Blackboard Rules for Coordinating Context-aware Applications in Mobile Ad Hoc Networks

Jacquet, Jean-Marie; Linden, Isabelle; Staicu, Mihail-Octavian

Published in:
EPTCS

DOI:
10.4204/EPTCS.91.5

Publication date:
2012

Blackboard Rules for Coordinating Context-aware Applications in Mobile Ad Hoc Networks

Jean-Marie Jacquet Isabelle Linden Mihail-Octavian Staicu
University of Namur
Namur, Belgium
jmj@info.fundp.ac.be isabelle.linden@fundp.ac.be msc@info.fundp.ac.be

Thanks to improvements in wireless communication technologies and increasing computing power in hand-held devices, mobile ad hoc networks are becoming an ever-more present reality. Coordination languages are expected to become important means in supporting this type of interaction. To this extent we argue the interest of the Bach coordination language as a middleware that can handle and react to context changes as well as cope with unpredictable physical interruptions that occur in opportunistic network connections. More concretely, our proposal is based on blackboard rules that model declaratively the actions to be taken once the blackboard content reaches a predefined state, but also that manage the engagement and disengagement of hosts and transient sharing of blackboards. The idea of reactiveness has already been introduced in previous work, but as will be appreciated by the reader, this article presents a new perspective, more focused on a declarative setting.

1 Introduction

On the front-line of current research areas, mobility related technologies are being developed in an exponential manner as a natural response to our information needs, regardless of space or time. Such information is mostly related to the context in which the user finds himself and usually dependent on his current needs. In the field of mobility, applications must cope with constant context changes in order to provide relevant information on demand. With current development strongly tackling the mobile connectivity domain, we can understand why the seminal equation

$$programming = coordination + computation \text{ [7]}$$

remains of great relevance. *Coordination* is thus an equal partner in the *programming* process as *computation*. This is even an essential element in the fields of distributed systems and concurrency. To be as generic as possible, coordination acts as a middleware layer between a consumer and a producer. It must place a demand to an entity able to solve it and afterwards ensures that an answer is returned. Departing from these two ideas we can see how *mobility* and *coordination* are key concepts for which proper mechanisms of interaction are required. As such, our objectives in this paper are to tackle two important needs: supporting context awareness and solving mobility issues in ad hoc networks related to the unpredictability of their connections and topology. We propose a solution in the form of *rules* spanning over one or more device data spaces (henceforth *blackboards*):

$$in(a,X),\ nin(b,Y) \ \longrightarrow\ in(c,W),\ nin(d,Z) \tag{1}$$

with the intuitive meaning that as long as some information X is present on a blackboard a and some information Y is not present on blackboard b, then we can assume that some information W will become available on blackboard c and some information Z will no longer be available on blackboard d.

Natallia Kokash and António Ravara (Eds.): 11th International Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA'12)
EPTCS 91, 2012, pp. 63–78, doi:10.4204/EPTCS.91.5

By providing two operational readings of such declarative rules, we aim at obtaining both of these needed functionalities. First, by means of a *forward reading* (inferred from applying the rule from left to right), we model the reaction to context changes occurring on the blackboards of mobile devices. Intuitively, departing from the previous rule, a blackboard context is defined in terms of information being present and/or absent and can model a user's specific need. For example, the rule

$$in(a, \langle location,\ Brussels \rangle),\ nin(a, \langle weather,\ sunny \rangle) \quad \longrightarrow \quad in(a, \langle list,\ museums \rangle)$$

states that if a tourist is in Brussels and the weather is not sunny, he would like to receive a list of available museums. Second, through a *backward reading* (inferred from using the rule from right to left), we represent the links between blackboards that would model the connection state of two devices. Let us assume the same tourist who is visiting Brussels, owning a mobile device, throughout the city info posts can make available a broad range of information consisting of weather data, nearby parking places, public transportation, etc. As he walks around the city, his mobile device roams between different such info posts and eventually connects to these when coming within connection range, which can depend on the wireless technology used for the connection. By exploiting the events raised at the networking level, a backward reading rule of the form $in(b,X) \quad \longrightarrow \quad in(a,X)$ is generated on the tourist's blackboard in order to define the connection state of his mobile device (containing the blackboard a) and the info post (containing the blackboard b). This rule acts as a logical link and allows for the transient sharing of the two blackboards, without the need to physically transfer the information between the blackboards. In conjunction with this rule, the previous forward reading rule can be resolved since the info post will provide the needed information about the weather.

The rest of this paper is organized as follows. Section 2 presents the concept of multiple blackboard coordination through rules. Section 3 shows an operational implementation of the rules on the Bach language. Section 4 presents some technical and architecture details of our mobile implementation. Section 5 presents related work in the field of mobile coordination and how our work relates to it. Section 6 presents some related work in terms of reactivity and how our proposal is positioned with respect to these. Section 7 draws the conclusions and presents the expectations for future work.

2 Blackboard coordination through rules

Reactivity in coordination languages relies on the basic idea of triggering an event when a predefined condition is met. The purpose is to enrich the capabilities of classical tuple spaces in order to transform them from simple tuple containers into context-aware entities, responsive to the interactions performed by the agents. Extensions have been explored, either by defining more complex triggering conditions or by taking a more complex set of actions.

Our approach, based on the principle of reactivity, is used to model the behavior of the system due to context changes on the tuple space and to handle the execution of mobile agents in scenarios that involve physical mobility of the hosts. Following the ideas exposed in LIME [15], when two or more devices come into each-other's range, their tuple spaces are merged in terms of information that is publicly available. A hard to grasp notion is the one of *"close enough"* for two devices to be in range and thus to be connected. Likewise, the idea of merging blackboards, although intuitive, is actually not a clearcut one. Let us address these two points in turn. On the one hand, closeness in practical terms is to be defined by the presence of a physical communication link, be it wired or wireless, that would enable a communication channel between the devices. From an abstract perspective, modeling such a notion is very difficult to express; however our solution benefits from the events raised by the operating system of the device and captured by the middleware. On the other hand, merging the blackboards should not be seen straightforward in the sense of transferring tuples over the network, but by providing *logical links*, as it was defined in [3]. These are in essence pointers that indicate the merging state with other blackboards. Taking a declarative approach, having a local blackboard merged with a remote one means that information is in some way available locally, assumed from the presence or absence on other blackboards.

