Coordination and Conversation Protocols in Open Multi-Agent Systems

Abdelkader Gouaich

Coordination and conversation protocols in open multi-agent systems

Abdelkader GOUAICH

Laboratoire, Informatique, Robotique et Micro électronique
Montpellier, France,
gouaich@lirmm.fr

Abstract. This paper presents an approach to formally link a simple dependency-based coordination model to its related conversation protocol. Hence, by observing conversation among coordinating entities, an external observer is able to recognise valid conversations that do not break the established norms on coordination. This may help in building reliable autonomous agent based systems in open and untrustred environments.

1 Introduction

Technological evolutions achieved in several computing fields such as hardware, networking and software engineering have enlarged the potential use of software-based services. Hence, current and future software systems are expected to be communicative and collaborative systems defined everywhere and offering their services at anytime to help users in their daily activities. Still, both changes on expectations and technological evolutions have also implied changes on software systems properties and hypothesis previously formulated on them. Hence, current and future software systems exhibit the following properties [27] :

- Situatedness: Having local communication and local interaction properties, a software system's global functionality depends on the local context of its elementary components.
- Openness: The software system has no longer a defined clear barrier and static structure. It is permanently evolving by merging or rejecting sub-components.
- Locality in interaction: Interaction among components can be conducted locally without requiring a centralised communication infrastructure. This raises the dynamics of the software system and allows emergence of new observable functionalities defined by elementary components.
- Heterogeneity: Obviously in a large connected world several actors with different goals build software components. Hence, several heterogeneous entities have to collaborate or coordinate their actions when placed together in a certain context in order to achieve their local goals.
- Autonomy of the components: Autonomy of a software component is an important feature that should be remembered when designing large open distributed software systems. Hence, software components have to be considered as black boxes, and lack of knowledge on their internal structure make them behaving autonomously since their reaction to an external stimulus is not fully predictable [1].

In this context, agent-based software engineering offers an interesting approach for designing and building this class of software systems. In fact, most of the presented features have already been considered as axioms in the theoretical foundation and definition of the agency [26, 11]. This paper focuses on coordination and conversation aspects of multi-agent systems and tries to make a formal link between coordination protocols and their related conversation protocols.

2 Autonomy and Responsibility of software agents

An autonomous software agent is defined as a self-governed entity that decide what action to do according to its local goals and rules [5]. Starting from this autonomy interpretation and by considering communication and interaction as another fundamental feature of software agents, [1] has established the need of an automatic entity that does not modify its internal rules to achieve actual interactions among autonomous agents. In fact, the integrity of the software structure encoding the agent's computational behaviours was found as a sine qua none condition in order to implement autonomous agents without having any assumption on their internal architecture. So, to interact an agent is not allowed to modify other agents' software structure and delegate this risky action to a trusted automatic entity, namely the *deployment environment*. On the other hand, in order to meet requirements of autonomous agents and guarantee virtual agent society norms on coordination and conversation, it was found useful to consider the multi-agent system as a composition of two orthogonal components: autonomous agents and the deployment environment [1]. Autonomous agents are considered as unknown software components that interact in order to achieve their private goals. The term unknown refers to software components that hide completely the software structure implementing their behaviours. As seen above, this feature has to be considered as an axiom in large distributed systems. Besides, the deployment environment may encode the agent virtual society laws and norms independently from agents that populate it and thus establishes a referential to establish agents responsibility. For instance, interaction and conversation policies may be implemented by this entity. Since, the deployment environment is an automatic entity its internal laws are static and identifiable. Due to this property, the responsibility of the deployed unknown autonomous agents is established when their actions challenge the norms of the agent virtual society [1]. As an example of such deployment environment an algebraic model named MIC* has been introduced in [13].

2.1 {Movement, Interaction, Computation}* (MIC*):

MIC* is an algebraic structure where autonomous, mobile and interacting entities are deployed. Within this framework, all interactions are conducted by explicitly exchanging interaction objects through interaction spaces. Hence, agents do not alter directly the deployment environment or the perceptions of other agents, but send their attempts as interaction objects. Interaction objects are structured: in fact, a formal addition law can compose them commutatively $+$ to represent simultaneous interactions. Furthermore, abstract empty interaction object 0 can be defined to represent no interaction.

The less intuitive part of the structure of the interaction objects concerns negative interaction objects. Negative interaction objects are constructed formally and may have no interpretation in the real world. However, they are useful for the internal model definitions and implementation of the deployment environment. For instance, the deployment environment can cancel any action, x, of the agent simply by performing an algebraic operation, $x + (-x) = 0$, that is expressed within the model notations. Finally, interaction objects defines a structure of a commutative group $(\mathcal{O}, +)$, where $\mathcal{O}$ represents the set of interaction objects and $+$ the composition law. Interaction spaces, represented by $\mathcal{S}$, are defined as abstract locations where interaction between agents holds. They are active entities that control their local and specific interaction rules. For instance, interaction object that are sent inside an interaction space may be altered if they violate the interaction norm. Agents, represented by $\mathcal{A}$, are autonomous entities that perceive interaction objects and react to them by sending other interaction objects. As said before, agents' actions are always considered as attempts to influence the deployment environment structure. These attempts are committed only when they are coherent with the deployment environmental rules of evolution. Having these elementary definitions, each MIC* term is represented by the following matrices:

- Outboxes Matrix: The rows of this matrix represent agents $A_i \in \mathcal{A}$ and the columns represent the interaction spaces $S_j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ is the representation of the agent A_i in the interaction space S_j.
- Inboxes Matrix: The rows of this matrix represent agents $A_i \in \mathcal{A}$ and the columns represent the interaction spaces $S_j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ defines how the agent A_i perceives the universe in the interaction space S_j.
- Memories vector: Agents $A_i \in \mathcal{A}$ represent the rows of the vector. Each element m_i is an abstraction of the internal memory of the agent A_i. Except the existence of such element that is proved using the Turing machine model, no further assumptions are made in MIC* about the internal architecture of the agent.

