Introducing Mobility into Agent Coordination Patterns

Sergio Esparcia[1] and Ichiro Satoh[2]
[1]*Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Valencia, Spain*
[2]*National Institute of Informatics, Chiyoda-ku, Tokyo, Japan*

Keywords: Mobile Agents, Coordination, Interaction, KQML.

Abstract: This paper proposes coordination patterns that support matchmaking, communication and interaction among mobile agents in addition to stationary ones. Mobile agent technology is a powerful implementation technique of distributed systems, but we need to manage migrations of agents, including their current and destination locations. The proposed patterns enable us to define coordination between mobile agents or between mobile and stationary agents without explicitly knowing their migrations between locations. They are mostly based on the Knowledge Query and Manipulation Language, one of the most extended Agent Communication Languages, but also new patterns are proposed. Additionally, a case of study about tourism is presented.

1 INTRODUCTION

Agent **coordination** is an important issue when some agents are required to work together towards a global goal (e.g., if these agents belong to a Multi-Agent System (MAS) (Ferber, 1999)). Commonly, agents have different abilities and capabilities, so they have to collaborate to achieve this global goal, which will satisfy also the individual goals of the agents. There exists the possibility that agents cannot achieve their goals if they are attached to a specific environment or machine (i.e., they are stationary). For this reason, **mobile agents** (Lange and Oshima, 1999) are implemented to be able to move around different machines and continue their execution there. Mobile agents have to be able to understand the languages and protocols of the destination machine. Mobility also brings new issues to tackle when dealing with coordination. For example, mobile agents may not know their next destinations, because they may not have the topology of the current network. Thus, the capabilities, services, and agents of reachable computers are unknown to them. Therefore, a matchmaking mechanism that enables mobile agents to discover other stationary or mobile agents and destinations is required.

To solve this problem, a matchmaking mechanism for stationary agents is extended with support to mobility of agents. **Agent communication languages** (ACL) (Labrou et al., 1999), such as the *Knowledge Query and Manipulation Language* (KQML) (Finin et al., 1994), present interaction patterns and performatives that focus on the matchmaking and communication between stationary agents that cannot change their position and already know the location of their communication partners. The KQML itself does not support mobility of agents, but it is useful for new visiting agents because since it is a well-known ACL by the community of agent researchers, their interaction patterns have became widely used not only in situations where KQML is used as the ACL employed by agents, but in different scenarios where other ACLs are used. Then, the KQML patterns are a suitable interaction and coordination mechanism to be extended with new patterns supporting mobile agents.

The objective of this paper is to define and present new interaction patterns for KQML that extend the original stationary features with new patterns that are intended to improve coordination and interaction between mobile agents. These patterns give mobile agents the capability of correctly managing their interactions, no matter that the agents move from their previous locations to a new ones, thus improving the performance of the agents and helping them to achieve not only their individual goals, but also when agents are required to fulfil a set of global goals.

The rest of this paper is structured as follows: Section 2 describes the background of this work. Section 3 presents the main contribution of this work, the patterns that support interaction and coordination between mobile agents. Section 4 presents a case of study based on tourism. Finally, Section 5 describes our conclusions on this topic and the future work.

Esparcia S. and Satoh I..
Introducing Mobility into Agent Coordination Patterns.
DOI: 10.5220/0004813401310138
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 131-138
ISBN: 978-989-758-016-1
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

2 BACKGROUND

This section defines the concepts that are part of the patterns that enhance KQML with mobility features. Agents and mobile agents are defined, as well as the KQML agent communication language, and the environment where our proposal is deployed.

Agents (Wooldridge and Jennings, 1995) are autonomous software entities that act representing users. These agents have functionalities in order to help the users they represent. Agents are able to be stationary or mobile, depending on whether they can move around different computers or not. Also, they are able to communicate with other agents. The agents used in this paper are defined as intelligent, autonomous, social, able to learn, and mobile.

A **mobile agent** (Lange and Oshima, 1999) is a process that can transport its state from one environment to another, with its data intact, and be capable of performing appropriately in the new environment the same activities as in the original one. Mobile agents decide when (e.g., to look for a better resource allocation, a less hostile environment to work, etc.) and where to move. When a mobile agent decides to move, it saves its own state, transports this saved state to the new host, and resumes execution from the saved state. This makes them a powerful technique for implementing distributed applications. Advantages of mobile agents are: computation bundles, parallel processing, dynamic adaptation, tolerance to network faults, flexible maintenance, and portability.

