Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 97 (2004) 67–96
www.elsevier.com/locate/entcs

Verifying Properties of Coordination by Well-Structured Transition Systems

Mirko Viroli[a,1]

[a] *DEIS, Università degli Studi di Bologna*
via Venezia 52, 47023 Cesena (FC), Italy

Abstract

Coordination models and languages are introduced to effectively rule and govern the interactions in those systems that feature complexity, distribution, opennes and high dynamics. These characteristics, however, traditionally impose a number of constraints on the engineering process: most notably, they make system specifications infinite, thus complicating – and sometimes preventing – the successful automatic verification of properties.
In the field of verification for infinite state systems, the notion of *well-structured transition systems* has recently being introduced and studied. Its framework not only unifies a number of existing results in the context of infinite verification, but also introduces general concepts and methodologies, such as *upward-closure* and *backward analysis*, that show a great potential applicability for concurrent and interactive systems in general.
In this paper, we evaluate the applicability of this framework to the context of coordination, formally defining the notion of *well-structured coordination*. A coordinated system adhering to this notion is amenable to a description in terms of a well-structured transition system, where interesting properties concerning termination, boundedness, safety, and liveness are decidable. An example of application to the LINDA coordination model is studied, focussing on a methodology for proving the safety properties of coordinated systems.

Keywords: Coordination Models, Linda, formal verification, transition systems

1 Introduction

Coordination models and languages are claimed to be an effective means to handle the complexity of interactions in those software systems characterised by crucial features such as distribution, large-scale, opennes, highly-unpredictable dynamics, and reliable behaviour over time. However, exactly

[1] Email: mviroli@deis.unibo.it

doi:10.1016/j.entcs.2004.04.032

these very features are the source of a number of complications, which, as a matter of facts, existing engineering methodologies not easily cope with.

An evident example is the automatic verification of properties, which is sometimes still considered an academic prerogative, but is instead a fundamental issue in the engineering of complex systems. In general, the few works tackling this aspect in the context of coordination provide results concerning rather specific models and problems. An example is the verification of properties using the SPLICE tool [17] and the μCLR language, which have been exploited to study bisimulation properties of tuple space implementations [37] and JavaSpaces [43]. On the other hand, the work in [14] describes a model checking technique for checking safety and liveness properties of the PoliS coordination model. In fact, these solutions often suffer from a traditional problem of automatic verification: when dealing with systems characterised by an infinite state they either simply fail to be effective ([14]), or they provide solutions to rather specific problems ([37]).

However, the above features that coordination is meant to cope with turn out to often introduce a decisive increase in the state space, until making it conceptually unbounded. Just to mention opennes, the ability of accepting new incoming coordinated entities over time entails infiniteness of the state space. As this problem is shared with virtually any research field concerning the engineering of today software systems, we believe that coordination can and should provide a suitable general framework to practically address the problem, and in particular, to vehicle existing theoretical results towards their effective application to the engineering of systems.

The problem of verifying infinite state systems is receiving more and more attention, and has already witnessed a significant number of positive results [42], mostly concerning traditional theoretical models such as Petri nets, process algebras and string rewriting systems. A crucial step towards finding more widely applicable results has been made by the so-called *parameterized verification* field [1,25], and in particular by the methodology underlying the notion of *well-structured transition systems* [26]. This framework applies an abstraction process over an infinite system configuration, finding a finite way to represent and manipulate it. By this approach, it is possible to decide a number of interesting properties: not only *covering*, which is the one we mostly focus on in this paper – and which is exploited to state a system safety –, but also inevitability, termination and boundedness. These results have been applied to restate in a common framework a number of existing results and verification methodologies already presented in the literature, concerning models such as Petri nets, process algebras, and string rewrite systems. Moreover, it has also been applied to verify the safety of various kinds of protocols,

including cache coherence [18], mutual-exclusion [5], and security protocols [21].

This recent framework can be considered as one of the very first attempts to define a unifying methodology for the verification of infinite state systems, providing general decidability results and corresponding verification methodologies independently of the specific model they are applied to. On the other hand, existing applications seem to show that the framework of well-structured transition systems is particularly suitable for describing concurrent and interacting systems in general – which are actually the basic setting on which coordination is based. So, the goal of this paper is to start evaluating the applicability of the framework of well-structured transition systems to the verification of properties in the context of coordination models. In particular, our aim here is not to specifically provide new technical results for the verification field. Rather, we want to *(i)* show examples of properties of coordination systems that are amenable to an automatic verification by this framework, *(ii)* devise a methodology for systematically applying the existing results to coordination models, and *(iii)* outline some of the main open issues and expectations of parameterized verification in the coordination context.

The remainder of the paper is organised as follows. Section 2 provides a self-contained description of the framework of well-structured transition systems mostly based on [26], introducing those concepts that will be used in the rest of the work. Section 3 outlines the basic applicability of this framework in the context of concurrent systems such as Petri nets and process algebras. In particular, we focus on the verification of safety properties of a system: the prototype verification tool devised by Delzanno in [19] is briefly introduced that is exploited throughout the paper for some experiments. Based on the viewpoint of *coordination as a service* presented and discussed in [46], in Section 4 we define a general framework for describing coordinated systems, on which sufficient conditions are given that allow for the decidability of meaningful properties – identifying what we shall call the notion of *well-structured coordination*. Section 5 puts to test this result to a paradigmatic, traditional case of coordination: the problem of the *dining philosophers* [24] (also referred to as "hurried philosophers") governed by a LINDA tuple space [29]. In spite of the opennes character of our example – that makes it an intrinsically infinite state space one – safety properties such as mutual exclusion and deadlock freeness are automatically proved to be satisfied. Finally, to discuss open issues, in Section 6 we consider an ongoing research on the framework of well-structured transition systems, the MSR(C) notation (multiset rewriting with constraints) [20]. An example application is shown to outline the potential of this model for the verification of properties in more complex coordination

scenarios, featuring an unbounded number of resources and aspects related to value-passing, such as communication and tuple matching. Section 7 provides concluding remarks and sketches a roadmap of future works.