For example, the rule $in(b_1,X) \longrightarrow in(b_2,X)$ states in a forward reading that any tuple X present on b_1 should also be considered present on b_2. By applying a backward reading we obtain a similar semantic which asserts that X is to be considered present on b_2 for as long as X is present on b_1. The common ground between the two readings is represented by the two expressions that comprise the rule, with respect to the $\longrightarrow$: the *left-hand side* (henceforth LHS) represents the rule's activation condition and the *right-hand side* (henceforth RHS) represents the effect produced by the rule's activation. To make such a rule more suitable for coping with context-awareness and transient sharing, we develop the LHS and the RHS to a more generic form. To this end, we extend the declarative equation 1 into the more general expression represented in equation 2. The context is represented by a sequence of template tuples that should be present on (by use of the *in* primitive) or absent from (by use of *nin* primitive) the blackboards. In the same way, the RHS can define the tuples that should be considered present or absent on several blackboards. There is no restriction for the blackboards defined in the LHS and the RHS in the sense that they don't necessarily need to be the same. The more general formal representation of the declarative rules is presented below:

$$\begin{aligned} in(b_1,t_1),\cdots,in(b_m,t_m),nin(b_{m+1},t_{m+1}),\cdots,nin(b_n,t_n) \\ \longrightarrow \; in(b_{n+1},t_{n+1}),\cdots,in(b_p,t_p),nin(b_{p+1},t_{p+1}),\cdots,nin(b_q,t_q) \end{aligned} \tag{2}$$

The meaning of this rule is that the presence of the tuples $t_1,\cdots,t_m$ on the blackboards $b_1,\cdots,b_m$ respectively and the absence of the tuples $t_{m+1},\cdots,t_n$ on the blackboards $b_{m+1},\cdots,b_n$ respectively, implies the presence of tuples $t_{n+1},\cdots,t_p$ on the blackboards $b_{n+1},\cdots,b_p$ respectively and the absence of tuples $t_{p+1},\cdots,t_q$ on the blackboards $b_{p+1},\cdots,b_q$ respectively. In the scenario of opportunistic mobile ad hoc networks, the operational translation of the LHS requires the acquiring of a lock on distributed blackboards, which is not feasible due to the unpredictability of the connections. As we will see in section 3.2, the solution amounts to restricting a rule's activation condition to a single blackboard.

The implicit behaviour of a rule upon activation is that information is consumed from the LHS in order to obtain the effect defined in the RHS. One important remark to be made is that for the *nin* primitive the destructive behaviour actually results in producing the template tuple on the respective blackboard. However, when using the rules to represent the transient sharing between blackboards such destructive behaviour is not desired. In order to preserve the characteristics of a logical link, the content of the blackboards defined in the rule must not be altered. To this extent we introduce a semantic notation in the form of square brackets surrounding the primitives *in* or *nin*, obtaining the guarded variants $[in(b,t)]$ and $[nin(b,t)]$. In order to fine tune the effect of a rule's activation, both the LHS and the RHS can have sets of simple primitives or guarded ones.

By providing two operational readings of equation 2 it is possible to obtain two different functionalities: *(i)* context awareness, by means of a *forward reading* and *(ii)* coordination in mobile ad hoc networks, by means of a *backward reading*. The detailed description of each reading will be provided in the next sections.

Note that other work, such as TuCSoN [19] or LIME [14], incorporate *IF pattern THEN actions* mechanisms. However, as we will show in section 3, our declarative approach allows to have a more refined view. For the moment, let us simply stress that in contrast to such an operational mechanism, we provide two multi directional views out of a declarative implication.

3 Language design

Having presented the declarative definition of the blackboard rules in section 1, we dedicate this section to describe how they can be incorporated in the Bach coordination language. With this aim, the language is first briefly presented in subsection 3.1. In subsection 3.2 we expand the idea of *rule reading* in order to model different functionalities and explain the constraints imposed by MANETs. We conclude with subsection 3.4 by presenting examples for each of the two readings. A section on formal operational semantics has also been developed, but due to page limitations could not be introduced in this version of the paper. The interested reader can refer to the full version of the paper which is available online [1].

3.1 Bach

Introduced in [9], the Bach coordination language is built upon the principle of a central *blackboard* (the equivalent of Linda's *tuple space*) represented by a shared memory space through which agents can communicate. The interaction with the blackboard is ensured by the use of four primitives: *tell* for outputting information on the blackboard, *ask* for querying the presence of information, *get* for retrieving information and *nask* for querying the absence of information. In order to express more complex actions, the four primitives can be linked by composition operators, namely: ";" for sequential execution, "||" for parallel execution and "+" for nondeterministic choice execution.

We improve the language definition by adding a set of four similar primitives for handling the rules over a network:

$$tellr(b,r) \qquad askr(b,r) \qquad getr(b,r) \qquad naskr(b,r)$$

having the respective semantics of adding a rule r to a blackboard b, querying the presence, retrieving, querying the absence.

3.2 Forward and backward reading

As illustrated by other declarative languages (Prolog [23], Haskell [24], etc.), the declarative reading of the blackboard rules must be completed by an operational one. To that aim let us now present in detail the two readings of the blackboard rules introduced in equation 2.

Forward reading ($LHS \longrightarrow_f RHS$). A first reading, named subsequently *forward reading*, is obtained by reading the general rule expression from left to right and by acting accordingly: provided that the condition in the LHS is met, then for each possibility in which the condition can be expressed from the blackboard's context the statements in the RHS are made true. Operationally, this means the following:

- For each tuple t_i $(1 \le i \le m)$ added on b_i and for any tuple t_j not present on b_j $(m+1 \le j \le n)$, then for any set of tuples $(t_1, \cdots, t_m)$ from the set of blackboards $(b_1, \cdots, b_m)$, the corresponding tuples t_k $(n+1 \le k \le p)$ should be created on b_k $(n+1 \le k \le p)$ and the corresponding tuples t_l should be removed from b_l $(p+1 \le l \le q)$.
- For each tuple t_i $(1 \le i \le m)$ removed from b_i and for any tuple t_j not present on b_j $(m+1 \le j \le n)$, then for any set of tuples $(t_1, \cdots, t_m)$ from the set of blackboards $(b_1, \cdots, b_m)$, the corresponding tuples t_k $(n+1 \le k \le p)$ should be created on b_k $(n+1 \le k \le p)$ and the corresponding tuples t_l should be removed from b_l $(p+1 \le l \le q)$.

