Fundamenta Informaticae 73 (2006) 507–534
IOS Press

Coordination as a Service

Mirko Viroli* **and Andrea Omicini**
DEIS, Alma Mater Studiorum – Università di Bologna
via Venezia 52, 47023 Cesena, FC, Italy
mikro.viroli@unibo.it, andrea.omicini@unibo.it

Abstract. Coordination models like LINDA were first conceived in the context of closed systems, like high-performance parallel applications. There, all coordinated entities were known once and for all at design time, and *coordination media* were conceptually part of the coordinated application. Correspondingly, traditional formalisations of coordination models — where both coordinated entities and coordination media are uniformly represented as terms of a process algebra — endorse the viewpoint of *coordination as a language* for building concurrent systems.

The complexity of today application scenarios calls for a new approach to the formalisation of coordination models. Open systems, typically hosting a multiplicity of applications working concurrently, require coordination to be imposed through powerful abstractions that *(i)* persist through the whole engineering process — from design to execution time — and *(ii)* provide *coordination services* to applications by a shared infrastructure in the form of coordination media.

As a unifying framework for a number of existing works on the semantics of coordination media, in this paper we present a basic ontology and a formal framework endorsing the viewpoint of *coordination as a service*. By this framework, coordination media are characterised in terms of their interactive behaviour, and are seen as primary abstractions amenable of formal investigation, promoting their exploitation at every step of the engineering process.

1. Introduction

Coordination models like LINDA were first conceived in the context of closed systems [19], like high-performance parallel applications. In this framework, the entities subject to coordination are known once and for all at design time, and coordinating entities — called *coordination media* henceforth —, such as tuple spaces in LINDA or channels in MANIFOLD, are built by a compiler for the specific application,

* Address for correspondence: DEIS, Alma Mater Studiorum – Università di Bologna, via Venezia 52, 47023 Cesena, FC, Italy

to which they conceptually belong. In this context, it is natural to consider coordination in terms of the language for building the interactive part of concurrent applications, which is designed at its best so as to be independent from the computational language used to define each application subpart [32]. Correspondingly, the most traditional approach to the formalisation of coordination models is based on the same framework used for specifying semantics of concurrent languages, such as foundational *calculi* for interaction: most notably CCS [35] and π-calculus [36]. There, different entities and abstractions involved in the coordination process are given a uniform representation as terms of a process algebra [6], whose composition defines the whole coordinated system. Semantics is given to the coordination model by describing the admissible evolutions of the coordinated system, typically by means of a Plotkin's-like SOS semantics (structural operational semantics) [48]. There, each transition rule represents a system subpart evolution, corresponding to the execution of a coordination primitive. This approach to formalisation, adopted by well-known works such as [9, 13, 14, 26], is said here to endorse the viewpoint of *coordination as a language*.

In spite of this traditional approach, today application scenarios call for a new approach to the semantics of coordination models and systems. First, the complexity of today systems requires design abstractions to be clearly defined and handled as first-class entities: not only they do impact on system specification, but also they are meant to affect the engineering process down to execution so as to support essential features like incremental refinement, run-time administration, and dynamic adaptation. Also, the typical multi-application setting (a multiplicity of applications sharing the same host environment) calls for infrastructures addressing common application needs, and providing applications with effective and reliable services. Once recognised that coordination is the activity of ruling and governing interaction among system components, and that the issue of coordination is critical in any non-trivial application scenario, coordination is one of the most obvious services that an infrastructure should provide. In particular, coordination infrastructures such as JavaSpaces [29] and TuCSoN [41] are built around coordination abstractions (JavaSpaces and tuple centres, respectively) that embody most of the coordination model semantics, and are provided at run time as services to agents. In principle, coordination media are shared by different components of an application, and by different co-existing applications as well. By choosing a certain coordination medium, a coordinated agent or application implicitly commits both to the model endorsed by the infrastructure providing the medium, and to the specific coordination laws embodied by the medium itself [40]. This viewpoint is what we call *coordination as a service*.[1]

Whereas most formalisations of coordination models adopt the notion of coordination as a language, some works do exist that describe the semantics of coordination in terms of the service provided by the medium abstraction — [33, 37, 54] to cite some. However, there is no common ontology for describing the behaviour of coordination media, for relating different approaches, and for comparing these models with respect to the traditional formalisations based on the notion of coordination as a language.

So, in this paper we define an ontology endorsing the notion of coordination as a service, developing on the ideas described in [22, 37, 44], and providing a formal framework grounded on this ontology. In this ontology, the entities and abstractions involved in the coordination process are clearly separated in three different roles, namely the *coordinated entities*, the *interaction space*, and the *coordination media*. Evolution of the whole system is expressed in terms of the interactive behaviour of the single coordination medium, which is explicitly characterised in terms of production and consumption of

[1] The term "service" is used here in a broader acceptation than the one used in the context of Web Services [21, 43]. However, we believe that the work presented in this paper could actually promote and extend the applicability of Web Services technologies to the coordination of complex systems. Section 7 provides for more details on this issue.

communication events through the interaction space. According to this acceptation, the coordination medium is intended as a first-class entity amenable to a formal investigation, promoting its exploitation at every step of the engineering process, from design — as a design abstraction defined by the coordination model — to execution time — as a run-time abstraction provided to coordinated systems by a coordination infrastructure.

The remainder of this paper is organised as follows. Section 2 describes the traditional approach to the formalisation of coordination, underlying the concept of "coordination as a language". Section 3 discusses related works and motivations, and introduces the ontology and formal framework supporting the viewpoint of "coordination as a service". Section 4 provides a detailed discussion of how the different aspects of a coordinated system are amenable to a formalisation within this framework. Indeed, this part is also meant to highlight that coordination media are the ideal *loci* where the key semantic aspects of a coordination model are conceptually encapsulated. Section 5 puts the framework to test by modelling a LINDA coordination infrastructure: by the way, this example is meant to providing some hints on how the specification of a coordination model as a service can be derived from its specification as a language. In order to show that the "coordination as a service" framework is at least as expressive as the traditional "coordination as a language" one, Section 6 states an equivalence result between the two. Finally, Section 7 provides the reader with some perspectives of future work, along with the concluding remarks.

2. Coordination as a Language

The most traditional foundation for coordination models is based on the idea of representing coordination through a concurrent language for modelling the interactions of *agents* in a parallel scenario [24]. Examples of researches endorsing this viewpoint are the seminal work on operational semantics for coordination languages in [24], the many papers by Busi et al. ([13, 14, 15] to cite some), the algebraic study of LINDA in [26], the study of expressiveness of coordination in shared dataspaces in [11], the formalisation of MANIFOLD [9], and so on. In general, this approach focuses on the idea of a coordination language added to an existing computational language in an orthogonal way [32], by means of a set of coordination primitives. In this section we survey the main semantics aspects of this well-known framework, so as to highlight its key features and benefits, as well as some of its limits.

2.1. The Formal Framework

The traditional approach to the formalisation of a coordination model is based on expressing a coordinated system in terms of a process algebra [6], describing its admissible evolutions by a semantics based on Plotkin's SOS approach [48]. The distributed state of a coordinated system is seen as a parallel composition of *agents* — interacting through coordination primitives — and *items* of the interaction space — also called the shared dataspace [11, 13, 16]: both agents and items are uniformly represented as processes in the algebra. Execution of coordination primitives is modelled similarly to synchronous communications such as in CCS [35].

As a simple example, consider the following formalisation of a portion of a LINDA system, following the style of [13]. The dataspace is represented as a multiset of data items $t \in T$, representing the tuples occurring in the tuple space. Requests for inserting a data item t in the dataspace are represented by the

item $\langle t \rangle$. Agents are seen as finite, sequential processes performing usual coordination primitives `in`, `rd`, and `out` without matching — that is, specifying a concrete tuple instead of a tuple template as in LINDA.

$$\beta ::= \mathtt{in}(t) \mid \mathtt{rd}(t) \qquad \alpha ::= \beta \mid \mathtt{out}(t) \qquad P ::= 0 \mid \alpha.P \qquad L ::= 0 \mid P \mid t \mid \langle t \rangle \mid (L \parallel L)$$

A LINDA coordinated system $L \in \mathcal{L}$ is modelled as a parallel composition of data items t, requests $\langle t \rangle$ for inserting a new data item, and agents P inserting (`out`), reading (`rd`), and consuming (`in`) items from the dataspace.[2] As in the standard formulations, coordinated systems are equipped with the following congruence relation, equating agent specifications that are to be considered syntactically equivalent:

$$L \parallel 0 \equiv L \qquad L \parallel L' \equiv L' \parallel L \qquad L \parallel (L' \parallel L'') \equiv (L \parallel L') \parallel L''$$

These rules makes it possible to get rid of terminated systems (when composed to other terms), and to handle parallel composition $\parallel$ as a n-ary, commutative operator.

The admissible dynamics of a system described in this way can be defined by proper SOS rules, which are easily understood as the operational semantics of the coordination language defined by the three primitives above. This is defined in terms of a transition system $\mathbb{L} = \langle \mathcal{L}, \longrightarrow_{\mathcal{L}} \rangle$ where $\longrightarrow_{\mathcal{L}} \subseteq \mathcal{L} \times \mathcal{L}$ describes how the coordinated system evolves from one state to another as an agent performs the operation associated to a coordination primitive. For instance, the above subset of LINDA can be given semantics by means of the following rules:

$$\begin{array}{lllllllllll}
L & \parallel & \mathtt{rd}(t).P & \parallel & t & \longrightarrow_{\mathcal{L}} & L & \parallel & P & \parallel & t \qquad [L - RD] \\
L & \parallel & \mathtt{in}(t).P & \parallel & t & \longrightarrow_{\mathcal{L}} & L & \parallel & P & & \qquad [L - IN] \\
L & \parallel & \mathtt{out}(t).P & & & \longrightarrow_{\mathcal{L}} & L & \parallel & P & \parallel & \langle t \rangle \qquad [L - OUT] \\
L & \parallel & \langle t \rangle & & & \longrightarrow_{\mathcal{L}} & L & \parallel & t & & \qquad [L - PND]
\end{array}$$

The first rule defines the semantics of the `rd` primitive: when the requested tuple t occurs in the space, then the process successfully executes operation $\mathtt{rd}(t)$, and its continuation P is allowed to carry on. The second rule handles the case of the `in` primitive: as for the `rd` primitive the process continuation carries on when the request tuple t occurs, but then t is also removed. The third and fourth rules define the semantics of `out` primitive, which corresponds to the *unordered semantics* in the classification reported in [17, 14]. This semantics is meant to capture the idea that the order of requests for inserting a tuple into a tuple space may not be preserved at the receiver side, and in particular, may not correspond to the actual order on which tuples appears in the tuple space. By exploiting the intrinsic non-determinism of SOS semantics this is modelled by first reifying the tuple request as a an item $\langle t \rangle$, which later eventually evaluates to the actual data item t in the dataspace.

2.2. The Viewpoint of Coordination as a Language

Foundational calculi for interactive programming languages, such as CCS [35] and π-calculus [36], are traditionally formalised by a process algebra along with its SOS semantics [47]. This approach has generally proved to be quite expressive to capture key aspects related to the interaction issue, including synchrony, communication, non-determinism, composition, and so on.

[2] Here, `in` and `rd` are grouped altogether by symbol β to allow query primitives to be denoted separately from `out`.

