Fundamenta Informaticae XX (2005) 1–28
IOS Press

Supporting Secure Coordination in SecSpaces*

Roberto Gorrieri

Roberto Lucchi

Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura Anteo Zamboni 7, I-40127 Bologna, Italy.
e-mail:{gorrieri, lucchi, zavattar}@cs.unibo.it

Abstract. In this paper we investigate security problems which occur when exploiting a Linda-like data driven coordination model in an open environment. In this scenario, there is no guarantee that all the agents accessing the shared tuple space are trusted. Starting from a formalization of some typical security properties in the standard Linda coordination model, we present a novel data-driven coordination model which provides mechanisms to support the considered security properties. The first of these mechanisms supports logical partitions of the shared repository: in this way we can restrict the access to tuples stored inside a partition, simply by limiting the access to the partition itself. The second mechanism consists of adding to the tuples some extra information which permits to authenticate the producer of a tuple or to identify its reader/consumer. Finally, we support the possibility to define access control policies based on the kind of operations an agent performs on a tuple, thus discriminating between (destructive) input and (non-destructive) read permissions on each single tuple.

1. Introduction

New networking technologies are calling for the definition of models and languages adequate for the design and management of new classes of applications. Innovations are moving towards two directions. On the one hand, the Internet is the candidate environment for supporting the so-called wide area applications. On the other hand, smaller networks of mobile and portable devices, such as mobile ad-hoc

* Work partially supported by MEFISTO Progetto "Metodi Formali per la Sicurezza e il Tempo", Microsoft Research Europe and by NAPOLI Progetto "Network Aware Programming: Oggetti, Linguaggi, Implementazioni".

networks or peer-to-peer systems, support applications based on agents or components which interact according to a dynamically reconfigurable communication structure. In both cases, the challenge is to develop applications without knowing, at design time, the overall structure of the system as well as the entities that will be involved. Indeed, these systems are usually open to new agents or components which are unknown beforehand.

Coordination models and languages, which advocate a distinct separation between the internal behaviour of the entities and their interaction, represent a promising approach for the development of applications for this class of dynamic and open systems. For instance, we assist to a renewed interest in data-driven coordination infrastructures originated by Linda [Gel85] as exemplified by recent commercial products, such as JavaSpaces [Sun02] and TSpaces [WMF98], which are two coordination middlewares for distributed Java [GJS96] programming proposed by Sun Microsystem and IBM, respectively.

Both proposals exploit the so-called *generative communication* [Gel85]: a sender communicates with one or more receivers through a shared tuple space (TS for short), where emitted tuples are collected; a receiver can read or consume the tuples from the TS; a tuple generated by an agent has an independent existence in the tuple space until it is explicitly withdrawn by a receiver; in fact, after its insertion in TS, a tuple becomes equally accessible to all agents, but it is bound to none. This form of communication is referred to as generative communication because when a datum is produced, it has an existence which is independent of its producer, and it is equally accessible to all agents.

In open systems, new critical aspects come into play such as the need to deal with a hostile environment. In particular, also untrusted agents may enter the system and, according to the data-driven approach, they can access the repository in order to read/remove data, as well as maliciously produce new data.

In this paper we propose an extension/modification of the tuple space coordination model in order to support some form of control in the execution of the coordination primitives that the agents can perform to access the TS.

The most interesting proposals of data-driven coordination languages which provide some form of access control are KLAIM [NFP98, NFP97] and SecOS [VBO03].

KLAIM exploits classic access control policies to manage the access of agents to tuple spaces: types are used to indicate the access rights of the agents, i.e., the operations that each agent may perform on each of the available tuple spaces. In open systems, especially in those with a high level of dynamicity, the managing of these information may be a critical task, mainly because the system should support a rapid and sometimes uncontrolled evolution of the agent community. More precisely, new agents may frequently enter the system, as well as old agents may rapidly exit, in an uncontrolled manner. Moreover, in some applications it could be useful to have a finer grained control, e.g., at the level of tuples and not at the level of spaces. For example, we may want to ensure that an agent cannot read tuples with a private contents, but it can read all of the other tuples.

SecOS follows a quite different approach. The access rights are not associated to the agents, but all control information are stored inside the data. More precisely, SecOS supports two forms of locks which are called *symmetric* and *asymmetric*. The former exploits the same key to protect and access the information, while the latter uses a pair of keys, one to protect and another one to access. This two locking techniques can be applied to protect either one single field inside a tuple or the whole tuple. In the first case the used locks are called *Field-locks*, while in the second one, they are called *Object-locks*.

In SecOS a read operation is defined as the juxtaposition of an input followed by an output operation, namely

$$rd\ e\ x.P \stackrel{def}{=} in\ e\ x.(out\ x \mid P)$$

This fact has the two following consequences:

- There is no discrimination between non-destructive and destructive input access policies. More precisely, there is no way to discriminate between the readers and the consumers of a datum. This could be an undesired feature in many applications. Consider, e.g., a system in which an information can be accessed by any agent, but it can be removed only by specialized garbage collectors.

- An agent able to access a datum is also able to reproduce that datum, possibly creating new instances of that datum. This means that an agent which is a reader of a datum, is implicitly also a potential producer for that datum. Also this feature could be undesired in many applications. Consider, e.g., a master/worker system in which we want to ensure that tasks can be produced only by masters and consumed only by workers. According to the SecOS approach, there is no direct way to avoid a malicious worker to reproduce new instances of the tasks it consumes.

Starting from these observations, we propose a novel coordination model called `SecSpaces` which aims at supporting the advantages of both the approaches. In particular, as in SecOS we introduce the access information inside the data, but we refine the access controls supported in SecOS by permitting to discriminate, e.g., the rd from the in permissions.

We introduce `SecSpaces` incrementally. We initiate by reporting in Section 2 a formal representation of our starting point, that is the Linda coordination model. We first formally define the coordination medium, i.e. the TS, then we formally define the effect on the TS of the execution of the coordination primitives. After, we introduce a framework that permits to model some of the security properties typically considered for channel based communication. Namely, we consider data *secrecy*, data *integrity*, and data *availability*. We conclude our analysis of standard Linda by showing that the considered security properties (in particular data availability) cannot be ensured. Our proof permits to show that the basis of the security deficiencies of Linda are in the possibility to use wildcards in the pattern matching algorithm used in data retrieval operations. For example, each tuple in the TS can be read or even consumed by any of the agents in the system, simply by using wildcards accordingly.

In Section 2 we introduce a first extension of the standard Linda pattern matching. We consider an extra information, associated to the tuples, that cannot be matched by wildcards. Thus, in order to access a tuple, an agent must show a proof of knowledge of this extra information. We call this extra information *partition field* because its effect is to logically partition the TS. By means of a simple example, we show that data secrecy, data integrity and data availability now can be supported simply by exchanging the data through private partitions.

However, we show that partition fields are not sufficient for programming those applications in which it is necessary to authenticate the producer of a certain datum, as well as to identify the possible readers/consumers of that datum. For example, consider a client-server application in which the clients ask for services emitting tuples, and the servers execute the services described in the tuples. According to the access control supported by the partition fields, a malicious client could remove the service requests produced by any other client before that the server execute them. To solve this lack of the model, in

Section 4 we introduce a more sophisticated access mechanism to the logical partitions, which is based on two asymmetric partition handlers: the first one to be used for data introduction, and the second one for data retrieval. We call these handlers *asymmetric partition fields*. We show by means of a simple example, that this enhanced mechanism supports the authentication of the producers as well as the identification of the readers/consumers of the data exchanged through these new form of logical partitions.

Finally, in Section 5 we observe that the two above mechanisms do not permit to distinguish between the access policy to use for non-destructive and destructive access operations. For example, an agent able to read a datum is also able to remove that datum from the TS. Thus, we complete the definition of our new coordination model `SecSpaces` by associating to each tuple a pair of partition fields, and a pair of asymmetric partition fields. The first element of the pairs is considered when non-destructive read operations are executed, the second is used when destructive input operations are performed.

The main contribution of `SecSpaces` is in the definition of an advanced matching rule which, exploiting the three mechanism described above, is the basis of a novel approach for programming, in a more secure way, distributed applications based on tuple spaces. The examples that we report in the paper prove the generality and the flexibility of this matching rule, that can used to achieve several access control mechanisms. However, acting only at the level of the matching rule, we do not support any control on the execution of the output operations. Due to the asynchrony of the communication supported by the tuple space, we consider the problem of controlling the ability to produce tuples as a different and orthogonal task with respect to the analysis of advanced matching rules to be used to retrieve those tuples. For this reason, we do not consider in this paper the problem of the definition of new policies for the control of output operations, that we leave for future work.

The last two Sections 6 and 7 report a more detailed comparison with related work and some conclusive remarks, respectively.

2. Security Properties in Linda

In this section we initiate our investigation of the security problems which arise when exploiting Linda-like tuple spaces for supporting the coordination of agents in an open application. We first introduce a formalization of the standard Linda coordination model, then we redefine in this new scenario some of the typical security properties which are usually considered in the context of secure channel communication. More in detail, the coordination model is introduced first by modeling the coordination medium (Section 2.1) and then (Section 2.2) by modeling the agent. An agent is characterized by a state and a knowledge, that is the set of data it knows at that given state of the computation. This is an unusual approach that separates the coordination medium from the agents accessing the space; the advantage is that the *enemy* can be easily modeled as a special agent that can perform any Linda primitive, at any time, by using any template and entry it can build from its knowledge (that can grow during the execution of the system). The enemy will be formally defined in Section 2.3 where the main security properties are defined.

It is also worth noting that, in order to separate the coordination medium from the coordinated agents, our labeled semantics slightly differs from standard labeled semantics for process calculi. We need to use two labels for each transitions, the first label describes the operation which is executed, the second label denotes the returned value (typically, the tuple actually accessed by an input or a read operation). This is a small complication that we need to introduce in order to benefit from the formal distinction between the coordination medium and the agents described above.

2.1. Modeling the coordination medium

Linda supports coordination via a shared tuple space, where a tuple is a finite sequence of data.[1] Formally, let $Data$, ranged over by d, d_1, ..., be a denumerable set of data. Let $Entry$, ranged over by e, e', ..., be the set of finite sequences of data taken from $Data$ and denoted by $\langle d_1, \ldots, d_n \rangle$, with $n \geq 1$.