2 Verifying Infinite Parameterized Systems

A common approach to describe the evolution of software systems is by transition system (TS) semantics [40,30]. A TS can be simply defined as a couple $\langle S, \longrightarrow \rangle$. S is the carrier set of the TS: it represents the set of all possible configurations of the system of interest – or at least, of the part of the system whose evolution is of concern. The binary relation $\longrightarrow \subseteq S \times S$ describes whether it is possible for a system configuration to evolve into another. In particular, $\langle s, s' \rangle \in \longrightarrow$ is written $s \longrightarrow s'$ and means that from configuration $s \in S$ the system may move to configuration $s' \in S$.

Sometimes the definition of set S also includes unfeasible configurations; in this case the above TS is equipped by a set of admissible initial states $S_0 \subseteq S$, so that the feasible configurations are indirectly characterised as those reachable in one or more steps from elements in S_0 through relation $\longrightarrow$. To complete the notation, write $Prec(s)$ for the set $\{t \in S : t \longrightarrow s\}$ of immediate predecessors of s and $Succ(s) = \{t \in S : s \longrightarrow t\}$ for the immediate successors. Also, define $\longrightarrow^+$ as the transitive closure of $\longrightarrow$, and $\longrightarrow^*$ as the reflexive and transitive closure of $\longrightarrow$, and correspondingly define $Prec^+(s)$ and $Prec^*(s)$.

The most common approach to formally state properties of interest over systems is *model checking* [16]. In its most abstract setting, a model checking problem is formulated as the problem of determining the set of configurations of a TS over which a given predicate p holds, that is, determining the set $S_p = \{s \in S : p(s)\}$. The naive way to solve the task is to start checking which initial states in S_0 satisfy the predicate, then considering their successors, then the successors of the successors, and so on until all configurations have been considered. However, this approach leads to the intended result only if *(i)* for any $s \in S$ there is an effective way to evaluate $p(s)$ and to compute $Succ(s)$, and if *(ii)* S is finite. While the former property is typically easily satisfied, the latter is more critical.

In the general case, it is impossible to determine S_p when S is infinite. This has not prevented the study of solutions to specific problems, such as the decidability of liveness and safety properties of Petri nets [31] and of subsets of the CCS algebra [13] – just to mention some of the most relevant. Providing a complete survey of the results achieved in this context is a goal of the "Roadmap of Infinite Results" project by Jiri Srba [42].

Most relevant to the scope of this paper, in [1] a number of general results are proved that enable the verification of various kind of infinite state systems, underlying the notion of *parameterized verification* – namely, verifying system properties independently of one or more parameters. In [26], these results are extended providing a conceptually unified framework: it is shown that the approach has not only the potential for verifying new systems, but is also a means to classify and restate a number of seemingly heterogeneous existing results and verification procedures. The key notions of this approach are here presented in a compact and self-contained way summarising and elaborating on [26].

2.1 Well-Structured Transition Systems

The key idea to overcome the infiniteness of the carrier set S is to focus on infinite sets that are semantically finite, that is, that can be characterised (represented, defined, manipulated) in a finite way, e.g. in terms of a representative finite subset of them.

As a first step, equip the carrier set S with a transitive and reflexive relation $\leq$, called a *quasi-order* (or a preorder) [32]. The notion of *updward-closed* set then naturally comes in to characterise those sets $I \subseteq S$ that are "closed under upward seek", that is, such that $s \in I$ and $s \leq t$ entails $t \in I$. A *basis* I^b of an upward-closed set I is a sufficient generator for I, i.e. if $t \in I$ then there exists $s \in I^b$ such that $s \leq t$. This generation is realised by virtue of the operator $\uparrow$ defined as $\uparrow I^b = \{t \in S : \exists s \in I^b, s \leq t\}$: I^b is a basis for I iff $\uparrow I^b = I$. The idea that a set of configurations can be denoted in a unique way by a representative (sub)set justifies the terminology of "parameterized systems": a finite basis is used to identify in a parameterized way an infinite set of system configurations.

A particular case of quasi-order, called *well-quasi-order*, enjoys crucial properties that enable the effective verification of infinite state systems. A well-quasi-order $\leq$ over S is a quasi-order such that for any infinite sequence $s_1, s_2, \ldots$ in S there exists indexes $i < j$ such that $s_i \leq s_j$. Notice that well-quasi-orders prevent infinite sequences of elements from being indefinitely (strictly) decreasing. Well-quasi-orders entail the following two properties:

(i) Finiteness: Any updward closed set admits a finite basis, hence it is semantically finite.

(ii) Stabilisation: For any infinite increasing sequence of upward-closed sets $U_0 \subseteq U_1 \subseteq U_2 \subseteq \ldots$ there is an $i > 0$ such that $U_i = U_j$ for each $j > i$.

While finiteness enables effective computation, stabilisation allows certain monotonic algorithms to be provably terminating.

