Agent Coordination Context: From Theory to Practice

Alessandro Ricci, Mirko Viroli, Andrea Omicini
DEIS, Università degli Studi di Bologna
Via Venezia 52, 47023 Cesena, Italy
email: {aricci, mviroli, aomicini}@deis.unibo.it

Abstract

The Agent Coordination Context abstraction (ACC) has been recently introduced as a mean to model the environment and the agent-environment interaction in the context of Multi-Agent Systems (MASs). In this article we discuss how the ACC notion has been embedded in a MAS infrastructure, in terms of the new runtime structures and services. As a reference, we will take TuCSoN coordination infrastructure, which is extended with the ACC framework to integrate organisation and security support with coordination.

1 Introduction

The notion of Agent Coordination Context (ACC) has been introduced in [Omicini, 2002] as a first class abstraction to model agent environment and agent-environment interaction. This is a fundamental issue involving the engineering of MAS and the infrastructures supporting MAS [Sycara et al., 2003; Ferber et al., 2003; Omicini and Ossowski, 2003]. The complexity of the application scenarios aimed to be engineered with MAS calls for an explicit infrastructural support for defining agents' relationships with their social environment, in particular in terms of organisation and coordination. This implies the adoption of MAS infrastructures providing not only the means for enabling agent interaction and interoperability, but also for governing the interaction, in particular with first class abstractions shaping MAS organisation structures & rules and MAS social (coordination) activities.

Then, the ACC can be adopted as a key abstraction for modelling organisation and security issues for MAS governing infrastructure. An ACC is meant to (1) work as a model for the agent environment, by describing the environment where an agent can interact, and (2) enable and rule the interactions between the agent and the environment, by defining the space of the admissible agent interaction. The ACC can be conceived as the conceptual place where to set the boundary between the agent and the environment, so as to encapsulate the interface that enables agent actions and perceptions inside the environment. Then, the ACC can be suitably understood as an infrastructural abstraction, i.e. a first class abstraction created and released to agents by a MAS infrastructure in order to explicitly define agent presence within the MAS environment and society.

In this paper, we discuss how the ACC notion has been embedded in practice in our MAS coordination infrastructure, TuCSoN. First we recall the features of the ACC notion in theory for modelling MAS (Section 2), then we describe in detail how the ACC notion has been exploited in practice, to extend the basic TuCSoN infrastructure model (Section 3). In order to clarify the approach, a simple example using ACCs in a Contract Net Protocol scenario is sketched (Section 4). Finally, conclusion, related and future works are provided (Section 5).

2 Agent Coordination Contexts in Theory

The ACC abstraction can be suitably used to model the presence of the agent inside a social/structured environment from different viewpoints. In particular:

Modelling Organisation – The ACC abstraction makes it possible to explicitly model the presence of an agent in an organisational context, where specific structures and rules are defined: for the purpose, the ACC can embed and reify the relationships that link an individual agent with respect to his position in the structure. In particular, in a role-based organisational context, the ACC makes it possible to represent the role(s) that an agent is playing inside an organisation, and then his responsibilities, permissions and the interactive behaviour (protocols, conversations). The ACC is then particularly suitable in those organisational frameworks where the role characterisation is provided solely in terms of the expected or allowed observable behaviour of the agents, not relying on agent internal (e.g. cognitive) aspects. This is particularly useful for modelling organisational frameworks dealing with heterogeneous agent models.

∗This work has been partially supported by MIUR, project COFIN 2003 (ex 40%) “Fiducia e diritto nella società dell’informazione”, and by MIPAF, project SIPEAA “Strumenti Integrati per la Pianificazione Eco-compatibile dell’Azienda Agricola.”
From a dynamic point of view, the ACC suitably embodies the idea of an agent working session inside the organisation: in order to participate and actively work inside an organisation, an agent must first obtain an ACC, framing his position (roles). Obtaining an ACC is a negotiation process: the agent proposes to join the organisation playing some specific roles (possibly in specific groups/societies); the agent request can be refused because of the organisational rules defined in the specific organisation (such as incompatibility of roles, cardinality constraints, etc), hence requiring the agent to formulate a refined proposal. In the case of a successful negotiation, a properly configured ACC is released to the agent, enabling his concrete participation inside the organisation. The ACC released can be inspected by the agent in order to know the details of his participation (roles, allowed actions, etc), promoting form of agent awareness and reasoning. It is worth remarking that If the negotiation fails, the agent has no means to interact with the organisation resources, since he has no ACC providing the access.