In order to perform input and read operations, templates are used by an agent to specify the kind of entry the agent is interested in. Templates may use wildcards to let some field unspecified; syntactically, a wildcard is denoted by $null$ (which we assume to be not in $Data$). In the following, dt, dt_1, ..., range over $Data \cup \{null\}$. Formally, let $Template$, ranged over by t, t', ..., be the set of finite sequences composed of data and wildcards denoted by $\langle dt_1, \ldots, dt_n \rangle$, with $n \geq 1$.

A matching rule between entries and templates is used to indicate when an entry satisfies a certain template. The standard Linda matching rule is as follows.

Definition 2.1. Matching rule - Let $e = \langle d_1, \ldots, d_n \rangle$ be an entry and $t = \langle dt_1, \ldots, dt_n \rangle$ be a template; we say that $e \ matches \ t$ if the following condition holds:

- for each $1 \leq i \leq n$, either $dt_i = d_i$ or $dt_i = null$.

The coordination primitives we consider are the classical Linda operations: $out(e)$ which introduces the entry e in the shared repository; $rd(t)$ which returns a copy of a matching entry e (i.e. $e \ matches$ t) which is currently available in the shared repository ; $in(t)$ which is the destructive counterpart of $rd(t)$. For the sake of simplicity, we usually omit the angular parenthesis from the denotation of in and rd operations: e.g. we use $in(d_1, null)$ instead of $in(\langle d_1, null \rangle)$.

Observe that the operations, excluding $out(e)$, have a return value: $rd(t)$ and $in(t)$ return a copy of the matching entry e (in the case of $in(t)$ this matching entry e is also removed from the repository).

Let $Action = \{out(e), rd(t), in(t) \mid e \in Entry, t \in Template\}$ (ranged over by α) be the set of operations that an agent can perform, and let $ReturnValue = Entry \cup \{-\}$ (ranged over by r) be the set of return values of the operations. We use $-$ to denote the empty return value of the out operation.

The current state of the shared tuple space is modeled by a multiset of entries: formally, the state of the tuple space is modeled with $TS, TS', \ldots$, which range over $\mathcal{M}(Entry)$, i.e. the set of the possible multisets on $Entry$. To describe the effect of the operations on the shared repository, we use a labeled transition system $(\mathcal{M}(Entry), Label, \longrightarrow)$ where $Label = Action \times ReturnValues$. The typical transition of this labeled transition system is denoted with $TS \xrightarrow[r]{\alpha} TS'$, whose meaning is as follows: when the state of the tuple space is TS, and an action α is performed, then the new state of the repository is TS' and the return value is r. The labeled transition system is defined as the smallest one satisfying the rules in Table 1.

The first rule describes the effect of the execution of an $in(t)$ operation: one of the available matching entry e is removed from the TS, and it is returned as result. The second rule describes $rd(t)$ operation: observe that the returned entry is not removed from the tuple space. The last rule describes the execution of an $out(e)$ operation whose effect is to add the entry e to the tuple space.

[1]In real Linda implementations, a tuple is a finite sequence of typed fields; for the sake of simplicity we consider a unique generic type $Data$.

$TS \oplus \{e\} \xrightarrow[e]{in(t)} TS$	$e\ matches\ t$
$TS \oplus \{e\} \xrightarrow[e]{rd(t)} TS \oplus \{e\}$	$e\ matches\ t$
$TS \xrightarrow[-]{out(e)} TS \oplus \{e\}$	

Table 1. Semantics of the coordination medium.

It is interesting to observe that wildcards can be used to access all the information available in the tuple space. Consider, e.g., a generic entry $\langle d_1, d_2 \rangle$ that can be read by any agent simply performing $rd(d_1, null)$ or even $rd(null, null)$. In particular, the operation $rd(null, null)$ exploits wildcards to match with all possible entries having exactly two data fields.

This simple example shows the main difficulty that the application developer encounters when he needs to bound the scope of visibility or to reduce the accessibility to information which are introduced in an entry of the tuple space. The typical solution to this problem could be to encrypt the information, in such a way that the information can be decoded only by the expected receiver of the information. However, this is not completely satisfactory, because there is no guarantee that the information will be removed from the tuple space only by the expected consumers, because any agent could exploit wildcards to remove the entry containing the encrypted information. In other words, the model cannot ensure that the entry is always available and delivered to the expected agent. This problem will be discussed and formalized more in detail in Section 2.3. In the remainder of this section we formalize our investigation of these security problems encountered when the standard Linda coordination model is adopted.

2.2. Modeling the agents

In order to introduce a formal definition of the security properties we need to provide, besides the modeling of the tuple space reported above, also a modeling of the agents in the system.

An agent has its own identity, an internal state, and a knowledge. The knowledge is used to formalize the information an agent is aware of, in order to be able to verify, e.g., whether an information is secret or known by all the agents in the system.

Formally, let $AgentId$, ranged over by A_1, A_2, $\ldots$, be a denumerable set of agent identifiers; let $State$, ranged over by S, S', $\ldots$, be a denumerable set of states. The knowledge of an agent is usually denoted with the set of data it is aware of; formally, we use ϕ, ϕ', $\ldots$, ranging over $Knowledge = \wp(Data)$, i.e. the powerset of $Data$. An agent with identity A, a state S, and a knowledge ϕ is denoted with A_S^ϕ. A configuration of agents Ag, Ag', $\ldots$, is a set of agents such that there exist no two different agents with the same identity; formally, Ag, Ag', $\ldots$, range over the set

$$Agent = \{\ \{A_1{}_{S_1}^{\phi_1}, \ldots, A_n{}_{S_n}^{\phi_n}\} \mid \text{for each } i \neq j \text{ then } A_i \neq A_j\}$$

In the following we will omit parenthesis for set of agents containing a single agent, e.g. we use A_S^ϕ instead of $\{A_S^\phi\}$.

In the following we will adopt the following notation:

- $arg(\alpha)$ denotes, given an action α, the argument of α; that is, $arg(out(e)) = e$ and $arg(op(t)) = t$ if $op \in \{rd, in\}$;
- $dataset(e)$ and $dataset(t)$ denote, given either an entry e or a template t, the set of data fields stored inside it, e.g., $dataset(\langle d_1, d_2\rangle) = \{d_1, d_2\}$;
- $D(\phi)$ denotes, given a knowledge ϕ, the entries and the templates that an agent with knowledge ϕ can generate, more precisely, the entries and templates which contain the data in ϕ or the wildcard $null$; for example, $\langle d_1, d_2\rangle$ and $\langle d_1, null\rangle$ are in $D(\{d_1, d_2\})$ while $\langle null, d_3\rangle$ is not in $D(\{d_1, d_2\})$.

In order to be as general as possible we do not specify the internal behaviour of the agents. We simply assume given a relation act which indicates the possible actions that an agent can take according to its current state and knowledge.

Formally, let $act \subseteq State \times Knowledge \times Action \times ReturnValue \times State$ be a relation composed of 5-ples (S, ϕ, α, r, S') having the following meaning: if an agent has an internal state S and a knowledge ϕ, then the agent is willing to perform the action α and, in the case the return value is r, then the next state will be S'. Note that act does not depend on the agent identifier, hence agents having the same state (and the same knowledge) are behaviourally equivalent.

We make the following two assumptions on any of the 5-ples (S, ϕ, α, r, S') of act. The former is that the argument of the action α should be compliant with the current knowledge ϕ of the agent; formally, we assume that $arg(\alpha) \in D(\phi)$. The latter is that the next state should be defined for any possible return value r', even if different from r; formally, if $TS \xrightarrow[r']{\alpha} TS'$ (for some TS and TS'), then also a 5-ple $(S, \phi, \alpha, r', S'')$ is in act for any $r' \neq r$.

We are now ready to formalize a system as a pair (Ag, TS) composed of a set of agents Ag and the current state TS of the tuple space; formally, let $System = Agent \times \mathcal{M}(Data)$. The semantics of the systems is described by a labeled transition system $(System, Label, \rightarrow)$, where $Label = Action \times ReturnValue \times AgentId$, with transitions denoted as $(Ag, TS) \xrightarrow[r]{\alpha}_A (Ag', TS')$. The interpretation for the transition is that given a system (Ag, TS), then this can evolve to (Ag', TS') as effect of the execution of the operation α, with return value r, performed by the agent with identity A.

The labeled transition system is defined as the smallest one satisfying the rule in Table 2.

It is worth noting that the knowledge of the agent performing the operation may change in the case new information is achieved as result of a read (or input) operation.

With an abuse of notation, we use $\longrightarrow$ for both the transition systems defined in Tables 1 and 2. However, this does not introduce confusion because the current meaning of $\longrightarrow$ will be made clear from the context.

In the following we will use $(Ag, TS) \rightarrow^* (Ag', TS')$ to denote the fact that the system (Ag, TS) may evolve to (Ag', TS'); namely, there exists a sequence of transitions

$$(Ag, TS) \xrightarrow[r_1]{\alpha_1}_{A_{i_1}} (Ag_1, TS_1) \xrightarrow[r_2]{\alpha_2}_{A_{i_2}} \ldots \xrightarrow[r_{n-1}]{\alpha_{n-1}}_{A_{i_{n-1}}} (Ag_{n-1}, TS_{n-1}) \xrightarrow[r_n]{\alpha_n}_{A_{i_n}} (Ag', TS')$$

$$\frac{TS \xrightarrow[r]{\alpha} TS'}{(Ag \cup A_S^{\phi}, TS) \xrightarrow[r]{\alpha}_A (Ag \cup A_{S'}^{\phi \cup dataset(r)}, TS')} \quad \text{if } (S, \phi, \alpha, r, S') \in act$$

Table 2. Semantics of the agents.

We write $(Ag, TS) \xrightarrow[r]{\alpha}_A$ to state that there exists (Ag', TS') such that $(Ag, TS) \xrightarrow[r]{\alpha}_A (Ag', TS')$.

2.3. Formalizing the security properties

We have already observed that when tuple spaces are used as coordination infrastructures in open applications, security problems occur due to the fact that also untrusted agents may access the tuple space, and perform any kind of coordination primitives. In order to formalize the effect of the presence of these untrusted entities in the system, we consider a specific agent, called the *enemy*, having the ability to execute any possible action at any possible instant. The enemy is formally modeled as an agent with a specific identity E and a specific state S_E. In order to permit to the enemy to perform all the possible actions that the untrusted unexpected entities may execute, we assume that the state S_E has the following property: for any knowledge ϕ and any return value r, the set $\{(S_E, \phi, \alpha, r, S_E) \mid arg(\alpha) \in D(\phi)\}$ is a subset of act, i.e. this state allows to non-deterministically perform all the operations permitted by the current knowledge of the agent.