In order to exploit well-quasi-orders to verify properties of systems evolutions, it is necessary to make them fitting with the TS semantics. This leads to the notion of *well-structured transition system* (WSTS). This is a triple $\langle S, \longrightarrow, \leq \rangle$ where $\leq$ is a well-quasi-order and where $\longrightarrow$ has *upward compatibility* w.r.t. $\leq$, that is, for all $s_1 \longrightarrow s_2$ and $s_1 \leq t_1$ there exists a transition $t_1 \longrightarrow t_2$ such that $s_2 \leq t_2$. In particular, this upward compatibility property is referred to as strong compatibility [1], for it requires the existence of precisely one transition moving t_1 to t_2. Figuratively, upward compatibility can be defined in terms of the commutativity of the following diagram [2]:

$$\begin{array}{ccc} t_1 & \longrightarrow_S & t_2\ (\exists) \\ \leq\uparrow & & \uparrow\leq \\ (\forall)\ s_1 & \longrightarrow_S & s_2 \end{array}$$

As far as verification procedures are concerned, in the following we suppose that $\leq$ is decidable, and that $Prec(s)$ and $Succ(s)$ are effectively computable.

2.2 Deciding Covering

The main result achieved in the context of WSTSs is the decidability of the *covering* problem. A configuration $s \in S$ is said to cover a configuration $t \in S$ if it reaches a greater element (a $t' \in S$ such that $t \leq t'$), or equivalently, if it reaches (a state into) $\uparrow \{t\}$. This notion is in general applicable to state the reachability problem for upward-closed sets: in order to check whether a configuration s reaches some element in the upward-closed set I, suffices it to check whether s covers at least one element of the (finite) basis of I.

The algorithm used to solve this problem is referred to as *backward (reachability) analysis*, and works as follows. Simply denote K_0 as the finite basis of the upward-closed set I, and for any $n > 0$ build K_{n+1} as the union of K_n and a finite basis of $\uparrow Prec(\uparrow K_n)$ – notice that the predecessors of an upward-closed set form another upward-closed set. The sequence of finite sets K_n is increasing: each upward-closure $\uparrow K_n$ contains those elements of S that may reach I in n or less steps. For the main property of WSTSs the sequence $\uparrow K_n$ eventually stabilises, correspondingly, K_n stabilises to a finite basis of $Prec^*(I)$. Thus, checking if s may reach I simply amounts to verify whether $s \in Prec^*(I)$, that is, whether $i_k \leq s$ for some i_k in the finite basis of $Prec^*(I)$.

[2] Commutative diagrams are a notion frequently exploited in the context of category theory [39]: similarly to [26] in our notation the down-left side of the diagram is the universal quantified one, the up-right side is the existential quantified one – this hypothesis will be assumed in the remainder of the paper as well.

Notice that while covering is decidable, pointwise reachability is *not*, namely, in general one cannot prove that for any two states $s, t \in S$ there is a sequence of transitions $s \longrightarrow^* t$, evolving s into t.

2.3 Other Decidable Properties

Even though covering is one of the properties that has received more interest in the context of WSTSs, it worth briefly outlining other decidability results that have been obtained.

Another verification approach for WSTSs is based on the idea of considering an initial state s, and building a tree of the states that can be reached from s. The leafs of this tree are either the deadlock nodes – which admits no more transitions – or the *subsumed nodes* – a leaf t is said to be subsumed by the ancestor t' if $t' \leq t$. The tree obtained in this way is called *finite reachability tree* of s, it is finite by definition of WSTS, and can be exploited to analyse the states reachable from s, in particular to state the following properties:

- **Termination.** A WSTS is said to have *transitive compatibility* if for all $s_1 \leq t_1$ and $s_1 \longrightarrow s_2$, there exists a (non void) sequence of transitions $t_1 \longrightarrow^+ t_2$ such that $s_2 \leq t_2$. Notice that this property is more general than strong compatibility of WSTSs. For these systems termination is decidable: there exist a non-terminating computation starting from s iff the finite reachability tree of s has some subsumed node.
- **Inevitability.** A WSTS is said to have *stuttering compatibility* if for all $s_1 \leq t_1$ and $s_1 \longrightarrow s_2$, there exists a (non void) sequence of transitions $t_1 \longrightarrow t_2 \longrightarrow \ldots \longrightarrow t_n$ such that $s_2 \leq t_n$ and $s_1 \leq t_i$ for all $i < n$. For these systems, which are less general than WSTSs with transitive compatibility, the following liveness property is decidable: given an upward-closed set I, any computation starting from s eventually goes outside I. To prove this property, called *inevitability* (whereas its opposite is called *control-state maintainability*), suffices it to show that any maximal path in the finite reachability tree of s has one node outside I.
- **Boundedness.** A WSTS is said to have *strict (reflexive and transitive) compatibility* if for all $s_1 < t_1$ and $s_1 \longrightarrow s_2$, there exists a sequence of transitions $t_1 \longrightarrow^* t_2$ such that $s_2 < t_2$. This properties is less general than (non-strict) compatibility. In the case $\leq$ is also a partial order (i.e, if it is also anti-symmetric) the *boundedness* problem is decidable: the set of reachable states of s is finite iff no leaf t in the finite reachability tree of s is subsumed by an ancestor t' such that $t' < t$.

