Coordination in Introspective Multi-Agent Systems

Yasmine Charif and Nicolas Sabouret
Université Pierre et Marie Curie-Paris6, UMR 7606, LIP6,
4 Place Jussieu, Paris, F-75005 France
firstname.lastname@lip6.fr

Abstract

This paper proposes a multi-agent system architecture based on introspective agents provided with an expressive communication language and reasoning capabilities on their skills. The agents’ introspective capabilities make it possible to agents to autonomously decompose a task according to their competencies. Using a protocol, the agents can then coordinate with other agents to as to collectively and dynamically cover the performance of a complex task with interrelationships and dependence among its actions. We describe in this paper the introspective agents’ structure and the coordination protocol that supports the distributed and decentralized task achievement.

1. Introduction

Multi-agent systems consist of a set of autonomous and interactive agents whose aim is to solve collectively a given problem or perform a given task. One of the most common approaches is to use agent communication languages and interaction protocols [2, 4]. However, existing protocols like [3, 10] assume that all tasks to perform are atomic. That is, each agent can either integrally execute it or not execute it at all. For instance, in the FIPA-CNP [3], a manager agent sends a call for proposal to participant agents who may propose to accept or refuse to perform the overall task.

On the contrary, the task can require agents to perform several sub-tasks with inter-dependencies between them (precedence, incompatibility, etc). This requires agents to decompose the overall task and coordinate with each other to properly cover its execution. Several solutions have been proposed to this purpose. One common approach is to ask the programmer to provide a description of the workflow [8]. This solution has a reduced cost. However, it leads to ad-hoc solutions that strongly depend on the application. Such a result sharing strategy requires to explicitly specify action relationships (e.g. the result of one agent’s action is needed by others to perform another action) [1]. Another adopted approach consists in using distributed planning methods [1, 5]. The user expresses its needs as a goal to the system. A manager agent builds a plan that can involve several actions allocated to different agents in the system. However, one cannot guarantee the availability of agents capable of achieving a subplan and several iterations of decomposing and distributing a task may be necessary before agents genuinely start to execute a plan. Moreover, in such techniques, no assumption is made on the way the agents will perform their subtask(s). This is precisely the issue addressed by our paper.

Indeed, there is a strong need for dynamic task decomposition and automatic and dynamic coordination among agents [7]. In our work, we pitch on an interactive approach based on agents introspection and dialogues. We assume that each agent is capable of reasoning on the set of actions it is able to perform. Thus, it can decompose a task according to its capabilities, and interact with other introspective agents to overcome its limitations.

Nevertheless, such intricate and introspective capabilities are not provided to agents developed using current development frameworks such as JADE\(^1\) or JACK\(^2\). Additionally, current agent communication languages (ACL) lack a mean to express questions or assertions about agents’ capabilities [6, 12]. Finally, current interaction protocols cannot take into account dependencies between subtasks originally from an overall task decomposition [9].

This is why we propose to design a multi-agent system (MAS) architecture in which agents are able to dynamically decompose a task according to their skills and coordinate with other agents so as to achieve the overall task in a consistent, distributed and decentralized way. We present in this paper the agent’s structure and describe the introspective capabilities making it possible for agents to decompose a task according to their actions. Then we describe the coordination protocol allowing agents to interact and collaborate so as to realize the required task in a decentralized way.

\(^1\)http://jade.tilab.com
2. Introspective Agents’ Structure

2.1. Overview

The MAS we aim to design relies on agents offering a set of functionalities or services (such as household appliance selling, money exchange, delivery, etc).

Each agent is able to know which actions it is able to perform and to reason on the task it is asked to achieve regarding to its actions. Thus, if it cannot solve a request by its own, it should be able to know if it can at least take part to its realization, and which information it requires to achieve it in case it is dependent on another sub-task.

As shown in figure 1, our agents are composed of three main layers:

1. The communication layer. It manages the messages transportation through an agent communication language (ACL) allowing to encompass expressive message contents about the agents’ capabilities.

2. The knowledge layer. It includes the agent database and set of actions. Moreover, it contains an ontology gathering the definition of the concepts and action names handled by the agent.