Modelling Access Control – The governing behaviour enacted by the ACC on the agent makes this abstraction suitable to model forms of dynamic access control to environment resources. In particular, Role-Based Access Control Model (RBAC) models, currently considered the most effective approach for modelling security in advanced information systems and complex organisations [Sandhu et al., 1996], can be suitably modelled with ACCs [Omicini and Ricci, 2003].

Modelling the Quality of Interaction – An interface, the ACC is the conceptual place where non-functional properties related to the quality of the interaction/communication can be suitably modelled. The quality of the interface can be related to issues such as performances, latencies, but also reliability, security, and all the aspects that could require a different kind of spatial/temporal resource allocation for the infrastructure releasing the ACC.

Modelling Relationships between Agents and Institutions – The ACC represents a contract between the agent and the institution (organisation) which released it. Then, when formally specified and certified by an external authority, the ACC can be used as a legal artifact for defining what kind of interaction service the infrastructure promised to the agent, and, conversely, what kind of actions an agent was expected to execute depending on his role(s).

3 ACC in Practice: Extending the TuCSoN Model

The ACC abstract concept has been brought to practice embedding it into the TuCSoN coordination infrastructure for MAS. TuCSoN was at beginning conceived as a purely coordination infrastructure for (mobile) agent based systems, orthogonal to the agent computational model [Omicini and Zambonelli, 1999]. The infrastructure has been then extended to cover security issues in synergy with coordination [Cremonini et al., 1999], and now the ACC concept is used to integrate a role-based organisation model and extend security toward the RBAC approach. The infrastructure model can be summarised describing four main related aspects: the coordination model, the organisation model and the security model.

TuCSoN coordination model is based on the coordination as a service approach, that is promoting the exploitation of general purpose coordination artifacts/services (tuple centres) that can be dynamically customised (programmed) in order to support agent coordination [Omicini and Zambonelli, 1999].

From the topology point of view, TuCSoN coordination artifacts are collected in infrastructure nodes, distributed over the network, organised into articulated domains [Cremonini et al., 1999]. A domain is characterised by a gateway node and a set of nodes called places. Briefly, a place is meant to host tuple centres for the specific applications/systems, while the gateway node is meant to host tuple centres used for domain administration, keeping information on the places. A place can belong to different domains, and can be itself a gateway for a sub-domain.

From the organisation point of view, TuCSoN adopts a role-based organisational model [Omicini and Ricci, 2003]. Each system/application living on the infrastructure is structured as an organisation composed by a set of societies, where specific roles are defined. A detailed description of current organisation ontology can be found in [Omicini and Ricci, 2003]. An agent participates to the organisation (system) always by playing one or multiple roles, in the same or different societies. The participation is fully dynamic: agents can dynamically enter and exit (also physically, in the case of mobile agents) from an organisation, activating different sets of roles through ACC re-negotiation. Each organisation is mapped onto a domain, and so it involves at least one gateway and a typically not empty set of places. Dynamic information about the organisation structures and rules (available societies, roles, the roles governing the organisation) are stored as logic tuples in the $ORG($OrgID$)$ tuple centre of the gateway, where the $OrgID$ is the identifier of the organisation.

The security model is strictly related to the topology and the organisation model. The role notion makes it possible to enact role-based access control, analogously to RBAC models, as found in the security management of complex information systems, extending the basic authentication and authorisation framework defined in [Cremonini et al., 1999]. The roles determine the set of actions and interaction protocols that an agent playing the roles is allowed to execute [Omicini and Ricci, 2003].