[1] http://info.fundp.ac.be/~msc/articles/FOCLASA_2012_full.pdf

The activation condition of a rule represents a context defined on multiple blackboards. However, a full implementation of that general rule would involve a costly mechanism and heavy network loading in order to check whether the condition becomes active or not. Furthermore, in mobile ad hoc networks the communication links between devices are unstable and unpredictable. In this hypothesis, the wise choice is to restrict the context definition to only one blackboard. As such, the operational forward reading is represented as:

$$in(b_1,t_1),\cdots,in(b_1,t_m),nin(b_1,t_{m+1}),\cdots,nin(b_1,t_n)$$
$$\longrightarrow_f \quad in(b_{n+1},t_{n+1}),\cdots,in(b_p,t_p),nin(b_{p+1},t_{p+1}),\cdots,nin(b_q,t_q)$$

Backward reading ($LHS \longrightarrow_b RHS$). A second reading is obtained in a backward fashion by reading the general rule from right to left. The behavior is also different from the forward reading since as long as the conditions in the LHS are met, the statements in the RHS are verified. As such,

- The presence of t_k on b_k $(n+1 \leq k \leq p)$ can be deduced from the presence of a set of tuples $(t_1,\cdots,t_m)$ on the set of blackboards $(b_1,\cdots,b_m)$ and the absence of a set of tuples $(t_{m+1},\cdots,t_n)$ on the set of blackboards $(b_{m+1},\cdots,b_n)$.
- The absence of t_l from b_l $(p+1 \leq k \leq q)$ can be deduced from the presence of a set of tuples $(t_1,\cdots,t_m)$ on the set of blackboards $(b_1,\cdots,b_m)$ and the absence of a set of tuples $(t_{m+1},\cdots,t_n)$ on the set of blackboards $(b_{m+1},\cdots,b_n)$.

For the backward reading, restrictions must be imposed in the RHS as well: only one *in* or *nin* primitive is permitted. The reason is that the presence or absence of a tuple on a local blackboard can be inferred from the context of another blackboard, thus providing pointers to virtual tuples that can be used in the evaluation of *ask*, *get*, *nask* primitives. To this extent, the operational backward reading has two possible forms:

$$in(b_1,t_1),\cdots,in(b_1,t_m),nin(b_1,t_{m+1}),\cdots,nin(b_1,t_n) \quad \longrightarrow_b \quad in(b_2,t_{n+1})$$
$$in(b_1,t_1),\cdots,in(b_1,t_m),nin(b_1,t_{m+1}),\cdots,nin(b_1,t_n) \quad \longrightarrow_b \quad nin(b_2,t_{n+1})$$

3.3 Refinements

It may be possible at a given time for two or more rules to become active at the same time. The order in which the rules are handled is nondeterministic and the handling is an atomic operation. As such, handling a rule may lead to data consumption which could render inactive some of the other previously active rules. By default, the *in* and *nin* have a destructive behavior: once the rule activated, *in* consumes information and *nin* produces information. Since the interest of the backward reading is to provide logical links between blackboards it may not be desired to consume existing information or produce new one. To this aim we introduce a semantic notation in the form of *square brackets* surrounding the *in* and *nin* primitives in order to inhibit their *destructive behavior*. As such, [*in*] will not destroy information and [*nin*] will not produce new information once the rule is activated. More details will be provided in the following section.

3.4 Examples

By exploiting the expressiveness of the two operational readings, we can define special operations that would model the way in which blackboards are connected to each-other and the way in which they react to given contexts. Let us now present some practical uses of the rules in the context of distributed coordination.

The *forward rule*, defined as

$$forward(b_1, b_2) \equiv \{in(b_1, X) \longrightarrow_f in(b_2, X)\},$$

would redirect the tuples destined for b_1 to b_2. This is particularly interesting in scenarios of automatic information sharing between devices that enter each other's communication range. Other uses may include load balancing or for memory management. We can imagine for example that if a blackboard has a strict memory quota limit it could use the forward rule to bypass tuples to a remote blackboard in order to prevent overflow. A variant can be represented as a *copy rule* by simply adding brackets in the LHS:

$$copy(b_1, b_2) \equiv \{[in(b_1, X)] \longrightarrow_f in(b_2, X)\}.$$

Every new tuple arriving on b_1 will be transmitted to b_2 without removal from b_1. Used in TCP/IP, the broadcast or multicast functionalities can also be modeled with the help of the following rule:

$$broadcast(b, b_1, \cdots, b_n) \equiv \{in(b, X) \longrightarrow_f in(b_1, X), \cdots, in(b_n, X)\}.$$

The converse of a broadcast rule would be the merger one, with the purpose of collecting all the tuples from several blackboards to a single one, with particular interest in the scenarios involving sensor networks:

$$merge(b_1, \cdots, b_n, b) \equiv \{in(b_1, X) \longrightarrow_f in(b, X), \cdots, in(b_n, X) \longrightarrow_f in(b, X)\}.$$

The *inherit rule*, defined as

$$inherit(b_1, b_2) \equiv \{[in(b_1, X)] \longrightarrow_b [in(b_2, X)]\},$$

states in a direct form that b_1 inherits b_2 or b_1 has access to all the tuples present on b_2. This type of rule enables a logical link between two blackboards and defines the connection between two mobile devices. By adding inheritance to the forward rule we allow the blackboard that is forwarding the tuples to also keep a pointer in case it ever needs to access those tuples again. As such, the enriched forward rule is:

$$forward(b_1, b_2) \equiv \{in(b_1, X) \longrightarrow_f in(b_2, X), [in(b_1, X)] \longrightarrow_b [in(b_2, X)]\}$$

It should be noted that the *inherit rule* must have the primitives in the LHS and RHS guarded by square brackets in order to keep the significance of a logical link and not to change the contents of the blackboards.

4 Implementation

4.1 General principles

Being a language designed for distributed systems, Bach has been implemented in a client-server fashion, but only in terms of the architecture. Each device acts independently and is responsible for establishing connections with neighboring devices. There are no devices designated as central managers for transactions. No hypothesis is made on a-priori knowledge of the network architecture. The server-side and the client-side represent just a separation of concepts and tasks that are to be performed on each device.

The server-side component is responsible for handling the communication, the blackboard and its operations, a rule space for containing the reaction rules associated with the blackboard, a request space for the agents that need to be processed and a solved request space for storing the results of the agents execution.

The client-side handles the parsing of string representations of agents, dispatches requests to a local or remote blackboard depending on needs and receives the replies. Parsing an agent returns a tree-like structure with the nodes consisting of the composition operators that link the primitives, which are stored in the leaves. The processing of the tree begins at the root node with the recursive creation of sub-agents until the leaves are reached, moment at which requests are formed towards the server.