The decidability results described here, which form the basic contribution of [26], are pictorially represented in Figure 1. Notice that covering does not

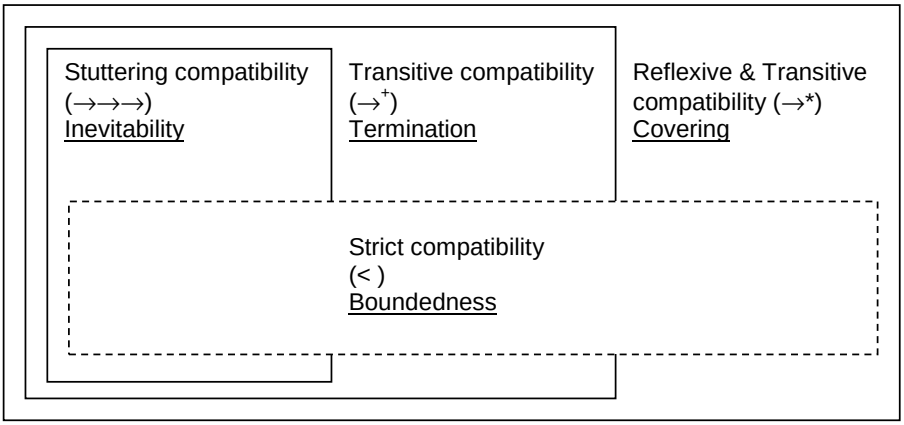

Fig. 1. Decidable properties in WSTSs

require strong compatibility, but rather the more general notion of reflexive and transitive compatibility: if for all $s_1 < t_1$ and $s_1 \longrightarrow s_2$, there should exist a sequence of transitions $t_1 \longrightarrow^* t_2$ such that $s_2 \leq t_2$. To summarise, while strong compatibility entails decidability of covering, termination, and inevitability, strict strong compatibility also entails decidability of boundedness. Based on the initial idea of [1], in [26] the notion of strong compatibility is extended to labelled transition systems (LTS) as well. A LTS $\langle S, \longrightarrow, A\rangle$ with well-quasi-order $\leq$ has *strong (labelled) compatibility* if for all $s_1 \leq t_1$ and $s_1 \xrightarrow{a} s_2$ for some $a \in A$, there exists a transition $t_1 \xrightarrow{a} t_2$ such that $s_2 \leq t_2$.

The notion of donwnard-WSTS described in [26] – which is used to entail decidability of the so-called *sub-covering* problem – is not discussed here for brevity, and for it currently seems to have more limited applicability.

3 Applications to Concurrent Systems

Even though the above formulation has been only recently introduced, there are already a number of relevant applications, some on traditional frameworks and formalisms such as Petri nets and process algebras [26], others to new and ad hoc models for specific systems, such as security models [21] and cache coherence protocols [18]. In order to understand the applicability of these results in the context of coordinated systems, it is interesting to provide a brief survey of some of these applications.

3.1 On the Multiset Nature of Concurrent Systems

A first observation is that a common way to structure the carrier S of a transition system $\langle S, \longrightarrow\rangle$ is as a set of multisets of atoms. This holds in

particular for the applications that are of interest here, namely, for concurrent systems: each atom can be considered as the state of a system subpart that can be replicated many times.

A first example of formalism that follows this schema is Petri nets [31]. A Petri net can be considered as a triple $\langle P, T, F\rangle$ where P is a finite set of places, T is a finite set of transitions, and F is a function of the kind $(P \times T) \cup (T \times P) \mapsto \mathbb{N}$, determining how many arcs goes from a place to a transition or vice-versa. The state of a Petri net at a given time, also called *marking*, is a function from places to natural numbers representing how many *tokens* reside in each place, which can be dually expressed as a multiset of places. Considering the set of places $P = \{p_1, p_2, p_3\}$, a valid marking can be expressed as the function $\{p_1 \mapsto 2, p_2 \mapsto 1, p_3 \mapsto 0\}$ or equivalently as the multiset $\{p_1, p_1, p_2\}$.

A second example is that of process algebras such as ACP [2], CCS [33] and π-calculus [38]. Considering as a simple example the following subset of ACP, represented by the syntax

$$P ::= 0 \mid a_1 \mid a_2 \mid \ldots \mid a_n \mid (P||P)$$

and the congruence rules:

$$0||P \equiv P \qquad P||P' \equiv P'||P \qquad P||(P'||P'') \equiv (P||P')||P''$$

An example of process is $a_1||a_1||a_2$, representing the parallel composition of three processes respectively executing actions a_1, a_1 and a_2, which is again in the form of a multiset of atoms.

3.2 Upward Compatibility of Transitions

In this framework, it is natural to exploit as quasi-order the inclusion relation over multisets, which is also shown to be a well-quasi-order [23] – an infinite sequence of multisets cannot indefinitely decrease. To devise a well-structure, then, it is necessary to identify those transition relations that satisfy given upward compatibility properties with respect to one such ordering.

3.2.1 Basic Petri net transitions

We start considering again the basic framework of Petri nets. A marking evolves as transitions are sequentially fired. Each transition can be characterised by the incoming arcs – each associated to the place where it comes from – and outgoing arcs – each associated to the place where it goes to. Hence, any transition can be described by a couple of multisets of places, or of markings, e.g. by the notation $\{p_1, p_1, p_2\} \longrightarrow \{p_1, p_2, p_3\}$, where $\{p_1, p_1, p_2\}$ is referred

to as the *pre-condition* and $\{p_1, p_2, p_3\}$ as the *effect*. Intuitively, a transition can be applied if the current marking includes the precondition, and when applied leads to the new marking obtained by removing the precondition and adding the effect. It comes not unexpected that the operational semantics of process algebras is defined by rules with a very similar syntax (and corresponding semantics). For instance, a rule of the kind $p_1||p_1||p_2 \longrightarrow p_1||p_2||p_3$ means that a process of the kind $P||p_1||p_1||p_2$ may evolve to the process $P||p_1||p_2||p_3$ – independently of P.

It turns out that if the transition system can be defined by rules of this kind only, then it has strong (and strict) compatibility with respect to the inclusion ordering. In fact, considering the process algebraic notation without loss of generality, from $P_1 < Q_1$ and $P_1 \longrightarrow P_2$ it turns out that $Q_1 = P_1||P_0$ (for some $P_0 > 0$), hence we have also $Q_1 \longrightarrow P_2||P_0$ with $P_2 < P_2||P_0$. This compatibility entails decidability of covering, termination, inevitability, and boundedness.