As a result, describing the semantics of coordination models using this mathematical framework implicitly assumes the viewpoint of coordination as a concurrent language for building parallel applications, where the interactions among the system's subparts are suitably governed by the coordination primitives [24]. For this reason, we refer to the viewpoint endorsed by this semantic approach as *coordination as a language*. By adopting the *chemical soup* metaphor for process algebras promoted in [7], a coordinated system can be viewed as a soup where a number of processes float just like molecules. Processes include agents subject to coordination, and also any other entity involved in the coordination process, such as coordinating entities, items of a shared dataspace, pending requests, and so on. Pictorially, evolution of the whole system takes place when compatible molecules get near to each other, so that a local transformation occurs that changes their state. This is a convenient description of SOS rules, as reflected e.g. in the rule for the `in` primitive in Section 2.1: when the molecule $\texttt{in}(t).P$ gets near to molecule t — namely, when the agent performs operation `in` and tuple t occurs in the space — interaction may take place, and the two molecules join into new molecule P — representing tuple t being dropped and agent continuation P carrying on. Uniformity is the basic flavour of this framework: abstractions of different kinds are represented in the same fashion, supporting power and simplicity in the description of even complex coordination features.

This general framework has been shown to effectively support the specification of the evolution of coordinated systems, allowing the semantics of many coordination models to be formally described in a quite compact and uniform way. In particular, this framework provides a good abstraction level for studying properties of interest and for reasoning about coordination models. Some remarkable examples exist that highlights the need for describing coordination features by the suitable formal model, instead of relying on an informal specification. In [12] three different semantics are described for the `out` primitive of LINDA — namely, instantaneous, ordered, and unordered. As a matter of fact, existing informal specifications for LINDA do not mention this issue, and LINDA implementations do not all adopt the same semantics: this emphasises the fact that relevant aspects may be underspecified if not represented at the proper abstraction level.

Moreover, in [18] Busi et al. go beyond and show how informal specifications can even lead to incorrect systems. There, the integration between the management of transactions, expiring data, and test-for-absence primitive are studied in the context of shared dataspaces, proving the design of JavaSpaces coordination model [29] to be incorrect, in that the claimed serialisability of transactions is not guaranteed. The most important lesson learnt by that experience is that complex systems — such as coordination infrastructures — are not always amenable to an informal description. Instead, they often require formal specifications grounded on top of suitable frameworks — process algebras being the most relevant one for coordination models.

2.3. The Notion of Expressiveness

One of the notions supported by this formal framework, and strongly driven by the viewpoint of coordination as a language, is that of *expressiveness* of coordination models: the expressiveness of a coordination model is defined in terms of the "possible evolutions" of a coordinated system adopting the model. Roughly speaking, a coordination model is considered relatively more expressive than another if it allows for more evolutions when the corresponding coordinated systems are considered. Indeed, this notion resembles very much the idea of expressiveness traditionally used in the context of automata, where it is defined e.g. in terms of Turing-like transformations.

Most results on this subject come from the works by Busi et al. that study the expressiveness of LINDA coordination primitives relying on the notion of expressiveness based on *encodings*. A coordination language defined by the transition system $\langle X, \longrightarrow_X \rangle$ is considered more expressive than another $\langle Y, \longrightarrow_Y \rangle$ if there exists a function (encoding) $\left|.\right|$ from Y to X so that:

$$y \longrightarrow_Y y' \quad \Rightarrow \quad \left|y\right| \longrightarrow_X^+ \left|y'\right|$$

that is, the encoding makes any single-step evolution of Y be simulated by a sequence of transitions of X. Variations of this kind of encoding have been used to study relative notions of expressiveness of different coordination primitives, and to provide separation results between two models by devising behaviour properties — such as termination and divergence — decidable for one language but not for the other. For instance, in [14] a Turing equivalence property for LINDA is established. Considering agents performing the primitives `out`, `in`, `rd`, `inp`, `rdp`, or repeatedly the `in` primitive (by the operator $!\texttt{in}(a).P$ resembling CCS semantics), and assuming the ordered semantics for `out`, then an encoding of LINDA coordinated systems into Turing programs exists so that any Turing transformation can be mapped upon the evolution of a LINDA coordinated system. The different semantics for `out` primitives have been shown in [14] to have different expressiveness as well, in that Turing equivalence does not hold under the unordered interpretation for `out`. An analogous framework has been used to analyse expressiveness of other features such as event-notification, `copy-collect` primitive, and transient data — we forward the interested reader to [16] for an example and other references.

Another notion of expressiveness used for coordination languages that comes from the traditional field of computer languages is that of *embedding* [52]. First of all, given two languages (or models) X and Y, their comparison is defined in terms of the *observation criteria* $O \in X \mapsto Obs$ and $O' \in Y \mapsto Obs'$, associating to each element of the language a representation of it according to a given abstraction level. According to these observation criteria, X is said to *embed* Y if there exists an encoding of languages $C \in Y \mapsto X$ and a decoding of observations $D \in Obs \mapsto Obs'$ so that for any $y \in Y$ we have $D(O[C(y)]) = O'[y]$. In the context of concurrent languages, this embedding is generally required to be *modular* [8], that is, to satisfy some properties that are meant to lead to a better notion of expressiveness:

(i) since typically observations are powersets, decoding D should be defined elementwise,

(ii) encoding C should be compositional with respect to the non-deterministic operators of the languages (such as choice and parallel composition), and

(iii) a notion of termination invariance on observations should be preserved by the decoding.

The study in [11], for instance, compares expressiveness of languages including subsets of four coordination primitives and three different semantics for them, modelling features of LINDA-like languages, multiset-rewriting languages such as Gamma [5], languages based on communication transactions such as Shared Prolog [10], and concurrent constraint-programming languages. The observation criteria chosen there associate to each set of coordinated entities the multiset of tuples they have produced at the time they reach a final state, and record whether that final state represents a success — the final state is the termination process 0 — or a failure — represented by a deadlock state. Embeddings provided in [11] are shown to be modular with respect to that observation criteria: among the various comparative re-

sults, for instance, test-for-absence primitive is shown to make languages with communication transactions semantics more expressive than multiset-rewriting languages.

2.4. Limits

The formal framework endorsing the notion of coordination as a language provides a uniform viewpoint over seemingly different entities living into a coordinated system. Therefore, clear separation is not promoted between the entities subject to coordination (coordinated entities) and the coordinating entities (coordination media). Whereas this lack of distinction does not seem to be problematic in simple cases, when dealing with non-trivial coordination mechanisms it is likely to make it hard to understand the roles played by the entities within the whole coordinated system — which is definitively a critical issue in complex systems.

In general, this happens because the framework falls short in providing the suitable abstraction level to capture the very notion of coordination medium as a first-class abstraction. Consider the case of the unordered semantics for the `out` primitive in LINDA described in Section 2.1. In this formalisation, execution of $\texttt{out}(a)$ creates a pending request $\langle a \rangle$ waiting to be evaluated into the actual data item a. In some sense, $\langle a \rangle$ may be interpreted as a sort of agent spawned by the process in charge of actually inserting the tuple into the tuple space. There, it is unclear whether this transient agent (or datum) has to be considered as part of the dataspace — modelling the request pending within the tuple space —, or as a transient effect of the interaction process — modelling the fact that the underlying communication infrastructure does not preserve ordering of messages.

A similar situation occurs in the model of transactions in [18], or the notification mechanism in [15]. There, both active transactions and event listeners are represented as entities living in the system in parallel to data and agents, without clearly representing how the coordination medium is meant to manage them. Only the final effect of the system evolution is preserved, according to the goal of the underlying formal framework.

Similar arguments can be issued not only for the separation between entities and coordination medium, but for the interactions between them as well. In fact, the semantics of a coordination model is described in terms of changes to the coordinated system as a whole, while communication acts typically occurring in the system between the medium and the entities are only partially accounted for and not explicitly represented. For example, the semantics of the `in` primitive as described in Section 2.1 provides for the atomic consumption of a data item, whereas in real systems handling the `in` primitive generally requires two distinct phases: sending a consumption request and later receiving a reply containing the removed tuple. Only the reply part of this interaction protocol is actually represented: the request is left implicit, since in the formal model the `in` primitive is executed only when the required data item actually occurs in the space — just as if a pending request were already sent and were waiting for the tuple to be inserted. However, as pointed out in Section 2.2, rather than an inadequacy of the framework, this should be considered as a result of an explicit choice of the desired abstraction level, which allows the evolution of a coordinated system to be represented in a compact and uniform way.

Notice that there are some works on the expressiveness of coordination models such as [11] where a distinction between dataspace and coordinated entities is clearly remarked, but where run-time interactions are not fully accounted for. As a result, we consider these works as still adhering to the viewpoint of coordination as a language, since they basically rely on the above formal framework: separation of entities into distinct roles is reduced as a simple syntactic feature, with no clean ontological distinction.

3. Coordination as a Service

In contrast to the view of coordination as a language, naturally born in the context of closed parallel systems, in this section we introduce background, ontology, and a formal framework for the viewpoint of coordination as a service. In this framework, coordination media are promoted to first-class entities in coordinated systems, amenable to an explicit characterisation in terms of an interactive abstraction, with the goal of supporting the design of coordination models down to the deployment of coordination infrastructures.

3.1. Motivation and Related Works

The idea of coordination as a service introduced in this paper is meant to provide a unifying and basic framework for a number of existing works developed around the notion of *coordination medium*, firstly introduced in [22]. There, Ciancarini argues that a coordination model can be "constructively" defined by describing *(i)* coordination entities, which are the types of the agents subject to coordination, *(ii)* coordination media, components and abstractions making communication among agents possible, and *(iii)* coordination laws, describing how agents coordinate themselves through the media using some coordination primitives. In particular, the notion of coordination medium is promoted as a first-class entity in systems adopting coordination models, providing a clean separation with respect to agents, and identifying the conceptual *locus* where coordination takes place. Interestingly enough, in [22] a clear distinction is also made between two approaches for providing coordination to a software designer — namely, through either a coordination language (orthogonally added to an existing computational language) or a coordination architecture (through channels or blackboards). In the latter case, which is the one demanded by today applications and promoted by our framework, the notion of coordination medium comes in as a model for those abstractions that build up the run-time architecture supporting coordination — namely, the coordination infrastructure. A similar point of view is generally endorsed by the so-called control-driven approach to coordination [44], exploited for instance in MANIFOLD [2] and Reo [3]. According to [42], control-driven coordination models include components (units of computation) and connectors (specifying interactions and communication patterns), with components coordinating each other through connectors by posting and receiving events.

By adopting the general setting of these works, a semantics framework for coordination models is developed in [37]. Communication between coordinated entities and coordination media takes place through an interaction space, where communication events occur as an effect of some entity generating an output (speaking phase) and are consumed when some entity opens to the interaction space (listening phase). Coordination media are supposed there to be reactive abstractions — where internal activities are fired only due to a listening —, which are suitable for an operational description in terms of a transition system, where (observable) actions are events consumption and production. According to [37], this characterisation is meant to help driving the implementation process and easing verification of correctness — e.g., by these techniques the soundness of the observable behaviour can be checked *a priori* on the design, instead of *a posteriori* on the implementation. This general framework has been exploited e.g. in the context of the coordination infrastructure based on ReSpecT tuple centres [39]— which are tuple spaces equipped by a programmable reactive behaviour —, in order to formally define the semantics of the ReSpecT coordination language [38], and to prove the Turing completeness of the coordination laws it can embed [28]. The idea of coordination media as interactive abstractions is also exploited in

[54], were a parametric transition system is defined based on an ontology for *observation*, which can be specified so as to model many aspects of existing coordination models, including LINDA, JavaSpaces, and tuple centres.