We are now ready to formally define, in our scenario, three security properties. Namely, we consider *data secrecy*, *data integrity* (also called *data authenticity*, see [FGM00]) and *data availability*. Informally, secrecy corresponds to the ability of a secure channel to maintain confidential, i.e. unknown to the enemy, the information which are exchanged through the channel. Data integrity means that the contents of the data which are communicated through the channel cannot be modified by any possible enemy. Finally, data availability means that the enemy cannot remove any datum from the secure channel.

In order to formalize data secrecy, we consider a specific system with a confidential datum; then, we considered the same system extended with the enemy, and we require that the enemy has no way to add the confidential datum to its knowledge.

Definition 2.2. Data secrecy - Let (Ag, TS) be a system, and let ϕ_E be the initial knowledge assigned to the enemy E. Let $d \in Data$ be a datum: we say that $Secrecy((Ag, TS), \phi_E, d)$ holds if and only if for any ϕ'_E such that $(Ag \cup E_{S_E}^{\phi_E}, TS) \rightarrow^* (Ag' \cup E_{S_E}^{\phi'_E}, TS')$, we have that $d \notin \phi'_E$.

Observe that the test is performed assuming a specific datum d, whilst other secrecy definitions, see e.g. [AG99], extend the test applying the universal quantifier to d, requiring that the test holds for any $d \in Data$. However, our approach is more flexible because, when it is necessary, permits to extend the secrecy test by universally quantifying the datum d on a specific subset of $Data$ (and not only on the whole set $Data$). For the same reason, also the following properties that we are going to define do not consider universal quantification on the data.

Note that data secrecy does not ensure that the data will always be delivered to one of the honest agents. Moreover, the knowledge of the agents in Ag is completely disregarded, that is, it is not important which agent knows the secret datum. This reflects the fact that, in this context, the same datum (even if confidential) can be non-destructively accessed by more than one agent. On the other hand, in secure channel communication we usually assume that each channel has a single expected reader and a single expected consumer.

We now consider data integrity. Suppose that in a system (Ag, TS) there is a datum M that an agent A should read. The integrity of the datum is supported only if it is ensured that the agent A receives exactly M, and not a different datum M'. In order to formally check that data integrity is supported, following the approach used in [FGM03], we assume that after the agent A has read the datum for which integrity is assumed, it communicates this datum, say d, by performing the operation $out(\langle commit, d\rangle)$. Integrity is actually guaranteed if it is ensured that d corresponds to the expected M.

It is worth noting that in real distributed contexts Linda primitives and the corresponding parameters, as well as return values, are transmitted on communication channels between the agents and the tuple space implementation. The formalization of data integrity we propose exploits the assumption that the enemy does not attack the communication channels. Such an assumption could be guaranteed, for instance, by using well known security protocols supporting integrity on channels (see, e.g., [Sch96]). In other words we do not admit the case in which the agent A receives M', inserts the special entry $\langle commit, M'\rangle$ which is received on TS as $\langle commit, M\rangle$.

Definition 2.3. Data integrity - Let (Ag, TS) be a system, and let ϕ_E be the initial knowledge assigned to the enemy E. Let $M \in Data$ be a datum. $Integrity((Ag, TS), \phi_E, M)$ holds if and only if for any agent identifier $A \neq E$ such that $(Ag \cup E_{S_E}^{\phi_E}, TS) \rightarrow^* (Ag' \cup E_{S_E}^{\phi'_E}, TS') \xrightarrow[-]{out(\langle commit, d\rangle)}_A$ then $d = M$.

In line with the observation reported above, several agents could read the datum M; this is formally reflected by the fact that the agent identity A is universally quantified in the formal definition. It is worth noting that the formal definition does not consider the fact that M is not in the knowledge of the enemy, hence data integrity does not ensure the confidentiality of datum M.

The last security property that we consider is data availability. This property is important when we would like to ensure that a datum d cannot be removed by the enemy. Suppose that $Ids = \{A_1, \ldots, A_n\}$ is the set of the identities of the agents interested in reading the datum d. Data availability is ensured if, after that the datum is introduced in the tuple space, it remains available at least until all the agents have accessed the datum.

Definition 2.4. Data availability - Let (Ag_0, TS_0) be a system, and let ϕ_E be the initial knowledge assigned to the enemy E. Let Ids be a set of agent identities and let $d \in Data$ be a datum. $Availability((Ag_0, TS_0), \phi_E, Ids, d)$ holds if and only if for any agent identifier $A \in Ids$ and any possible computation of the system with enemy

$$(Ag_0 \cup E_{S_E}^{\phi_{E_0}}, TS_0) \xrightarrow[r_1]{\alpha_1}_{A_{i_1}} (Ag_1 \cup E_{S_E}^{\phi_{E_1}}, TS_1) \xrightarrow[r_2]{\alpha_2}_{A_{i_2}} \ldots \xrightarrow[r_n]{\alpha_n}_{A_{i_n}} (Ag_n \cup E_{S_E}^{\phi_{E_n}}, TS_n)$$

one of the three following conditions holds:

- the datum d is not introduced in the tuple space, i.e., $d \notin \bigcup_{e \in TS_i} dataset(e)$ for any $0 \leq i \leq n$;
- after that the datum d is introduced in the tuple space, the agent A has always the capability to read/consume it; formally, if $0 \leq i \leq n$ such that $d \in \bigcup_{e \in TS_i} dataset(e)$, then for any $j \geq i$ we have that there exist $e \in TS_j$ (with $d \in dataset(e)$) and a template $t \in D(\phi_{A_j})$ (where ϕ_{A_j} is the knowledge of the agent A in Ag_j) such that $e \; matches \; t$;
- the agent A accesses the datum d during the step j of the computation; formally, there exists $1 \leq j \leq n$ such that $A_{i_j} = A$ and $d \in dataset(r_i)$.

Also in this case the formal definition does not assume that the datum d is not in the knowledge of the enemy; thus, data availability may be ensured even if the enemy can access the datum d. Moreover, the definition does not make any assumption on the number of occurrences of the datum d; hence, data availability may be ensured even if the enemy removes datum d, provided that it has no way to remove all of its instances.

We are now ready to formalize the main general security deficiencies of the Linda coordination model. We will consider data secrecy and availability, while we do not take data integrity into account. Indeed, integrity is a property which strongly depends on the specific communication protocol executed by the agents, and it is not a general property. For example, a system in which no agent performs the $out(\langle commit, d \rangle)$ operation trivially satisfies data integrity.

Proposition 2.1. Let (Ag, TS) be a system and Ids a set of identities of agents in Ag. For any $d \in \bigcup_{e \in TS} dataset(e)$ we have that $Secrecy((Ag, TS), \phi_E, d)$ and $Availability((Ag, TS), \phi_E, Ids, d)$ do not hold.

Proof:
As $d \in \bigcup_{e \in TS} dataset(e)$, we have that there exists an entry e in TS, such that $e = \langle d_1, \ldots, d_n \rangle \in TS$ and $d = d_i$ for some $1 \leq i \leq n$. Consider now the template $t = \langle null, \ldots, null \rangle$ with exactly n wildcards. Trivially we have that $t \in D(\phi_E)$, thus the following computation can occur $((Ag \cup E_{S_E}^{\phi_E}, TS) \overset{rd(t)}{\underset{e}{\longrightarrow}}_E (Ag \cup E_{S_E}^{\phi'_E}, TS')$. We have that $d \in \phi'_E$, thus secrecy is not supported.

The same reasoning can be used in order to prove that also availability is not supported. It is enough to consider all entries $e_1, \ldots, e_m$ containing the datum d, and observe that the enemy can use wildcards and the in operation in order to remove all these entries. □

This proposition shows that wildcards, and the matching rule, are the basis for the security deficiencies of Linda. A first approach for resolving these problems could be to exploit standard cryptographic techniques [Sch96] in order to introduce encrypted data inside tuples. Such a solution is used in Klava [BN03], that will be discussed in Section 6. This approach could, e.g., support secrecy, provided that the enemy has no ability to decrypt the confidential data. However, this solution is not completely satisfactory when also data availability must be ensured.

In the remainder of the paper we present extensions of the matching rule, and we will show that these extensions will permit to support secrecy and availability even without exploiting cryptographic techniques.

3. Partitioning the Tuple Space

As observed in the previous section, in Linda systems each tuple in the shared repository may be potentially read or consumed by each agent having access to that repository. In many applications this could be an undesired feature; consider, e.g., applications in which the agents are divided into classes and it is necessary to distinguish among the entries of one class and the entries of another class. A possible approach for supporting this separation could be to use one repository for each group, that can be easily implemented in KLAIM. In this section we present a simpler approach which permits to logically divide the whole dataspace in separated partitions. Even if logical partitions may resemble similar to physical partitions (such as those of KLAIM), they have the advantage of beeing conservative extensions of Linda. For instance, existing applications based on the standard Linda primitives do not require to be revisited in order to be compliant with the new coordination model extended with logical partitions. Moreover, we propose a stepwise extension of the classical Linda model in order to guarantee an increasing number of security features.

3.1. Partition fields

The idea is to extend the entry structure with a special field, named *partition field*. The partition field is the only one in which the matching rule does not accept wildcards. Therefore, an agent executing a read operation on an entry with a partition field is forced to deliver a proof of knowledge of that partition field.

Formally, let $Partition$ be a denumerable set of partition field values (ranged over by c, c_t, $c_s, \dots$). We also assume that $Partition$ contains a special default value $\#$ used when the partition field has no importance (the actual use of $\#$ will be described in the following). An entry e, and a template t with the partition field, are defined as follows:

$$e = \langle d_1, \dots , d_n \rangle^{[c]}$$

$$t = \langle dt_1, \dots , dt_n \rangle^{[c_t]}$$

with $n \geq 1$. Let $Entry^c$ and $Template^c$ be the set of entries and templates with partition field, respectively. In the following, e, e', ..., range over $Entry^c$, and t, t', ..., range over $Template^c$. The tuples $\langle d_1, \dots , d_n \rangle$ and $\langle dt_1, \dots , dt_n \rangle$, $n \geq 1$, are also denoted in the compact form $\langle \vec{d} \rangle$ and $\langle \vec{dt} \rangle$, respectively. We also define $e|_d$ as the operator that, given an entry $e = \langle \vec{d} \rangle^{[c]}$, returns the tuple of data $\langle \vec{d} \rangle$.

Definition 3.1. Matching rule with partition fields - Let $e = \langle \vec{d} \rangle^{[c]}$ be an entry, and $t = \langle \vec{dt} \rangle^{[c_t]}$ be a template; we say that $e\ matches^c\ t$ when the following conditions hold:

- $c_t = c$,
- $\langle \vec{d} \rangle\ matches\ \langle \vec{dt} \rangle$.