The ACC is the key abstraction to support at the infrastructure level the organisation and security model. An ACC is modelled as a runtime entity released by the TuCSoN infrastructure as interface to access the tuple centres of an organisation. An agent aiming at joining an organisation, must first obtain an ACC, properly configured, from one of the gateways, exploit-
ing the service provided by the TuCSoN infrastructure. Once obtained the ACC, the agent can use it by executing the available actions (coordination primitives) so as to access the tuple centres of the organisation, according to the constraints enforced by the specific ACC configuration. When the agent wants to exit from the organisation, he quits from the ACC, ending his working session. Multiple ACCs can be held by an agent simultaneously, one for each organisations where the agent is actively playing.

3.1 The ACC Structure and Behaviour

An ACC is defined by an Interface, a Contract, and a Contract State. The ACC Interface defines the set of operations that the agent can use to act inside a TuCSoN organisation. The ACC Contract is a description of the relationships between the agent and the organisation, in particular of the policy enacted by the ACC ruling agent actions and interaction protocols. The ACC Contract State is a description of the runtime state of the ACC, concerning the evolution of ongoing interaction protocols. Both the ACC Contract and the ACC Contract State information are meant to be inspectable in order to promote agent reasoning about his current position inside the organisation.

The ACC Interface

The abstract model of the ACC Interface has the following structure:

\[
\text{ACC Interface ::= doAction}(\text{+ACCAction}, \text{-ActionID})|\text{isActionCompleted}(\text{+ActionID},\{\text{true}\text{,false}\})|\text{isActionFailed}(\text{+ActionID},\{\text{true}\text{,false}\})|\text{action}(\text{+ActionID},\text{ACCAction})|\text{actionResult}(\text{+ActionID},\text{ACCActionCompleted})|\text{getACCSpec}(\text{-ACCSpec})|\text{getACCState}(\text{-ACCState})
\]

\[
\text{ACCAction ::= CoordinationOp | Tid ? CoordinationOp | Tid@Node ? CoordinationOp}
\]

\[
\text{CoordinationOp ::= out}(T)|\text{in}(T)|\text{rd}(T)|\text{rdp}(T)|\text{set}\_\text{spec}(T)|\text{get}\_\text{spec}(T)
\]

The doAction request is used to execute an ACC action, that is the execution of an operation on a specific tuple centre. If the action is not allowed according to the ACC policy, an exception is generated, preventing action execution; otherwise, the action is executed and an action reference is released to the agent. The coordination operations are the tuple centre coordination primitives, for inserting, reading and retrieving tuples and for inspecting/changing the behaviour specification of a tuple centre [Omicini and Denti, 2001]. isActionCompleted checks if an executed action has been completed, or if its completion is still pending: this is the case, for instance, of the invocation of an in or rd operation on a tuple centre without the matching tuples. isActionFailed checks instead for possible failures, happened after action execution and before completion. action request is provided to retrieve the ACC action, given its identifier. In order to inspect the information possibly resulting from an action completion, the actionResult request is provided, which is meant to be used for actions which reached their completion. It is worth noting that there are no blocking actions: the doAction returns immediately also when invoking operations like in or rd – which are blocking operations in the basic Linda model [Gelernter, 1985]. Finally, the interface provides the means to inspect the ACC Contract and the ACC Contract State (getACCSpec and getACCState, respectively).

The ACC Contract

The contract contains the description of the relationships established between the agent and the organisation, encoded in the Prolog language in the form of a logic theory. The information include the name of the organisation releasing the ACC (organisation(ID)), the roles actively played by the agent (role(ID,SocietyID)), the release date of the ACC (release_date(Date)), and – possibly – its period of validity (validity(DateFrom,DateTo) or validity(LeaseTime)).

Most importantly, the contract contains the policy that the ACC uses to establish if an agent action can be executed or not. This ACC policy is obtained by composing the individual role policies, which are stored as logic tuples inside the $ORG tuple centre of the gateway admitting the agent.