The usefulness of operational characterisations of coordination media is also studied in Gelernter and Zuck's [33], where the behaviour of a LINDA implementation is described in terms of the admissible histories of incoming and outgoing messages. In this paper, too, the idea of a coordination model described in terms of the run-time behaviour of a coordination medium is promoted — which, according to the authors, should emphasise "what exactly LINDA means" — supporting validation of implementations and evaluating new applications for LINDA coordination model.

To this extent, it is worth mentioning also the work on LINDA implementation reported in [51, 27]. There, it is argued that starting from an abstract semantics for LINDA primitives — e.g., the formalisation sketched in previous section — many different implementations can be provided, that vary in the allowed dynamics of requests and replies, resulting in different efficiency. These implementations are amenable of a characterisation in terms of the interaction history manifested from the tuple space to any single coordinated entity. In that work it is argued that an efficient implementation of LINDA may feature a delayed removal of tuples due to `in` operations, even though a positive reply is earlier sent to the sending entity — that is, relying on a *optimistic* approach (see a more detailed discussion in Section 6). This implementation schema is claimed to still adhere to LINDA abstract semantics, even though such an argument is not given a formal evidence. As discussed in this paper, too, studying these issues generally helps understanding the meaning of a LINDA tuple space run-time, which can be generally captured by the idea of a medium interactive behaviour.

In general, the traditional formalisation of coordination models endorses the viewpoint of coordination as a language, naturally promoted by the very nature of closed systems on which they were originally conceived. On the other hand, the need to deal with design abstractions representing the entities that actually provides coordination at run-time calls for a new viewpoint of coordination. The aforementioned works are examples of researches toward this direction, trying to clearly characterise the run-time abstractions providing coordination to the subparts of a system, and to reason about their interactive behaviour.

So, in this paper we introduce a common ontology and a formal framework for notions such as coordinated entity, coordination medium, interaction space, medium interactive behaviour, events production and consumption. In particular, the ontology is meant to improve the understanding of the behaviour of coordination media, to foster comparison with the many existing works adopting the notion of coordination as a language, and to promote the development of engineering methodologies for coordination infrastructures grounded on formal specifications.

3.2. Ontology

Starting from the work of Ciancarini in [22], and elaborating on the semantic framework developed by Omicini in [37] for tuple-based coordination models, we introduce in the following an ontology for coordination models, whose main ingredients are depicted in Figure 1.

A system adopting a given coordination model is called a *coordinated system*. This is composed by three separated spaces, the *coordinated space*, the *interaction space*, and the *coordination space*, that include entities and abstractions playing different roles in the coordination process, each with a clear characterisation as run-time design abstraction.

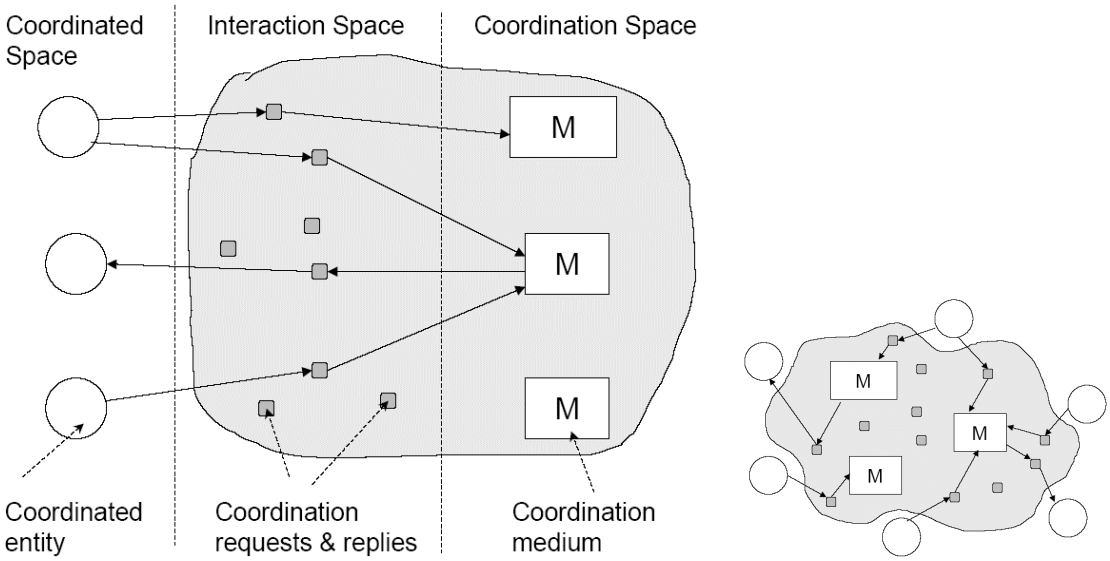

Figure 1. Coordination as a Service: logical architecture (left) and topology (right)

The coordinated space is the part of the system including those entities to which coordination services are provided, namely the *coordinated entities*. According to the standard terminology [32], coordinated entities are those activities whose interactions are governed and ruled by the coordination model. The behaviour of each coordinated entity can be understood in terms of issuing *coordination requests*, opening itself to the reception of *coordination replies*, or performing internal computation — namely *silent actions*. No further hypothesis is made on any other computational activity carried on or any other kind of interactive behaviour. So, from the viewpoint of coordinated entities, coordination takes place as an interactive service made of requests and replies exchanged with the environment. In general, a coordinated entity is not required to be aware of the others living in the coordinated space: direct interaction between them is not represented, and can only indirectly occur through a mediation of the environment, through a coordination activity. Finally, the coordinated space is generally composed of a set of coordinated entities, each with its unique identity, which is used by the coordinating activities to keep track of who issued requests and who has to receive replies. We call *situated (coordinated) entity* a coordinated entity to which a unique identity is assigned by the coordinated system, which in principle enables it to access the coordination services. Notice that in order to deal with openness, the set of coordinated entities should not be fixed once and for all, but can vary so as to take into account coordinated entities entering and leaving the coordinated space. In this case, an entity becomes situated — associating a new unique identifier to it — when entering the system. Even though the very issue of openness is not discussed further in this paper for the sake of simplicity, only slight accommodation is required in order to deal with it.

The interaction space represents the environment for coordinated entities, and handles their requests and replies. Its main goal is to materialise — or reify — the actions performed by the entities and abstractions of the coordinated systems into *(communication) events*, that can be later consumed by other entities or abstractions. In the basic ontology, events can be of the two kinds: *request events* and

reply events. From the viewpoint of a coordinated entity, request events are issued toward the interaction space, and reply events are consumed that occur on the interaction space. So, the coordination process amounts at consuming requests and producing replies through the interaction space. Each reply event is explicitly directed to a given coordinated entity — e.g., to the one that previously issued a given request —, which can then open itself to the reception of that reply.

Consumption of request events and production of reply events, along with underlying coordination activity, is performed within the coordination space. This is made of a number of *coordination media*, which are the abstractions carrying on coordination activities. Since coordination media can be conceptually associated to a subset of the coordinated entities, reflecting different topologies at run-time, a filtering mechanism is assumed to exist that let an event be consumed by a coordination medium according to some *matching condition* — a predicate over either the issuing coordinated entity, or the actual content of the request, or the current state of the medium. After consumption, the coordination medium may produce one or more events that materialise in the interaction space. They can be either reply events, meant to be consumed by coordinated entities, or request events again, which are then directed to other coordination media. This feature is used to take into account communication between coordination media, which is a technique more and more used within coordination infrastructures to support the management of dependencies between multiple coordination flows — see for instance the LOGOP coordination model [53], the extension of ReSpecT tuple centres with the `out_tc` primitive [50], LIME [46], and so on. Dually to coordinated entities, the behaviour of a coordination medium is understood in terms of consuming *coordination requests*, issuing *coordination replies*, and performing silent actions representing the internal computation of coordination activities.

Our ontology provides a simple and expressive model for the run-time abstractions living in a coordinated system: the coordination space models the coordination infrastructure, the interaction space models the enabling communication infrastructure, and the coordinated space models those agents within the system that may enjoy the coordination services provided by the infrastructure, by choosing to interact through the coordination media [40]. As a relevant feature, this ontology also separates mere communication aspects from coordination aspects — that is, enabling from governing interaction —, the former conceptually localised in the interaction space, the latter handled within the coordination space. As a result, this framework promotes the coordination medium as the fundamental design abstraction providing coordination.

3.3. Basic Formal Framework

The following syntactic conventions are adopted. Unless differently specified, in a set X we generally let meta-variable x range over X (and analogously, variable any ranges over the set Any), $\widehat{x}$ range over the set $\widehat{X}$ of finite sequences over X, and $\overline{x}$ over the set $\overline{X}$ of multisets over X. Union of multisets $\overline{x}$ and $\overline{x}'$ is denoted by $\overline{x} \parallel \overline{x}'$, and concatenation of sequences $\widehat{x}$ and $\widehat{x}'$ by $\widehat{x}; \widehat{x}'$, void multiset and empty sequence by $\bullet$. We sometimes use the convenient notation of multisets also when dealing with sets. A transition system is modelled as a triple $\langle P, \longrightarrow, A \rangle$ where P is the set of processes (the states of the component of interest), A is the set of (observable) actions, and relation transition $\longrightarrow$ is a subset of $P \times A \times P$.

In our framework, the set of (states of) coordinated systems $\mathcal{S}$ is a triple $\langle Ms, Is, Cs \rangle$. A coordinated system state is denoted by the syntax $S = ms \oplus is \oplus cs$, and specifies the current state of the coordination space ms, of the interaction space is, and of the coordinated space cs.

The coordinated space is made of coordinated entities. Each coordinated entity is seen as an abstraction interacting with its environment — represented by the interaction space — by sending (coordination) requests $req \in Req$ and receiving (coordination) replies $rep \in Rep$, or by processing the internal silent action τ. So, coordinated entities are modelled as processes whose dynamics is described by the transition system $\langle \mathcal{C}, \longrightarrow_{\mathcal{C}}, Act_{\mathcal{C}} \rangle$, where $\mathcal{C}$ is the set of states for the coordinated entities, ranged over by meta-variable C, and $Act_{\mathcal{C}} ::= \uparrow req \mid \downarrow rep \mid \tau$ denotes the set of actions. Note that details about $\mathcal{C}$ and $\longrightarrow_{\mathcal{C}}$ are not relevant here for they pertain agent internal architecture, which is typically abstracted away when dealing with coordination models: suffices it to characterise the syntax of $Act_{\mathcal{C}}$ as reported above. Within a coordinated system, coordinated entities are contextualised by associating a unique identifier $id \in Id$ to each of them. So, we define the coordinated space Se as $\langle Id, \mathcal{C} \rangle$, made of situated entities of the kind $se = \langle id, C \rangle$.

A request $\uparrow req$ performed by a situated entity $\langle id, C \rangle$ is reified in the interaction space as a request event $id \uparrow req$, which ranges over the set $E^{\uparrow}$. Conversely, when a reply event $id \downarrow rep$ in $E^{\downarrow}$ occurs in the interaction space, the situated entity id may consume it by performing the reply action $\downarrow rep$. The set of events E, ranged over by meta-variable e, is defined as the union of request and reply events ($E = E^{\uparrow} \cup E^{\downarrow}$). So, the interaction space can be generally modelled by the transition system $\langle Is, \longrightarrow_{\mathcal{I}}, Act_I \rangle$, where actions can be either production or consumption of one event, namely $Act_I ::= prod(e) \mid cons(e)$ — $prod(e)$ when an entity produces an event *over* the interaction space, and conversely $cons(e)$ when an entity consumes an event *from* the interaction space. Notice that this framework does not represent silent actions for the interaction space: in our model it is not sensible to consider computational activities for the interaction space, as it just represents the communication infrastructure upon which the coordination model is grounded.