3. The request processing and protocol management layer where messages are processed according to the knowledge layer and other interaction parameters presented in section 4.

2.2. Agent Actions in the Knowledge Layer

Each agent action is defined by a tuple \((\text{name, } \mathcal{P}, \mathcal{E})\) where:

- \text{name} is the action’s name (e.g. deliver equipments, record TV programs, ...);
- \(\mathcal{P}\) is a set of preconditions;
- \(\mathcal{E}\) is a set of effects, \textit{i.e.} message sending and/or data modifications.

In our model, the introspection capabilities for automatic coordination of agents focus on preconditions only. We can outline two kinds of preconditions:

1. Event patterns that describe events, each of which is an event name and a set of fields, which trigger the action. When an agent receives an event, he tries to match it with one of its event patterns. If the events names matches but some fields are missing, the agent can compute this set of missing fields and return it to the sender. We later explain how this feature allows the dynamic task decomposition.

2. Guard condition which are boolean expressions that must be verified for the action to be executed. This makes it possible for agent to have proactive behaviours.

3. The Agent Communication Language

3.1. The Request Model

The starting point of the agents coordination is the user needs. These needs are formalised as a set \(\mathcal{R}\) of requests \(r \in \mathcal{R}\), each of which expresses a question about agent’s data (e.g. “what is the price of this camera”) or an action to perform given a set of parameters, \textit{i.e.} an event (e.g. “record the TV program broadcasted on channel 5 at 8.30 p.m.”).

These requests appear both in human-agent and in inter-agent dialogues.

A request is a pair \(r = (\alpha, \gamma)\) where \(\alpha\) is the request performative and \(\gamma\) is the request content. The request performative \(\alpha\) can take the following values:

- What-is for requests that represent questions about the agent’s data.
- Assert-is for assertions about agent’s data values (in answer to What-is requests).
- Order for requests that represent actions to perform.
- Ack for acknowledgment when an action has been performed (in answer to an Order request).
- Assert-cannot to express that the agent is unable to perform a command.
- Assert-can to express that the agent can perform a command, but misses a field value to be able to do so.
- Unknown for asserting that the agent doesn’t know a field or the overall action to perform.

3.2. Communication between Agents

The requests structure presented above is also used by agents in their communications. Indeed, these requests formalize the contents of the messages exchanged by agents.
each of which is structured as follows:

\[ m = [i, C, \text{sender}, \text{receiver}, \text{in-reply-to}] \]

Where \( i \) is the message id, \( C \) is the message content, \( \text{sender} \) and \( \text{receiver} \) are the sender and receiver AIDs (agents’ ids) and the in-reply-to slot is the thread of the message associated to the initial user request.

A message content \( C = \{r_k\}_{k \in [1,s]} \) is in fact a set of requests. This allows the agents not only to express a wide range of information, but also to process directly the demand of the user interacting with them.

4. Dynamic Agent Coordination

4.1. Requests’ Processing

The agents are provided with a Request Processing Module (RPM) which is in charge of building for questions on agent’s data (requests What-is) and commands (requests Order) their corresponding answers according to the agent skills following table 1. The respective request ranges processing is then as follows:\(^3\):

- If an agent received a What-is request, it builds in answer a request Assert-is if it actually contains the asked data in its knowledge layer. Otherwise, it answers by an Unknown request, either to refer that it doesn’t know the overall data, or one of the fields defining it.

- If it received an Order request, it builds in answer an acknowledgement Ack if it is able to perform the asked action; an Assert-cannot request otherwise; and an Assert-can request if it can perform the action but needs to do so to have some missing fields values.

Notice that only questions and orders are managed by the RPM. For all the other request ranges, the processing has to be carried out at the protocol level where other interaction parameters (such as previously received messages) are taken into account.

Since the content of a message that an agent receives is a set of requests \( \{r_k\}_{k \in [1,s]} \), the built answer will be a message which content is the set of the corresponding answers \( \{p_k\}_{k \in [1,s]} \). Each request \( r_k \) is first being processed by the RPM following table 1. Then if the range of the received request doesn’t belong to \{What-is(γ),Order(γ)\}, the processing is performed at the protocol level, as we will present it hereafter.