3.2.2 Transfer and reset arcs

Similar compatibility and decidability properties hold also if we consider for this framework transitions following the semantics of transfer arcs and reset arcs of Petri nets [15]. Transfer arcs can be used to transfer all the tokens of a place into another. In the process algebraic notation this mechanism can be denoted by the syntax proposed in [20]. For instance, the rule

$$P[P_1 \hookrightarrow Q_1, P_2 \hookrightarrow Q_2, \ldots, P_n \hookrightarrow Q_n] \longrightarrow Q$$

with $P_i, Q_i > 0$ and distinct P_i means that when applying transition $P \longrightarrow Q$, also any subprocess P_i is then transformed into a Q_i. Similarly, we can also consider the idea of reset arcs that drop all the markings from a place, by considering the above notation and allowing a Q_i to be the void process. For instance, the rule

$$P[P_0 \hookrightarrow 0] \longrightarrow Q$$

means that after applying transition $P \longrightarrow Q$, all the subprocesses of the kind P_0 must be dropped from the system state. It is easy to show that transfer arcs (transfer rules) preserves strict and strong compatibility, while reset arcs make the system losing strictness, making boundedness be no longer decidable.

3.2.3 Inhibitor arcs

One can also consider inhibitor arcs, whose rules are of the kind:

$$P \longrightarrow Q \qquad \text{if } P_0 \not< P$$

for a given P_0, the transition $P \longrightarrow Q$ is enabled only if P does not include some subprocess P_0 [3]. Introducing inhibitor arcs makes the system losing all its upward compatibility properties [4].

3.2.4 Rules with constraints: the MSR(C) notation

A further example of transitions, studied in [20], is provided by the formalism called MSR(C) (multiset rewriting with constraints). In this framework, rules are of the kind $\mathcal{N} \longrightarrow \mathcal{M} : \phi$. $\mathcal{N}$ and $\mathcal{M}$ – respectively referred to as the head and the body of the rule – are multisets of atomic formulae of the kind $p(x_1, \ldots, x_n)$, where p is a predicate name and elements x_i are variables. In this notation, symbol $|$ is used for composition and ϵ for the void multiset. ϕ is a predicate representing a constraint on the variables occurring in the head and the body, having the form of a so-called *difference constraint*:

$$\phi ::= \phi \wedge \phi \,|\, x = y + c \,|\, x > y + c \,|\, x \geq y + c \,|\, true, \quad c \in \mathbb{Z}$$

The notation $\mathcal{N} : \phi$ can be used to denote the upward-closed set of states that are greater than $\mathcal{N}$ (in the sense of multiset inclusion) and satisfy ϕ (see [5] for a formally precise definition).

This formalism is particularly appealing because it allows quite sophisticated interaction protocols to be easily expressed, including mutual exclusion and cache coherence policies. Moreover, we observe that as far as term substitution is concerned, this formalism holds a great promise in terms of supporting the verification of systems featuring communication – i.e., value-passing –, which is indeed a relevant issue of coordination. An example of application of this framework to coordination problems is analysed in Section 6.

Unfortunatelly, devising proper well-quasi-orders in this setting is quite challenging. In general, a system expressed in MSR(C) is not a WSTS, hence the convergence of backward reachability analysis is not ensured. In [5], a particular case of MSR(C) is proved to entail decidability of covering, featuring *(i) name constraints* – which are all the constraints of the kind $\phi \wedge \phi \,|\, x > y \,|\, x \geq y \,|\, x = y \,|\, true$ –, *(ii)* a *pointwise* entailment relation on constraints, and *(iii)* specific restrictions on the occurrence of variables in predicates. However, these results are quite recent and deserve a more thorough analysis, especially concerning the trade-off between applicability and efficiency of automatic verification.

[3] In particular, such a rule has to be interpreted as describing a global behaviour: P and Q are meant to model the whole system configuration instead of subparts of it.

[4] However, in particular cases such as BPP algebra [13] – which are nets where the left side of rules is composed by only one atom – adding inhibitor arcs enable downward compatibility, in which the sub-covering problem is decidable [26].

$$init \longrightarrow tick \quad [\text{start}] \qquad 0 \longrightarrow process \quad [\text{spawn}]$$
$$process||tick \longrightarrow acc \quad [\text{access}] \qquad acc \longrightarrow tick \quad [\text{release}]$$

Fig. 2. A simple locking mechanism for mutual exclusion

3.3 Safety and Covering

Among all the properties that become decidable in the framework of WSTSs, covering is the one that has received the greatest interest, since it is the most significant improvement provided by WSTSs to the infinite verification field. By the decidability of covering, given any upward-closed set U it is possible to compute a finite basis of $Prec^*(U)$, hence one can effectively check whether from a state s it is possible to reach an element in U – namely, to cover an element in the basis of U. Fortunatelly, it turns out that covering has a significant potential impact on the engineering of concurrent systems. The experience shows that often a concurrent system can be declared unsafe when given conditions hold locally to some of its processes, that is, when the system configuration includes an unsafe subpart. In this case, the unsafe states can be clearly characterised in terms of a certain upward-closed set.

As a very simple example, consider a system composed by a data structure and by processes that are willing to access that data. A frequent requirement is mutual exclusion: only one process at a time should be allowed to access the data. In this case, unsafe states are those that contain two processes that are accessing the data, denote this system subpart by $acc||acc$, hence the upward-closure of $acc||acc$ is the set of unsafe states according to this mutual exclusion requirement. Moreover, the set $Prec^*(\uparrow \{acc||acc\})$ is the set of those states that may eventually reach $\{acc||acc\}$. So, if some of the initial system states belongs to this set the whole system should be declared unsafe.