Observe that the definition of $matches^c$ is a conservative extension of $matches$ (see Definition 2.1).

A consideration about performance is of interest. Given a template $\langle \vec{dt} \rangle^{[c_t]}$, if there exists a matching entry e in the space, then $e = \langle \vec{d} \rangle^{[c]}$ and $c_t = c$. We can exploit c_t in order to reduce the search space to the partition identified by c_t.

Example 3.1. matches^c evaluation - A list of entry matching examples follows:

template	**does it match with $\langle d_1, d_2\rangle^{[c]}$?**
$\langle null\rangle^{[c]}$	no
$\langle null, null\rangle^{[c]}$	yes
$\langle d_1, null\rangle^{[c]}$	yes
$\langle d_1, d_2\rangle^{[c]}$	yes
$\langle d_1, d_2\rangle^{[c']}, c' \neq c$	no
$\langle null, null, null\rangle^{[c']}$	no

The new operational semantics, which takes into account the presence of the partition fields, is obtained simply by replacing in Table 1 $matches$ with $matches^c$, and the e used to describe the return value with $e|_d$ (in this context $ReturnValue = \{e|_d \mid e \in Entry^c\} \cup \{-\}$). Observe that the return value does not contain the partition field, as we use this information only to control the access to the entries.

The partition field and the new matching rule induce a partitioning of TS. Every $c \in C$ identifies a subspace, which is composed of all those entries in TS with partition field set to the value c. Clearly, in order to access the subspace characterized by c, an agent must know c. However, the special value $\#$ provides the agents with a default subspace to be used when no specific partition has been defined as working space. For the sake of simplicity, when an entry (template) has the partition field set to $\#$, we omit it. For example, the Linda system defined in the previous section can be seen as a subcase of the new system, provided that only the default partition $\#$ is used.

In order to adapt to the new setting the security properties defined in the previous section, we need to modify the definition of the agents. First of all, we need to redefine the representation of knowledge, taking also partition fields into account: Let $Knowledge = \wp(Data) \cup \wp(Partition)$. Under the new interpretation of knowledge, $D(\phi)$ returns the set of entries and templates that can be composed using as data fields the data in ϕ (or $null$) and as partition field one of those in ϕ (or $\#$). For example, $\langle d_1, null\rangle^{[c]} \in D(\{d_1, c, c'\})$ and $\langle d_1, d_2\rangle^{[c'']} \notin D(\{d_1, d_2, c, c'\})$. The functions $dataset(e)$ and $dataset(t)$ returns the set of data fields stored inside the argument, hence it does not return the partition field. Once adopted these new definitions, the semantics of the new systems is given by the smallest labelled transition system satisfying the rule in Table 2.

We have previously observed that the partition field induces a partitioning of TS. This fact can be used in order to limit the access to a partition; it is enough to distribute the knowledge of the corresponding partition field value to the group of agents that has access permission to the partition. Exploiting this form of access control, the following example shows that, differently from Linda, we can define a system composed of two agents, say A and B, in which A sends an entry to B and the data are exchanged in a secure way.

Example 3.2. Private partition - Let (Ag, TS) be a system with empty tuple space, i.e. $TS = \emptyset$, and two agents A and B which exchange a datum through a partition with name c_s. More precisely, $Ag = \{A^{\{d,c_s\}}_{S_A}, B^{\{commit,c_s\}}_{S_B}\}$. Suppose that the first agent produces the entry $\langle d\rangle^{[c_s]}$, while the second one consumes an entry performing an $in(\langle null\rangle^{[c_s]}), r, S'_B)$ operation, and then communicates the consumed datum, say d', producing the entry $\langle commit, d'\rangle^{[c_s]}$. Formally, we assume that the unique actions the agents can perform are

$(S_A, \{d, c_s\}, out(\langle d\rangle^{[c_s]}), -, S_t) \in act$
$(S_B, \{commit, c_s\}, in(\langle null\rangle^{[c_s]}), r, S'_B) \in act$ (for any r)
$(S'_B, \{commit, d', c_s\}, out(\langle commit, d'\rangle^{[c_s]}), -, S_t) \in act$ (for any d')
where we assume that S_t is a terminating state, i.e. a state that does not permit the execution of any action.

If we assume that only A and B know c_s, we have that the partition corresponding to c_s can be accessed only by the two agents. In other words, if ϕ_E is the initial knowledge of the enemy, and if $c_s \notin \phi_E$, we have that all the properties $Secrecy((Ag, TS), \phi_E \setminus \{d\}, d)^2$, $Integrity((Ag, TS), \phi_E, d)$ and $Availability((Ag, TS), \phi_E, \{B\}, d)$ hold.

The proofs are rather simple, we just remind that, according to the definition of $matches^c$, in order to read or remove the entry written by A the enemy must be able to generate a template having the partition field set to c_s. But this is not possible because $c_s \notin \phi'_E$ for each ϕ'_E such that $(Ag \cup E^{\phi_E}_{S_E}, TS) \rightarrow^* (Ag' \cup E^{\phi'_E}_{S_E}, TS')$. Consequently, secrecy and availability hold because the enemy cannot neither read nor remove that entry.

To prove data integrity, we reason in a similar way, observing that the enemy cannot introduce any entry with partition field set to c_s. Thus, for each TS' such that $(Ag \cup E^{\phi_E}_{S_E}, TS) \rightarrow^* (Ag' \cup E^{\phi'_E}_{S_E}, TS')$, we have that the unique entry with partition field set to c_s is $\langle d\rangle^{[c_s]}$. This ensures that the unique datum that the agent B may commit is the expected datum d.

Finally, note that in order to prove the properties we have not made any assumption on d, hence they hold for any $d \in Data$.

Following this approach, we may implement also a form of *secure group communication*, simply by limiting the knowledge of a specific partition to the group of agents that need to communicate in a secure way.

More precisely, suppose we have the groups of agents $G_1, \ldots, G_n$. We associate to each group G_i a partition name $c_i \in C$, with $c_i \neq c_j$ for any $i \neq j$. We assume that c_i is known only by the agents that are members of G_i, and that they keep c_i secret.

In order to write or consume a datum into the space dedicated to G_i, it is sufficient to execute $out(\langle \vec{d}\rangle^{[c_i]})$ or $in(\langle \vec{dt}\rangle^{[c_i]})$, respectively. The rd operation can be treated similarly to in. Following the proofs above, it is possible to show that, if the knowledge of the enemy E does not contain any of the partition fields used by the groups, the data exchanged in those partitions satisfy secrecy, integrity and availability.

3.2. Scope of the partition fields

In the Example 3.2 we have assumed that the initial tuple space TS is empty. This assumption is rather unnatural, because in general applications running in open systems cannot impose the state of the tuple space when they start. However, the assumption is important for the proofs of the security properties because, in the case the partition name c_s is stored in some of the entries in the tuple space, the enemy could add it in its knowledge. Another important assumption we make in the example is that both A and B keep the name c_s secret, i.e. they do not introduce it in the tuple space.

We have made no assumption concerning the relation between $Data$ and $Partition$ so far, and this clearly influences the scope of partition fields. More precisely, consider that the two sets are separated,

[2] Obviously, if d is in the initial knowledge of E $Secrecy$ does not hold.

i.e. $Partition \cap Data$ is empty; in this case it is not possible to introduce in the tuples any partition field value. Under this assumption, we have that the scope of partition fields is *static*, i.e. the agents cannot change their knowledge of partition fields during the computation.

Under static scope of the partition fields, the assumptions taken in the example are not necessary because the enemy has no way to add to its knowledge the secret partition field c_s.

On the other hand, one could consider the possibility to introduce partition field values inside the tuples. Formally, we could consider $Partition \subseteq Data$. In this case we have that the scope of partition fields is *dynamic*, because the agents can increase the amount of partition fields that they know.

Differently from the static case, under dynamic scope of the partition fields, the assumptions taken in the example are necessary; consider, e.g., the case in which TS contains an entry e such that $c_s \in dataset(e)$. Using wildcards the enemy E can get the name c_s, and then it can add, read or remove entries from the partition related to c_s, thus invalidating the properties claimed in the example.

On the one hand, the static case offers the possibility to robustly control the access to the entries located in the partitions; on the other hand, the mechanism offers a limited flexibility because the permissions must be settled at the start-up of the system. On the contrary, the dynamic case is more vulnerable, but it permits a more flexible way to manage the access permissions; indeed, differently from the static case, they can be changed (even if only monotonically) every time the system needs to do it.

These two approaches clearly influence the way partition fields are distributed. E.g., in the Example 3.2 we have assumed the distribution of c_s as a precondition. In the static case, we have no manner to distribute the partition fields using the tuple space, thus another form of communication is needed to establish the partition to use. In the dynamic case, partition fields can be exchanged through the tuple space; the open problem is how to perform this phase in a secure way.

More precisely, when the partition fields are exchanged, it is important to authenticate the producer and the receiver of the entries containing this information. For example, an agent A that wants to communicate with B in a secure way through a fresh partition, i.e. a partition which is not currently used and known by the other agents in the system, settles this new partition and then communicates the corresponding partition field value through the tuple space. The agent A needs some guarantees that only B can read/consume this value from the tuple space, and also B wants to be sure to receive the name established by A, and not other partition value produced by malicious agents.

In the following subsection we formally model authentication of producer/receiver of an entry, and we prove that the partition fields are not expressive enough to ensure also these security properties.

Besides the distribution of the private partition field identifiers, another issues regards the "freshness" of the partition field. In Example 5.1 we will discuss a possible implementation of a partition field generator service that guarantees freshness and secure distribution of the partition values.

3.3. Producer and Receiver Authentication

In the Example 3.2 we have shown that private partitions can be used to exchange data in such a way that secrecy, integrity, and availability are supported. There exist other interesting properties that secure channels should usually support, e.g. the possibility to authenticate the producer and/or the receiver of a particular information communicated through the channel.

Consider, for example, an entry in the tuple space that represents an electronic payment sent from the agent A to the agent B. The agent A wants to be sure that the entry will be received only by B and, on the other hand, the agent B requires reasonable guarantees that A is the actual producer of the entry.

For example, it is important that A is not going to repudiate the payment, as well as that B is unable to recycle the payment, reintroducing it in the tuple space after its initial withdrawal. We refer to these security properties as *producer authentication* and *receiver authentication*, respectively.

Formally, we can model producer and receiver authentication, as follows.