Knowledge Query and Manipulation Language (KQML) (Finin et al., 1994) is an ACL to support communication between intelligent agents. Each KQML message contains the action to be carried out (also known as the performative) as well as the parameters of this communication (e.g., sender, receiver, etc.). In this work, we propose a subset of KQML patterns that require a *facilitator*: *recommend*, *broker*, *subscribe*, *recruit*, and *forward*; and the *point-to-point communication* pattern, which does not require it.

In the recommend pattern, the service provider agent advertises one of its capabilities to the facilitator. Then, the client agent queries the facilitator about an agent that has to be able to develop the same capability. In this case, the facilitator will send the recommendation to the client agent, and finally this agent will send the query to the service provider. The broker pattern works in a similar way as the recommend pattern. That is, the service provider advertises one of its capabilities, and then the client queries the facilitator to act as a broker for the same capability. Then, the facilitator will ask the service provider and then collects its answer and sends it to the client agent. This is, the facilitator acts as a mediator in the communication process between client and service provider.

The subscribe pattern consists on the service provider querying the facilitator to subscribe to all the answers of a query. Then, the facilitator sends the client all the answers it receives to the query. The recruit pattern is similar to the broker pattern, but the service provider sends the answer to the client directly. Finally, in the forward pattern, the client sends a query to the facilitator, but asking it to forward the query to a specific agent, because the client agent knows a specific service provider agent, but for any reason it does not know its location.

For the definition of the **environment**, this proposal relies on the *Agents & Artifacts* conceptual framework (Ricci et al., 2007). This framework represents and structures the environment by means of an aggregation of *workspaces*. The workspace where an agent is located is considered as its local environment, and its granularity can be chosen by the system designer (e.g., a workspace could be a portion of a machine, the whole machine, or a network of machines), being possible to make a scalable system.

Workspaces provide a physical description of the environment, in a similar way as the real world is described. Each workspace has an absolute position inside the environment. Workspaces can be intersected and nested between them, features that allow an agent to be located in different workspaces at the same time, and to make the system extensible (adding new workspaces, merging or splitting existing workspaces, etc.). An agent is placed, at least, in one workspace, so this agent has a specific location within the environment. It is also possible for an agent to move inside a workspace if the workspace is big enough. In this case, the position of the agent will not be the position of the workspace it is located, but a position inside the workspace. The proposed approach assumes that each mobile agent knows the facilitator located in their current environment using a service discovery protocol via multicast communications. For example, each facilitator periodically issues heartbeat messages with its agent identifier so newly visiting agents receive the messages to know the facilitator that is available in their current locations.

The *A&A* framework also includes artifacts, entities located in the environment that are reactive (but not proactive), and provide functionalities to agents (that are able to use artifacts in order to achieve their objectives). Artifacts feature: (i) *observable properties*, which are able to be checked by agents but not directly modified by them; (ii) *operations*, which are functions that agents can execute; and (iii) *link operations*, similar to operations but they require another artifact to be completed. We rely on the use of artifacts instead of agents in situations where proactivity is not required, but it is necessary to be provided with entities providing specific functionalities.

3 COORDINATION PATTERNS FOR MOBILE AGENTS

Since mobile agents can migrate between computers, other agents and external systems that want to interact with them need mechanisms to locate their current positions. However, such mechanisms are complicated and costly, so that they should be supported by particular agents or services. Agents also should select their partners, which may be mobile or stationary, according to their functions. Therefore, we use agents corresponding to facilitators studied in the literature of agents (e.g., KQML), and distributed objects (e.g., CORBA (Vinoski, 1997)). However, existing approaches do not support mobility of agents. KQML provides a set of agent communication patterns that rely on stationary agents, so an agent knows the location of the agent it is communicating with, or the location of the facilitator. Therefore, agent mobility is an interesting feature to be supported.