The theory describing a role policy is composed by a set of can_do rules:

\[
\text{can_do(CurrentState,Action,NextState):-Conditions.}
\]

This rule means that the action Action can be executed in the role state CurrentState if conditions Conditions value, and – in that case – next role state is NextState. The concept of role state is used as a way to easily express interaction protocols; any Prolog term – also structured, partially specified – can be used to denote the role state. By default, the starting state is denoted by the init atom. CurrentState and NextState can be omitted, using the any Prolog symbol (_): omitting the CurrentState information accounts for stating the validity of the rules for every possible state; omitting the NextState information accounts for keeping current state as next state. Finally, Action denotes an agent action, as described in the ACC Interface.

Actually, Conditions can contain also built-in predicates useful to describe context-aware (with respect to local time, space and identity/positions of the agents) policies. Among the predicates, here we mention: agent_id(~ID) (which re-
trieves the identity of the agent owner the ACC),

**local_node(Node)** (retrieves the node hosting the agent), **session_time(TimeInMillis)** (which retrieves the number of milliseconds passed since the release of the ACC)\(^2\).

So, given the ACC policy, an action is admitted if and only if there is a can do rule which holds for it; in other words, an action Action can be demonstrated, given the theory. So, for instance a rule of the kind: can do(,,,,), means that every action is allowed. can do(,,), means that the action Action can be always executed. By specifying the body of the rules it is possible to express also specific forbidden actions: can do(,,,):not(Act=Action), means that every action can be executed but actions matching with Action template.

The adopted Prolog encoding makes it easy to map the formal language based on process algebras defined in [Omicini et al., 2003].

As a simple example of ACC policy, consider a message box service in which the tuple centre msg box is used as to exchange messages represented by tuples msg(DestID,Content). A suitable ACC policy for users can be then:

can do(,,,,),
can do(,,),:- msg(DestID,Content),:- agent_id(DestID).

The policy allows a user to send messages to any other one, but to retrieve only messages sent to him.

An example of an ACC with a timed action is:

can do(,,),:- session_time(ST), ST < T.

The action Action can be executed only within a period of time T.

Finally, an example of context-aware policy is:

can do(,,),:- local_node(Node).

The rule asserts that only agents located at the node Node can insert tuples into the tuple centre Tid.

The ACC Contract State

The ACC Contract State describes the dynamic state of an ACC, in terms of the contextual information related to agent location (both spatial and temporal), and the logic state of ongoing interaction. Also this information are encoded as a Prolog theory, composed by a set of facts. Current ontology includes current_time(Time) and current_node(Node), which describe respectively current time and node of the ACC (and then of the agent using it), and current_state(RoleState), which describes current logic state as defined for the ACC policy.

### 4 A Case Study: a constrained Contract Net Protocol with ACCs

For sake of concreteness, we briefly illustrate a more articulated example of ACC policy, considering a simple version of a well-known agent interaction protocol, the Contract Net (CNP) [Smith, 1979].

---

\(^2\)This value refers to the time in which the ACC accepts the request execution for the action.

Suppose to be in a task allocation scenario, inside an organisation called acme.org. A society for task allocation coordination activity is defined, called **task_distribution**. The society involves two roles, master and worker, and a coordination artifact used to support the task allocation coordination activity, the **tasks** tuple centre. This tuple centre encapsulates the coordination rules involved in the task allocation, actually embodying a CNP-like protocol, where masters are the managers and workers are the contractors. The coordinating behaviour of the tuple centre is defined by the reactions shown in Table 1 (bottom) expressed in the ReSpecT language [Omicini and Denti, 2001], and is briefly described in the caption of the table.

The interactive behaviour of masters and workers is described in the pseudo-code in Table 1 (top). For readability, instead of using the doAction for expressing action request and then actionResult for retrieving the result, the direct invocation of the coordination primitives is used. Agents aiming at playing the role of masters negotiate an ACC with the master role, instead workers specify the **worker** role. The negotiation process is simplified in the enterACC(Gateway,ACCSpec) request (line 0), in which the agent specifies the target gateway, the organisation to join and the role(s) to activate. A master announces a task to be performed (line 1) and waits for bids: when the related timeout expires, it retrieves the list of bids from potential contractors (line 3), selects the best bid according to its evaluation (line 4), awards the contract (line 5), and receives the result of the performed task (line 6). On the other side, a worker (Table 1, right) waiting for possibly-interesting task announcements (line 1) evaluates his capability to respond to this request (line 2), issues his bid, and starts waiting for an answer. If the bid is accepted by the master (answer awarded), the worker performs the task (line 5) and outputs the computed result (line 6).