Definition 3.2. Producer and Receiver Authentication - Let (Ag, TS) be a system and consider two agents $A^{\phi_A}_{S_A}, B^{\phi_B}_{S_B} \in Ag$. We assume $Ag = \{A_1{}^{\phi_1}_{S_1}, \ldots, A_n{}^{\phi_n}_{S_n}\}$.

- Let $A_i = A$ and e be an entry we want to test if only A can produce; we say that $ProdAuth((Ag, TS), A, e)$ holds if and only for any computation
  $(\{A_1{}^{\phi_1}_{S_E}, \ldots, A_{i-1}{}^{\phi_{i-1}}_{S_E}, A^{\phi_A}_{S_A}, A_{i+1}{}^{\phi_{i+1}}_{S_E}, \ldots, A_n{}^{\phi_n}_{S_E}\}, TS) \rightarrow^* (Ag', TS') \overset{out(e)}{\underset{-}{\longrightarrow}}_C$
  we have that $C = A$;

- Let $A_j = B$ and e be an entry we want to test if only B can read/remove; we say that $RecvAuth((Ag, TS), B, e)$ holds if and only if for any computation
  $(\{A_1{}^{\phi_1}_{S_E}, \ldots, A_{j-1}{}^{\phi_{j-1}}_{S_E}, B^{\phi_B}_{S_B}, A_{j+1}{}^{\phi_{j+1}}_{S_E}, \ldots, A_n{}^{\phi_n}_{S_E}\}, TS) \rightarrow^* (Ag', TS') \overset{op(t)}{\underset{r}{\longrightarrow}}_C$
  with $op \in \{rd, in\}$ and $e\ matches^c\ t$, we have that $C = B$.

The first part of this property ensures that if an agent writes e, then it is A. Note that this condition is tested for any behaviour that the agents of the system can have (keeping their original knowledge). This is obtained by substituting the states S_i with S_E. In this way we guarantee that, even if some components of the system are malicious, A is the only agent that can write e. The second property regards the ability to read a specific entry e; it ensures that, even assuming malicious behaviour of the the agents different from B, if an agent performs a rd/in operation using a template that matches with e, then it is B.

It is interesting to remark that, differently from the security properties given in Section 2.3, we define the producer/receiver authentication without explicitly introducing any enemy agent, but we introduce malicious behaviours of the agents composing the system. The same property could be equivalently defined performing the test on the system with enemy simply by fixing its initial knowledge equal to the union of the single knowledges of the agents in Ag (excluding that of A for producer authentication, and that of B for receiver authentication).

We are now ready to prove that the partition fields are adequate to support authentication only in the trivial case in which the producer and the consumer are the same agent. Such a result motivates the new kind of partition (the asymmetric partition) that we introduce in the following section; asymmetric partitions allow us to cover this deficiency for the producer/receiver authentication problem.

Proposition 3.1. Let (Ag, TS) be a system. Let $A^{\phi_A}_{S_A}$ and $B^{\phi_B}_{S_B}$ be two agents in Ag that want to exchange data via an entry $e = \langle \vec{d} \rangle^{[c]}$ in such a way that both $ProdAuth((Ag, TS), A, e)$ and $RecvAuth((Ag, TS), B, e)$ hold. If there exists a communication session in which A and B exchange the entry e, i.e.

$$(Ag, TS) \rightarrow^* (Ag', TS') \overset{out(e)}{\underset{-}{\longrightarrow}}_A (Ag'', TS'') \rightarrow^* (Ag''', TS''') \overset{op(t)}{\underset{\langle \vec{d} \rangle}{\longrightarrow}}_B ,$$

with $e\ matches^c\ t$, then $A = B$.

Proof:
It is enough to consider the property $ProdAuth((Ag,TS),A,e)$ which states that A is the unique producer of e even if all the agents but A are malicious (i.e. their state correspond to S_E). Suppose, proceeding by contradiction, that $B \neq A$. Consider now the agent $B^{\phi_B}_{S_E}$ that can perform also all the action of the agent $B^{\phi_B}_{S_B}$ of the initial system. Thus, by hypothesis, also this agent has the ability to execute (after the production of the entry e) an input/read operation $op(t)$, with $e\ matches^c\ t$, which returns $\langle\vec{d}\rangle$. After the execution of this operation this agent surely has in its knowledge both c and $\vec{d}$, hence it can produce the entry $\langle\vec{d}\rangle^{[c]} = e$. The fact that $B \neq A$ contradicts the hypothesis $ProdAuth((Ag,TS),A,e)$. □

4. Asymmetric Partitioning of the Tuple Space

In the previous section we have proposed to logically partition the tuple space, in such a way that an agent can access a partition only if it is able to show a proof of knowledge of the partition field value. If an agent can access a partition, then it can perform write, read, and take operations on that partition.

We have formally proved that even if this approach can support secrecy, authenticity, and integrity, it is possible to support producer and receiver authentication only in the trivial case in which the producer and the consumer are the same agent.

This limitation of partition field is a consequence of the fact that they are not expressive enough to discriminate between the producers and the readers/consumers of a datum. As described in the proof of Proposition 3.1, when an agent is able to read an entry, then it is also able to reproduce that entry. The same holds also for producers; when an agent produces an entry, it can also read or consume that entry.

However, many applications need to support this form of discrimination. Consider, for example, a client-server service (such as a print service): clients send the requests and the server is the only agent that collects them. Using a tuple space to implement the model, the client submits the work writing an entry, and the server consumes the requests. It is important to allow the permission to read the submitted jobs only to the server. Furthermore, several applications need to authenticate the sender or the receiver of a specific entry. For example, the distribution of software: a software house distributes its software products, upgrades and patches, writing them into the space. In order to avoid to download and install uncertified code, the receiver requires the authentication of the data producer.

In order to support these class of applications, the following section introduces entries with a new field of control, named *asymmetric partition field*.

4.1. Asymmetric partition fields

As commented above, we intend to use asymmetric partition fields to distinguish between write and read/input permissions, and to provide a manner for authenticating the sender ("who has written the entry"), and the receiver ("who is executing the read/input").

The idea is to consider a new class of partitions which have the following characteristics: given a partition, there exists two corresponding partition field values K and $\overline{K}$, the former used to introduce data in the partition, and the latter used to retrieve data. In other words, given an entry containing the control information K, an agent can access (read or remove) that entry only if it is able to show a proof of knowledge of $\overline{K}$. In this way, if K is known only to the agent A, and $\overline{K}$ is known only to the agent B, then the produced and the consumer of the entry are ensured to be A and B, respectively.

Formally, let $APartition$, ranged over by K, K', K_t, $\ldots$, be a denumerable set of asymmetric partition fields. We also assume that $APartition$ contains a special default value ?, whose meaning is similar to # for partition fields. Let $\bar{\cdot} : APartition \rightarrow APartition$ be a function, that is used to map K into $\overline{K}$, such that if $\overline{K} = K'$ then $\overline{K'} = K$ (and $\overline{?} = ?$).

The idea is to exploit this new kind of fields in the matching rule, in order to capture a more sophisticated relation between entries and templates. In particular, we exploit the fact that the information that the agents must know in order to either read/input or write an entry are different. Hence, limiting the knowledge of this information, we can distinguish the write and the read/input permissions. An entry e, and a template t, with asymmetric partition (and partition) fields, are defined as follows:

$$e = \langle \vec{d} \rangle^{[c]}_{[K]}$$

$$t = \langle \vec{dt} \rangle^{[c_t]}_{[K_t]}$$

Let $Entry^c_k$ and $Template^c_k$ be the set of entries and templates with asymmetric partition (and partition) fields, respectively. In the following, $e, e', \ldots$, range over $Entry^c_k$, and $t, t', \ldots$, range over $Template^c_k$. We define $e|_d$ as the operator that, given an entry $e = \langle \vec{d} \rangle^{[c]}_{[K]}$, returns the tuple of data $\langle \vec{d} \rangle$.

Definition 4.1. Matching rule with asymmetric partition fields - Let $e = \langle \vec{d} \rangle^{[c]}_{[K]}$ be an entry, and $t = \langle \vec{dt} \rangle^{[c_t]}_{[K_t]}$ be a template; we say that $e \; matches^c_k \; t$ if the following conditions hold:

- $K = K_t = ?$ or $\overline{K} = K_t$;
- $\langle \vec{d} \rangle^{[c]} \; matches^c \; \langle \vec{dt} \rangle^{[c_t]}$.

The definition of $matches^c_k$ is a conservative extension of $matches^c$ (see Definition 3.1).

Observe that the asymmetric partition field ? is used as a default *do-not-care* field; it can be used, e.g., to produce and retrieve entries for which it is not important to show the knowledge of any specific asymmetric partition field. Therefore, for the sake of simplicity, the asymmetric partition field is sometimes omitted when it is ?.

Example 4.1. matches^c_s evaluation - Let $K \neq ?$ and $K_t = \overline{K}$ be asymmetric partition fields. A list of entry matching examples follows.

template	**does it match with $\langle d \rangle_{[K]}$?**
$\langle null \rangle_{[?]}$	no
$\langle null \rangle_{[K_t]}$	yes
$\langle d \rangle_{[K']}$	no, if $K' \neq K_t$

The semantics of the primitives, using the entry (template) with asymmetric partition field and the matching rule $matches^c_k$, is obtained simply by replacing in Table 1 $matches$ with $matches^c_k$, and the e, used to describe the return value, with $e|_d$ (in this context $ReturnValue = \{e|_d \mid e \in Entry^c_k\} \cup \{-\}$). Observe that also in this case the asymmetric partition fields are used only to control the access to the entries, and the return value does not include them.

In this case, the fact that the additional control fields of an entry is not returned as the result of a read/consume operation is a crucial aspect. Consider e.g. an application in which we want to use the asymmetric partition field K to authenticate the entries produced by an agent with identity A. To this aim, it is fundamental that K is a secret known only by the agent A during the overall computation. The agent A will add the asymmetric partition field K to those data that will need to be authenticated. Suppose now that another agent reads/consumes the authenticated entry; in the case it receives also the information K, then K will become part of its knowledge, thus contrasting the expected secrecy of K.

The asymmetric partition field presents some similarities with the asymmetric field and object lock used by SecOS. However, differently from SecOS, our control fields are associated only to the entries (and not to the data fields) and, more important, the return value does not contain the control fields. A more detailed comparison is made in Section 6.