Our purpose being to provide a fully functional framework for mobile devices, that would support all the features of the proposed coordination language, we found Python a good choice of programming language, since it ensures easy portability of the code on different mobile platforms: Symbian, iOS, Android.

4.2 Implementation techniques

The blackboard rules are implemented as a structure comprised of two ordered arrays, one for the LHS and one for the RHS, and a boolean variable depicting whether the rule is a forward reading, when true, or a backward reading, when false.

For defining the activation condition of a rule it is not mandatory to have a context composed solely of different tuples. Depending on the needs, several instances of the same tuple may be required. This would translate by placing a sequence of $in(b,t)$ primitives in the condition. In order to avoid writing repeatedly one *in* primitive for each needed instance of the tuple t we provide a small extension allowing to add an index to specify the total number of instances. Subsequently, $in_c(b,t)$ states that a total count of c instances of tuple t are required on blackboard b. Testing the absence of a tuple amounts to having 0 instances of the blackboard. Hence, no counters are needed for the *nin* primitives. By complementing the above definitions with this notation we obtain the following general rules:

$$in_{c_1}(b_1,t_1),\cdots,in_{c_m}(b_1,t_m),nin(b_1,t_{m+1}),\cdots,nin(b_1,t_n)$$
$$\longrightarrow_f \quad in_{c_{n+1}}(b_{n+1},t_{n+1}),\cdots,in_{c_p}(b_p,t_p),nin(b_{p+1},t_{p+1}),\cdots,nin(b_q,t_q)$$
$$in_{c_1}(b_1,t_1),\cdots,in_{c_m}(b_1,t_m),nin(b_1,t_{m+1}),\cdots,nin(b_1,t_n)$$
$$\longrightarrow_b \quad in(b_2,t_{n+1})$$
$$in_{c_1}(b_1,t_1),\cdots,in_{c_m}(b_1,t_m),nin(b_1,t_{m+1}),\cdots,nin(b_1,t_n)$$
$$\longrightarrow_b \quad nin(b_2,t_{n+1})$$

A rule's activation condition is defined by the minimum number of tuple instances specified in the *in* primitives and by the absence of tuples specified in the *nin* primitives. Concretely, rules are handled through an *activation vector* which has the generic form:

$$activation\ vector = [c_1,\cdots c_m,0,\cdots,0]$$

At the same time, each rule must keep track of the changes occurring on the blackboard. With the execution of $tell$ or get primitives, tuples are respectively added or removed. By keeping track of the tuples that transit the blackboard it is possible to determine the moment when the context is met for the rule to become active. To this purpose, a separate vector must be used in order to count the number of tuple instances present on the blackboard and which match the tuples defined in a rule's LHS. More specifically, *blackboard vectors* are of the form:

$$blackboard\ vector = [bc_{t_1},\cdots,bc_{t_m},bc_{t_{m+1}},\cdots,bc_{t_n}]$$

where bc_{t_i} $(1 \leq i \leq m)$ represents the counter for the number of tuples present on the blackboard that match the tuples t_i of the *in* primitives and bc_{t_j} $(m+1 \leq j \leq n)$ represents the counter for the number of tuples present on the blackboard that match the tuples t_j of the *nin* primitives. This incremental style of computing allows to observe only the modifications that occur on the blackboard. So instead of comparing the entire LHS of a rule with the entire contents of the blackboard, only the tuple that was used in the *tell* or *get* is the subject of this comparison.

By using these two definitions it is possible to formally define a rule's activation condition:

$$bc_{t_k} \geq c_k,\ k = 1 \ldots m \quad and \quad bc_{t_k} = 0,\ k = m+1 \ldots n$$

It is worth to notice that in the case of the backward reading the rule stays active as long as the above condition is met, while in the case of the forward reading the rule acts as a reaction rule by producing or deleting tuples as depicted by the RHS. Taking into account the definitions presented at the beginning of this section we observe that if the activation context may be obtained in several ways from the tuples of the blackboard, then the forward reading rule can be executed several times. In fact, this number is determined by the total number of combinations in which the context could be obtained from the blackboard's content:

$$\prod_{k=1}^{m} \binom{bc_{t_k}}{c_k}$$

This is of course a maximum number, depending on the usage of brackets on the *in* and *nin* primitives, because if tuples are consumed or produced on the blackboard due to the rule execution this may deactivate the condition of the LHS.

4.3 Examples

Forward reading example: Let us consider two blackboards, b_1 and b_2, initially empty of tuples and the forward rule on blackboard b_1:

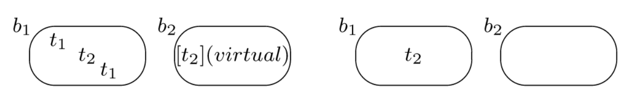
$$in_2(b_1,t_1),\ [in(b_1,t_2)],\ nin(b_1,t_3) \longrightarrow_f in(b_2,t_2)$$

which states that if at least two instances of tuple t_1, one instance of tuple t_2 and no instance of t_3 are present on blackboard b_1, then one instance of tuple t_2 will be created on blackboard b_2. For this rule the activation vector is $[2,1,0]$. Given that b_1 is empty, the blackboard counter for this rule is initially $[0,0,0]$. Assuming a set of primitives executed in the sequence depicted in figure 1(a), we obtain after step 5 a state of the blackboard vector that enables the rule.

By applying the combinatorial formula, we obtain a number of three possibilities in which the rule can be fired. However, after the first execution, two instances of the tuple t_1 will be consumed from b_1 and one instance of the tuple t_3 will be produced on b_1. In the meantime, due to the RHS of the rule, one tuple t_2 is added to b_2. After this execution the state of the two blackboards is as presented in figure 1(b) with the blackboard vector having the values $[1,1,1]$, which implies the deactivation of the rule.