Table 2 shows the ACC Contracts for the master and the worker role. The ACC policy for the master enforces an interaction protocol composed by four different role states (represented by the Prolog terms **init, task_announced, award_bidder, get_result**), which correspond to different protocol stages (initialisation, task announcement, bidder awarding and result retrieving). The policy enforces a master to access only information concerning the task he announced, in every step of the protocol (line 3,4,7), and to award an agent which was necessarily among the bidders of the specific task announcement (line 6). The ACC policy for the worker enforces a protocol composed basically by two states (**init, awarded**), which correspond to bidding and result provision stages. The policy avoids non-awarded bidders (workers) to provide fake results of the task to be allocated (line 4): only the awarded bidder is allowed to insert the tuple concerning task results.

The example considers only a simplified version of the contract net: more complex cases – for instance involving time constraints – would need more articulated role policy specifications.
Table 1: (Top) The behaviour of master (left) and worker (right) agents in the CNP example (Bottom) The ReSpecT code defining the behaviour of the tuple centre tasks, implementing the coordination rules of the Contract Net Protocol. A brief explanation: When a master issues a task announcement, reaction 2 is triggered and coordination data are set up in the tuple centre. Each time a worker makes a bid, reaction 1 stores information about the new proposal and updates the bid list. When the master eventually collects the bids, reaction 3 removes the task announcement tuple: so, no more bids are considered. Finally, when a master awards the contract, reaction 4 emits the tuple bid(Task, MyBid, awarded) to notify the specific worker waiting for that answer, then triggers reaction 5 and 6: the first places the refuse others tuple, which is used to collect and remove the information about other bidders (tuples contractor), while reaction 6 notifies the other contractors by emitting the bid(Task, TheBid, 'not-awarded') tuple.

Table 2: ACC contract for the master role (Top) and worker role (Bottom).
5 Conclusions and Future Work

In this paper we discussed how the notion of Agent Coordination Context can be used in practice in a MAS coordination infrastructure, providing support for organisation and role-based access control models. TuCSoN has been considered as reference infrastructure: however, conceptually, the ACC framework could be applied in the context of any agent-based systems, since it is neutral with respect to the specific agent and interaction model.

The notion of Agent Coordination Context is similar to the notion of controller as found in the Law Governed Interaction (LGI) model [Minsky and Ungureanu, 2000]. Generally speaking, LGI is a message-exchange mechanism that allows an open group of distributed agents to engage in a mode of interaction governed by an explicitly specified and strictly enforced policy, the interaction law of the group. Law enforcement is decentralized, and carried out by a distributed set of controllers, one for each member of the community. As the LGI controller, the ACC enforces rules constraining the action/perception space of the agent exploiting it, enabling the enactment of policy that are local to the agent. In our approach, global coordination policies—meant to model laws concerning the society as a whole, not related to specific agent actions but to the global agent interaction space—are embedded and enforced by the coordination artifacts (tuple centres).

Several works exist in literature about the notion of contracts and institution in MAS ([Noriega and Sierra, 2002; Weigand et al., 2003; Dellarocas, 2000]): mostly, these works provide formal models useful for the design of MAS organisation. Our approach is more oriented to the engineering of the systems, in particular to the infrastructure support for their construction and runtime management.

Several issues are worth to be investigated in future works, extending the basic ACC model depicted in the paper. Currently we are exploring the link between current ACC language based on Prolog theory and the formal semantics based on process algebras described in [Omicini et al., 2003]. We plan to support heterogeneous languages and ontologies—in particular standard ones, based on XML and related semantics framework—as ACC language for contract specification and dynamic state description. Finally, some application contexts that have been engineered in the past with TuCSoN—mainly distributed workflow management for virtual organisations and smart environment scenarios—will be reconsidered and re-engineered, stressing the benefits of the new support for organisation and access control provided by the infrastructure.