In order to model the agents under the new kind of control fields, we can adopt the rule in Table 2, simply by extending the set $Knowledge$ that now must contain also the asymmetric partition fields, i.e. $Knowledge = \wp(Data) \cup \wp(Partition) \cup \wp(APartition)$, and assuming that $D(\phi)$ returns the set of entries and templates that can be generated by using the data fields, the partition and the asymmetric partition fields in ϕ (plus $null$, $\#$, and ?).

Also in this case, we can assume the scope of the asymmetric partition field either static or dynamic. For the reason described in the Section 3.2, the dynamic approach is more flexible; for this reason, we consider the dynamic approach.

We are now ready to show that the addition of asymmetric partition fields is enough to support both producer and receiver authentication. Indeed, if we consider two asymmetric partition fields K and $\overline{K}$, and we assume that K is known only by the producer while $\overline{K}$ is known only by the receiver, then all the data exchanged using the asymmetric partition fields K and $\overline{K}$ are authenticated. The following proposition defines sufficient conditions which implies authentication.

Proposition 4.1. Let (Ag, TS) be a system, $A^{\phi_A}_{S_A}, B^{\phi_B}_{S_B} \in Ag$ be two agents. Let K and $K' = \overline{K}$ be two asymmetric partition fields. If the following conditions hold:

i) $K \in \phi_A$;

ii) $K' \in \phi_B$;

iii) $Secrecy((Ag, TS), \phi_E, K)$ holds for $\phi_E = \bigcup_{e \in TS} dataset(e) \cup \bigcup_{C^{\phi}_{S} \in Ag, C \neq A} \phi$;

iv) $Secrecy((Ag, TS), \phi_E, K')$ holds for $\phi_E = \bigcup_{e \in TS} dataset(e) \cup \bigcup_{C^{\phi}_{S} \in Ag, C \neq B} \phi$.

then for any entry e with asymmetric partition field set to K, we have that $ProdAuth((Ag, TS), A, e)$ and $RecvAuth((Ag, TS), B, e)$ hold.

Proof:
To prove producer authentication it is enough to observe that if during a computation an agent $C^{\phi}_{S} \in Ag$ performs $out(e)$, then the agent C must have $e \in D(\phi)$, thus also $K \in \phi$. By iii) only A can have K in its knowledge during all the computation, thus $C = A$.

Similarly, to prove receiver authentication we simply observe that a template t matching with e must have asymmetric partition field set to K', but iv) ensures that K' can be in the knowledge of no agent but B. □

4.2. An implementation of asymmetric partition fields

We have described the asymmetric partition fields as a generic set of names equipped with a particular function $\bar{\cdot}$ which associate to each field K the corresponding matching field $\overline{K}$.

There are two fundamental features that the function $\bar{\cdot}$ must satisfy. The former is that the coordination middleware (that is the implementation of the tuple space) should be reasonably able to check whether two fields K and K' match (i.e. verify if $K' = \overline{K}$). The latter regards the impossibility for an agent to guess $\overline{K}$ starting from the knowledge of K, more formally, any agent that has in its knowledge K, has no reasonable way to guess $\overline{K}$ until it explicitly reads this information from the tuple space.

The section presents a possible way to implement the asymmetric partition fields and the function $\bar{\cdot}$ exploiting asymmetric cryptography (see e.g. [Sch96]). To this end, we first give some formal definitions:

- The set of plaintexts *PlainText*, ranged over by $p, p', \ldots$;

- The set of encryption keys *Key*, ranged over by k, k', $PrivK$, $PubK$, $\ldots$, containing private and public keys. As usually done when cryptographic protocols are analysed (see e.g. [DY83]), we assume a perfect version of cryptographic operations. Hence, in the following, when we refer to pairs of private and public keys $(PrivK, PubK)$, we assume that a plaintext encrypted with $PubK$ (resp. $PrivK$) can be decrypted only using $PrivK$ (resp. $PubK$);

- The set $Ciphertext$, ranged over by s, $s_t, \ldots$, that contains the ciphertexts obtained encrypting $p \in PlainText$, with $k \in Key$, denoted with $\{p\}_k$.

Any non trivial asymmetric partition field, i.e. those different from ?, is encoded as a triple $(p, PubK, s)$ composed of a plaintext p, one public key $PubK$ and one ciphertext s. Suppose that we have two asymmetric partition fields $(p, PubK, s)$ and $(p', PubK', s')$; two fields match if and only if the decryption with $pubK'$ of the ciphertext s produces the plaintext p' and, symmetrically, the decryption with $pubK$ of the ciphertext s' produces the plaintext p. This condition holds if and only if s corresponds to $\{p'\}_{PrivK'}$ and s' corresponds to $\{p\}_{PrivK}$.

More precisely, the implementation is as follows: the asymmetric partition field ? is implemented with a default value $?'$, while all the other fields are encoded as triples $(p, PubK, s)$. The function $\bar{\cdot}$ is defined as follows:

- given $?'$, we have that $\overline{?'} = ?'$;

- given the triple $(p, PubK, s)$, we have that $\overline{(p, PubK, s)} = (p', PubK', s')$ if $s = \{p'\}_{PrivK'}$ and $s' = \{p\}_{PrivK}$.

Observe that this implementation requires that, in order to produce an asymmetric partition field K, an agent should know a private key $PrivK'$ as well as the public key $PubK$ corresponding to the private key $PrivK$ used to produce $\overline{K}$. In light of this observation, we can conclude that it is not reasonably possible for an agent to generate $\overline{K}$ given only K, because asymmetric cryptography ensure that it is not reasonably possible to produce the private key $PrivK$, given the corresponding public key $PubK$.

4.3. Examples

In order to prove the adequacy of asymmetric partition field to solve the security issues that we have tackled, this section presents several examples. In particular, we hint how to solve problems such as distinguishing the *write* and *read* permission, as well as the authentication of the sender and of the receiver of entries.

4.3.1. Write and Read/Input Permissions

Using asymmetric partition fields it is possible to avoid the unauthorized replication of entries, i.e. avoid that a reader/consumer of an entry is always able to reproduce that entry. This feature can be useful in several applications, e.g., to distinguish write and read/input permissions. On the contrary, if the readers can write the entries that they read, this means that read permissions implies also write permissions.

Let K and $K' = \overline{K}$ be asymmetric partition fields. Suppose that only agent A knows and keeps secret K, whilst K' is known to all the other agents. Thus, we have that A can write an entry of the form $\langle\vec{d}\rangle_{[K]}$ (assuming that also $\vec{d}$ is in its knowledge) that any other agent can read (or consume). By definition, the return value of the read (or input) operation does not include the asymmetric partition field. Hence, the reader can subsequently produce a new entry with the same data fields $\vec{d}$, but a different asymmetric partition field. In this way, the new entry cannot be exactly the same of the initial one. Therefore, excluding the case when the reader is the sender, the reader cannot reintroduce the entry in the space.

Formally, let (Ag, TS) be a system in which $A^{\phi}_{S} \in Ag$, $\langle\vec{d}\rangle_{[K]} \in D(\phi)$ and $TS = \{\langle K'\rangle, \langle\vec{d}\rangle\}$. We suppose that K is known only by the agent A; i.e. for any $B^{\phi'}_{S'} \in Ag$ with $B \neq A$ then $K \notin \phi'$. We consider that the agent A simply produces the entry $\langle\vec{d}\rangle_{[K]}$, i.e. the unique actions in act starting from the state S is $(S, \phi, out(\langle\vec{d}\rangle_{[K]}), -, S_t)$ (where S_t is a terminating state).

It is not difficult to prove that $Secrecy((Ag, TS), \phi_E, d)$ holds for any ϕ_E that does not contain K (i.e. $K \notin \phi_E$). As a consequence of this fact we have that if $(Ag, TS) \rightarrow^* (Ag', TS') \xrightarrow[\underline{\ }B]{out(\langle\vec{d}\rangle_{[K]})}$ then we have that $B = A$, because only A knows K (i.e., it is the unique agent with the necessary knowledge to set K as asymmetric partition field).

Symmetrically, one may wonder if write permissions include also read/input permission, that is, if an agent can always read/consumes the entries that it produces. The answer is negative, in particular, this is not permitted when an entry has K as asymmetric partition field and the producers does not know the corresponding $\overline{K}$. For instance, all agents of the system described above have the ability to produce the entry $e = \langle\vec{d}\rangle_{[K']}$ because K' and $\vec{d}$ are public. However, only the agent A holds K, thus it is the unique agent that can read e.

Therefore, using the asymmetric partition field, we obtain the maximum flexibility to assign the write and read/input permissions simply by distributing, in the correct way, the knowledge of the asymmetric partition fields.

4.3.2. Secure channels

In the Example 3.2 we have shown that private partitions can be used to exchange data in such a way that secrecy, integrity, and availability are supported. Here, we show that a secure channel can be simulated using the implementation of the asymmetric partition fields in term of asymmetric cryptography presented in Section 4.2. This encoding of secure channels guarantees also producer and receiver authentication.

We make the standard assumption that the involved agents have settled their private keys, and published the corresponding public keys. Let $PrivK_A$ (resp. $PrivK_B$) be the private key of A (resp. B), and let $PubK_A$ (resp. $PubK_B$) be the public key of A (resp. B). The emission of a datum d on the secure channel can be performed by the agent A simply be executing an $out(\langle d\rangle_{[p,PubK_B,\{p\}_{PrivK_A}]})$ operation. Symmetrically, a receive operation on the secure channel can be performed by B simply executing an $in(\langle null\rangle_{[p,PubK_A,\{p\}_{PrivK_B}]})$ operation.

According to the discussion about the write and read/input permissions reported above, formalized in Proposition 4.1, we can clearly see that A is the unique agent which has write capabilities on the channel, while B is the unique agent with the input capability.

Differently from the channel implemented with private partition of Example 3.2, here we can exploit the different permissions to authenticate the producer and the consumer of a datum.

A consideration on p is needed; p is a plaintext on which A and B agrees in order to implement the secure channel. Different approaches could be followed in order to settle p. On the one hand, p could be a public datum which is always used by A and B everytime they need to communicate through the secure channel. On the other hand, p could be refreshed during each communication sessions simply by, e.g., transmitting a new plaintexts p' (to be replaced to p) within each tuple communicated through the secure channel. The latter approach is inspired by the typical technique of *nonces* (numbers used only once) used in cryptographic protocols in order to ensure the freshness of the exchanged messages, thus contrasting the so-called reply attacks.

4.4. Partition/Asymmetric Partition Fields and Symmetric/Asymmetric Cryptography

Security protocols based on channel based communication, typically use symmetric and asymmetric cryptography, in order to guarantee security properties (e.g. secrecy, authentication). In this subsection we discuss interesting similarities between symmetric and asymmetric cryptography with the partition fields and asymmetric partition fields that we have introduced in the context of Linda-like coordination.