The set of all MIC* terms is represented by $\mathcal{T}$.

Dynamics of MIC*: The dynamics of the deployment environment is described as a sequence of elementary evolutions: $\mathcal{T} \rightarrow \mathcal{T}$. Three main classes of evolutions were characterised in MIC*:

- Movement μ: A movement is a transformation μ, of the environment where both inboxes and memories matrices are unchanged, and where outboxes matrix interaction objects are changed but globally invariant. This means that the interaction objects of an agent can change positions in the outboxes matrix and no interaction object is created or lost.
- Interaction φ: The interaction is characterised by a transformation φ that leaves both outboxes and memories matrices unchanged and transforms a row of the inboxes matrix. Thus, interaction is defined as modifying the perceptions of the entities in a particular interaction space.
- Computation γ: An observable computation of an entity transforms its representations in the outboxes matrix and the memories vector. For practical reasons, the

inboxes of the calculating entity are reset to 0 after the computation to distinguish interaction objects that were involved in different computations.

$$t_0 \in \mathcal{T} \overset{f_0 \in \gamma \cup \varphi \cup \mu}{\longrightarrow} t_1 = f_0(t_0) \in \mathcal{T} \overset{f_1 \in \gamma \cup \varphi \cup \mu}{\longrightarrow} t_2 = f_1(t_1) \in \mathcal{T} \rightsquigarrow t_{n+1} = f_n(t_n) \in \mathcal{T}$$

Fig. 1. Evolution of a MAS deployment environment starting from an initial term t_0 until t_{n+1} by following elementary transformations of type μ, φ or γ.

The main idea of MIC* approach is that the dynamics of the deployment environment can be modelled as a composition of evolutions that can be considered as being a movement, interaction or computation. Consequently, as presented in figure 1, any state of the deployment environment can be reached by following a path of evolutions of type μ, φ or γ. Hence, the deployment environment dynamics is no more an unknown and chaotic evolution, but a structured sequence of elementary transformation.

3 Background

3.1 Generalised study of coordination:

Malone and Crowson in [19] have noticed that coordination among autonomous entities is common to several independent fields such as computer science, economics, operational research and organizational theory. Hence, they have tried to study it in a single framework: *the coordination theory*. According to their definition, coordination is viewed as managing dependencies between activities. Hence, entities coordinate their actions in order to manage dependencies that exist between their activities. So, to understand what is coordination, one has to understand and study what are the existing dependency situations among activities.

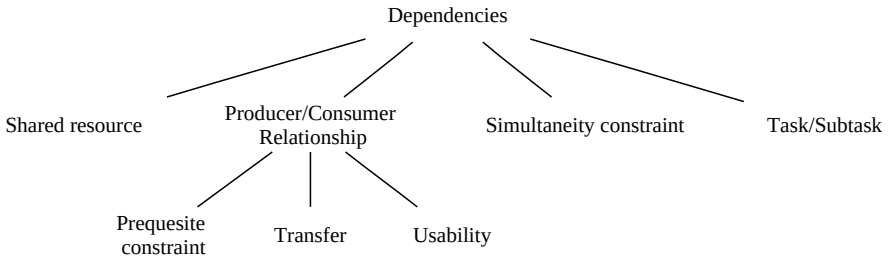

Fig. 2. Malone and Crowson [19] categorisation of common dependencies among activities.

As shown in figure 2, Malone and Crowson sketch a categorisation of the dependencies among autonomous entities as follows:

Shared resource: When several activities share some limited resource a particular *resource allocation* strategy is needed in order to handle this dependency. Several independent fields have identified this dependency and proposed some specific resource allocation strategies. For instance, computer science field has studied how available hardware and software resources are shared among computational entities. Hence, multi-threaded operating systems require allocation strategy and algorithms to share the computation units (processors, input/output devices) on a computer. Another example of limited resource dependency is the optimal sharing of available network bandwidth using a network protocol.

Producer/Consumer Relationship: This dependency represents situations when an activity produces something that is used by another activity. This relation implies the following dependencies:

- **Prerequisite constraint:** This dependency expresses an order that exists between the producer and the consumer.
- **Transfer:** When the producer activity produces something that has to be used by the consumer activity, the produced object has to be transferred from the production point to the consumer point. Managing this dependency involves defining how to *transport* things between the producer and consumer activities. For instance, in case of software programs where the produced thing is information, the transfer dependency is resolved using communication protocols as a transportation mechanism.
- **Usability:** The other dependency implied by producer/consumer relationship is the usability. In fact, the consumer should be able to correctly use the producer's products. *Standardisation* is a common way to resolve this dependency. Hence, both producer and consumer agree on what are the requirements to consider a product as consumable and try to meet these specifications. Syntax of messages and the semantics of commands are examples of standardisation when transferring information between software entities.

Simultaneity Constraint: Some activities are timely constrained, so they have to be executed at the same time (or cannot occur in the same time). This dependency expresses these situations. *Semaphores* are examples of a mechanism used in computer science to handle synchronised activities.

Task/Subtask This relation basically expresses the hierarchical decomposition of an activity in several sub-activities. So, the main activity depends on its sub-activities and finishes when they are completed.