Finally, the coordination space is modelled by a set of coordination media, each represented by a process of the transition system $\langle \mathcal{M}, \longrightarrow_{\mathcal{M}}, Act_{\mathcal{M}} \rangle$, where $\mathcal{M}$ is the set of states of the coordination medium. The set of actions $Act_{\mathcal{M}}$ is defined by the syntax $Act_{\mathcal{M}} ::= get(e^{\uparrow}) \mid put(e) \mid \tau$, respectively representing reception of a request, production of an event — either a reply to a coordinated entity or a request to another coordination medium —, and execution of an internal action.

Differently from coordinated entities, coordination media are contextualised in the coordinated system by associating to each of them a matching predicate $\pi \in \Pi \subseteq E^{\uparrow}$, representing the subset of request events that the medium is enabled to consume. In the following we denote by π_M the matching predicat associated to medium M, supposing it can somehow be extracted from the current medium state M. Indeed, the notion of matching predicate provides a more flexible mechanism for controlling interactions than the unique identifier of coordinated entities, which is useful to properly shape the coordination space.

According to the above definitions, the state of a coordinated system is of the kind $\overline{M} \oplus is \oplus \overline{\langle id, C \rangle}$, containing the set of (contextualised) coordination media $\overline{M}$, the interaction space state is (generally containing the pending events), and the set of situated entities $\overline{\langle id, C \rangle}$.

Even if the above characterisation should be sufficient to describe and understand the role played by each abstraction involved in the coordination process, we proceed by equipping the set of coordinated systems by an operational semantics $\mathbb{S} = \langle \mathcal{S}, \longrightarrow \rangle$. On the one hand, this allows us to precisely specify how each abstraction interacts with the others, giving a comprehensive view of the evolution of a coordinated system. On the other hand, by providing an operational semantics, we make it possible to compare specifications in the proposed framework with those of the traditional framework of coordination as a

language, as we develop in Section 6. In particular, the operational semantics is defined by the rules:

$$\frac{C \xrightarrow{\uparrow req}_{\mathcal{C}} C' \qquad is \xrightarrow{prod(id \uparrow req)}_{\mathcal{I}} is'}{\overline{M} \oplus is \oplus \overline{se} \parallel \langle id, C \rangle \longrightarrow_{\mathcal{S}} \overline{M} \oplus is' \oplus \overline{se} \parallel \langle id, C' \rangle} \quad [REQ]$$

$$\frac{C \xrightarrow{\downarrow rep}_{\mathcal{C}} C' \qquad is \xrightarrow{cons(id \downarrow rep)}_{\mathcal{I}} is'}{\overline{M} \oplus is \oplus \overline{se} \parallel \langle id, C \rangle \longrightarrow_{\mathcal{S}} \overline{M} \oplus is' \oplus \overline{se} \parallel \langle id, C' \rangle} \quad [REP]$$

$$\frac{C \xrightarrow{\tau}_{\mathcal{C}} C'}{\overline{M} \oplus is \oplus \overline{se} \parallel \langle id, C \rangle \longrightarrow_{\mathcal{S}} \overline{M} \oplus is \oplus \overline{se} \parallel \langle id, C' \rangle} \quad [COMP]$$

$$\frac{is \xrightarrow{cons(e^{\uparrow})}_{\mathcal{I}} is' \qquad e^{\uparrow} \in \pi_M \qquad M \xrightarrow{get(e^{\uparrow})}_{\mathcal{M}} M'}{\overline{M} \parallel M \oplus is \oplus \overline{se} \longrightarrow_{\mathcal{S}} \overline{M} \parallel M' \oplus is' \oplus \overline{se}} \quad [GET]$$

$$\frac{M \xrightarrow{put(e)}_{\mathcal{M}} M' \qquad is \xrightarrow{prod(e)}_{\mathcal{I}} is'}{\overline{M} \parallel M \oplus is \oplus \overline{se} \longrightarrow_{\mathcal{S}} \overline{M} \parallel M' \oplus is' \oplus \overline{se}} \quad [PUT]$$

$$\frac{M \xrightarrow{\tau}_{\mathcal{M}} M'}{\overline{M} \parallel M \oplus is \oplus \overline{se} \longrightarrow_{\mathcal{S}} \overline{M} \parallel M' \oplus is \oplus \overline{se}} \quad [COORD]$$

Rules [REQ] and [REP] model a coordinated entity interacting with the interaction space, either by posting a request event or by consuming a reply event. Rule [COMP] allows coordinated entities to perform internal silent actions. Rules [GET] and [PUT] deal with coordination media getting or putting events in the interaction space. Finally, the rule [COORD] accounts for the activity executed inside a coordination medium. Notice that rules [REQ], [REP], [GET], and [PUT] represent interactions of some abstraction with the interaction space, rule [COMP] is the one responsible for describing the computational activities carried on in the coordination system, while [COORD] deals with the coordination activities governing the interaction between all the coordinated entities. For instance, the evolution of a system where a coordinated entity issues a LINDA `in` primitive and then receives the matching tuple found in the coordination medium would be described as follows: *(i)* the coordinated entity produces a request event [REQ], *(ii)* the coordination medium receives the request [GET], *(iii)* the coordination medium finds the tuple [COORD], *(iv)* the coordination medium produces a reply event [PUT], and finally *(v)* the coordinated entity consumes the reply event [REP]. So, this model not only provides a clean separation between computational and coordination activities — which is one of the key strength of the whole coordination approach [32] —, but also conceptually separates the coordination activities from the issues related to the communication infrastructure.

4. Modelling a Coordinated System

In this section we provide a more detailed discussion about the basic issues to be faced when trying to represent a coordination infrastructure by the framework here presented. In particular, each different sort of run-time abstraction is separately analysed, showing how typical and well-known aspects related to the interaction and coordination issues — as well as features of existing coordination models — can be suitably represented. Indeed, our goal here is to stress the idea that while all the abstractions define aspects that are relevant to the end of specifying the overall system behaviour, only the coordination media defines the key aspects of coordination, namely, how coordination activities are carried on. This idea supports the intuition that the coordination medium is the design abstraction that actually maps onto the run-time entities embodying the coordination services as provided by the coordination infrastructure.

4.1. The Coordinated Space

In the basic version of the framework discussed in Section 3.3, the coordinated space is seen as a collection of situated coordinated entities, each with a unique identifier within the coordinated system. Whereas details on the internal behaviour of such entities are not relevant in our framework — for this is not related to coordination —, we simply stick to the idea that their interactive behaviour is represented by a sequences of issued requests, received replies, and internal computations. This is accomplished by simply imposing coordinated entities to be modelled by a transition system $\langle \mathcal{C}, \longrightarrow_{\mathcal{C}}, Act_{\mathcal{C}} \rangle$, whose set of actions is the set $Act_{\mathcal{C}} ::= \uparrow req \mid \downarrow rep \mid \tau$.

Without any further constraint, however, this means that requests and replies are allowed to occur in any order, i.e., that coordinated entities can be understood — supposing them to have finite and deterministic behaviour — by the algebra with the following syntax and semantics:

$$C ::= 0 \mid act_{\mathcal{C}}.C \qquad act_{\mathcal{C}}.C \xrightarrow{act_{\mathcal{C}}}_{Act_{\mathcal{C}}} C$$

In this case, the implicit model assumed for a coordinated entity is either that of a multi-threaded activity, or that of a single-threaded activity whose requests do not block, e.g. which is able to send many synchronous requests simultaneously, handling reception of all replies. In those cases where the execution of a coordination primitive is supposed to block the sender — as in the case of the `rd` primitive in LINDA — and when entities are modelled as single-threaded activities, not all sequences of interactions are actually allowed. For instance, after requesting a `rd` operation, the next action allowed for the entity is only the reception of the corresponding reply: neither other requests, replies, nor internal computations can occur, for the entity being simply blocked. So, in order to specify a coordinated system taking into account blocking conditions within the coordinated entities, a constraint on their interaction histories has to be specified.

Another reason why not all sequences of interactions are generally allowed comes from the fact that each request and reply event can be considered in the context of a coordination protocol, which constrains the occurrence of certain events at a given time. The simplest example is that of synchronous requests, such as the primitives `in`, `rd`, `inp`, and `rdp` in LINDA: there, it is fair for an entity not to expect to receive replies if no requests were issued. Other interaction patterns can actually occur that involve a more complex contract between entities and medium, especially as far as event-based scenarios are concerned [56]. If an entity registers for being notified each time a tuple is inserted in the dataspace, after a registration request many notification replies can be received, at least until a de-registration request is

sent. In this case, interaction histories can be constrained so as to specify a fair behaviour for coordinated entities — e.g., not to send de-registrations before sending registrations —, defining their compatibility with respect to the interaction patterns set up by the services provided by the medium.

These kinds of constraints can be easily described by providing an algebra for coordinated entities with a more refined behaviour than the above one. Consider the interactive behaviour of a mono-threaded coordinated entity for a LINDA medium. Its set of admissible states can be defined by the algebra below:

$$C ::= 0 \mid \tau.C \mid \mathtt{out}(t).C \mid \mathtt{rd}(t).C \mid \mathtt{in}(t).C \mid \mathtt{getreply}(t).C$$

with relation $\longrightarrow_{\mathcal{C}}$ defined by the 5 straightforward rules:

$$\tau.C \xrightarrow{\tau}_{\mathcal{C}} C \qquad \mathtt{out}(t).C \xrightarrow{\uparrow \mathtt{out}(t)}_{\mathcal{C}} C \qquad \mathtt{in}(t).C \xrightarrow{\uparrow \mathtt{in}(t)}_{\mathcal{C}} C$$
$$\mathtt{rd}(t).C \xrightarrow{\uparrow \mathtt{rd}(t)}_{\mathcal{C}} C \qquad \mathtt{getreply}(t).C \xrightarrow{\downarrow \mathtt{rep}(t)}_{\mathcal{C}} C$$

Then, the initial state should be constrained to range in the set defined by:

$$C_0 ::= \tau.C_0 \mid \mathtt{out}(t).C_0 \mid \mathtt{rd}(t).\mathtt{getreply}(t).C_0 \mid \mathtt{in}(t).\mathtt{getreply}(t).C_0 \mid 0$$

so as to impose the proper sequence of requests and replies. Notice that as far as blocking and synchrony issues for coordinated entities need to be explicitly represented in the formal model of a coordinated system, our ontology promotes their specification within the coordinated space, that is, conceptually separated from the coordination media.

4.2. The Interaction Space

Detailing the description of the interaction space is also of some relevance, here, since it may impact some basic aspects which are traditionally ascribed to the coordination model. As an example, in the remainder of this section we show how the interaction space can be defined so as to be responsible for ensuring ordering of messages — which reflects in the properties of coordination as perceived by the coordinated entities.

In our framework the interaction space is modelled as an interactive abstraction from which events can be consumed and produced. A simple specification of its behaviour can be given as an unstructured store of events, where $Is = \overline{e}$, and by the simple semantics:

$$\overline{e} \xrightarrow{prod(e)}_{\mathcal{I}} \overline{e} \parallel e \qquad\qquad \overline{e} \parallel e \xrightarrow{cons(e)}_{\mathcal{I}} \overline{e}$$

This is an unordered model for the interaction space, representing a communication infrastructure that does not guarantee to preserve the ordering of messages. Notice that this hypothesis is the most general and reasonable one in the context of open and distributed systems, which are the typical application scenario for coordination infrastructures [37]. In the case of multi-threaded coordinated entities, unordering of messages may seriously affect the semantics of any request or reply, while in the case of single-threaded entities it just influences the semantics of asynchronous requests. In fact, in the case of synchronous requests such as `in` or `rd` in LINDA, the entities indeed wait for the reply before sending new requests, so from the point of view of a coordinated entity, arrival order is preserved independently from the interaction space, and similarly for replies. Instead, an unordered interaction space affects the

semantics of `out` primitive: sending a sequence of `out` requests provides no information about their consumption order, or if some of them is consumed at all. This is why the in the context of LINDA, the ordering issue applies only to primitive `out`, as addressed in [14].