Suppose that the agents A and B in a cryptographic system are the unique agents who knows the symmetric key K_{AB}; then they are also the unique agents that can encrypt data with K_{AB}, as well as decrypt the obtained ciphertext. Similarly, if two agents A and B in our coordination system share (secretly) the partition c_{AB}, then only A and B can write, read, and consume any entry $e = \langle\vec{d}\rangle^{[c_{AB}]}$. Thus, we have that either solutions can be used to guarantee the secrecy of the data exchanged between A and B. Moreover, in either solutions there is no way to authenticate who produces, or who receives, the data.

We have that asymmetric cryptography supports interesting extra functionalities: (i) signature of data in order to allow the receiver to authenticate the sender (by encrypting the data with the private key of the sender); (ii) emission of encrypted message, in such a way that it is guaranteed that only the expected receiver can decrypt it (by encrypting the data with the public key of the receiver). We have shown above that the same functionalities can be obtained in coordination infrastructures exploiting asymmetric partition fields.

A final remark regards a more detailed comparison between the coexistence in our coordination model of both partition fields and asymmetric partition fields. It is not difficult to see that the former can be implemented in the latter. Let us consider, e.g., a system (Ag, TS), and a partition field c; we encode this partition field using two related asymmetric partition fields, namely K_c and $\overline{K_c}$. Given an agent $A_S^{\phi} \in Ag$ who knows c (i.e. $c \in \phi$), we replace ϕ with $\phi' = \phi \setminus \{c\} \cup \{K_c, \overline{K_c}\}$. Any output operation $out(\langle\vec{d}\rangle^{[c]})$ on partition c is encoded as $out(\langle\vec{d}\rangle_{[K_c]})$, whilst any input operations $in(\langle\vec{dt}\rangle^{[c]})$ is encoded as $in(\langle\vec{dt}\rangle_{[\overline{K_c}]})$ (and similarly for read operations).

Even if the standard partition fields can be simply encoded in terms of the asymmetric partition fields, we continue to support both for two main reasons. The first one concerns efficiency: the verification of perfect matching between two partition fields is surely more efficient than the verification of correspondence of two partition fields K and K' (i.e. verify whether $K' = \overline{K}$). For example, in the implementation described in Section 4.2 this latter check requires to use a decryption (or encryption) algorithm twice. A second interesting aspect is related to an interesting use that can be made of these two kinds of partition fields. For example, under public key infrastructures, a communication session could be based on (more efficient) symmetric keys, which are freshly generated before the session, and exchanged using (the less efficient) asymmetric keys. Also in our coordination infrastructure, asymmetric partition fields could be used in order to exchange newly generated partition fields, that can be used to share data during a specific session.

5. Destructive and Non-Destructive Retrieve Operations

There are examples of applications in which several (or all) agents can read some data, but only a subset of them can delete such data. For example, a person can publish some data (e.g. name, phone, e-mail, ..), writing an entry in the tuple space. It is reasonable to assume that this information cannot be removed by any other person, excluding the owner (for example, in order to update the published data). To support this, it is necessary to distinguish between read and input permissions.

To accomplish this goal, the asymmetric partition fields are not sufficient. For example, if an agent A has the ability to read a specific entry e, this means that the agent A can generate a template t such that $e\; matches_k^c\; t$; this template can be used also for an input operation.

In this section we present how to extend the entries in order to discriminate among their readers and consumers. The idea is simply to use two groups of control fields, and use the first group for non-destructive (rd) operations, and the second one for destructive (in) operations.

Formally, let $Entry_k^c(rd, in)$ be the set, ranged over by e, e', $\dots$, of entries with read and input permissions, defined as follows:

$$e = \langle\vec{d}\rangle^{[c]_{rd}[c']_{in}}_{[K]_{rd}[K']_{in}}$$

We define $e|_d$ as the operator that, given an entry $e = \langle\vec{d}\rangle^{[c]_{rd}[c']_{in}}_{[K]_{rd}[K']_{in}}$, returns the tuple of data $\langle\vec{d}\rangle$.

The control fields with the subscript $_{rd}$ are used in the case of rd operations; those with subscript $_{in}$ are used in in operations. The template continues to be defined as in the previous section (that is $t \in Template_k^c$); indeed templates are used in operations $rd(t)$ or $in(t)$, and the indication of the kind of retrieve operation (either destructive or non-destructive) is inherited by the operation itself (either in or rd, respectively).

Definition 5.1. Matching rule with permissions - Let $op \in \{rd, in\}$ be the primitive on which the matching must be evaluated, t be a template and $e = \langle\vec{d}\rangle^{[c]_{rd}[c']_{in}}_{[K]_{rd}[K']_{in}}$ be an entry; we define e $matches_k^c(op)\; t$ as follows:

- $\langle\vec{d}\rangle^{[c]}_{[K]} matches_k^c\; t$, if $op = rd$.

- $\langle\vec{d}\rangle^{[c']}_{[K']} matches^c_k\ t$, if $op = in$.

In other words, $matches^c_k(op)$ is reduced to $matches^c_k$ using only the control fields of e corresponding to op.

It is trivial to prove that $matches^c_k(op)$ is well defined; in both cases $matches^c_k$ is applied to entry $e \in Entry^c_k$ and template $t \in Template^c_k$.

The semantics of the primitives, using the entry with permission and the matching rule $matches^c_k(op)$, is obtained simply by replacing in Table 1 $matches$ with $matches^c_k(op)$, and the e used to describe the return value with $e|_d$ (in this context $ReturnValue = \{e|_d \mid e \in Entry^c_k(rd, in)\} \cup \{-\}$).

The calculus we have defined in this section comprises the standard partition fields, the asymmetric partition fields, and permits also the indication of two different policies for read and input operations. As it comprises all the three new mechanisms we propose, this is the calculus which formally define the coordination model `SecSpaces`.

5.1. Examples

It is interesting to observe that the combination of the possibility to discriminate between out and rd/in permission (supported by the asymmetric partition fields as discussed in Subsection 4.3.1) plus the possibility to discriminate between the rd and in permissions introduced in this section, provide the maximal flexibility in the definition of the whole spectrum of permissions.

The two examples in this section report two typical patterns of interactions: a one-to-many communication in which there is a producer and many possible consumers, and a many-to-many communication in which the receivers can only read the data, that can be removed only by the producers.

Example 5.1. Distribution of fresh partition names - In the Example 3.3 we have discussed about the problem of generating fresh partitions to be used as private partitions during a secure communication session. Here, we present the description of a possible solution to this problem based on a server that produces the fresh partition field identifiers.

The idea is that a trusted agent behaves as the server responsible for the generation of the fresh partition field identifiers. When a client agent in the system needs to use a fresh partition field, it simply consumes one of the identifiers produced by the server. Following this approach, it is important to guarantee the authentication of the server, in order to give guarantees to the clients that they have consumed an entry generated by the server. The other important issue is that two different agents should not access the same fresh partition identifier. This can be ensured by giving to the clients the input permission, and not the read permission to the tuples containing the fresh partition field identifiers.

The specification of the service is as follows. Let (Ag, TS) be a system, let K, K' and K_s, K_r be two pairs of asymmetric partition fields such as $K' = \overline{K}$ and $K_r = \overline{K_s}$. We assume that K, K' and K_s are known only to the server who keep them secret, while K_r is public.

Each fresh partition field c can be communicated by the server simply by producing the entry $\langle c\rangle_{[K]_{rd}[K_s]_{in}}$. The clients can get a fresh partition field simply by performing the operation $in(\langle null\rangle_{[K_r]})$.

Observe that producer authentication is ensured by the fact that the server is the unique agent who knows K_s. Moreover, the clients have only input permission on these entries and not rd permission because they are aware of K_r while they do not know K'. Only the server (the unique agent who knows K') has rd permissions.

Example 5.2. Many-to-many communication - In this example we show how to encode the many-to-many communication model, distinguishing the permission of writing, reading and consuming the entries. The model we present extends the secure group communication discussed in the Example 3.2 by adding an access control policy that discriminates between the read, remove and output permission.

Let (Ag, TS) be a system, K_r and $K_{w_r} = \overline{K_r}$, K_{w_i} and $K_i = \overline{K_{w_i}}$ be two pairs of asymmetric partition fields. The partition fields K_r and K_i are used to access the entries during read and input operations, respectively (e.g. $rd(\langle null\rangle_{[K_r]})$ and $in(\langle null\rangle_{[K_i]})$). On the other hand, K_{w_r} and K_{w_i} are used to produce the entries (e.g. $out(\langle d\rangle_{[K_{w_r}]_{rd}[K_{w_i}]_{in}})$).

We assume that the agents in Ag are divided into three groups: $Ag_w \subseteq Ag$ is the group of writers; $Ag_r \subseteq Ag$ is the group of agents that can perform only rd operations; $Ag_i \subseteq Ag$ is the group of agents that can perform only in operations. The partition fields identifiers are distributed as follows:

i) if K_{w_r} (resp. K_{w_i}) is in ϕ then also K_{w_i} (resp. K_{w_r}) is in ϕ and $A_S^\phi \in Ag_w$;

ii) if K_r is in ϕ then $A_S^\phi \in Ag_r$;

iii) if K_i is in ϕ then $A_S^\phi \in Ag_i$.

We assume also that these asymmetric partition fields identifiers are not present in the initial tuple space TS (i.e. $K_r, K_{w_r}, K_{w_i}, K_i \notin \bigcup_{e\in TS} dataset(e)$).

To be as general as possible, without specifying the whole system, we simply assume that agents do not distribute the access permissions K_r, K_{w_r}, K_i and K_{w_i} and use the correctly asymmetric partition fields, that is:

- for any (Ag', TS') such that $(Ag, TS) \rightarrow^* (Ag', TS') \overset{out(\langle\vec{d}\rangle_{[K]_{rd}[K']_{in}})}{\underset{-}{\longrightarrow}}{}_A$ then $K_r, K_{w_r}, K_{w_i}, K_i \notin dataset(e)$, $K = K_{w_r}$ and $K' = K_{w_i}$;

- for any (Ag', TS') such that $(Ag, TS) \rightarrow^* (Ag', TS') \overset{op(\langle\vec{dt}\rangle_{[K]})}{\underset{r}{\longrightarrow}}{}_A$ then if $op = rd$ then $K = K_r$, if $op = in$ then $K = K_i$.