3.2 Models and methodologies to specify coordination protocols in software systems:

To express coordination protocols among software components several models and notations have been introduced. This paragraph gives an overview of these models.

Software engineering methodologies: Usually software engineering methodologies such as MERISE and UML include in their specification model some graphical

and textual descriptions to specify coordination of activities to be implemented by software programs.

Statecharts: Statecharts [14] is an extension of state transition diagrams with three elements dealing with hierarchy, concurrency and communication. This formalism is used for the specification and design of complex discrete-event systems such as distributed systems and communication protocols.

Flowchart: A flowchart system shows the overall structure of a software system by tracing the flow of information, work and by highlighting key processing and decision points. It also includes the physical media on which data are inputted, outputted and stored.

Process Algebra: This family of formal models (CSP [16], CCS [21], Pi calculus [25], Ambient [4], Join calculus [12]) implicitly expresses coordination among processes by synchronising communication and actions using algebraic operations. So, coordination mechanisms are implicitly defined in the semantics of the operators and rewriting rules. The observable result is still an ordered execution of the distributed activities. For instance, a Pi-calculus process stops its activity waiting for an incoming message on a named communication channel; when another process sends a message on this named channel, the blocked process continues its execution. This synchronisation operation implicitly defines a partial order relationship among activities and can also be viewed as a producer/consumer relation between processes where messages are the exchanged resources.

Language of Temporal Ordering Specifications (LOTOS): LOTOS is a formal description technique standardised as ISO/IEC8807. It was developed by the International Organisation of Standardisation (ISO) to support standardisation of Open Systems Interconnections (OSI). In fact, formal description technique helps in diminishing ambiguities on OSI produced specifications that are distributed worldwide and used by several software engineering actors. LOTOS formalism is mainly based on process algebra to express the implicit order and dependencies among the activities. Besides, LOTOS allows abstract specification of data type using ACT ONE. This abstract data type specification language can be viewed as a way to resolve the usability dependency previously presented by Malone and Crowson.

Petri Nets: Petri Nets is a formal and graphical oriented language for design, specification, simulation and verification of systems in which communication, synchronisation and resource sharing are important. Examples of application areas of Petri nets are communication protocols, distributed systems, workflow analysis. The Petri Net model is a digraph composed by two classes of nodes: transitions and places representing respectively activities and perquisite relations between activities. Hence, each place contains tokens that are produced by the ancestor activities; a particular activity may be executed when all its preceding places contain tokens.

Specification and Description Language (SDL): SDL is a specification language standardised by the International Telecommunication Union (ITU) . SDL coordination mechanisms are expressed using extended finite state machines. SDL specifications include also abstract data type specifications described using ACT ONE as in LOTOS.

Extended State Transition language (ESTELLE): ESTELLE is a formal description technique standardised by ISO. It is suitable for the specification of coordination

mechanisms among distributed software systems and was used to describe OSI services and protocols. An ESTELLE specification is a hierarchy of communicating non-deterministic state machines. By specifying the used type of communication, synchronous or asynchronous, both dependencies and execution order are implicitly defined between the state machines in ESTELLE.

Message Sequence Charts (MSCs) : Message Sequence Charts (MSCs) [17] are graphical and textual language for the description and specification of the interactions between different components normalised by the (ITU). They are usally used in combination with SDL specifications or used as sequence diagram in UML. MSC denotational semantics is expressed using an adaptation of process algebra formalism. This helps in defining partial order relations between events, such as message exchange and actions, occurring in the system. Another interpretation of MSC denotational semantics is done using graph grammars.

High-Level Message Sequence Charts (HMSC): They are considered as formalism to compose MSCs automatons using sequence, parallel and alternative operators. These compositions are given two different semantics within HMSC: strong composition, where all events of a scenario must hold before executing the next scenario; and weak composition where events are executed when possible. Since, MSCs semantics is defined as a partial order between events occurring in a system, the semantics of HMSC is defined as an extension of these partial order relations built following the semantics of the composition operators expressing sequential composition, alternative and iteration. HMSC denotational semantics is expressed using grammar graph that helps in studying some general properties of the system using model-checking techniques and to translate the system specification on other specification languages such as SDL, ESTELLE or Statecharts.

3.3 Software architectures to implement coordination in software systems:

Coordination (software) community has also defined some coordination architecture implementing generic coordination mechanisms such as synchronisation and producer / consumer relationships. Entities of a coordination architecture are the following: coordinables, representing entities to be coordinated; coordination media, representing the media used to coordinate the entities; and finally coordination laws that define how the coordination media reacts in response to coordinables actions. A survey of coordination architectures can be found in [24]. It categorises coordination architectures in two main categories:

Data driven In data-driven coordination models, the coordination media is represented by an addressable storage space shared between the coordinables. Thus coordinables interact by storing and retrieving data structures from the shared data space. Besides, the coordination laws define how these data structures are represented, stored and consumed by the coordinables.

Control driven In control-driven coordination models, the coordination media is represented by a set of input/output communication ports linking coordinables and enabling their interactions. In order to achieve these interactions, the coordination media considers coordinables as black boxes and check their state changes and

events occurring on their communication ports. These events are then propagated to other coordinables following some specific coordination rules with no concern for their internal data and representation

3.4 Conversational aspects of coordination:

When coordinables are distributed, coordination is necessarily achieved through communication and interaction. Hence, entities communicate by exchanging messages in order to meet the requirements of their coordination protocol. When observing these exchanged messages among coordinating entities, an external observer remarks that the structure of the conversation, defined as an ordered sequence of messages, is guided by the underlying coordination protocol. Specifying this dialogue structure is known as establishing a *conversation protocol*. Among formalisms that are used to specify conversation protocols one can find the following: finite state machines (FSM), as in works of [2, 3, 20]; state transition diagrams (STD), as in works of [18]; coloured petri nets (CPN), as in works of [6, 10].