In order to define a coordinated system supporting ordered semantics for the asynchronous primitives, a different model for the interaction space should be provided. For example, a queue structure can be introduced for each coordinated entity where request events are inserted as they are produced: only requests on top of a queue can be consumed, so that requests order is always preserved. In particular, in this case the interaction space has to be seen as a couple $\langle \overline{e^{\downarrow}}, \overline{q} \rangle$, containing a multiset of reply events and a multiset of requests queues of the kind $q \in Q = \langle Id, \widehat{Req} \rangle$. Its semantics is then defined by the rules:

$$\langle \overline{e^{\downarrow}}, \overline{q} \parallel \langle id, \widehat{req} \rangle \rangle \xrightarrow{prod(id \uparrow req)}_{\mathcal{I}} \langle \overline{e^{\downarrow}}, \overline{q} \parallel \langle id, req; \widehat{req} \rangle \rangle$$

$$\langle \overline{e^{\downarrow}}, \overline{q} \parallel \langle id, req; \widehat{req} \rangle \rangle \xrightarrow{cons(id \uparrow req)}_{\mathcal{I}} \langle \overline{e^{\downarrow}}, \overline{q} \parallel \langle id, \widehat{req} \rangle \rangle$$

$$\langle \overline{e^{\downarrow}}, \overline{q} \rangle \xrightarrow{prod(e^{\downarrow\prime})}_{\mathcal{I}} \langle \overline{e^{\downarrow}} \parallel e^{\downarrow\prime}, \overline{q} \rangle$$

$$\langle \overline{e^{\downarrow}} \parallel e^{\downarrow\prime}, \overline{q} \rangle \xrightarrow{cons(e^{\downarrow\prime})}_{\mathcal{I}} \langle \overline{e^{\downarrow}}, \overline{q} \rangle$$

It is straightforward to tune this model so as to provide the management of queues only for asynchronous requests, or to extend it to provided ordering to replies as well. Notice that queues for requests and replies could also be thought of as associated to coordination media instead of coordinated entities, yet leading to a similar semantics. In this case, unique identifiers have to be provided for media as well so as to track their association to queues within the interaction space, and the matching predicate is to be tested when the request event is produced instead of when it is consumed.

4.3. The Coordination Space

In the Sections 4.1 and 4.2 we showed how a number of well-known aspects that are typically related to coordination, such as ordering, blocking, and synchronisation, can be seen as conceptually disjoint from the actual part of the system realising coordination activities. Whereas they can highly contribute to the design of a coordinated system and may strongly affect the semantics of coordination, our framework promotes their separation from the activities carried on within the coordination space, which are those mainly responsible for governing interactions between coordinated entities. So, in this section we describe the structure of the coordination space, where coordination media live and govern interaction by consuming and posting events of the interaction space.

4.3.1. Single vs. Multiple Coordination Media

It can be the case that the coordination space is populated by only one coordination medium, responsible for coordinating the activities of all coordinated entities living in the system. This is the case, for instance, of a closed system exploiting a single LINDA tuple space. Our framework deals with this case by associating to the only coordination medium a matching predicate that is satisfied for all request events, and that only produces reply events to coordinated entities. In this case, whichever request an entity issues, this is intercepted and evaluated by the medium, which is then in charge of providing the proper reply.

On the other hand, it is also interesting to analyse the other case, where many coordination media live in the system. In the context of data-driven coordination models, where coordination takes place through a shared dataspace, many models have been introduced where the dataspace is not conceptually centralised, but is spread over a distributed system, reflecting various kinds of topology. Examples of such models are JavaSpaces [29], TSpaces [57], the extension of LINDA toward multiple tuple spaces described in [31], the LOGOP extension to LINDA [53], the transient shared tuple spaces of LIME [46], the TuCSoN architecture for tuple centres [41], and so on. Also, in the context of control-driven coordination models such as MANIFOLD [2] and Reo [3], coordination activities take place through many coordination media, that is, through channels dynamically interconnecting two or more coordinated entities.

Some general patterns can be identified in such interconnections, which can be handled in our framework by properly designing matching predicates of coordination media. The coordinated entities may have limited knowledge of the media topology, and just issue requests to the local medium, such as in LIME for the private space of a mobile agent. This case can be handled by associating to each media a predicate satisfied only by the requests coming from a statically given coordinated entity. Analogously, the matching predicate may associate to a coordination medium a cluster of entities, as happens in the management of the so-called interface tuple spaces in LIME.

More frequent is the case where the coordinated entity explicitly specifies an identifier of the medium it wants to issue requests to, such as in JavaSpaces and TuCSoN. There, a unique identifier in $idm \in Idm$ is statically associated to each media, and requests are considered of the kind $\langle idm, r \rangle$ — specifying the medium identifier and the actual request r. The matching predicate of a given medium just checks whether a request is meant to be directed to it.

Dynamic association of a coordination medium to coordinated entities, as occurs e.g. in MANIFOLD, can be generally obtained by matching predicates evolving as the coordination medium evolves.

4.3.2. On Media Inter-Communication

In our basic framework coordination media are allowed to directly communicate one to each other: a coordination medium can post a new request event in the interaction space, which can later be consumed by another coordination medium, depending on its matching predicate. In general, media inter-communication is a feature that allows to flexibly support coordination services, and to deal with the distribution issue. As the LOGOP coordination model shows, for instance, it may be interesting to invoke primitives working on different tuple spaces, requiring all of them to share their information and to co-operate so as to provide a single reply to the requesting entity. For instance, in LIME information is exchanged by tuple spaces in order to provide shared access to the information transparently from mobility. Another example is the extension of ReSpecT tuple centres [39] presented in [50], where a primitive for inter-media communication called `out_tc` is studied that allows one tuple centre to insert a tuple into another, supporting the management of multiple coordination flows in architectures such as TuCSoN.

In all these cases, the predicate matching mechanism can be exploited to support the proper bridging between events posted and received by media.

4.4. The Coordination Medium

In our ontology, the coordination space is the fundamental locus where coordination activities are carried on. There, coordination media are in charge of providing coordination services by accepting requests and reactively providing replies to coordinated entities. Therefore, a concise and foundational description of a coordination model can be generally given by abstracting away from details of the coordinated space and the interaction space, while focussing on the coordination media interactive behaviour.

In contrast to the traditional process algebraic approach to the semantics of coordination described in Section 2, where a coordination model semantics is described by the admissible evolution of the coordinated system as a whole, the framework of "coordination as a service" promotes the description of coordination in terms of the interactive behaviour of the coordination medium, and provides a clean separation between entities playing different roles in the coordinated system. This point of view generalises the idea endorsed by existing works on the formal semantics of coordination models, as discussed in Section 3.

Under this new framework it is still possible to represent the admissible overall evolutions of the coordinated system by a transition system $\mathbb{S} = \langle \mathcal{S}, \longrightarrow_{\mathcal{S}} \rangle$, following the interpretation of coordination as a language. However, this framework also puts the focus on the key issues of coordination by the transition system $\langle \mathcal{M}, \longrightarrow_{\mathcal{M}}, Act_{\mathcal{M}} \rangle$, which describes the operational semantics of the coordination medium — namely, its interactive behaviour.

From the ontology defined in Section 3.2, then, such a behaviour is represented by actions of three kinds: accepting request events, replying events (either requests to other media or replies to coordinated entities), and silent actions representing internal activities — namely, coordination activities.

By this framework, a coordination medium can be characterised in terms of its observational behaviour according to a given semantics (see e.g. the survey on observational semantics in [34]). Adopting for instance a trace semantics as in [33] or [55], a coordination model can be characterised in terms of the set of sequences of actions that its coordination medium exhibits starting from a given initial state.

This opens an interesting issue in the field of coordination models, we believe, in that a wide set of aspects of coordination models are amenable to a suitable description in terms of admissible histories, which is a quite different viewpoint with respect to the usual semantics.

5. LINDA as a Service

In order to foster comparison between the two frameworks discussed in this paper, and to show a full example of modelling coordination as a service, we describe the LINDA service. According to our ontology, we focus on clearly separating the coordination medium abstraction from the other parts of the coordinated system. Indeed, it is our goal here to show that the new approach described in this paper is at least as expressive as the traditional one. The whole discussion is also meant to provide some clues on how existing models interpreting coordination as a language can be turned into models interpreting coordination as a service, and which are the challenges in doing so.

5.1. The LINDA Coordinated System

In order to suitably represent the coordinated system corresponding to the model shown in Section 2.1 we stick to similar assumptions. First of all, coordinated entities are here either supposed to be finite and se-

quential — i.e., mono-threaded — processes performing coordination primitives, abstracting away from the tuple matching mechanism. Then, since only unordered semantics has to be supported, the interaction space is supposed to be an asynchronous store of events, which does not guarantee preservation of ordering. Finally, we suppose that only one coordination medium exists in the space — modelling the original single LINDA tuple space — and that it accepts any request event and produces only reply events.

The set of requests is defined as $Req ::= \mathtt{out}(t) \mid \mathtt{rd}(t) \mid \mathtt{in}(t)$, while the only reply is $Rep ::= \mathtt{ok}$, representing the acknowledgement to a tuple being found correspondingly to a `in` or a `rd`. So, following the discussion provided in Section 4.1, coordinated entities can be represented by the transition system $\langle \mathcal{C}, \longrightarrow_{\mathcal{C}}, Act_{\mathcal{C}} \rangle$, with $\mathcal{C}$ defined by the syntax

$$C ::= 0 \mid \tau.C \mid req.C \mid rep.C$$

by the semantics

$$\tau.C \xrightarrow{\tau}_{Act_{\mathcal{C}}} C \qquad req.C \xrightarrow{\uparrow req}_{Act_{\mathcal{C}}} C \qquad rep.C \xrightarrow{\downarrow rep}_{Act_{\mathcal{C}}} C$$

and considering as set of valid initial states:

$$C_0 ::= 0 \mid \tau.C_0 \mid \mathtt{out}(t).C_0 \mid \mathtt{in}(t).\mathtt{ok}.C_0 \mid \mathtt{rd}(t).\mathtt{ok}.C_0$$

The interaction space can be easily modelled as a store of multiset of events, by the transition system $\langle Is, \longrightarrow_{\mathcal{I}}, Act_I \rangle$, where $Is ::= \overline{e}$ and with $\longrightarrow_{\mathcal{I}}$ defined by the rules:

$$\overline{e} \xrightarrow{prod(e)}_{\mathcal{I}} \overline{e} \parallel e \qquad\qquad \overline{e} \parallel e \xrightarrow{cons(e)}_{\mathcal{I}} \overline{e}$$

5.2. The LINDA Coordination Medium

As a further and fundamental step, we provide a LINDA coordination medium specification in terms of the transition system $\langle \mathcal{M}, \longrightarrow_{\mathcal{M}}, Act_{\mathcal{M}} \rangle$, that is meant to represent all and only the system evolutions described by the formalisation reported in Section 2.1. Clearly, we consider the set of actions $Act_{\mathcal{M}}$ as being defined as $Act_{\mathcal{M}} ::= get(e^{\uparrow}) \mid put(e^{\downarrow}) \mid \tau$ so as to make the medium compatible with the events managed by the coordinated entities.