Step	Primitives	Counter vector	Comments
1	$tell(b_1,t_1)$	[1, 0, 0]	
2	$tell(b_1,t_3)$	[1, 0, 1]	
3	$tell(b_1,t_1)$	[2, 0, 1]	
4	$tell(b_1,t_2)$	[2, 1, 1]	
5	$get(b_1,t_3)$	[2, 1, 0]	**Rule is active**

(a) The evolution of the counter vector



(b) The blackboards before and after executing the rule

Figure 1: Forward reading example

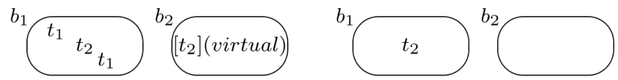
Backward reading example: As with the forward reading, we use the same two initially empty blackboards, b_1 and b_2 and associate a backward reading rule on blackboard b_2:

$$in_2(b_1,t_1),\ [in(b_1,t_2)] \longrightarrow_b [in(b_2,t_2)]$$

having its activation vector $[2,1]$. Suppose an agent wants to execute the primitive $ask(b_2,t_2)$. Given the fact that b_2 is empty, the agent suspends its execution waiting for an instance of t_2 to become available. In the meantime let us suppose that we execute a set of primitives on blackboard b_1, as shown in figure 2(a). Since at step 3 the rule becomes active, blackboard b_2 can assume the presence of an instance of tuple t_2, due to which the *ask* primitive previously suspended continues its execution.

Step	Primitives	Counter vector	Comments
1	$tell(b_1,t_1)$	[1, 0]	
2	$tell(b_1,t_1)$	[2, 0]	
3	$tell(b_1,t_2)$	[2, 1]	**Rule is active**
4	$ask(b_2,t_2)$	[0, 1]	**Rule is fired**

(a) The evolution of the counter vector



(b) The blackboards before and after firing the rule

Figure 2: Backward reading example

By analyzing these two examples in more details we observe that in the case of forward reading the rule needs to be associated only with the blackboard defined in the LHS, whereas the backward reading rules require to be present on the blackboard from the LHS as well as the RHS.

4.4 Performance analysis

In order to have an image on how our approach is placed with respect to other implementations, we ran a comparative performance test on one representative of reactive rule based model, TuCSoN. More precisely, we have modeled the synchronization of multiple processes, which has been presented as a practical application in the field of workflow management [18]. Assuming each process places a tuple on the blackboard to signal that it had finished processing, the synchronization is achieved by reacting to the presence of all the tuples on the blackboard. In our approach, this amounts to a forward reading rule of the type:

$$in(\langle task,1\rangle),\ \ldots,\ in(\langle task,n\rangle) \longrightarrow_f in(\langle task,final\rangle)$$

In TuCSoN, since the reaction is observed only on one primitive execution, the test scenario implies the use of n rules, one for each different tuple:

$$reaction(out(task(1)),\ completion,\ (in(task(1),\ldots,in(task(n),out(task(final))))).$$
$$\ldots$$
$$reaction(out(task(n)),\ completion,\ (in(task(1),\ldots,in(task(n),out(task(final))))).$$

We ran several tests, increasing the value of n from 1 to 200. Two execution steps can be identified: *(i)* add and parse the reaction rules, *(ii)* add in sequence the tuples corresponding to the tasks: $\langle task,1\rangle,\ \ldots,\ \langle task,n\rangle$.

The tests have been conducted on two different machines, one on which the tuples 1 to n were outputted and the other on which the final task was outputted. For metrics, we have considered the total execution time. The results are plotted in figure 3 on a semilogarithmic scale where the abscissa holds the different values of n and the ordinate the execution time expressed in seconds.

We observe that, in most cases, up to $n = 20$, the results are comparable. The execution times differ greatly as the value of n increases due to the fact that in TuCSoN, to achieve the same functionality, there is the need to encode n rules and apply all of them each time a new tuple is added to the blackboard. Moreover, TuCSoN requires additional processing for the transactional mechanism in the body of the reaction. On the other hand, our approach makes use of an incremental computation in order to avoid testing the entire rule activation condition after each operation performed on the blackboard.

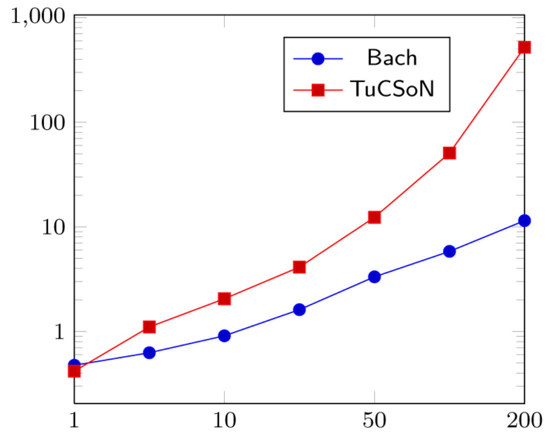


Figure 3: Comparative performance result

TuCSoN	Bach
`reaction(out(task_result(taskA,X),( in_r(task_result(taskA,X)), in_r(task_result(taskB,Y)), out_r(task_todo(taskC,args(X,Y))))).` `reaction(out(task_result(taskB,Y),( in_r(task_result(taskB,Y)), in_r(task_result(taskA,X)), out_r(task_todo(taskC,args(X,Y))))).`	$in(\langle taskA, ?X\rangle), in(\langle taskB, ?Y\rangle) \longrightarrow_f in(\langle taskC, !X, !Y\rangle)$

Table 1: Syntax translation from TuCSoN to Bach

The reciprocal translation, from TuCSoN to Bach, based on an example in [18] and presented in table 1, amounts for the particular case of $n = 2$ in our test scenario.

Likewise, comparable results have been obtained for the examples presented in section 3.4. This was expected since they are particular cases of the scenario presented above.

Based on these results, we can conclude that our approach presents advantages in terms of performance and expressivity, mostly in scenarios which require the verification of complex contexts.

5 Mobile devices

By using the rule mechanism we can model in Bach the connection of remote blackboards when a physical connection between their hosts exists. Being given two devices A and B, the blackboard b_A of host A is linked to the blackboard b_B of host B by providing the backward reading rule $[in(b_B,X)] \longrightarrow_b [in(b_A,X)]$. Symmetrically, on b_B, $[in(b_A,X)] \longrightarrow_b [in(b_B,X)]$ defines the link between the blackboards b_B and b_A. The disengagement is modeled by simply deleting the rules that define the link between the blackboards.

To intermediate between the physical connection of the hosts and the logical linking of blackboards we introduce the notion of *events* that are triggered by dedicated processes that monitor the network activity. The pair of tuples $(connect, h)$ and $(disconnect, h)$ can be generated when the local host connects to a remote host h, respectively disconnects from h.