3.5 Discussion

As presented above, several works have studied the coordination problem in computer science. These works can be classified in the following categories:

Activity centric: Works of this category focuse on the activity part of coordination. Hence, their main goal is to know what activity to execute and when to execute it. This category is represented by the models for the specification of coordination protocols presented in §3.2.

Conversation centric: Works of this category focus their interest on the conversational aspect of coordination. Hence, their main goal is to know what message to send and when to send it in order to coordinate distributed activities. Works of 3.4 are examples of this category.

Implementation centric: This category of works propose generic software architectures and middlewares abstracting coordination mechanisms to easily design, implement and deploy software systems where concurrent activities have to be coordinated.

Generalisation efforts: Works of this category sketch a general framework trying to include all aspects of coordination and linking several research fields sharing same concepts and interests. Malone and Crowson's works are an example of such generalisation effort.

By observing the literature it seems that activity centric work, conversational centric works and architecture centric constitute blocks with some synergy among them. Still, there is no established formal link between them. Malone and Crowson coordination theory is a first step to link all these works in a single framework. Hence, this paper starts from their works in order to refine a simple dependency based-coordination model expressing dependencies among activities using a directed graph as mathematical model. Having this refined coordination model some concepts such as *role*, and

role-cut are introduced. Conversation protocols are then defined formally as a sequence of messages recognised by a rewriting grammar. This grammar is defined by the structure of the coordination model.

4 Simple dependency-based coordination model

Figure 2 presented common dependencies shared by several research fields. This categorisation is very helpful to understand why autonomous entities coordinate their actions. However, since Malone and Crowson seek for generality, they make no precise assumptions on the properties of the addressed system. Besides, some of the presented dependencies are redundant and could be expressed with other dependencies. So, by having more assumptions on the entities to be coordinated, would it be possible to define a more refined model of coordination where all dependencies are canonically expressed? To answer this question, let us first set some hypothesis on the considered systems to be coordinated:

- The considered system is a software system composed of either software or hardware autonomous and interacting components;
- The autonomy axiom implies that the internal structure of the entities is never accessible and modified by another entity. Consequently, communication and actions of the entities are asynchronous and explicitly represented as attempts of actions[1]. The autonomy axiom prevents also from knowing how entities actually behave and achieve their goals. Only observable aspects of their computation are studied.
- Communication among components is represented as an explicit exchange of data, or messages, through a communication medium and the communication process is not assumed to be synchronous.

Under these assumptions, all dependencies presented in figure 2 can be expressed canonically as producer/consumer relationships. For instance, the shared resource dependency is modelled by introducing a resource manager entity that gives authorisation to consumer to access the resource according to a particular resource management strategy. The consumers of the resource and the resource manger are obviously in a producer/consumer dependency. Similarly, simultaneity dependency is expressed as a consumer/producer dependency between a monitor and the concurrent components. Finally, task/sub-tasks dependency is also expressed as a set of producer/consumer dependency between the global task and its sub-activities. Consequently, the refined model of coordination has to express only the consumer/producer dependency with the following elements:

Activities: As said previously, coordination is defined between activities; thus, they have to be explicitly represented in the coordination model.

Transferable: According to results presented above, all common dependencies among activities can be considered as producer/consumer dependencies. A producer/consumer dependency induces both a transfer and prerequisite dependency. Notice that prerequisite dependency is also a consumer/producer dependency where the notification is considered as the thing to be transported.

Usability dependency: According to Malone and Crowson's, each producer/consumer dependency induces a usability dependency. The usability dependency guarantees that the transferable is usable by the consumer after being produced by the producer. Consequently, for each transferable a conformity-checking function should be provided to model the usability dependency. Using Malone and Crowson's terms this function represents the standard resolving the usability dependency between activities.

Roles: Functionalities of agents in their artificial society are usually abstracted as *roles*. Roles are considered in the refined coordination model in order to situate where the activities are executed during the coordination process.

Given these elements, the coordination model can be represented as a directional graph described as follows:

- Two types of vertices are identified representing activities, graphically symbolised as boxes, and transferables graphically symbolised as circles;
- For each transferable node a usability-checking function is associated to check the conformity of the transferable according to the usability convention that was established by the producer and the consumer;
- An oriented edge links an activity to a transferable to represent a production of a transferable by that activity and a transferable to an activity to represent a consumption of a transferable by that activity. A dependency relation is then completely defined by having the producer vertex, the transferable and the consumer vertex;
- As previously mentioned, a prerequisite dependency among activities is expressed as a producer/consumer dependency with a special notification transferable noted ϵ.
- The coordination graph can be split in several connected components that are linked by transferable vertexes to represent activities associated with each role.