A simple example of transition system that is amenable to such a formal treatment is reported in Figure 2. A locking mechanism is exploited to regulate accesses: processes access the data by consuming a ticket (rule [access]), and repost it at the end of the work (rule [release]). Notice that the ability of spawning processes (rule [spawn]) makes this system infinite, and thus requires the abstraction supported by the WSTSs framework.

In very simple cases, one can try to perform a backward reachability analysis by hand, but even for the simplest locking mechanism this would imply a considerable work. Relying to some automatic tool is mandatory for verifying any non-trivial property. As an example, Delzanno developed an automatic verification tool for MSR(C), implemented as a Sicstus Prolog program exploiting the CLP library [20,19]. This prototype takes a set of unsafe states U and computes a finite basis for $Prec^*(\uparrow U)$. By applying the tool to the system

of Figure 2 we obtained the finite basis

$$\{acc||acc, tick||acc, init||acc, tick||tick, init||tick, init||init\}.$$

Since the initial state *init* is not in the upward-closure of this set, the whole system can be declared safe: independently of the dynamics of spawning and interactions, mutual exclusion is guaranteed.

4 Well-Structured Coordination

We observe that while the main practical applications of the framework of WSTSs are currently in the security, mutual-exclusion protocols, and cache coherence protocols areas, studying applicability to the context of coordination may both generalise them as well as provide a number of new and interesting usages. The main intuition behind the idea is that coordination problems seem to satisfy the conditions for applying WSTSs as outlined in the previous section. On the one hand, the semantics of coordination models and languages, as well as the behaviour of coordinated systems and infrastructures, is traditionally formalised by transition systems. On the other hand, these models typically take into account concurrency aspects and interaction, which are the basic features that can be taken into account by WSTSs.

In particular, our goal in this section is to focus on a general enough set of coordinated systems and models, and define a methodology for modeling their entities and abstractions as WSTSs, and for understanding the usefulness of the corresponding decidability results.

4.1 A General Framework for Coordinated Systems

We consider a general setting, where a coordinated system is composed by a *coordinated space* – populated by a number of coordinated entities – and a coordination space – where a *coordination medium* governs and rules the interactions between such entities. Without loss of generality, coordinated entities are supposed to evolve by performing actions enabled by the coordination medium. These actions are amenable to various interpretations, including the invocation of coordination primitives, the act of sending or receiving messages and signals, and so on. The resulting model endorses the notion of *coordination as a service* introduced in [46], that is, it interprets coordination as an interactive service provided to coordinated entities by an infrastructure. See [46] also for a thorough discussion about the applicability and usefulness of this conceptual and formal approach to coordination, and for a comparison with other viewpoints.

Formally, a coordinated system configuration $S \in \mathcal{S}$ is of the kind $M \otimes C$, where $M \in \mathcal{M}$ is the state of the coordination medium and $C \in \mathcal{C}$ is the configuration of the coordinated space. In particular, the coordinated space is seen as a set of coordinated entities, with syntax:

$$C ::= 0 \mid P \mid (C||C)$$

where $P \in \mathcal{P}$ is the state of a coordinated entity, also called here a *process*. The usual congruence rules for operator $||$ are also assumed.

Since we are interested in providing the greatest abstraction over the internal structure of either processes and coordination medium, we simply suppose that they are equipped by a specifically shaped labelled transition system semantics. In particular, the admissible evolutions of the coordination medium are described by a structure of the kind $\langle M_0, \mathcal{M}, \rightarrow_{\mathcal{M}}, A \cup \{\tau\} \rangle$, where $M_0 \in \mathcal{M}$ is the initial state of the coordination medium, A is the set of actions allowed – ranged over by metavariable a and its decorations –, and τ is the internal, silent action as in the usual process algebras. The transition relation is of the kind $\rightarrow_{\mathcal{M}} \subseteq \mathcal{M} \times (A \cup \{\tau\}) \times \mathcal{M}$: the coordination medium evolves as actions in A or the silent action occur. Similarly, for processes we consider the structure $\langle \mathcal{P}, \rightarrow_{\mathcal{P}}, A \rangle$, with $\rightarrow_{\mathcal{P}} \subseteq \mathcal{P} \times A \times \mathcal{P}$.

Silent actions are not considered for processes since they would represent the purely computational (i.e., algorithmic) aspects of the system, which are here abstracted away. On the other hand, silent actions in the coordination medium take into account the computations occurring in the coordination space, which actually involves the activity of governing and ruling interactions that coordination is all about.

To describe the evolution of the coordinated space, we also introduce the initial configuration $C_0 \in \mathcal{C}$, which includes the initial processes in the system, and the set $\mathcal{P}_0$ of processes that may enter the system at any time, tacking into account system opennes. The whole coordinated system evolution can be defined by the rules of transition system $\langle S_0, \mathcal{S}, \rightarrow_{\mathcal{S}} \rangle$, which are defined in terms of $\rightarrow_{\mathcal{M}}$ and $\rightarrow_{\mathcal{P}}$ as follows:

$$\begin{array}{lll} M \otimes C \rightarrow_{\mathcal{S}} M' \otimes C & \text{if} & M \xrightarrow{\tau}_{\mathcal{M}} M' \\ M \otimes C||P \rightarrow_{\mathcal{S}} M' \otimes C||P' & \text{if} & M \xrightarrow{a}_{\mathcal{M}} M' \text{ and } P \xrightarrow{a}_{\mathcal{P}} P' \\ M \otimes C \rightarrow_{\mathcal{S}} M \otimes C||P_0 & \text{if} & P_0 \in \mathcal{P}_0 \end{array}$$

The first rule handles silent actions in the coordination medium, the second rule deals with the interaction between a process and the coordination medium, while the third rule is about a new process P_0 entering the system.