Since this work focuses on agent mobility, the KQML patterns have been adapted to mobile agents, and this section presents the modification of well-known patterns. Most of the included patterns rely on a facilitator agent which helps in the communication process, but the point-to-point communication pattern is also included. Notice that it is also presented a pattern which is not included in the KQML specification, the footprint pattern, but it is also interesting when dealing with mobile agents.

These patterns represent basic communicative structures between agents featuring mobility properties. The patterns are able to be combined (e.g., by executing them sequentially), thus creating more complex patterns to deal with more complex problems and situations. The patterns are presented in a generic way in order to facilitate both, its combination and its usage in different frameworks that use different ACLs.

Since these patterns focus on basic communication structures, it can be seen that some entities participating in the patterns are not proactive, so they could be represented as artifacts instead of agents, at least from the point of view of one specific pattern. However, from the point of view of the global system (due to pattern combination) this entity could require to be proactive, so it has to be an agent. Therefore, these entities are represented as agents in this paper, giving system designers the final decision about representing them as agents or artifacts.

The following subsections describe the proposed patterns by means of a text description and for most of them a figure is provided. Figure 1 presents the legend of the graphical notation. The number written next to each message depicts the order number.

Agent
Artifact
Workspace

Figure 1: Legend of the graphical notation.

3.1 Point-to-Point Communication

In a *point-to-point communication* (Figure 2), agent A makes a query directly to agent B. Agent B knows where A is located because of the source point of the received message. However, if the location of agent A changes after it queried, at the moment agent B answers A's query, B would not know where to send the message, or it would send the message to an empty location. In order to solve this situation, A could send a message to B indicating its current location while A does not receive any answer from B. This solution could maybe overload the network, but it is required to avoid failures when sending the answer message.

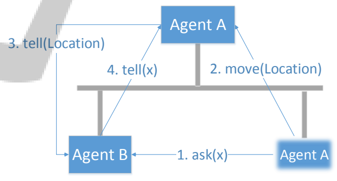

Figure 2: Point-to-point communication pattern.

3.2 Register Pattern

This pattern is used by agents who *join a new workspace* (Figure 3, left). The facilitator in charge of that environment has no information about the new agent and the latter has no information about the facilitator. Therefore, we use two approaches as service discovery protocols, e.g., UPnP and Jini (Allard et al., 2003). Using UPnP, the new agent informs its profile information (identifier, functionalities, and so on) to the facilitator by using a multicast communication protocol (Banavar et al., 1999) e.g., UDP multicasting. If the workspace is big enough, the new agent has to include its location inside its profile information. With Jini, the facilitator periodically sends messages with its address within the workspaces that it supports through a multicast protocol. When the new agent receives the message returns its profile information to the facilitator specified in the address of the message.

3.3 Unregister Pattern

The *unregister pattern* is the inverse version of the register pattern. It has to be used by an agent when it is intended to leave the workspace that it is populating. Unregistering from a workspace means to cancel all the pending activities with the facilitator of that workspace. Depending on how the facilitator is implemented, it could maintain the agent data inside its *Knowledge Base* (KB), since this information could be interesting for solving future queries.

However, a problem arises with this pattern if a message is sent to agents that unregistered from the environment, because the message will not arrive to its addressee agent. This problem is handled with the use of the footprint pattern (see subsection 3.11). If a message arrives to an agent which has unregistered from the workspace, it will be forwarded by the footprint artifact to the current location of the agent.

3.4 Transport Address Pattern

An agent can also move around its current workspace (i.e., its local environment) if its size and architecture allows to do so (Figure 3, right). When an agent moves inside a workspace, it has to communicate its new location in the workspace to the facilitator using the *transport address performative*.

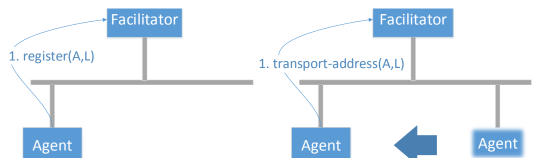

Figure 3: Register pattern (left); and Transport address pattern (right).

3.5 Broker Pattern, Mobile Service Provider Agent

The *broker pattern* (Figure 4, left) is used when the client agent wants the facilitator to tell it an appropriate agent who could be able to provide a solution for its query. In this pattern, client and provider do not communicate to each other directly. Therefore, the client is not informed about the position of the service provider since it does not require the position for communication purposes. For the service provider agent it is required to send updated information about its location to the facilitator because the facilitator has to be able to locate the service provider at any time, to answer the query of the client.