Each device is responsible for the discovery of its neighbouring devices, achieving this by use of the dedicated functionalities of the network layer. In the same way in which Bluetooth publishes its available services to inquiring devices, one host can publish the list of its public blackboards to other inquiring hosts. As a result, the hosts become self-aware of the neighbouring blackboards which could be uniquely identified by a construct of type $blackboard\_name@host\_name$, where the host's name can be represented by it's MAC or IP address. We achieve the self-awareness and self-adaptation because there is no central manager that handles the engagement and disengagements of hosts. However, for external programming of the blackboards an *a priori* knowledge of their names is necessary. Generic rules such as *inherit*, *broadcast*, *merge* (introduced in Section 3.4) are not affected by such a restriction since they can be kept up to date by the events mentioned in the previous paragraph.

In mobile ad-hoc networks, sudden disconnections may occur without notice. When such an event occurs in the course of an active rule, executing primitives on remote blackboards which are no longer connected would fail. To prevent such an event from blocking the local blackboard it is fair to assume a reasonable connection timeout. This is easily achievable by using the timed primitives introduced in [9] and refined in [11].

As means of communication between hosts we are currently using the Bluetooth technology in RFCOMM mode. It is fair to say that Bluetooth has become a standard on current smartphones. Also, if not present by default on portable computers the low price of Bluetooth dongles makes it a viable choice for short range communication. One drawback of Bluetooth is that devices cannot announce their presence, but try to discover devices around them. For this reason, the concept of two devices being *"close enough"* to establish a connection can only be modeled by periodically scanning for devices within range.

The philosophy we aim to achieve is the one expressed by the BUMP application for mobile devices [10]. Two devices having the BUMP application installed can be physically bumped-into each other in order to get them connected. Once connected, it is possible to share contacts, photos, etc. by means of drag-and-drop from one device to the other. Our experiments show that the application works only in the presence of a functional internet connection. Because of this requirement we concluded that BUMP could not be classified as an application destined for mobile ad hoc networks, even if the two devices are physically close. In response, the Bach language would allow the development of applications that would offer: support more than two devices at once, automatic connection to compatible devices in range, local communication without the need of an active internet connection.

From the point of view of coordination languages the mobility issue can be seen from two perspectives: *logical* and *physical*, as classified in several works [14, 15, 21, 28]. The logical mobility aims at providing to an agent (a program fragment) the means for navigating or exploring an existing connection topology in order to pursue its execution. The physical mobility concerns actual devices or hosts that form the nodes of a network through which the agents roam. The latter is particularly interesting in the case of MANETs (Mobile ad hoc Networks) since the devices moving in and out of each-other's range introduce a high degree of unpredictability in the network topology. The challenge arises from the need to correctly adjust the execution of the agents in the new hypothesis consisting of expanded or contracted tuple spaces.

The KLAIM language [16] has been among the first proposals by considering processes in the same way as data: transferable between computing environments. Departing from the method of invoking remote procedures, KLAIM allows for code fragments to be sent and executed on remote nodes by interacting with other processes via the local tuple space. Since the transient sharing of tuple spaces is not supported, processes need to move to new locations in order to access new tuple spaces.

As stated in [15], in order to keep a high degree of generality at the model level, LIME [15, 20] does not use explicit mechanisms for handling the mobility issue, be it logical or physical. Instead, mobility is inferred from the changes that occur on the host-level or federated tuple spaces as a result of an agent engaging or disengaging a group. We believe that certain limitations are introduced by the need of having a group leader that handles the operations of engagement and disengagement since MANETs are characterized by highly unpredictable connections and disconnections.

TuCSoN [19], designed as a coordination model for internet agents, uses the notion of programmable tuple spaces and uses the hypothesis of permanent communication links. Mobile agents roam the nodes of hierarchically organized networks to query for information and retrieve it if found.

Because of the challenges posed by MANETs we consider that physical mobility has a crucial impact over the logical one. In order to achieve greater flexibility for the agent's execution, these must not be encumbered with the heterogeneity of the surrounding blackboards possibly containing relevant information. To support this idea we propose an alternative solution in the sense that the blackboard on which the agent is destined to be executed should provide the means in order to ensure connections to neighboring blackboards, which would allow the agent faster and more transparent access to information. For this purpose, we present in the following section a rule based mechanism that would enable mobile agent coordination independent of the topology of the network. We believe this to be a different point of view from the approaches presented above in the sense of shifting the focus from handling the mobility of agents to managing the mobility of hosts.

In this light we consider our approach as having a clear advantage with respect to other related pieces of work. The backward reading rule in particular offers all the necessary expressiveness in order to link remote blackboards and to provide simple and efficient access between them.

6 Related work

Let us now offer a bird's eye-view over existing lines of research related to the idea of reactivity and see how they are placed with respect to our proposal.

6.1 Chemical models

The road of reactivity was paved by the GAMMA model [1] which introduced the idea of transforming multi-sets of data by means of mechanisms inspired by chemistry. Accordingly, the multi-set is metaphorically seen as a chemical solution on top of which different reaction rules are defined. The rules consist of pairs *(R,A)*, where *R* represents the reaction condition and *A* the action to be taken. The multi-set evolves as long as the reaction condition is met, after which a stable point is reached. Data is thus rewritten producing either an expansion or a reduction of the initial ones.

As an extension of GAMMA, the *chemical abstract machine* [2] or simply *cham*, added the notions of *membranes* and *airlocks*. Membranes act as containers for sub-solutions, thus enforcing local reactions. Airlocks enable the communication between these enclosed sub-solutions and their containing environment. Such a model is of particular interest to the coordination in mobile ad hoc networks since an analogy can be found between a molecule and a device, respectively between an airlock and the communication links connecting the devices.

More recent developments of the chemical metaphor are those related to the biochemical tuple spaces [25], which introduce a probabilistic approach. By using tuple concentration and chemical rates it is possible to model the likelihood of a given reaction occurring, service equilibrium, service decay, service competition or service composition [26]. In terms of pervasive ecosystems, the work in [27] explores how network nodes can be enriched with live semantic annotations which can be governed by global eco-laws in a chemical-like fashion. This proposition is similar to our current one from the point of view of declarative transformations. However, these eco-laws rely on an underlying framework covering the global space of neighbour devices. In our approach, rules are stored on the blackboards themselves, and connect them without other middleware. Moreover, LSA rules always consume their reactants (lhs) and produce their products (rhs), while ours allow a broader set of behaviours definitions.