As initial state we consider $S_0 = M_0 \otimes C_0$.

4.2 Well-Structure

We identify here sufficient conditions for well-structuring the above transition system, supporting a notion of upward-closure that may lead to the definition – and the verification – of meaningful safety properties of coordinated systems.

As a first step, we study a specific well-structure property for the coordination medium alone. Since the coordination medium is equipped with a labelled transition system we might simply exploit the strong (labelled) compatibility property reported in Section 2.3. However this compatibility is in general too severe: e.g., as far as only covering is concerned, silent actions may always be allowed, as they do not affect the evolution of coordinated entities. Hence, we introduce a *weaker* notion of compatibility – "weak" in the sense of similarity modulo usage of silent actions, as in the common terminology of observation semantics for labelled transition systems [30]. Therefore, we analyse how compatibility properties in the coordination medium impact the properties of the whole coordinated system. In particular, we provide results that can be easily obtained by the definition of compatibility, but which opens to the possibility of studying the well-structure of a coordinated system by focussing on the properties of its coordination media.

Given the labelled transition system $\langle X, \longrightarrow_X, A \cup \{\tau\}\rangle$, denote by $x \xrightarrow{(\tau)b}_X x'$ the existence of a non void sequence of transitions moving x to x', all labelled by τ actions except from exactly one action $b \in A \cup \{\tau\}$. Then, the well-quasi-order $\leq_X$ is said to have *weak compatibility*, if for all $x_1 \leq x_1'$ and $x_1 \xrightarrow{b}_X x_2$ with $b \in A \cup \{\tau\}$, there exists a non void sequence of transitions $x_1' \xrightarrow{(\tau)b}_X x_2'$ such that $x_2 \leq x_2'$.

Notice that *(i)* locally to a single transition, when b is τ itself, weak compatibility is equivalent to transitive compatibility, and moreover, *(ii)* the notion of weak compatibility is more general than strong compatibility (for labelled transition systems), and in particular, strong compatibility entails weak compatibility.

Then, the following theorem holds:

Theorem 4.1 *If the coordination medium $\langle M_0, \mathcal{M}, \rightarrow_{\mathcal{M}}, A \cup \{\tau\}\rangle$ has weak compatibility with respect to the well-quasi-order $\leq_{\mathcal{M}}$, then the binary relation $\leq_{\mathcal{S}}$ over $\mathcal{S}$ defined as*

$$M \otimes C \leq_{\mathcal{S}} M' \otimes C || C' \quad \Leftrightarrow \quad M \leq_{\mathcal{M}} M'$$

is a well-quasi-order, and makes $\langle S_0, \mathcal{S}, \rightarrow_{\mathcal{S}}\rangle$ be a WSTS with transitive com-

patibility.

Proof.

The well-quasi-ordering of $\leq_{\mathcal{S}}$ is easily shown to derive from the composition of the well-quasi-ordering of $\leq_{\mathcal{M}}$ and the well-quasi-ordering of the inclusion relation over multisets of processes applied to the coordinated system C. To show compatibility we must prove it separately for any of the three kinds of transitions for $\rightarrow_{\mathcal{S}}$: silent actions, spawning, and interaction.

- For silent actions of the coordination medium, compatibility derives from the (local) transitive compatibility of $\xrightarrow{\tau}_{\mathcal{M}}$.
- For actions involving the spawning of a new process, compatibility derives from the well-quasi-ordering of multiset inclusion.
- For interactions between the coordination medium and a process, consider any action $a \in A$ and the compatibility properties of coordination medium and processes alone. On the one hand, for any transition $M_1 \xrightarrow{a}_{\mathcal{M}} M_2$ and for $M_1 \leq_{\mathcal{M}} M_1'$, there exists an element M_2' such that $M_1' \xrightarrow{(\tau)a}_{\mathcal{M}} M_2'$ and $M_2 \leq_{\mathcal{M}} M_2'$. On the other hand, a similar construction is possible for the set of coordinated systems, supposing the existence of a process P moving to P' by action a and considering multiset inclusion. The two cases are represented as follows:

$$\begin{array}{ccc} M_1' & \xrightarrow{(\tau)a}_{\mathcal{M}} & M_2' \\ {\scriptstyle \leq_{\mathcal{M}}}\uparrow & & \uparrow{\scriptstyle \leq_{\mathcal{M}}} \\ M_1 & \xrightarrow{a}_{\mathcal{M}} & M_2 \end{array} \qquad \begin{array}{ccc} C||P||C' & \xrightarrow{a}_{\mathcal{C}} & C||P'||C' \\ {\scriptstyle \leq}\uparrow & & \uparrow{\scriptstyle \leq} \\ C||P & \xrightarrow{a}_{\mathcal{C}} & C||P' \end{array}$$

In particular, the diagram in the right side would commute even if instead of considering a single transition labelled with a in the upper side, one would consider the transition $\xrightarrow{(\tau)a}_{\mathcal{C}}$ of the left diagram, for coordinated systems being not affected by silent actions. Correspondingly, the following diagram commutes that is obtained by composing the two above:

$$\begin{array}{ccc} M_1' \otimes C||P||C' & \longrightarrow_{\mathcal{S}}{}^{+} & M_2' \otimes C||P'||C' \\ {\scriptstyle \leq_{\mathcal{S}}}\uparrow & & \uparrow{\scriptstyle \leq_{\mathcal{S}}} \\ M_1 \otimes C||P & \longrightarrow_{\mathcal{S}} & M_2 \otimes C||P' \end{array}$$

Hence the coordinated system is a WSTS with transitive compatibility.