Let us consider the system with the enemy; if ϕ_E is a knowledge set such that $K_r, K_{w_r}, K_i, K_{w_i} \notin \phi_E$; the system $(Ag \cup E_{S_E}^{\phi_E}, TS)$ is such that:

- only agents in Ag_w can write entries;
- only agents in Ag_r can read (without consuming) entries;
- only agents in Ag_i can consume entries.

The approach we have followed is similar to the encoding of the secure channel (see Subsection 4.3.2). However, the assumption that asymmetric partition fields K_r, K_i, K_{w_r} and K_{w_i} are known by more than one agent does not allow to authenticate the agents that write, read or consume the entries. In the special case in which Ag_w is composed by one agent only, the authentication of the producer of an entry holds. Similarly, if Ag_r (resp. Ag_i) is composed by one agent only, the authentication of the reader (resp. consumer) holds. The proof of these properties is essentially the same as the one given in Proposition 4.1.

6. Related work

In the Introduction we have already discussed two proposals, KLAIM and SecOS, that aim to solve the problem of adding access control policies to the primitives of Linda.

Here we want to add one further observation. The symmetric and asymmetric field locks of SecOS are used to select specific fields inside entries, hence they provide also a way to assign the permission for accessing the fields. Differently, in SecSpaces control fields limit the access to the entries. However, this does not prevent the possibility of limiting the access to a specific field of an entry. Let d_i be a field contained in the entry $e = \langle d_1, \dots, d_n \rangle^{[c]_{rd}[c']_{in}}_{[K]_{rd}[K']_{in}}$. In order to restrict furtherly the access to d_i, we can proceed simply by splitting e into two entries. The first one is $e' = \langle d_1, \dots, d_{i-1}, d_e, d_{i+1}, \dots, d_n \rangle^{[c]_{rd}[c']_{in}}_{[K]_{rd}[K']_{in}}$ where d_e is a fresh datum used as a pointer to the second entry $e'' = \langle d_e, d_i \rangle^{[a]_{rd}[a']_{in}}$. Agents that can read (resp. remove) e' are exactly those that can read (resp. remove) e. On the other hand, in order to read (resp. remove) the field d_i stored inside e'' agents must know the pointer d_e and the partition field a (resp. a'). Thus, we can exploit a (resp. a') to restrict the access to d_i. If necessary, instead of partition fields a and a', it is possible to use also asymmetric partition fields in order to control the access to d_i. Using this idea, we can repeatedly split the entry until we obtain a distinct access permission for any data field.

Finally, some comments on the different definition of return value follow. We have already observed that in SecOS an agent able to read an entry is also able to reproduce that entry. In SecOS the return value of a read (or input) operation is exactly the matching entry that has been found in the space, hence it contains both data fields and fields/objects locks. In this way, the knowledge of the agent contains all the fields, even if some of them are inaccessible. Thus, malicious agents may use these chunks of data even if they do not know the information they contain. On the other hand, in SecSpaces when an agent accesses an entry, the return value does not contain the control fields. In this way, it is possible to avoid the possibility for the reading agent to use these information in subsequent operations.

Here we report a list of other significant proposals present in the literature that consider security issues in the setting of tuple space coordination. Klava [BN03] exploits cryptography to support data secrecy. It introduces encrypted messages into the fields of the entries; the encryption of messages ensures that they can be correctly interpreted only by the allowed clients. Other two proposals introduce security mechanisms in Lime [MPR01] (Linda in a Mobile Environment), one at the implementation level [HR03] and the other one by extending the formal model [CVV]. The former is a secure implementation of Lime that provides a password-based access control mechanism at the level of tuples and tuple spaces. More precisely, the password-based system on tuple space and tuples permits the access only to the authorized agents, that is those knowing the password. Password-based access permissions on entries can be associated to the read and to the removal operations. In the case an agent is allowed to remove an entry (i.e. it knows the password associated to the removal operations), it has also the permission of reading that entry. The latter proposes a formal model describing basic features of Lime and then extends it in order to support security. In particular, it is extended by adding a capability system that allows the owner of a tuple space to grant capabilities to access that space. The proposed capability system has some similarities with KLAIM, indeed capabilities are at the level of tuple space and they express which operations are permitted to each process allowed to access the tuple space.

Two proposals that consider access rights as in KLAIM are [MMU01] and [COZ00]. In [MMU01] the permission policies (laws) are described using Prolog rules; the agents have the correct access rights if they satisfy the rules. This check is executed by policy-independent trusted controllers. In [COZ00] the relation between coordination and access control in open, distributed agents systems is investigated in the context of the TuCSoN [OZ99] coordination model. More precisely, the possibility to program the coordination medium and the hierarchical topology supported by TuCSoN are exploited to support an access control to the shared resources, which is expressed in terms of an access control matrix. On the other hand, in [Pin92] an access control policy is defined which is similar to the one supported by SecOS. The idea is to provide Linda with directed communication via private channels. The solution exploits asymmetric cryptography: at each entry corresponds a special data, the ticket. The matching is available only if the tickets of entry and template match (i.e. one of them is the public ticket and the other one is the corresponding private ticket).

A different approach is proposed in [ORV03], where a general model for coordination middlewares that exploits process handlers to control the behaviour of processes has been presented. More precisely, a context is associated to each process and it defines which operations are allowed to the process. In order to express allowed actions, a language derived from CCS is used to describe which operations processes can perform.

A final remark regards the fact that we support the definition of the access control policies by introducing extra information inside the tuples. These extra information are usually called *attributes*, and they can be used not only to support secure coordination but also in many other contexts. A survey of the possibilities provided by the attributes can be found in [Woo99] where several manners to introduce the attributes in tuple space coordination are discussed, in particular how to insert them to the level of fields, of entries and of tuple spaces.

7. Conclusion and Future Work

In this paper we have proposed `SecSpaces`, a Linda-like coordination model that permits to control the access to the entries stored in the shared data space, and to authenticate/identify the producer of an entry or its reader/consumer. `SecSpaces` essentially introduces a new advanced matching rule, which exploits two new kinds of fields, namely partition and asymmetric partition fields.

In order to be suitable for open applications, in `SecSpaces` all the agents can always access the tuple space using the coordination primitives, in other words, `SecSpaces` permits to define permissions at the level of tuples and not at the level of tuple space. For example, an agent can always introduce entries in the tuple space. In this way, e.g., the quality of the coordination service cannot be guaranteed: an agent, indeed, can write a huge amount of entries into the space degrading the performance of the coordination service itself. As discussed in the introduction, we consider the definition of access control policies at the level of tuple spaces an orthogonal task, and we leave this as a future work.

We want to conclude by remarking that we have considered a subset of the common Linda primitives. More precisely, we have not taken into account the non-blocking versions of rd and in operations, usually denoted as rdp and inp, respectively. The analysis of an extended model including these operations is left as future work. Actually, we feel that non-blocking operations increase the power of the enemy, because in some cases they could be used to test not only the presence, but also the absence, of a particular entry.

Acknowledgments

We thank Riccardo Focardi for useful comments on a preliminary version of this paper.

References

[AG99] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[BN03] L. Bettini and R. De Nicola. A Java Middleware for Guaranteeing Privacy of Distributed Tuple Spaces. In E. Astesiano N. Guelfi and G. Reggio, editors, *Proc. of FIDJI'02, Int. Workshop on scientific engineering of distributed Java applications*, volume 2604 of *LNCS*, pages 175–184. Springer-Verlag, 2003.

[COZ00] Marco Cremonini, Andrea Omicini, and Franco Zambonelli. Coordination and access control in open distributed agent systems: The TuCSoN approach. In António Porto and Gruia-Catalin Roman, editors, *Coordination Languages and Models*, volume 1906 of *LNCS*, pages 99–114. Springer-Verlag, 2000. 4th International Conference (COORDINATION 2000), 11–13 September 2000, Limassol (Cyprus), Proceedings.

[CVV] Bogdan Carbunar, Marco Tulio Valente, and Jan Vitek. Coordination and Mobility in CoreLime. *Mathematical Structures in Computer Science*. To appear.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, 1983.

[FGM00] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In U.Montanari and E.Welzl, editors, *International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*. Springer, July 2000.

[FGM03] R. Focardi, R. Gorrieri, and F. Martinelli. A Comparison of Three Authentication Properties. *Theoretical Computer Science*, Vol. 291(3):285–327, 2003.

[Gel85] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[GJS96] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[HR03] Radu Handorean and Gruia-Catalin Roman. Secure sharing of tuple spaces in ad hoc settings. In Riccardo Focardi and Gianluigi Zavattaro, editors, *Electronic Notes in Theoretical Computer Science*, volume 85. Elsevier, 2003.

[MMU01] N. Minsky, Y. Minsky, and V. Ungureanu. Safe Tuplespace-Based Coordination in Multi Agent Systems. *Journal of Applied Artificial Intelligence*, 15(1), 2001.

[MPR01] A. Murphy, G. Picco, and G.-C. Roman. A middleware for physical and logical mobility. In *21st International Conference on Distributed Computing Systems*, pages 524–533, 2001.

[NFP97] R. De Nicola, G. Ferrari, and R. Pugliese. Coordinating Mobile Agents via Blackboards and Access Rights. *In Proc. of the Second International Conference on Coordination Models and Languages, Lectures Notes in Computer Science 1282, Springer*, pages 220–237, 1997.

[NFP98] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998. Special Issue: Mobility and Network Aware Computing.

[ORV03] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Formal specification and enactment of security policies through agent coordination contexts. In Riccardo Focardi and Gianluigi Zavattaro, editors, *Electronic Notes in Theoretical Computer Science*, volume 85. Elsevier, 2003.

[OZ99] Andrea Omicini and Franco Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, September 1999. Special Issue: Coordination Mechanisms for Web Agents.

[Pin92] J. Pinakis. Providing directed communication in Linda. *In Proceedings of the 15th Australian Computer Science Conference*, pages 731–743, 1992.

[Sch96] B. Schneier. *Applied Cryptography*. Wiley, 1996.

[Sun02] Sun Microsystems, Inc. *JavaSpaces^{TM} Service Specification*, 2002. http://www.sun.com/jini/specs/.

[VBO03] Jan Vitek, Ciarán Bryce, and Manuel Oriol. Coordinating Processes with Secure Spaces. *Science of Computer Programming*, 46:163–193, 2003.

[WMF98] P. Wyckoff, S.W. McLaughry, and D.A. Ford. TSpaces. *IBM System Journal*, August 1998.

[Woo99] Alan Wood. Coordination with Attributes. *In Proceedings of the Third International Conference COORDINATION '99, Springer LNCS-1594*, pages 21–36, 1999.