Compared to these pieces of work, our proposal keeps the same idea of reactivity. However, it refines it by identifying forward and backward reading, by enhancing the patterns of the rules in distinguishing the presence or absence of information on both sides, by providing an efficient implementation and applying it to mobile ad hoc networks. We provide no counterpart for probabilistic reasonings, but consider this as orthogonal to our work. As a result, ideas from [25, 26] can be introduced directly in our work. Such ideas will be the subject of future work.

6.2 Reactive models

In another line of research, the articles [3, 8] explore models relying on the idea of reactive tuple spaces. Among others, they are used for the coordination of mobile agents. This has also been treated in a series of work, such as: MARS [4], TuCSoN [19], ReSpecT [5, 6], LIME [15, 20].

More concretely, the MARS model proposes reactions in the form of a four components set consisting of: *(i)* the reaction type, *(ii)* the tuple wild-card to be matched, *(iii)* the type of operation on the tuple space that should be monitored, *(iv)* the agent's identity. This mechanism is very flexible and is able to express a wide range of scenarios, from specific to more general ones. This is achieved in the way in which the four components are defined: the most general situation occurs when only the reaction type is specified and is rendered more precise by adding values to the other components.

In TuCSoN , the approach is to define programmable logic tuple centers which consist of a tuple space enhanced by the notion of *behavior specification*. The supervision and control of this behavior is achieved by specifying reactions to the communication events over the tuple space. Reactions act like a set of operations handling sets of tuples in the tuple center, either in the form of addition or deletion. Operations of a reaction are executed atomically, in the sense that even if multiple primitives are invoked they are perceived by the system as a single event.

ReSpecT implements the reactions in the form of two special types of tuples. The first represents an association between a communication primitive and a logical event, allowing for groups of primitives to be connected to one identical logical event or for one primitive to generate several events. The second is an association between the logical events and the actual reaction body, which consists of either state primitives, term predicates or primitives of the tuple space. The paper [17] augments the language with the introduction of a *guard*, that may enforce additional requirements for an event, such as its source, destination or trigger time. The multiple blackboard approach for ReSpecT has been explored in [12] by studying its interactions with LOGOP, introduced in [22] and later refined in [13]. LOGOP presents itself as an extension of LINDA for the management of multiple tuple space environments. The execution of primitives in such a hypothesis is supported by the introduction of composed tuple spaces in their definition. The composition is achieved by applying the logical operators AND, OR, XOR, NOT between multiple tuple spaces. A dedicated logOp tuple center uses the ReSpecT language to react to LOGOP primitives and form requests for each tuple center forming the composition. It also receives the replies and forms the final answer. We believe that this architecture, using a centralised manager, is not suitable in the unpredictable scenario of mobile ad-hoc networks.

Offering a different perspective from the previous models, LIME associates reactions to the context of the tuple space rather than the set of primitives executed on it. Reactions are triggered by matching tuples on the tuple space with given patterns, thus defining specific contexts.

To sum up, the main characteristics of related work are threefold: *(i)* the reaction condition is expressed only in terms of data being present on the blackboard, *(ii)* some reactions are triggered on the execution of primitives, *(iii)* the reaction rules mostly concern a single blackboard.

In our approach we provide a finer control over the reaction conditions, which can be defined in terms of data presence (by using the primitive called *in*) and data absence (by using the primitive called *nin*). This allows the specification of more precise and strict contexts, not possible in other works. In addition, our rule mechanism is designed to be used also in the multi-blackboard system proper to mobile ad hoc networks.

6.3 Final remarks

In summary, as may be appreciated by the reader, our work presents significant differences with respect to the related work. In addition to the chemical models, our approach allows the definition of more complex contexts consisting not only of information that needs to be present, but also of information that needs to be absent. In the terms of the chemical metaphor, our proposition offers the possibility to model the idea of an *inhibitor*, a substance capable of stopping or retarding a chemical reaction. By means of the *nin* primitive it is possible to express reactions which occur in the absence of given inhibitors.

With respect to the reactive models, in which the main focus is put on reacting to the execution of atomic primitives, our declarative approach, by means of context-awareness, offers greater flexibility and expressiveness in terms of coordinating mobile agents in mobile environments and in offering the necessary means to ensure logical links between remote blackboards. In particular, our blackboard rules allow for a fine grained tuning of actions to be taken when mobile devices come close enough. In addition to the merge of tuple spaces, offered by languages such as LIME, we are able to code alternative forms of coordination like *filtering* or *duplication* of blackboard contents.

7 Conclusions

In this paper, we propose a solution for coordination in mobile ad hoc networks by means of declarative rules. By providing two operational readings, we aim at rendering a double functionality. On the one hand, by means of *forward reading*, we offer support for context awareness and reactivity to context changes. By modelling the user's needs as contexts, it is possible to react promptly once the conditions are met. On the other hand, by means of *backward reading*, we provide a mechanism for supporting the transient sharing of blackboards of mobile devices that come within communication range. The *backward reading* rules build logical links between remote blackboards, which enables their unification without the need to physically replicate the tuples. In conjunction, the *forward reading* and *backward reading* support the possibility of interacting not only with the local blackboard, but with remote ones as well.

Certainly, the idea of rules and reactivity is not a novelty to coordination models. However, we have introduced new variants based on a declarative reading which we have shown to be expressive, yet being efficiently implementable. Taking an incremental approach on the computation bypasses the need for a transactional mechanism, since keeping track of how the blackboard content is reflected on the rule's activation condition there is no need to reevaluate it after each primitive execution.

For future work we aim at introducing a probabilistic reasoning in the sense of attaching reaction rates to the rules, similar to [25]. By exploiting the rules mechanism we also aim to obtain capabilities for complex event processing.

References

[1] (1988): *A Parallel Machine for Multiset Transformation and its Programming Style*. Future Generation Computer Systems 4(2), pp. 133 – 144, doi:10.1016/0167739X(88)90012X.

[2] G. Berry & G. Boudol (1990): *The Chemical Abstract Machine*. In: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL '90, ACM, New York, NY, USA, pp. 81–94, doi:10.1145/96709.96717.

[3] K. De Bosschere & J.-M. Jacquet (1996): *μ^2Log: Towards Remote Coordination*. In: Coordination languages and models, Springer-Verlag, pp. 142–159, doi:10.1007/3-540-61052-9_44.

[4] G. Cabri, L. Leonardi & F. Zambonelli (1999): *Reactive Tuple Spaces for Mobile Agent Coordination*. In: Proceedings of the Second International Workshop on Mobile Agents, Springer-Verlag, London, UK, pp. 237–248, doi:10.1007/BFb0057663. Available at `http://dl.acm.org/citation.cfm?id=647628.732565`.