The set $\mathcal{M}$ of configuration of a tuple space is described by the algebra:

$$M ::= 0 \mid e^{\uparrow} \mid e^{\downarrow} \mid t \mid (M \parallel M)$$

so that at any time the state of the medium is a finite composition of pending requests $e^{\uparrow}$, pending replies $e^{\downarrow}$, and tuples t. The transition relation $\longrightarrow_{Act_{\mathcal{M}}}$, which defines the internal behaviour of the medium, is defined by the rules:

$$\begin{array}{lllllllll}
M & & & \xrightarrow{e^{\uparrow}}_{\mathcal{M}} & M & \parallel & e^{\uparrow} & & [S-REQ] \\
M & \parallel & e^{\downarrow} & \xrightarrow{e^{\downarrow}}_{\mathcal{M}} & M & & & & [S-REP] \\
M & \parallel & id \uparrow \mathtt{out}(t) & \xrightarrow{\tau}_{\mathcal{M}} & M & \parallel & t & & [S-OUT] \\
M & \parallel & id \uparrow \mathtt{rd}(t) \parallel t & \xrightarrow{\tau}_{\mathcal{M}} & M & \parallel & t & \parallel \; id \downarrow \mathtt{ok} & [S-RD] \\
M & \parallel & id \uparrow \mathtt{in}(t) \parallel t & \xrightarrow{\tau}_{\mathcal{M}} & M & \parallel & id \downarrow \mathtt{ok} & & [S-IN]
\end{array}$$

Rules [S-REQ] and [S-REP] can be seen as standard: they mean that requests and replies are to be handled asynchronously. For the particular case of our LINDA system, furthermore, rule [S-REQ] implicitly defines unordering of operation `out`, in that an `out` request remains pending until eventually evaluated. Rules [S-OUT], [S-RD], and [S-IN] have a one-to-one, syntactical mapping with rules [L-INS], [L-RD], and [L-IN] of the formalisation of LINDA provided in Section 2.1. On the one hand, for a coordination activity to be fired by a primitive, its request must first reach the coordination medium, then the proper condition should be satisfied — e.g., a tuple occurring in the tuple space. On the other hand, after a primitive is executed, instead of just letting the process continuation carry on, a reply message is sent to the coordinated entity.

Notice that at a given time, the state of the coordination medium is of the kind $\overline{e} \parallel \overline{t}$, containing the multiset of events $\overline{e}$, representing pending requests and replies waiting to be served, and the multiset of tuples $\overline{t}$ occurring in the tuple space.

5.3. The Whole Coordinated Space

Given the above definitions, the configurations $S \in \mathcal{S}$ of a LINDA coordinated system can be expressed by the notation $\overline{e} \parallel \overline{t} \oplus \overline{e}' \parallel \overline{\langle id, C \rangle}$, orderly specifying the coordination medium's state, the interaction space's state formed by pending events, and the set of coordinated entities. By applying the operational rules reported in Section 3.3 we then obtain a model $\mathbb{S} = \langle \mathcal{S}, \longrightarrow_{\mathcal{S}} \rangle$ of a LINDA coordinated system, which is said to endorse the viewpoint of coordination as a service.

As an example, consider the following initial configuration:

$$S_0 = t_1 \oplus \bullet \oplus \langle id_1, \mathtt{out}(t_2) \rangle \parallel \langle id_2, \mathtt{in}(t_2).\mathtt{ok} \rangle$$

representing only tuple t_1 occurring in the tuple space, no pending event in the medium nor in the interaction space, and two coordinated entities id_1 and id_2, respectively willing to insert and consume the tuple t_2. One of the admissible evolutions of the system, where the two entities issue their request concurrently, can be represented by the transitions:

$$\begin{aligned}
S_0 \quad &\longrightarrow_{\mathcal{S}} \quad t_1 \oplus id_1 \uparrow \mathtt{out}(t_2) \oplus \langle id_1, 0 \rangle \parallel \langle id_2, \mathtt{in}(t_2).\mathtt{ok} \rangle \\
&\longrightarrow_{\mathcal{S}} \quad t_1 \oplus id_1 \uparrow \mathtt{out}(t_2) \parallel id_2 \uparrow \mathtt{in}(t_2) \oplus \langle id_1, 0 \rangle \parallel \langle id_2, \mathtt{ok} \rangle \\
&\longrightarrow_{\mathcal{S}} \quad t_1 \parallel id_2 \uparrow \mathtt{in}(t_2) \oplus id_1 \uparrow \mathtt{out}(t_2) \oplus \langle id_1, 0 \rangle \parallel \langle id_2, \mathtt{ok} \rangle \\
&\longrightarrow_{\mathcal{S}} \quad t_1 \parallel id_2 \uparrow \mathtt{in}(t_2) \parallel id_1 \uparrow \mathtt{out}(t_2) \oplus \bullet \oplus \langle id_1, 0 \rangle \parallel \langle id_2, \mathtt{ok} \rangle \;=\; S_1
\end{aligned}$$

Orderly, rules [REQ], [REQ], [GET], and [GET] are here applied to produce the two request events, and then to make them be accepted by the coordination medium. In this situation, the pending `out` eventually results in the production of the tuple t_2, which is then consumed by the pending `in`. This is respectively represented by two rules [S-COORD], specifying the coordination activity of inserting and removing the tuple, as follows:

$$\begin{aligned}
S_1 \quad &\longrightarrow_{\mathcal{S}} \quad t_1 \parallel id_2 \uparrow \mathtt{in}(t_2) \parallel t_2 \oplus \bullet \oplus \langle id_1, 0 \rangle \parallel \langle id_2, \mathtt{ok} \rangle \\
&\longrightarrow_{\mathcal{S}} \quad t_1 \parallel id_2 \downarrow \mathtt{ok} \oplus \bullet \oplus \langle id_1, 0 \rangle \parallel \langle id_2, \mathtt{ok} \rangle \;=\; S_2
\end{aligned}$$

In particular, in the first transition the medium applies rule [S-OUT] to transform the pending `out` into the tuple, while in the second transition rule [S-IN] is applied that removes the tuple and produces the

$$\left|\overline{e} \parallel \overline{t} \oplus \overline{e}' \oplus \prod_{i\in I}\langle id_i, C_i\rangle\right|_{\mathcal{L}} = \prod_{i\in I}\left|C_i\right|_{\mathcal{L},\mathcal{C}}^{id_i,\overline{e}\parallel\overline{e}'} \parallel \overline{t} \parallel \left|\overline{e} \parallel \overline{e}'\right|_{\mathcal{L},E}$$

$$\left|0\right|_{\mathcal{L},\mathcal{C}}^{id,\overline{e}} = 0 \qquad \left|\mathtt{ok}.C\right|_{\mathcal{L},\mathcal{C}}^{id,\overline{e}} = \left|C\right|_{\mathcal{L},\mathcal{C}}^{id,\bullet} \qquad \left|req.C\right|_{\mathcal{L},\mathcal{C}}^{id,\bullet} = req.\left|C\right|_{\mathcal{L},\mathcal{C}}^{id,\bullet}$$

$$\left|C\right|_{\mathcal{L},\mathcal{C}}^{id,id\uparrow\beta\parallel\overline{e}} = \beta.\left|C\right|_{\mathcal{L},\mathcal{C}}^{id,\bullet} \qquad \left|C\right|_{\mathcal{L},\mathcal{C}}^{id,id'\uparrow\beta\parallel\overline{e}} = \left|C\right|_{\mathcal{L},\mathcal{C}}^{id,\bullet} \; if \; id \neq id'$$

$$\left|\bullet\right|_{\mathcal{L},E} = 0 \quad \left|e^{\downarrow} \parallel \overline{e}\right|_{\mathcal{L},E} = \left|\overline{e}\right|_{\mathcal{L},E} \quad \left|id\uparrow \mathtt{out}(t) \parallel \overline{e}\right|_{\mathcal{L},E} = \langle t\rangle \parallel \left|\overline{e}\right|_{\mathcal{L},E} \quad \left|id\uparrow\beta \parallel \overline{e}\right|_{\mathcal{L},E} = \left|\overline{e}\right|_{\mathcal{L},E}$$

Figure 2. Encoding of LINDA model $\mathbb{S}$ into $\mathbb{L}$

reply event. Finally, the reply event is sent to the coordinated entity id_2 through the interaction space, by means of the rules [PUT] and [REP]:

$$\begin{array}{rcl} S_2 & \longrightarrow_{\mathcal{S}} & t_1 \oplus id_2 \downarrow \mathtt{ok} \oplus \langle id_1, 0\rangle \parallel \langle id_2, \mathtt{ok}\rangle \\ & \longrightarrow_{\mathcal{S}} & t_1 \oplus \bullet \oplus \langle id_1, 0\rangle \parallel \langle id_2, 0\rangle \; = \; S_3 \end{array}$$

6. Equivalence

In this section we provide an equivalence result between the model of LINDA presented in Section 2.1 in terms of the structure $\mathbb{L} = \langle \mathcal{L}, \longrightarrow_{\mathcal{L}}\rangle$, and the model presented in the previous section represented as $\mathbb{S} = \langle \mathcal{S}, \longrightarrow_{\mathcal{S}}\rangle$. This result is meant to show that the framework interpreting coordination as a service is at least as expressive as the traditional one taking the viewpoint of coordination as a language.

In the context of this paper, we rely on the notion of encoding instead of modular embedding (see Section 2.3). The main reason of this choice is that encodings make SOS semantics directly play a critical role in comparing the admissible system evolutions induced by two models. Instead, the key feature of modular embeddings is modularity with respect to composition operators to be applicable to the whole coordinated system, such as parallelism and choice, which have no direct ontological counterpart in the framework of coordination as a service — e.g., a composition of coordination media is not necessarily seen as a new coordination medium, in particular as far as interactive behaviour is concerned. On the other hand, we couple encodings with a notion of termination preservation resembling observations of modular embeddings, which is useful to guarantee semantic injectivity of the encodings.

The transition systems $\mathbb{L}$ and $\mathbb{S}$ are shown to express exactly the same set of admissible evolutions for a LINDA coordinated system. First of all, we define the encodings $\left|.\right|_{\mathcal{L}} \in \mathcal{S} \mapsto \mathcal{L}$ and $\left|.\right|_{\mathcal{S}} \in \mathcal{L} \mapsto \mathcal{S}$, respectively mapping configurations of $\mathbb{S}$ into configurations of $\mathbb{L}$, and viceversa.