The above elements are captured more formally by the following definition:

Definition 1. *Let $\mathcal{O}$ be a set of interaction objects (or messages), a dependency-based coordination model is defined as a digraph $G =< A, N, T, F, V^{\uparrow}, V^{\downarrow} >$. A represents the set of activity nodes. N and T are disjoint sets where elements of $\epsilon_i \in N$ are considered as notification nodes and elements $t \in T$ are transfer nodes. Arrows of the graph are defined by $V^{\uparrow}$ and $V^{\downarrow}$, such as $V^{\uparrow} : A \times (N \cup T)$ and $V^{\downarrow} : (N \cup T) \times A$ represent respectively production and consumption arrows. $F = \{f_{t \in T} : \mathcal{O} \rightarrow \mathcal{O}\}$ is a set of functions that check for each transfer node $t \in T$ the conformity of the actual exchanged transferable considered as an interaction object.*

When a transfer node is consumed by several activities, this is interpreted as an exclusive choice. Thus, only one activity may be executed after production of the transferable. On the other hand, when an activity produces several transferables, this is interpreted as simultaneous production of resources. Roles decomposition in a dependency-based coordination digraph is viewed as a particular partitioning of the activities in the digraph. Roles have to be independent and should not share activities. In fact, without this property, actual agents implementing these roles may be in conflict on activities and consequently lose their autonomy property.

Definition 2. *Having a coordination graph* $G =< A, N, T, F, V^{\uparrow}, V^{\downarrow} >$*, a role decomposition* d *is a subset of* $\mathcal{P}(A)$ *where:*

$$\forall x, y \in d, x \cap y = \emptyset$$

Roles decomposition divides the global coordination graph in several sub-graphs. Hence, for each role r in a role decomposition $d \subset \mathcal{P}(A)$, the associated role sub-graphs are built by including all activities of r and inserting all production and consumption arrows that are inside r. This is more formally defined as follows:

Definition 3. *A role sub-graph* $g_r =< A_r, N_r, T_r, F_r, V_r^{\uparrow}, V_r^{\downarrow} >$ *of a role* r *found in a decomposition* $d \subset \mathcal{P}(A)$ *of a dependency-based coordination digraph* $G =< A, N, T, F, V^{\uparrow}, V^{\downarrow} >$ *is defined as follows:*

$$\begin{aligned}
A_r &= r \\
N_r &= \{\epsilon \in N : \exists\, a, b \in A_r,\ (a, \epsilon) \in V^{\uparrow} \wedge (\epsilon, b) \in V^{\downarrow}\} \\
T_r &= \{t \in T : \exists\, a, b \in A_r,\ (a, t) \in V^{\uparrow} \wedge (t, b) \in V^{\downarrow}\} \\
F_r &= \{f_t \in F : t \in T_r\} \\
V_r^{\uparrow} &= \{(a, t) \in V^{\uparrow} : (a \in A_r) \wedge (t \in T_r)\} \\
V_r^{\downarrow} &= \{(t, a) \in V^{\downarrow} : (a \in A_r) \wedge (t \in T_r)\}
\end{aligned}$$

Role sub-graphs of a complete decomposition are linked by transferables defining dependencies among these roles. This part of the graph is named a *role-cut* and represents the glue that makes roles interdependent. Still, notification objects, $\epsilon \in N$, are considered as a special transferable with a special semantics. For instance, they do not induce a usability dependency between activities and can be implemented inside the same software entity without exchanging an explicit interaction object. So, notification nodes that do not always induce an explicit exchange of messages have not to be present in the role-cut. More formally, a roles-cut is specified as follows:

Definition 4. *Roles-cut of a dependency-based coordination graph* $G =< A, N, T, F, V^{\uparrow}, V^{\downarrow} >$*, is defined as the minimal set of transferable* $c \subset T$ *that if all its elements* $x \in c$ *were removed along with there related production and consumption vertexes this would produce a digraph with more connected components than the original digraph.*

4.1 Defining conversation protocol grammar from coordination protocol

Notations: Interaction among autonomous agents has to be explicitly represented. For instance, within the MIC* model, a formal deployment environment of autonomous agents [13], interaction is conducted through explicit interaction objects represented in the set $\mathcal{O}$. This set owns at least, for the purpose of this paper, a structure of a commutative group. Hence, interaction objects found in $\mathcal{O}$ can be summed commutatively with the $+$ law. This operation has not to be misinterpreted with the non-commutative concatenation law . (dot) that will be used in order to generate ordered sequences of messages represented by $\mathcal{O}^*$. When there is no ambiguity $x.y$ is simply represented by xy. In this section $\mathcal{O}^*$ represents the free monoid generated by the interaction object set $\mathcal{O}$ with the non-commutative dot law. Intuitively this represents words where the

alphabet is interaction object set. $\bar{\mathcal{O}}$ represents a marked set of interaction object defined as follows: $\bar{\mathcal{O}} = \{\bar{x} : x \in \mathcal{O}\}$. $\mathcal{O}'$ is defined as union set of $\mathcal{O}$ and $\bar{\mathcal{O}}$. $|\alpha|$ is a simple function defined on $\mathcal{O}'^* \rightarrow \mathcal{O}^*$ that retrieves marks from all marked interaction object found in a word α. The function $\pi_d : \mathcal{O}^* \rightarrow \mathcal{O}*$, where $d \subset \mathcal{O}$ is a projection of a word on a subset alphabet. In other words, this operation simply retrieves elements found in the word that are not in the subset alphabet d. For an activity node $a \in A$ in the dependency-based coordination graph, a^+ and a^- respectively represent the set of production arrows and the consumption arrows of a. $\overset{R}{\rightsquigarrow}$ represents a path or a finite composition of a relation $\overset{R}{\rightarrow}$.