3.5.1 Fetch Pattern

This is a broker situation (Figure 4, right) where the client asks the facilitator to *fetch* himself with a service provider agent to help it to achieve a goal. Before, the service provider agent had advertised that it is ready to fetch with another agent. Having received messages from the two agents, the facilitator sends to the service provider agent the information of the client who asked for fetching. Finally, the service provider agent moves to the location of the client.

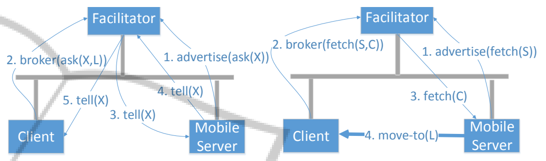

Figure 4: Broker pattern with mobile service provider agent (left); and Fetch pattern (right).

3.6 Broker Pattern, Mobile Client

Considering the *client to be mobile* is very similar as considering it as stationary. The only difference is that the client has to send its updated location to the facilitator, thus assuring receiving the answer to the query associated to the broker performative.

3.7 Recommend Pattern, Mobile Service Provider Agent

In the *recommend pattern* (Figure 5) client and service provider agents communicate in a direct way. First, the service provider will advertise to the facilitator the performative it is able to answer, as well as its location. The client will then ask for a recommendation to the facilitator including the query to be solved. The facilitator sends the client the data of the service provider agent, including its location, and then the client directly asks the server provider. Finally, the server tells its answer to the client. In this case, the service provider agent has to send updated information about its location to the facilitator.

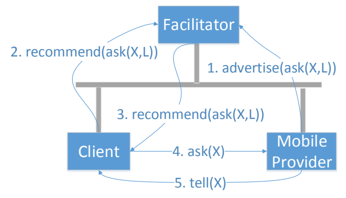

Figure 5: Recommend pattern with mobile service provider agent.

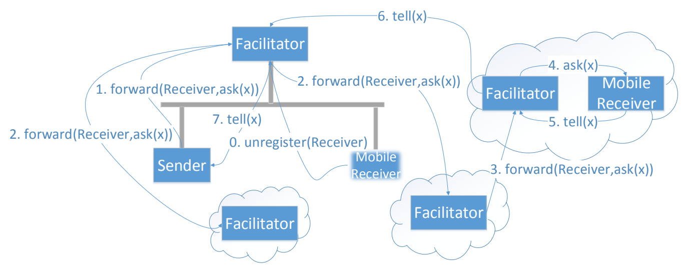

Figure 6: Forward pattern with mobile receiver (only to selected facilitators).

3.8 Recommend Pattern for Destination, Mobile Client

This case is similar to the case of the recommendation pattern for a mobile service provider agent. The difference here is that the content of the query made by the client is the specific location to get a requirement. In this case, the facilitator recommends the service provider to the client, and sends not only the name or identifier, but the location of this service provider. Therefore, after receiving this recommendation, the client makes use of its mobility features and moves where the service provider is located to get the requirement (resource, information, etc.) that it queried.

3.9 Forward Pattern, Mobile Receiver

In the *forward performative*, the sender of the message knows about the existence of a receiver that is able to fulfil the query it intends to solve, but it does not know the current location of the receiver, so the use of a facilitator is required. Nevertheless, since the receiver is mobile, it is possible to move to a different workspace. When this situation occurs, then the facilitator has to send a broadcast performative to ask the facilitators from other workspaces. Since all facilitators are stationary, they know the location of other facilitators, so they are able to communicate between them. Each facilitator will look around its workspace for the receiver agent. Once located, the receiver agent will answer the query, and the two facilitators will bring the response to the sender.

A more efficient solution for forwarding is to make a multicast, sending the message to a limited number of agents (Figure 6). The facilitators to send the multicast are chosen by proximity or by using the previous knowledge about the type of agents that populate the different environments. If the receiver agent is not located inside any of the environments of these facilitators, then they will forward the message to more facilitators until the receiver gets the message. Then, the facilitator in the environment of the receiver will send the answer to the environment of the client.

