurrency*, 8(2), 2000.

[22] V. Kotov. Towards service-centric system organization. Technical Report HPL-2001-54, Computer Systems and Technology Laboratory – Hewlett Packard (HP), Palo Alto, Mar. 2001.

[23] D. Li and R. R. Muntz. COCA: Collaborative objects coordination architecture. In *Computer Supported Cooperative Work*, pages 179–188, 1998.

[24] E. Lupu and M. Sloman. Reconciling role based management and role based access control. In *Proceedings of the second ACM workshop on Role-based access control*, pages 135–141. ACM Press, 1997.

[25] N. H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(3):273–305, 2000.

[26] A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer-Verlag, 2001.

[27] A. Omicini. Towards a notion of agent coordination context. In D. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, pages 187–200. CRC Press, 2002.

[28] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, Nov. 2001.

[29] OSGi – Open Services Gateway Initiatives Consortium. OSGi home page. `http://www.osgi.org/`.

[30] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[31] R. Simon and M. E. Zurko. Separation of duty in role-based environments. In *IEEE Computer Security Foundations Workshop*, pages 183–194, 1997.

[32] R. K. Thomas. Team-based access control (tmac): a primitive for applying role-based access controls in collaborative environments. In *Proceedings of the second ACM workshop on Role-based access control*, pages 13–19. ACM Press, 1997.

[33] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman. A coordination model for secure collaboration. In D. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, pages 1–20. CRC Press, 2002.

[34] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman. Design of a Policy-Driven Middleware for Secure Distributed Collaboration. In *In proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pages 393–400, July 2002.

[35] M. Viroli and A. Omicini. Coordination as a service: Ontological and formal foundation. In A. Brogi and J.-M. Jacquet, editors, *Foundations of Coordination Languages and Software Architectures – Papers from FOCLASA'02*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., 2002.

[36] W3C. Web services activity home page. `http://www.w3.org/2002/ws/`.

[37] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, July 1999.

[38] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.