Having these notations, the grammar of the conversation protocol is defined as a rewriting system where the rewriting relation, $\overset{R}{\rightarrow}$: $\mathcal{O}'^* \times \mathcal{O}'^*$, is defined as follows:

$$U_0 a_1 U_1 a_2 U_2 ... a_n U_n \overset{R}{\rightarrow} U_0 \bar{a_1} U_1 \bar{a_2} U_2 ... \bar{a_i} U_i U_{i+1} ... U_n$$
$$\Rightarrow ((\sum_{x=1}^{i} |a_x|, \sum_{x=i+1}^{n} |a_x|) \in R) \wedge (\forall j \in [1..i], a_j \notin \bar{\mathcal{O}})$$

$(a_i)_{i \in [1..n]} \in \mathcal{O}'$ are alphabet elements; $(U_j)_{j \in [0..n]} \in \mathcal{O}'^*$ are sequences belonging to the monoid.

Interpretation: $R : \mathcal{O} \times \mathcal{O}$ represents the set of reduction or simplification rules. The interpretation of these rules is as follows: $(\sum x, \sum y) \in R$ means that $\sum y$ messages appear *necessarily* after $\sum x$ messages. So, this defines a partial order relationship between messages. After a reduction, messages that have been used in the left part of a reduction relation are marked to avoid confusions and ambiguities: in fact, this mechanism ensures that the left part of a reduction relation is used once.

What is missing in the previous definition is the relation $R : \mathcal{O} \times \mathcal{O}$ of simplification rules. This set is constructed from the coordination graph $G =< A, N, T, F, V^{\uparrow}, V^{\downarrow} >$ as follows:

$$\forall a \in A, (\sum_{(x,a) \in a^-} x, \sum_{(a,x) \in a^+} x) \in R_G \tag{1}$$

Having these definitions, a conversation protocol is defined as the set of message sequences recognised by the grammar defined above. This is expressed more formally as:

Definition 5. *Let $\mathcal{O}$ be the interaction objects set; a conversation protocol $P_{(x_0,G)}$ for a dependency-based coordination graph and the axiom $x_0 \in \mathcal{O}^*$ is defined as follows:*

$$U \in P_{(x_0,G)} \iff \exists U' \in \mathcal{O}'^* : U' \overset{R}{\rightsquigarrow} x_0 \wedge |U'| = U$$

The problem with the given definition is that it includes all interaction objects that are exchanged among activities. However, as said before only interaction objects exchanged through a role-cut are observable and interesting to study. So, when defining a conversation protocol for a particular roles-cut the previous definition is expressed as follows:

Definition 6. *Let $\mathcal{O}$ be the interaction objects set; A conversation protocol $P_{(x_0,G,d)}$ defined on a role-cut d found in a dependency-based coordination graph G for the axiom $x_0 \in \mathcal{O}*$ is defined as follows:*

$$U \in P_{(x_0,G,d)} \iff \exists\, U' \in \mathcal{O}'^{*} : U' \overset{R}{\rightsquigarrow} x_0 \wedge |\pi_d(U')| = U$$

Thus, the projection operation ensures that only interaction objects observable on the role-cut are included in the definition of the conversation protocol.

4.2 Simple example

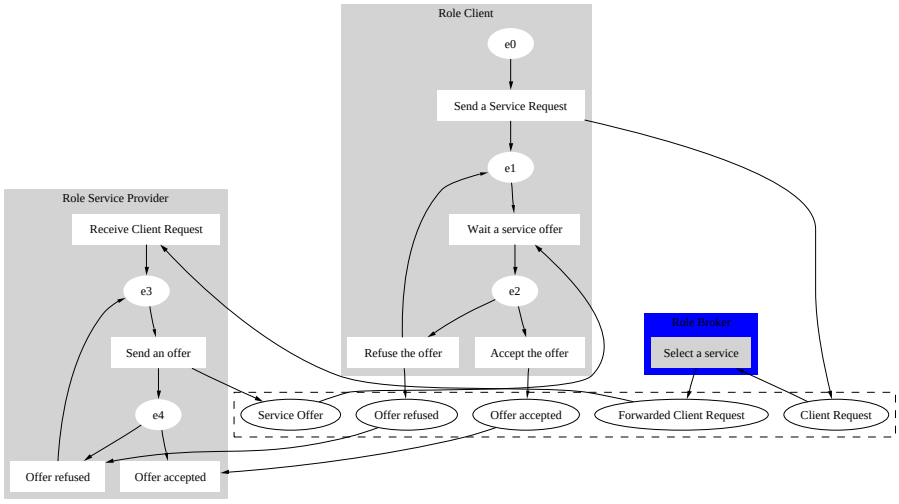

Fig. 3. *Simple Coordination protocol among three roles: Client, Broker and Service Provider, expressed using the dependency-based coordination graph. Activities nodes as represented as boxes and Transfer nodes as circles.*

Figure 3 presents a simple coordination protocol among three roles: the client, the service broker and the service provider. This coordination pattern in commonly used in both multi-agent systems and distributed systems, where the broker links clients to appropriate service providers. When connected to a client, the service provider recursively proposes offers to the client until the later accepts a good offer.

Figure 4 represents an abstracted view of the coordination protocol described in figure 3 where the roles, resources and activities are represented by abstract symbols. The roles-cut in this graph is $c_G = \{x, y, z, v, w\}$. By applying definition of equation 1, the simplification relations set, R_G, is calculated and equals to:

$$R_G = \{(e_0, x + e_1), (x, y), (e_1 + z, e_2), (e_3, z + e_4),$$
$$(e_2, e_1 + v), (v + e_4, e_3), (e_2, w), (y, e3)\}$$