There is a third possibility for solving this problem, the use of a directory. A directory contains the name and/or the identifier of an agent, and the current location of the agent. Therefore, the facilitator will send the forward message to the facilitator of the correct environment, or will directly ask to the agent. This approach supposes new problems because in order to keep these directories up-to-date, facilitators have to exchange messages between them. However, it improves the broadcast approach, since the number of messages using the directory is smaller than the number of messages when broadcasting.

3.10 Forward Pattern, Mobile Sender

In the case of the *forward pattern with mobile sender*, the sender has to send the facilitator its updated location information. This is critical when the message is a query that requires an answer from the receiver. Therefore, if the information about location that the facilitator has inside its KB is not up-to-date, sending the answer back will fail.

3.11 Footprint Pattern

When an agent wants to send a message to another agent, it must know its current location. Therefore, a mechanism is needed for tracking a mobile receiver. Immediately before the receiver moves into another workspace, it creates and leaves a *footprint artifact* behind. This artifact has a more updated location of the receiver and receives messages on behalf of it.

Two approaches to find the receiver are supported: (i) each footprint artifact registers the latest location of the receiver (as an observable property), so after receiving a query, the footprint artifact (using reactivity) sends it directly to it (Figure 7); and (ii) a footprint artifact only stores the next location of the mobile receiver, so the queried footprint artifact has to query another footprint artifact, and so on, until the agent is located and the query is sent to it (Figure 8).

This pattern is not part of the KQML specification, but it is included here because of its relevance

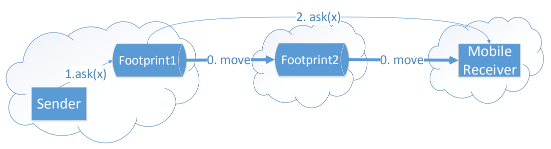

Figure 7: Footprint pattern (first approach).

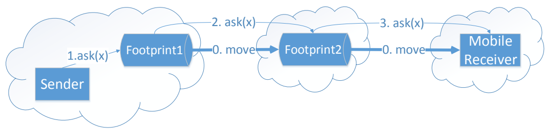

Figure 8: Footprint pattern (second approach).

on dealing with mobile agents. Differently from the forward pattern, using footprint artifacts reduces the number of messages that are required to be sent before finding the receiver, and avoids to use facilitator agents to find the receiver. However, this pattern does not substitute the use of forwarding, which is useful in other situations (e.g., if it is not allowed to place artifacts in the environment).

3.12 Subscribe Pattern, Mobile Service Provider Agent

The subscribe performative is used by a client to ask the facilitator about the truth of a statement. The facilitator waits until a confirmation of the statement is given by any agent of the workspace. However, this confirmation is not requested by the facilitator, so agents proactively inform the facilitator. The case of the *mobile service provider agent* (Figure 9, left) is slightly different from the case where this agent is stationary. The difference between both situations is that the service provider agent has to inform the facilitator about its movements, to make sure to be found by the facilitator when it needs to solve a query. However, in the case of subscription it is not a strong requirement because of the proactivity of the service provider agent when sending messages to the facilitator.

3.13 Subscribe Pattern, Mobile Client

Differently from the mobile service provider agent situation, in this case (Figure 9, right) the *mobility of the client* is a key aspect when an agent subscribes to a facilitator. It is important to notice about that because a client could send a subscribe performative to the facilitator, and then it could move to a different place. When a client agent moves from one workspace to another, it has to inform the facilitator, and to unsubscribe if it has any pending subscription queries. In the case that the client does not unsubscribe before

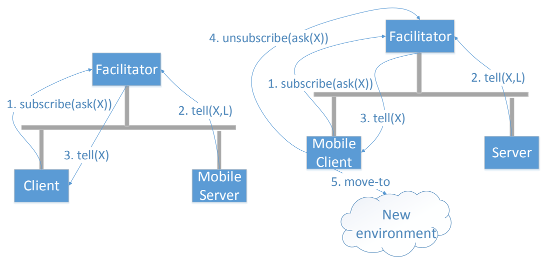

Figure 9: Subscribe pattern with: Mobile Service Provider Agent (left); and Mobile Client (right).

moving to a new workspace the network will be overflowed with messages from the facilitator while trying to locate the client. Therefore, it is an implementation decision to automatically unsubscribe if the agent leaves the workspace so as to keep the network load under control or to keep the subscription active until the client unsubscribes. This second option could be interesting if the agent has moved from the workspace only as a temporary situation and it plans to come back, so it wants to keep updated information coming from the workspace that it has temporarily left.