[5] E. Denti, A. Natali & A. Omicini (1998): *On the Expressive Power of a Language for Programming Coordination Media*. In: Proceedings of the 1998 ACM symposium on Applied Computing, SAC '98, ACM, New York, NY, USA, pp. 169–177, doi:10.1145/330560.330665.

[6] E. Denti & A. Omicini (1999): *Designing Multi-agent Systems around an Extensible Communication Abstraction*. In: Selected papers from the ESPRIT Project ModelAge Final Workshop on Formal Models of Agents, Springer-Verlag, London, UK, pp. 90–102, doi:10.1007/3-540-46581-2_7. Available at `http://dl.acm.org/citation.cfm?id=646599.698418`.

[7] D. Gelernter & N. Carriero (1992): *Coordination Languages and their Significance*. Commun. ACM 35, pp. 97–107, doi:10.1145/129630.129635.

[8] J.-M. Jacquet & K. De Bosschere (2001): *Blackboard Relations in the µLog Coordination Language*. New Generation Computing (19), pp. 23–56, doi:10.1007/BF03037533.

[9] J.-M. Jacquet & I. Linden (2007): *Coordinating Context-aware Applications in Mobile Ad Hoc Networks*. In: Proceedings of the first ERCIM workshop on eMobility.

[10] D. Lieb, A. Huibers & J. Mintz (2012): *BUMP*. Available at `http://bu.mp`.

[11] I. Linden (2007): *On the Introduction of Time in Coordination Languages: Semantics, Expressiveness and Programming Methodologies*. Ph.D. thesis, FUNDP - Faculte d'Informatique.

[12] R. Menezes, A. Omicini & M. Viroli (2002): *Have* ReSpecT *for* LOGOP. In Flavio De Paoli, Sara Manzoni & Agostino Poggi, editors: *AI*IA/TABOO Joint Workshop "Dagli oggetti agli agenti: dall'informazione alla conoscenza" (WOA 2002)*, Pitagora Editrice Bologna, Milano, Italy.

[13] R. Menezes, A. Omicini & M. Viroli (2004): *On the Semantics of Coordination Models for Distributed Systems: The* LOGOP *Case Study*. Electronic Notes in Theoretical Computer Science 97, pp. 97–124, doi:10.1016/j.entcs.2004.04.033. Available at `http://www.sciencedirect.com/science/article/pii/S157106610405042X`.

[14] A. L. Murphy, G. P. Picco & G.-C. Roman (2001): *LIME: A Middleware for Physical and Logical Mobility*. In: International Conference on Distributed Computing Systems, pp. 524–533, doi:10.1109/ICDSC.2001.918983. Available at `http://dl.acm.org/citation.cfm?id=876878.879249`.

[15] A. L. Murphy, G. P. Picco & G.-C. Roman (2006): *LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents*. ACM Trans. Softw. Eng. Methodol. 15(3), pp. 279–328, doi:10.1145/1151695.1151698.

[16] R. De Nicola, G. Ferrari & R. Pugliese (1997): *KLAIM: a Kernel Language for Agents Interaction and Mobility*. IEEE Transactions on Software Engineering 24, pp. 315–330, doi:10.1109/32.685256.

[17] A. Omicini (2007): *Formal ReSpecT in the A&A Perspective*. Electron. Notes Theor. Comput. Sci. 175(2), pp. 97–117, doi:10.1016/j.entcs.2007.03.006.

[18] A. Omicini, A. Ricci & M. Viroli (2006): *Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research*. In: *Software Engineering for Multi-Agent Systems IV*, *Lecture Notes in Computer Science* 3914, Springer Berlin / Heidelberg, pp. 71–90, doi:10.1007/11738817_5.

[19] A. Omicini & F. Zambonelli (1999): *Tuple Centres for the Coordination of Internet Agents*. In: *1999 ACM Symposium on Applied Computing (SAC'99)*, ACM, San Antonio, TX, USA, pp. 183–190, doi:10.1145/298151.298231.

[20] G. P. Picco, A. L. Murphy & G.-C. Roman (1999): *LIME: Linda Meets Mobility*. In: *Proceedings of the 21st international conference on Software engineering*, ICSE '99, ACM, New York, NY, USA, pp. 368–377, doi:10.1145/302405.302659.

[21] I. Satoh (2002): *Physical Mobility and Logical Mobility in Ubiquitous Computing Environments*. In: *Proceedings of Conference on Mobile Agents (MA'02)*, *LNCS*, Springer, pp. 186–202, doi:10.1007/3-540-36112-X_13. Available at `http://dl.acm.org/citation.cfm?id=647631.732730`.

[22] J. Snyder & R. Menezes (2002): *Using Logical Operators as an Extended Coordination Mechanism in Linda*. In: *Proceedings of the 5th International Conference on Coordination Models and Languages*, COORDINATION '02, Springer-Verlag, London, UK, UK, pp. 317–331, doi:10.1007/3-540-46000-4_29. Available at `http://dl.acm.org/citation.cfm?id=647017.713447`.

[23] L. Sterling & E. Shapiro (1994): *The Art of Prolog (2nd ed.): Advanced Programming Techniques*. MIT Press, Cambridge, MA, USA.

[24] S. Thompson (1999): *Haskell: the Craft of Functional Programming*. Addison Wesley.

[25] M. Viroli & M. Casadei (2009): *Biochemical Tuple Spaces for Self-Organising Coordination*. In John Field & Vasco T. Vasconcelos, editors: *Coordination Languages and Models*, *LNCS* 5521, Springer, Lisbon, Portugal, pp. 143–162, doi:10.1007/978-3-642-02053-7_8.

[26] M. Viroli & M. Casadei (2010): *Chemical-inspired Self-composition of Competing Services*. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, ACM, New York, NY, USA, pp. 2029–2036, doi:10.1145/1774088.1774514.

[27] M. Viroli, D. Pianini, S. Montagna & G. Stevenson (2012): *Pervasive Ecosystems: a Coordination Model Based on Semantic Chemistry*. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, ACM, New York, NY, USA, pp. 295–302, doi:10.1145/2245276.2245336.

[28] S. Zachariadis, C. Mascolo & W. Emmerich (2002): *Exploiting Logical Mobility in Mobile Computing Middleware*. In: *In Proceedings of the IEEE International Workshop on Mobile Teamwork Support, Collocated with ICDCS'02*, pp. 385–386, doi:10.1109/ICDCSW.2002.1030799.