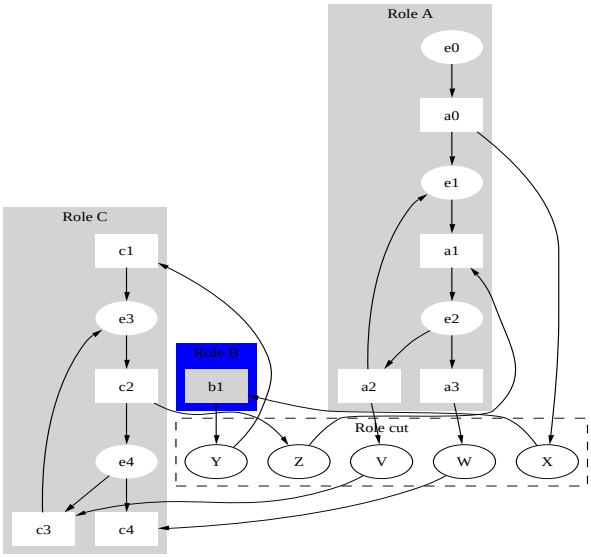

Fig. 4. *Abstract view of the coordination protocol presented in figure 3*

Having these elements, and by applying definition 6, the conversation protocol observable at the role-cut, c_G, among the *three* roles is fully defined. e_0 is considered as the axiom the conversation protocol. Notice that the definition of the conversation protocol depends only on the role-cut, no matter how many roles are linked by this roles-cut.
Let $u = xyzvzvzw \in \mathcal{O}^*$ be an observable conversation between A, B and C at the role-cut. This conversation is a valid conversation. In fact, by considering $\alpha_0\alpha_1\alpha_1\alpha_2$, where $\alpha_0 = e_0e_1xy$, $\alpha_1 = e_3e_4ze_2ve_1$ and $\alpha_2 = e_3e_4ze_2w$, this sequence can be reduced to the axiom e_0 as follows:

$$\alpha_0\alpha_1\alpha_1\alpha_3 \overset{R_G}{\rightarrow} \alpha_0\alpha_1\alpha_1e_3e_4z\bar{e_2} \overset{R_G}{\rightarrow} \alpha_0\alpha_1e_3e_4ze_2v\bar{e_1}e_3e_4\bar{z} \overset{R_G}{\rightarrow} \alpha_0\alpha_1e_3e_4ze_2v\bar{e_1}\bar{e_3}$$
$$\overset{R_G}{\rightarrow} \alpha_0\alpha_1e_3\bar{e_4}ze_2\bar{v}\bar{e_1} \overset{R_G}{\rightarrow} \alpha_0\alpha_1e_3\bar{e_4}z\bar{e_2} \overset{R_G}{\rightarrow} \alpha_0e_3e_4ze_2v\bar{e_1}e_3\bar{e_4}\bar{z} \overset{R_G}{\rightarrow} \alpha_0e_3e_4ze_2v\bar{e_1}\bar{e_3}$$
$$\overset{R_G}{\rightarrow} \alpha_0e_3\bar{e_4}ze_2\bar{v}\bar{e_1} \overset{R_G}{\rightarrow} \alpha_0e_3\bar{e_4}z\bar{e_2} \overset{R_G}{\rightarrow} e_0\bar{e_1}xye_3\bar{e_4}\bar{z} \overset{R_G}{\rightarrow} e_0\bar{e_1}xy\bar{e_3}$$
$$\overset{R_G}{\rightarrow} e_0\bar{e_1}x\bar{y} \overset{R_G}{\rightarrow} e_0\bar{e_1}\bar{x} \overset{R_G}{\rightarrow} \bar{e_0}$$

and $\pi_{c_G}(\alpha_0\alpha_1\alpha_1\alpha_2) = u$. The interpretation of this conversation is quite simple, after being connected with the service by the broker; the client has refused two offers and accepted the third one. By contrast, when no path is found to reduce a conversation into the axiom, this means that the partial order of messages has been violated and consequently the coordination protocol is not respected. For instance, yxz is an invalid conversation that will not be recognised by the grammar.

5 Discussion

The presented approach aims to check conformity of agents' conversation according to coordination protocols at the runtime. Hence, the conversation protocols (or interaction protocols) are not represented explicitly like in [9] but derived from the coordination protocol. The presented model of coordination is based on producer/consumer dependency relationship. The resource dependency model among roles/agents is becoming used even in agent design methodologies like *Tropos* [15] and can be assumed to be rich enough to express complex situations founds in multi-agent systems.

Like [9] we assume that the coordination protocols and consequently the conversation protocols should be public and represents a part of the multi-agent system social norms. However, each agent has a specific internal strategy that steers its behaviour and responses to external stimulus. This assumption constrains us to check conformity of coordination protocol by an external observer: the deployment environment or the coordination media for instance.

The *mediated interaction* [22, 7] offers the opportunity to observe all actions coming from the autonomous agents and to check their conformity according to the established norms. In our approach, the *interaction mediator* will be responsible of checking the conformity of an ongoing conversation by applying results of the definition 6. For instance, the next paragraph presents an example of how conversation protocols are checked within the MIC* framework.

5.1 Conforming interactions in MIC*:

Functionally, the conversation protocol checker can be considered as a Boolean function $f : \mathcal{O}^* \rightarrow \{\top, \bot\}$ taking as argument a sequence of interaction objects $o \in \mathcal{O}^*$ (or a conversation) and returning a value $v \in \{\top, \bot\}$. $\bot$ expresses the invalidity of a conversation while $\top$ expresses its validity. The set of conformity testing functions is noted CF. Using the approach presented in this paper, the conversation protocol checker has to reduce the conversation to the axiom by using the simplification rules obtained from the coordination protocol by applying the definition of equation 1. When the conversation can be reduced to the axiom, the conversation is considered as valid ($\top$) and invalid otherwise ($\bot$).