3.14 Recruit Pattern, Mobile Service Provider Agent

The *recruit pattern* (Figure 10, left) consists on a client sending a petition to the facilitator to find an agent who is able to find a solution for its query. Here, the answer to the query from the client is given directly by the service provider agent to the client. Mobility of the service provider agent is employed to assure the correct arrival of the information to the client. The service provider agent will move to the location of the client, thus delivering the answer to the query, and then it will come back to its initial position.

3.15 Recruit Pattern, Mobile Client

The main feature of the recruit pattern is that exists direct communication between the client and the service provider agent. Thus, mobility of any of the agents could suppose a failure in the process if the agents do not find each other. The case of a *mobile client* (Figure 10, right) is important because if it moves, the service provider agent would not be able to locate it. In order to avoid this problem to happen, when the client sends a recruiting performative the facilitator has to be updated with the current position of the client. When the facilitator sends the received query to the service provider agent, it also sends the current location of the client, thus being able for the service provider to send the message to the address where the

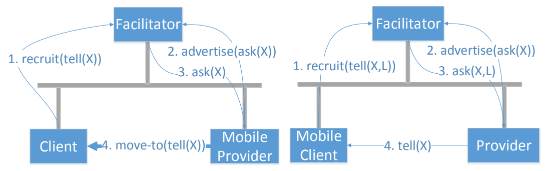

Figure 10: Recruit pattern with: Mobile Service Provider Agent (left); and Mobile Client (right).

client is located. In the case the client leaves the environment the recruit performative is canceled.

4 CASE OF STUDY: TOURISM

The **tourism market** is increasing its electronic business in the last years, with more products and services being offered online. However, the prices offered to the users of the tourism market are not stable, and they change due to the *fluctuation* of the demand at different moments. Additionally, *competence* between tourism providers is another factor that influences the prices, with sales and special discounts.

For this reason, clients are interested on finding the best available prices for the tourism products and services they require. Due to the aforementioned fluctuations on the prices, the users will spend so much time looking for the best available prices. Therefore, a tourism online market based on mobile agents is created. This market is open, including agents representing the clients, and agents representing the different tourism providers. A *client* agent has stored the preferences of the human client which it is representing (e.g., budget, preferred destinations, etc.) and its goal is to find a trip that satisfies these preferences. Each tourism *provider* agent offers products of a specific type (e.g., hotels, flights, train tickets, etc.). Clients and providers access the system in order to look for products and services (in the case of the clients), or for offering and publicizing them (in the case of providers). The environment of the system is distributed among different workspaces, one for each type of products or services. For example, there is a workspace for hotel reservations, another for flight tickets, etc. It is necessary for a client agent to move around different workspaces to get the products and services of the desired trip. Each workspace has its own *facilitator* that provides the functionalities described in the patterns. In case that the client agent was not able to move around different workspaces, it would be necessary to have more than one client agent per user, thus complicating the process of finding an appropriate trip, from both computational and temporal points of view. Therefore, using a mobile agent saves time and computational resources.

A scenario depicting the use of the **broker** and **forward** patterns in the tourism market is presented. Here, a mobile client agent has the goal of reserving a hotel and buying the flight tickets to the destination. In the example (Figure 11), there are two different workspaces (Flight and Hotel), where reservations for flights and hotels, respectively, are carried out. In each workspace, providers advertise their capabilities to the respective facilitators. The mobile client is located in the flight workspace, and queries the facilitator to act as a *broker* to find a proper flight. After making this query, the client *unregisters* from the flight workspace and *registers* into the hotel workspace (the register and unregister messages are not represented to improve the readability of the figure). Once in the hotel workspace, the client asks the hotel facilitator to work as a *broker* to find a hotel that matches its requirements. In this moment, there are two concurrent tasks (i.e., some messages have the same order number). On the one hand, the flight facilitator finds a flight provider that is offering flights that match the client expectations and gets a flight for the client. Then, the flight facilitator sends a *forward* message with the flight information to the hotel facilitator. On the other hand, the hotel facilitator sends the query of the mobile client to the hotel provider, which answers it back to the facilitator including the chosen hotel. Finally, the hotel facilitator sends the answer of both queries, flight and hotel, to the client, whose desired trip is now planned and reserved.