Figure 2 reports the mapping of configurations of $\mathbb{S}$ into configurations of $\mathbb{L}$, that is, basically erasing information about the state of the run-time abstractions living in the system and about the pending events. The first line of the definition encodes a generic system state $S = \overline{e} \parallel \overline{t} \oplus \overline{e}' \oplus \prod_{i\in I}\langle id_i, C_i\rangle \in \mathcal{S}$ into the composition of three terms. Orderly, they are *(i)* the set of processes representing the coordinated entities, obtained by propagating the encoding to each entity's state $C \in \mathcal{C}$, *(ii)* the set of tuples, which

$$\left| \prod_{i\in I} P_i \parallel \prod_{j\in J} \langle t_j \rangle \parallel \prod_{k\in K} t_k \right|_{\mathcal{S}} = \prod_{j\in J} id_0 \uparrow \mathtt{out}(t_j) \parallel \prod_{k\in K} t_k \oplus \bullet \oplus \prod_{i\in I} \langle id_i, \left| P_i \right|_{\mathcal{S},\mathcal{P}} \rangle$$

$$\left|0\right|_{\mathcal{S},\mathcal{P}} = 0 \qquad \left|\mathtt{out}(t).P\right|_{\mathcal{S},\mathcal{P}} = out(t).\left|P\right|_{\mathcal{S},\mathcal{P}} \qquad \left|\beta.P\right|_{\mathcal{S},\mathcal{P}} = \beta.\mathtt{ok}.\left|P\right|_{\mathcal{S},\mathcal{P}}$$

Figure 3. Encodings of LINDA model $\mathbb{L}$ into $\mathbb{S}$

are the same occurring in the configuration S, and *(iii)* the set of pending tuples (of the kind $\langle t \rangle$), obtained by propagating the encoding to the pending events.

The encoding on states of entities is denoted by the symbol $|.|_{\mathcal{L},\mathcal{C}} \in \mathcal{C} \mapsto \mathcal{L}$ and is reported in the second group of definitions. Basically, the state of a coordinated entity maps to a process by erasing the representation of replies (action `ok`), by keeping the representation of requests, and by rebuilding pending requests β (`in` or `rd`) from pending request events. Notice that in order to deal with the latter part we need to carry information on the current identifier of the entity as well as on the pending events in the system.

The third group of definitions handles the encoding on pending events, denoted by the symbol $|.|_{\mathcal{L},E} \in \overline{E} \mapsto \mathcal{L}$. A pending request for inserting a tuple $id \uparrow \mathtt{out}(t)$ is mapped to the pending tuple $\langle t \rangle$, independently from the fact that the pending request occurred in the interaction space, or it was already consumed by the coordination medium.

Figure 3 defines the opposite encoding $\left|.\right|_{\mathcal{S}} \in \mathcal{L} \mapsto \mathcal{S}$, taking a configuration L and building a coordinated system S divided into coordination medium, interaction space, and coordinated space. The coordination medium is obtained by considering all the tuples occurring in L and adding a pending request $id_0 \uparrow \mathtt{out}(t_j)$ for any pending tuple $\langle t_j \rangle$ occurring in L, where id_0 is any valid identifier. The interaction space obtained from the mapping is always void, in that pending tuples are materialised in the medium, while there is no representation of other pending events. Finally, processes are translated into coordinated entities by the encoding $|.|_{\mathcal{S},\mathcal{P}} \in \mathcal{P} \mapsto \mathcal{C}$, which generates new identifiers id_i that are supposed to be all different. Actions in P are translated into the same action in C, adding the action `ok` in order to wait for replies after requesting a `in` or a `rd`.

As argued in this paper, one of the key feature of the framework of coordination as a service is that it introduces new information about system configurations, about run-time aspects such as interactions, and about roles of the abstractions involved in the coordination process. This issue reflects in some key characteristics of the two encodings. First of all, the encoding from $\mathcal{L}$ to $\mathcal{S}$ is injective while the opposite is not, reflecting the idea that going from the viewpoint of services to that of languages erases information on run-time aspects. Also, the two encodings are related by the fact that given a configuration $L \in \mathcal{L}$ it is easy to see that $\left|\left|L\right|_{\mathcal{S}}\right|_{\mathcal{L}} = L$. On the other hand, given a configuration $S \in \mathcal{S}$, the new configuration $S' = \left|\left|S\right|_{\mathcal{L}}\right|_{\mathcal{S}}$ is generally different from S, in that it may represent a state obtained from S posting or consuming some event. As an example, we have:

$$S' = \left|\left|t \oplus id \uparrow \mathtt{in}(t) \oplus \langle id, \mathtt{ok}.0 \rangle\right|_{\mathcal{L}}\right|_{\mathcal{S}} = \left|t \parallel \mathtt{in}(t).0\right|_{\mathcal{S}} = t \oplus \bullet \oplus \langle id', \mathtt{in}(t).\mathtt{ok}.0 \rangle$$

The configuration S' obtained by the double encoding is similar to the starting one S (modulo renaming of identifiers): the only difference is that the pending request in the interaction space is "rolled-back" as a request in the coordinated entity.

Apart from these technical details, however, it is crucial to show that the encodings defined prove the two frameworks to be equally expressive according to the notion of expressiveness by encoding described in Section 2.3. In particular, the following result holds:

Theorem 6.1. Given any elements $S, S' \in \mathcal{S}$ and $L, L' \in \mathcal{L}$:

$$S \longrightarrow_{\mathcal{S}}^{+} S' \quad \Rightarrow \quad \left|S\right|_{\mathcal{L}} \longrightarrow_{\mathcal{L}}^{*} \left|S'\right|_{\mathcal{L}} \tag{1}$$

$$L \longrightarrow_{\mathcal{L}}^{+} L' \quad \Rightarrow \quad \left|L\right|_{\mathcal{S}} \longrightarrow_{\mathcal{S}}^{+} \left|L'\right|_{\mathcal{S}} \tag{2}$$

that is, any non-void sequence of transitions in $\mathbb{S}$ is simulated by a possibly void sequence of transitions in $\mathbb{L}$, while any non-void sequence of transitions in $\mathbb{L}$ is simulated by a non-void sequence of transitions in $\mathbb{S}$.

Proof:
The first part of the theorem is easily shown to hold by considering that there is an appropriate mapping of transitions in $\mathbb{S}$ to zero or one transitions of $\mathbb{L}$.

In particular, *(i)* the transition [REQ] involving an `out` primitive is associated to a transition [L-OUT], *(ii)* the transition [COORD] caused by the medium performing the transition [S-OUT] is associated to the transition [L-PND], *(iii)* the transition [REP] involving the `in` and `rd` primitives are associated to the transitions [L-IN] and [L-RD], orderly, and *(iv)* finally any other transition in $\mathbb{S}$ is associated to no transitions in $\mathbb{L}$.

An appropriate association of all transitions of $\mathbb{L}$ to a sequence of transitions in $\mathbb{S}$ proves the second part of the theorem. The transition [L-OUT] can be simulated by a transition [S-REQ] causing the reply event to be raised by the coordinated entity, followed by a transition [GET] causing that event to be consumed by the coordination medium. The transition [L-PND] materialising the tuple in the coordination medium is instead associated to the transition [COORD] caused by the medium performing the transition [S-OUT]. Finally, both transition [L-IN] and [L-RD] are simulated by a sequence of five transitions, including *(i)* the production of the request event ([S-REQ]), *(ii)* its consumption by the coordination medium ([S-GET]), *(iii)* the transition [S-COORD] where the pending request is served (by [S-IN] and [S-RD] respectively), *(iv)* the production of the reply event [S-PUT], and finally *(v)* its reception by the coordinated entity [REP].

□

Indeed, this result guarantees that the two encodings are compatible with respect to the operational semantics of $\mathbb{L}$ and $\mathbb{S}$. However, in this case this result is not enough. As argued above, the encoding $\left|\cdot\right|_{\mathcal{L}}$ is not injective, which is reflected by the fact that one transition in $\mathbb{S}$ is translated into zero or one transition in $\mathbb{L}$. In the worst case, in fact, $\left|\cdot\right|_{\mathcal{L}}$ could map all the elements S into a single element L_0, with the Property 1 in Theorem 6.1 still holding. So, we should also provide a result stating that this encoding is at least "semantically" injective, that is, it does not erase relevant information about the evolution of a coordinated system. In general, this further constraint for encodings is required so as to support comparison of models independently of the granularity of each one's transitions.

To this end, a technique exploited in the context of modular embedding can be here applied, namely, imposing the two encodings to preserve the termination state of configurations. Along with the Theorem 6.1, this suffices in providing an adequate notion of equivalence. In particular, let

$\mathcal{L}_F = \{L \in \mathcal{L} : L \not\rightarrow_{\mathcal{L}}\}$ and $\mathcal{S}_F = \{S \in \mathcal{S} : S \not\rightarrow_{\mathcal{S}}\}$ be the termination states of $\mathbb{L}$ and $\mathbb{S}$, and let $\mathcal{L}_F^+ \subseteq \mathcal{L}_F$ and $\mathcal{S}_F^+ \subseteq \mathcal{S}_F$ be the successful termination states. The latter are defined as those states where coordinated entities are terminated and pending requests are to be served, namely $\mathcal{L}_F^+ = \{\overline{t} \in \mathcal{L}\}$ and $\mathcal{S}_F^+ = \{\overline{t} \oplus \bullet \oplus \prod_i \langle id_i, 0\rangle \in \mathcal{S}\}$. In particular, the following theorem holds.

Theorem 6.2. Given any elements $S \in \mathcal{S}_F$ and $L \in \mathcal{L}_F$, then:

$$L \in \mathcal{L}_F^+ \quad \Leftrightarrow \quad \left|L\right|_{\mathcal{S}} \in \mathcal{L}_F^+ \tag{3}$$

$$S \in \mathcal{S}_F^+ \quad \Leftrightarrow \quad \left|S\right|_{\mathcal{L}} \in \mathcal{S}_F^+ \tag{4}$$

that is, successful termination states are encoded in successful termination states, and deadlock (i.e., unsuccessful termination) states are encoded into deadlock ones, by both encodings.

Proof:
This result straightforwardly follows from the definition of the encodings $\left|.\right|_{\mathcal{L}} \in \mathcal{S} \mapsto \mathcal{L}$ and $\left|.\right|_{\mathcal{S}} \in \mathcal{L} \mapsto \mathcal{S}$. □

The above property guarantees the two models $\mathbb{L}$ and $\mathbb{S}$ to allow for the same set of evolutions for the LINDA coordinated system. This example should show that the viewpoint shift from coordination as a language to coordination as a service does not limit the expressiveness of the model, while it allows the run-time aspects of the coordination process to be taken into account.

7. Future Work and Conclusions

In this work we provided a unifying framework for reasoning about a number of existing formal approaches to the semantics of coordination models, focussing on the idea of coordination as a service provided by interactive coordination media.

This framework is meant to improve the understanding of coordination medium behaviour, to foster comparison with the many existing works adopting the notion of coordination as a language, and to promote the development of engineering methodologies for coordination infrastructures grounded on formal specifications.

In the future, we plan to exploit this framework along many different research lines.

First of all, even though the framework seems quite general, it worth studying full encodings of other coordination models than LINDA, ranging from control-oriented ones such as Reo, and to models based on multiple tuple spaces such as TuCSoN — which we showed to provide a wide spectrum of interaction patterns involving coordinated entities and media. Indeed, comparison with the standard formalisations endorsing the notion of coordination as a language — briefly outlined in Section 6 — needs to be faced in further detail, studying e.g. the impact on the notion of expressiveness along the direction outlined in [55]. Then, we are also interested in studying the impact of this framework on the engineering of coordination infrastructures, especially as far as satisfaction of effectiveness and reliability properties is concerned.

Finally, it is also interesting to deepen the relationship between our approach and the whole scenario of Web Services. In particular, we believe the framework introduced in this paper could be used as an ontology for generally implementing coordination models in terms of Web Services. This actually

comes for free once recognised that coordination media are described in terms of an interactive behaviour made of request and replies, which naturally translates into a similar interactive behaviour of a Web Service. Indeed, studying the relationship between Web Services orchestration/choreography [45] and infrastructures for providing coordination services is a primary issue in this context, and a main subject for future work.

References

[1] Arabnia, H. R., Ed.: *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, CSREA Press, Las Vegas, NV, USA, 24–27 July 2002, ISBN 1-892512-89-0.