Interactions in MIC* (see §2.1) are defined as special evolutions of the deployment environment that modify the perceptions of agents, according to emissions of other agents in an interaction space. The goal here is to build interaction operations insuring the conformity of interactions. Hence, a conform interaction φ_f is built by the following application:

$$\begin{aligned} \varphi \times CF &\rightarrow \varphi \\ (f, g) &\mapsto f_g \end{aligned}$$

where, f_g is defined as following:

$$\forall x \in \mathcal{O}^* : f_g = \begin{cases} f \text{ if } g(x) = \top \\ Id \text{ if } g(x) = \bot \end{cases}$$

Id expresses the identity application that does not modify the deployment environment. So, given a standard interaction operation defined on MIC* and a conversation protocol checker, a conform interaction operation is defined on MIC*. This new interaction operation does not modify the deployment environment, when the conservation is invalid, and behaves as the standard interaction operation φ otherwise. Consequently, the deployment environment is never modified by invalid interactions and these *bogus* attempts are transparently rejected and never reach agents' perceptions.

The next step of our work is to offer to the deployment environment concrete solutions and tools to check if a conversation belongs or not to a certain protocol.

6 Conclusion

This paper has presented an approach in order to link conversation protocols to their corresponding coordination protocols. Hence, the conversation protocol is seen as structured sequences of messages of interaction objects recognised by the rewriting system. This rewriting system is considered as a conversation policy that can be integrated in a coordination architecture such as MIC* in order to validate conversations in a multi-agent system. Consequently, future works have to explore how to automatically and iteratively recognise valid conservations by observing messages among the coordinating entities. This will help in implementing reliable multi-agent systems where conversation laws and consequently coordination protocols are guaranteed by a trusted observer, namely the deployment environment, which established a referential for agents responsibility in large open and untrusted software systems.

References

1. GOUAICH Abdelkader. Requirements for achieving software agents autonomy and defining their responsibility. In *Autonomy Workshop at AAMAS 2003*, Melbourne, Australia, 2003.
2. Mihai Barbuceanu and Mark S. Fox. COOL: A language for describing coordination in multi-agent systems. In Victor Lesser, editor, *First International Conference on Multi-Agent Systems*, pages 17–24, San Francisco, California, 1995. AAAI Press/The MIT Press.
3. Mihai Barbuceanu and Wai-Kau Lo. Conversation oriented programming for agent interaction. In Dignum and Greaves [8], pages 220–234.
4. Luca Cardelli. Abstractions for mobile computation. *Secure Internet Programming*, pages 51–94, 1999.
5. C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. *Intelligent Agents: Theories, Architectures, and Languages*, 890:56–70, 1995.
6. R. Scott Cost, Ye Chen, Timothy W. Finin, Yannis Labrou, and Yun Peng. Using colored petri nets for conversation modeling. In Dignum and Greaves [8], pages 178–192.

7. Enrico Dente, Ricci Alessandro, and Rossella Rubino. Integrating and orchestrating services upon an agent coordination infrastructure. In Omicini et al. [23].
8. Frank Dignum and Mark Greaves, editors. *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Computer Science*. Springer, Lecture Notes in Computer Science, 2000.
9. Ulle Endriss, Lu Wenjin, Maudet Nicolas, and Stathis Kostas. Competent agents and customising protocols. In Omicini et al. [23].
10. Amal El Fallah-Seghrouchni, Serge Haddad, and Hamza Mazouzi. Protocol engineering for multi-agent interaction. In Francisco J. Garijo and Magnus Boman, editors, *MAAMAW*, volume 1647 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 1999.
11. Jaques Ferber. *Les Systemes Multi-Agents*. InterEditions, 1995.
12. Cedric Fournet. *Le Join-Calcul: Un Calcul Pour la Programmation Repartie et Mobile*. PhD thesis, Ecole Polytechnique, 1998.
13. Abdelkader GOUAICH, Yves GUIRAUD, and Fabien MICHEL. Mic*: An agent formal environment. 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), 7 2003. Orlando, USA.
14. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–271, June 1987.
15. Brian Henderson-Sellers, Giorgini Paolo, and Bresciani Paolo. Enhancing agent open with concepts used in the tropos methodology. In Omicini et al. [23].
16. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.
17. ITU. Recommendation z.120: Message sequence chart (MSC). Technical Report Z.120, International Telecommunication Union, 1993.
18. Ralf König. State-based modeling method for multiagent conversation protocols and decision activities. In Ryszard Kowalczyk, Jörg P. Müller, Huaglory Tianfield, and Rainer Unland, editors, *Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, volume 2592 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 2003.
19. Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, 26(1):87–119, 1994.
20. F. Von Martial. *Coordinating Plans of Autonomous Agents*, volume 610 of *Lecture Notes in AI*. Springer, Berlin, 1992.
21. Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
22. Andrea Omicini and Ricci Alessandro. Integrating organisation within a mas coordination infrastructure. In Omicini et al. [23].
23. Andrea Omicini, Paolo Petta, and Jeremy Pitt, editors. *Engineering Societies in the Agents World III*, volume 2577 of *LNAI*, 2003.
24. G.A. Papadopoulos. Models and technologies for the coordination of internet agents: A survey. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 2, pages 25–56. Springer-Verlag, 2000.
25. Milner Robin, Parrow Joachim, and Walker David. A calculus for mobile processes, parts 1 and 2. *Information and Computation*, 100(1), 1992.
26. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
27. Franco Zambonelli and H. Van Dyke Parunak. Signs of a revolution in computer science and software engineering. In *Agent Oriented Software Engineering Workshop at AAMAS 2002*, Bologna, 2002.