References


[Omicini and Denti, 2001] Andrea Omicini and Enrico Denti. From structures: however, conceptually, the ACC framework can be applied in the context of any agent-based systems, since it is neutral with respect to the specific agent and interaction model.

The notion of Agent Coordination Context is similar to the notion of controller as found in the Law Governed Interaction (LGI) model [Minsky and Ungureanu, 2000]. Generally speaking, LGI is a message-exchange mechanism that allows an open group of distributed agents to engage in a mode of interaction governed by an explicitly specified and strictly enforced policy, the interaction law of the group. Law enforcement is decentralized, and carried out by a distributed set of controllers, one for each member of the community. As the LGI controller, the ACC enforces rules constraining the action/perception space of the agent exploiting it, enabling the enactment of policy that are local to the agent. In our approach, global coordination policies—meant to model laws concerning the society as a whole, not related to specific agent actions but to the global agent interaction space—are embedded and enforced by the coordination artifacts (tuple centres).

Several works exist in literature about the notion of contracts and institution in MAS ([Noriega and Sierra, 2002; Weigand et al., 2003; Dellarocas, 2000]): mostly, these works provide formal models useful for the design of MAS organisation. Our approach is more oriented to the engineering of the systems, in particular to the infrastructure support for their construction and runtime management.

Several issues are worth to be investigated in future works, extending the basic ACC model depicted in the paper. Currently we are exploring the link between current ACC language based on Prolog theory and the formal semantics based on process algebras described in [Omicini et al., 2003]. We plan to support heterogeneous languages and ontologies—in particular standard ones, based on XML and related semantics framework—as ACC language for contract specification and dynamic state description. Finally, some application contexts that have been engineered in the past with TuCSoN—mainly distributed workflow management for virtual organisations and smart environment scenarios—will be reconsidered and re-engineered, stressing the benefits of the new support for organisation and access control provided by the infrastructure.

References


[Omicini and Denti, 2001] Andrea Omicini and Enrico Denti. From institutions: however, conceptually, the ACC framework can be applied in the context of any agent-based systems, since it is neutral with respect to the specific agent and interaction model.

The notion of Agent Coordination Context is similar to the notion of controller as found in the Law Governed Interaction (LGI) model [Minsky and Ungureanu, 2000]. Generally speaking, LGI is a message-exchange mechanism that allows an open group of distributed agents to engage in a mode of interaction governed by an explicitly specified and strictly enforced policy, the interaction law of the group. Law enforcement is decentralized, and carried out by a distributed set of controllers, one for each member of the community. As the LGI controller, the ACC enforces rules constraining the action/perception space of the agent exploiting it, enabling the enactment of policy that are local to the agent. In our approach, global coordination policies—meant to model laws concerning the society as a whole, not related to specific agent actions but to the global agent interaction space—are embedded and enforced by the coordination artifacts (tuple centres).

Several works exist in literature about the notion of contracts and institution in MAS ([Noriega and Sierra, 2002; Weigand et al., 2003; Dellarocas, 2000]): mostly, these works provide formal models useful for the design of MAS organisation. Our approach is more oriented to the engineering of the systems, in particular to the infrastructure support for their construction and runtime management.

Several issues are worth to be investigated in future works, extending the basic ACC model depicted in the paper. Currently we are exploring the link between current ACC language based on Prolog theory and the formal semantics based on process algebras described in [Omicini et al., 2003]. We plan to support heterogeneous languages and ontologies—in particular standard ones, based on XML and related semantics framework—as ACC language for contract specification and dynamic state description. Finally, some application contexts that have been engineered in the past with TuCSoN—mainly distributed workflow management for virtual organisations and smart environment scenarios—will be reconsidered and re-engineered, stressing the benefits of the new support for organisation and access control provided by the infrastructure.

References