4.2. MAS Behaviors

In our coordination protocol, all the agents play the same role. As each agent takes part to the user’s requests resolution, the processing is distributed. As it does not depend on any specific agent (all agents play the same role), it is moreover decentralized.

The overall MAS behavior to provide a consistent result to the user is then as follows:

1. When an agent receives a triggering message, that is the message containing the user’s requests, it first processes each request composing it by using its RPM, following table 1.

2. When agents send their answers to each other, they most likely receive requests which are not questions nor commands (i.e. Unknown, Assert-can and Assert-is requests). Each agent then uses specific dialectical rules, described hereafter, for such requests processing.

3. As far as the message recipients are concerned, if all the requests composing a message have been solved, the result can then be sent to the user. Otherwise, the agent forwards its message answer to all its acquaintances except those for which it knows that it’s useless. We define hereunder through the dialectical rules how an agent assesses the set of recipient agents for messages sending.

The result provided to the user is then computed as follows:

- For all the requests that have been solved, i.e. answered by an assertion or an acknowledgment, the agent returns to the user the set of candidate answers lists \( \mathcal{L}(r_k) \) each of which is defined by:

\[ \mathcal{L}(r_k) \equiv \{(b_j, \rho_k)\}_{j \leq |\text{participants}|} \]

where \( b_j \) is a bidder agent AID, and \( \rho_k \) is the assertion or acknowledgment it sent for the triggering request \( r_k \). It means that for each triggering request, the user will receive the set of answers together with the AID of the agents which sent them.

- The agent acknowledges the user about the missing fields. For instance, an agent could return to the user that it didn’t specify the delivery address.

In our interaction model, every agent is provided with a history table where sent and received messages are stored and assigned an identifier associating them to the user requests that triggered them.

\(^3\)The reader can find further details about the RPM algorithm in [11].
4.3. Dialectical Rules

Agents make use of dialectical rules taking into account previous exchanged messages during the agent’s interaction to process Assert-can, Unknown, Assert-is and unsolved Order requests. These past messages are retrieved from the agent’s history table. Indeed, when an agent receives:

- an Assert-can request, it retrieves the order it originally sent and replaces the required field stated in the Assert-can answer by its value if owned.
- an Unknown request, and if it just concerns one of the fields defining the data originally asked in a What-is request, it looks in its database or in its history table if it owns or has been sent such a value, and sends again the original What-is request completed.
- an Order request, and if it misses a field value to perform it, as we previously mentioned, it builds an Assert-can answer. However, if the sender couldn’t acknowledge the recipient agent about the required field, the latter builds a request What-is to ask other agents about the needed information.

The mechanism described here is the one used to decompose a single task into several sub-tasks according to an agent skills. Suppose for instance the request “I would like to download .avi videos”. In our model, it can be dynamically solved with an .mpeg provider agent and .mpeg to .avi converter agent. The latter would ask for the .mpeg videos to convert through What-is request. The .mpeg provider agent when receiving this request can answer it, and the converter agent, when receiving the answer, can then finish the overall task achievement.

- an Assert-is request, and if it’s in answer to a What-is request that it built itself, it means it has to retrieve the order for which it sent the What-is request. It can then trigger it using the information received trough the Assert-isanwer.

5. Conclusion and Future Work

In this paper, we proposed a MAS architecture of introspective agents capable to coordinate themselves and perform collectively a complex task in a dynamic way by reasoning on their skills. These agents make use of both an ACL allowing to encompass expressive message contents, and a dynamic coordination protocol managing the dependences among the actions to perform, so as to provide a consistent result to the user.

In future work we need to consider several improvements in order to achieve greater scalability in the agents coordination. In particular, our interaction model has to be adapted so as to take into account global constraints specified by the user. Indeed, it manages requests like “I want to buy and deliver equipments” but we want it to take into account global constraints like in “I want that the equipments and delivery costs sum doesn’t exceed $300”.

References