[2] Arbab, F., Herman, I., Spilling, P.: An Overview of MANIFOLD and its Implementation, *Concurrency: Practice and Experience*, **5**(1), February 1993, 23–70, ISSN 1040-3108.

[3] Arbab, F., Mavaddat, F.: Coordination through Channel Composition, in: Arbab and Talcott [4], 22–39.

[4] Arbab, F., Talcott, C., Eds.: *Coordination Languages and Models*, vol. 2315 of *LNCS*, Springer-Verlag, 2002, ISBN 3-540-43410-0.

[5] Banatre, J.-P., Le Metayer, D.: Programming by Multiset Transformation, *Communications of the ACM*, **36**(1), 1993, 98–111, ISSN 0001-0782.

[6] Bergstra, J. A., Ponse, A., Smolka, S. A., Eds.: *Handbook of Process Algebra*, North-Holland, 2001, ISBN 0-444-82830-3.

[7] Berry, G., Boudol, G.: The chemical abstract machine, *Theoretical Computer Science*, **96**, 1992, 217–248, ISSN 0304-3975.

[8] de Boer, F. S., Palamidessi, C.: Embedding as a Tool for Language Comparison, *Information and Computation*, **108**(1), 1994, 128–157, ISSN 0890-5401.

[9] Bonsangue, M. M., Arbab, F., de Bakker, J. W., Rutten, J. J. M. M., Scutella, A., Zavattaro, G.: A transition system semantics for the control-driven coordination language MANIFOLD, *Theoretical Computer Science*, **240**(1), June 2000, 3–47, ISSN 0304-3975.

[10] Brogi, A., Ciancarini, P.: The Concurrent Language Shared Prolog, *Transactions on Programming Languages and Systems*, **13**(1), 1991, 99–123, ISSN 0164-0925.

[11] Brogi, A., Jacquet, J.: On the Expressiveness of Coordination Models, in: Ciancarini and Hankin [23], 134–149.

[12] Busi, N., Gorrieri, R., Zavattaro, G.: Comparing Three Semantics for Linda-like Languages, *Theoretical Computer Science*, **240**, 1990, 49–90, ISSN 0304-3975.

[13] Busi, N., Gorrieri, R., Zavattaro, G.: A Process Algebraic View of Linda Coordination Primitives, *Theoretical Computer Science*, **192**(2), 1998, 167–199, ISSN 0304-3975.

[14] Busi, N., Gorrieri, R., Zavattaro, G.: On the Expressiveness of Linda Coordination Primitives, *Information and Computation*, **156**(1-2), January 2000, 90–121, ISSN 0890-5401.

[15] Busi, N., Gorrieri, R., Zavattaro, G.: Process Calculi for Coordination: From Linda to JavaSpaces, in: *Algebraic Methodology and Software Technology* (T. Rus, Ed.), vol. 1816 of *LNCS*, Springer-Verlag, 2000, ISBN 3-540-67530-2, 198–212.

[16] Busi, N., Rowstron, A., Zavattaro, G.: State- and Event-Based Reactive Programming in Shared Dataspaces, in: Arbab and Talcott [4], 111–124.

[17] Busi, N., Zavattaro, G.: Event Notification in Data-driven Coordination Languages: Comparing the Ordered and Unordered Interpretations, in: Carrol et al. [20], 233–239.

[18] Busi, N., Zavattaro, G.: On the Serializability of Transactions in JavaSpaces, *Electronic Notes in Theoretical Computer Science*, **54**, July 2001, ISSN 1571-0661.

[19] Carriero, N., Gelernter, D.: How to write parallel programs: a guide to the perplexed, *ACM Computing Surveys*, **21**(3), 1989, 323–357, ISSN 0360-0300.

[20] Carrol, J., Damiani, E., Haddad, H., Oppenheim, D., Eds.: *The 2000 ACM Symposium on Applied Computing (SAC 2000)*, ACM, Como, Italy, 19–21 March 2000, ISBN 1-58113-239-5, Special Track on Coordination Models, Languages and Applications.

[21] Chung, J.-Y., Lin, K.-J., Mathieu, R. G.: Guest Editors' Introduction: Web Services Computing - Advancing Software Interoperability., *IEEE Computer*, **36**(10), 2003, 35–37.

[22] Ciancarini, P.: Coordination models and languages as software integrators, *ACM Computing Surveys*, **28**(2), June 1996, 300–302, ISSN 0360-0300.

[23] Ciancarini, P., Hankin, C., Eds.: *Coordination Languages and Models*, vol. 1061 of *LNCS*, Springer-Verlag, April 1996, ISBN 3-540-61052-9.

[24] Ciancarini, P., Jensen, K. K., Yankelevich, D.: On the Operational Semantics of a Coordination Language, in: *Object-Based Models and Languages for Concurrent Systems* (P. Ciancarini, O. Nierstrask, O. Yonezawa, Eds.), vol. 924 of *LNCS*, Springer-Verlag, 1994, ISBN 3-540-59450-7, 77–106.

[25] Ciancarini, P., Wolf, A. L., Eds.: *Coordination Languages and Models*, vol. 1594 of *LNCS*, Springer-Verlag, 1999, ISBN 3-540-65836-X.

[26] De Nicola, R., Pugliese, R.: A Process Algebra based on Linda, in: Ciancarini and Hankin [23], 160–178.

[27] De Nicola, R., Pugliese, R., Rowstron, A.: Proving the Correctness of Optimising Destructive and Non-destructive Reads over Tuple Spaces, in: Porto and Roman [49], 66–80.

[28] Denti, E., Natali, A., Omicini, A.: On the Expressive Power of a Language for Programming Coordination Media, in: *1998 ACM Symposium on Applied Computing (SAC'98)*, ACM, Atlanta, GA, USA, 27 February – 1 March 1998, ISBN 0-89791-969-6, 169–177.

[29] Freeman, E., Hupfer, S., Arnold, K.: *JavaSpaces: Principles, Patterns, and Practice*, Addison-Wesley, 1999, ISBN 0-201-30955-6.

[30] Garlan, D., Le Métayer, D., Eds.: *Coordination Languages and Models*, vol. 1282 of *LNCS*, Springer-Verlag, 1997, ISBN 3-540-63383-9.

[31] Gelernter, D.: Multiple Tuple Spaces in Linda, in: *Parallel Architectures and Languages Europe (PARLE'89)*, vol. II: Parallel Languages, Springer-Verlag, 1989, ISBN 3-540-51285-3, 20–27.

[32] Gelernter, D., Carriero, N.: Coordination Languages and their Significance, *Communications of the ACM*, **35**(2), 1992, 96–107, ISSN 0001-0782.

[33] Gelernter, D., Zuck, L.: On What Linda is: Formal Description of Linda as a Reactive System, in: Garlan and Le Métayer [30], 187–219.

[34] van Glabbeek, R.: The Linear Time – Branching Time Spectrum I. The semantics of Concrete, Sequential Processes, in: Bergstra et al. [6], chapter 1, 3–100.

[35] Milner, R.: *Communication and Concurrency*, Prentice Hall, 1989, ISBN 0-13-114984-9.

[36] Milner, R.: *Communicating and Mobile Systems: The π-calculus*, Cambridge University Pres, 1999, ISBN 0-521-65869-1.

[37] Omicini, A.: On the Semantics of Tuple-based Coordination Models, *1999 ACM Symposium on Applied Computing (SAC'99)*, San Antonio (TX), 28 February – 2 March 1999, ISBN 1-58113-086-4.

[38] Omicini, A., Denti, E.: Formal ReSpecT, *Electronic Notes in Theoretical Computer Science*, **48**, June 2001, 179–196, ISSN 1571-0661.

[39] Omicini, A., Denti, E.: From Tuple Spaces to Tuple Centres, *Science of Computer Programming*, **41**(3), November 2001, 277–294, ISSN 0167-6423.

[40] Omicini, A., Ossowski, S.: Objective versus Subjective Coordination in the Engineering of Agent Systems, in: *Intelligent Information Agents: An AgentLink Perspective* (M. Klusch, S. Bergamaschi, P. Edwards, P. Petta, Eds.), vol. 2586 of *LNAI: State-of-the-Art Survey*, Springer-Verlag, March 2003, ISBN 3-540-00759-8, 179–202.

[41] Omicini, A., Zambonelli, F.: Coordination for Internet Application Development, *Journal of Autonomous Agents and Multi-Agent Systems*, **2**(3), 1999, 251–269, ISSN 1387-2532.

[42] Papadopoulos, G. A., Arbab, F.: Configuration and Dynamic Reconfiguration of Components Using the Coordination Paradigm, *Future Generation Computer Systems*, **17**(8), 2001, 1023–1038, ISSN 0167-739X.

[43] Papazoglou, M. P., Georgakopoulos, D.: Special Section: Service-oriented computing. Introduction, *Communications of the ACM*, **46**(10), 2003, 24–28.

[44] Papdopoulos, G. A., Arbab, F.: Coordination Models and Languages, *Advances in Computers*, **46**, 1998, 329–400.

[45] Peltz, C.: Web Services Orchestration and Choreography, *IEEE Computer*, **36**(10), 2003, 46–52.

[46] Picco, G. P., Murphy, A. L., Roman, G.-C.: LIME: Linda meets mobility, in: *The 1999 International Conference on Software Engineering (ICSE'99)*, ACM, Los Angeles, CA, USA, 16-22 May 1999, ISBN 1-58113-074-0, 368–377.

[47] Pierce, B. C.: Foundational Calculi for Programming Languages, in: *The Computer Science and Engineering Handbook* (A. B. Tucker, Ed.), CRC Press, 1997, ISBN 0-8493-2909-4, 2190–2207.

[48] Plotkin, G.: *A structural approach to operational semantics*, Technical Report DAIMI FN-19, Department of Computer Science, AArhus University, Denmark, 1991.

[49] Porto, A., Roman, G.-C., Eds.: *Coordination Languages and Models*, vol. 1906 of *LNCS*, Springer-Verlag, 2000, ISBN 3-540-41020-1.

[50] Ricci, A., Omicini, A., Viroli, M.: Extending ReSpecT for Multiple Coordination Flows, in: Arabnia [1], 1407–1413.

[51] Rowstron, A.: Optimising the Linda in Primitive: Understanding Tuple Space Run-times, in: Carrol et al. [20], 227–232.

[52] Shapiro, E. Y.: Embeddings Among Concurrent Programming Languages (Preliminary Version), in: *CONCUR '92, Third International Conference on Concurrency Theory* (W. R. Cleaveland, Ed.), vol. 630 of *LNCS*, Springer-Verlag, 1992, ISBN 3-540-55822-5, 486–503.

[53] Snyder, J., Menezes, R.: Using Logical Operators as an Extended Coordination Mechanism in Linda, in: Arbab and Talcott [4], 317–331.

[54] Viroli, M., Omicini, A.: Tuple-based Models in the Observation Framework, in: Arbab and Talcott [4], 364–379.

[55] Viroli, M., Omicini, A., Ricci, A.: On the Expressiveness of Event-Based Coordination Media, in: Arabnia [1], 1414–1420.

[56] Viroli, M., Ricci, A.: Tuple-Based Coordination Models in Event-Based Scenarios, in: *22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02)*, IEEE Press, 2–3 July 2002, ISBN 0-7695-1588-6, 595–601.

[57] Wyckoff, P., McLaughry, S. W., Lehman, T. J., Ford, D. A.: T Spaces, *IBM Journal of Research and Development*, **37**(3 - Java Techonology), 1998, 454–474, ISSN 0018-8670.