From the human point of view, this approach saves time and the agent is able to search in a wider scope than the human herself. From a computational view, having only one agent in control of reserving the trip facilitates the computation.

5 CONCLUSIONS AND FUTURE WORK

This paper has presented a proposal for introducing mobility concepts into the KQML interaction patterns, also including new patterns that support mobile agents. These patterns are aimed at improving coordination and interaction between mobile agents, since coordinating entities that are able to move around different environments supposes a challenge due to the uncertainty about the exact location of the agents participating in the communicative process. There are patterns that rely on a point-to-point communication, but most of the patterns rely on the existence of a facilitator that knows the agents populating its environment and is also able to communicate with facilita-

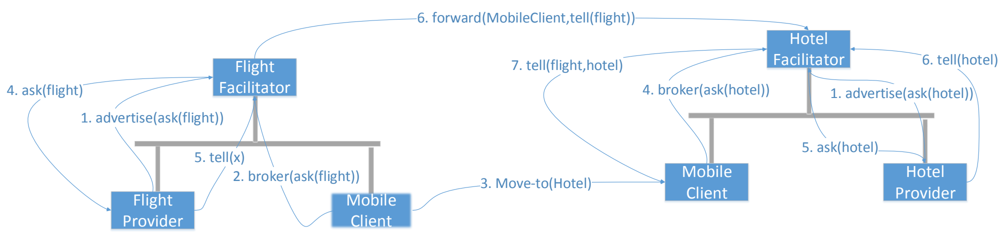

Figure 11: Example for the tourism market with a mobile client and two different workspaces.

tors in other environments, thus making possible for agents located in different environments to communicate through the facilitators. The coordination patterns proposed in this paper are also useful in the real world where agents, e.g., humans, can move between locations to work or meet others, like mobile agents.

As future work, the patterns presented here will be implemented in an agent platform that gives support to mobile agents, like MobileSpaces (Satoh, 2003), or to MAS, as THOMAS (Argente et al., 2011), along with other useful coordination patterns for mobile agents we found. Since patterns are defined in a generic way in this paper, they can be then expressed by means of the specific ACL of the framework where the patterns will be implemented in. We also plan to use the approach in physically mobile systems, and real societies whose people move between locations.

ACKNOWLEDGEMENTS

This work is supported by the NII International Internship Program, the MINECO/FEDER grant TIN2012-36586-C03-01, the TIN2009-13839-C03-01 project of the Spanish government, and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

REFERENCES

Allard, J., Chinta, V., Gundala, S., and Richard III, G. (2003). Jini meets upnp: an architecture for jini/upnp interoperability. In *Proc. Symposium on Applications and the Internet*, pages 268–275. IEEE.

Argente, E., Botti, V., Carrascosa, C., Giret, A., Julian, V., and Rebollo, M. (2011). An abstract architecture for virtual organizations: The thomas approach. *Knowledge and Information Systems*, 29(2):379–403.

Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R. E., and Sturman, D. C. (1999). An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. 19th IEEE Int. Conf. Distributed Computing Systems*, pages 262–272. IEEE.

Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). Kqml as an agent communication language. In *Proc. 3rd international conference on Information and knowledge management*, pages 456–463. ACM.

Labrou, Y., Finin, T., and Peng, Y. (1999). Agent communication languages: The current landscape. *Intelligent Systems and Their Applications, IEEE*, 14(2):45–52.

Lange, D. B. and Oshima, M. (1999). Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89.

Ricci, A., Viroli, M., and Omicini, A. (2007). Give agents their artifacts: the a&a approach for engineering working environments in mas. In *Proc. 6th int. conference on Autonomous agents and multiagent systems*, page 150. ACM.

Satoh, I. (2003). A testing framework for mobile computing software. *IEEE Transactions on Software Engineering*, 29(12):1112–1121.

Vinoski, S. (1997). Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35(2):46–55.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2):115–152.