□

So, if the coordination medium features weak compatibility, then both

covering and termination are decidable for the whole coordinated system. A more specific result can also be obtained, which allows for the decidability of inevitability as well:

Theorem 4.2 *If the coordination medium* $\langle M_0, \mathcal{M}, \rightarrow_{\mathcal{M}}, A \cup \{\tau\}\rangle$ *has strong (labelled) compatibility with respect to the well-quasi-order* $\leq_{\mathcal{M}}$*, then the binary relation over* $\mathcal{S}$ *defined as*

$$M \otimes C \leq_{\mathcal{S}} M' \otimes C||C' \quad \Leftrightarrow \quad M \leq_{\mathcal{M}} M'$$

is a well-quasi-order, and makes $\langle S_0, \mathcal{S}, \rightarrow_{\mathcal{S}}\rangle$ *be a WSTS with strong compatibility.*

Proof. Similar to the previous case. □

Since inevitability is now decidable, for any upward-closed set I it is always possible to state whether from the initial state any path will inevitably go outside I, or dually (control-state maintainability) whether at least one path indefinitely stays inside I.

Finally, boundedness can be accommodated by the following theorem:

Theorem 4.3 *If the coordination medium* $\langle M_0, \mathcal{M}, \rightarrow_{\mathcal{M}}, A \cup \{\tau\}\rangle$ *has strict strong (labelled) compatibility with respect to the well-quasi-partial-order* $\leq_{\mathcal{M}}$*, then the binary relation over* $\mathcal{S}$ *defined as*

$$M \otimes C \leq_{\mathcal{S}} M' \otimes C||C' \quad \Leftrightarrow \quad M \leq_{\mathcal{M}} M'$$

is a well-quasi-partial-order, and makes $\langle S_0, \mathcal{S}, \rightarrow_{\mathcal{S}}\rangle$ *be a WSTS with strict strong compatibility.*

Proof. Similar to the previous cases. □

In this more specific case boundedness is decidable as well. However, notice that if the set $\mathcal{P}_0$ is not void, then the system is always unbound, for the rule to spawn a process in $\mathcal{P}_0$ can always be fired. The results of these three theorems are summarised in the table of Figure 3.

4.3 The Case a Shared Dataspace with Regulated Agents

To make the above arguments more precise, we here narrow the framework to a more specific, yet relevant case of coordinated systems. First of all we consider the coordination medium as a shared, distributed dataspace [8], made of a multiset of atomic data. Then, we suppose that coordinated entities are agents playing given roles in their society, and are correspondingly constrained to interact according to specific, regulated protocols – e.g. constrained by

Coordination Medium	Coordinated System	Decidable Properties
Weak Comp.	Transitive Comp.	Ter + Cov
Strong Comp.	Strong Comp.	Ter + Cov + Inev
Strict Strong Comp.	Strict Strong Comp.	Ter + Cov + Inev + Bou

Fig. 3. Decidability Results for the Well-Structured Coordination

some kind of infrastructural support such as the notion of Agent Coordination Context [36,34].

A shared dataspace is modelled as a multiset of items in the finite set D, ranged over by metavariable d and its decorations, with syntax:

$$M ::= 0 \mid d \mid (M||M)$$

As discussed in [46], existing formalisations exploit items in the dataspace to encode various abstractions: tuples in tuple-based coordination models [9], pending requests and replies [44], pending transactions [12], triggers for notifications [10], programmed reactions [35], and so on.

Following the discussion in Section 3.2, when considering for the coordination medium transition rules of the general kind

$$M \xrightarrow{a}_{\mathcal{M}} N$$

or even the more general case

$$M[M_1 \hookrightarrow N_1, M_2 \hookrightarrow N_2, \ldots, M_n \hookrightarrow N_n] \xrightarrow{a}_{\mathcal{M}} N$$

along with multiset inclusions as well-quasi-order, strong and weak compatibility are automatically entailed, and so is the well-structure of the whole coordinated system.

Coordinated entities can be modelled as non-deterministic processes that sequentially invoke coordination primitives. So, we consider the syntax

$$P ::= 0 \mid a \mid P; P \mid P + P$$

the corresponding transition rules

$$a.P \xrightarrow{a}_{\mathcal{P}} P \qquad P + P' \xrightarrow{a}_{\mathcal{P}} P'' \text{ if } P \xrightarrow{a}_{\mathcal{P}} P'' \qquad P; P' \xrightarrow{a}_{\mathcal{P}} P''; P' \text{ if } P \xrightarrow{a}_{\mathcal{P}} P''$$

and the congruence rules

$$0 + P \equiv P \qquad P + P' \equiv P' + P \qquad P + (P' + P'') \equiv (P + P') + P''$$
$$(P + P'); P'' \equiv P; P'' + P'; P''$$

As usual, operator ; is used for sequential composition and + for nondeterministic choice. For instance, the process $P_1 = (a+b+c); (a+b+c); (a+b)$ is the process that can initially execute either actions a, b, or c, then again one of the three, and finally either a or b. Similarly, the process $P_2 = (a; b; b) + (c; c; c)$ can either execute the sequence $a; b; b$ or $c; c; c$.

Under this structure for coordinated entities, it is quite natural to assign to C_0 and $\mathcal{P}_0$ certain notions of role, understood as the actions that agent are allowed to execute. In particular, the initial configuration C_0 can be seen as the composition of all the processes playing the *initial roles* in the system – that is, roles which are statically assigned at startup-time. For instance, $C_0 = P_1||P_1$ means that at startup, the system should feature two instances of the process P_1. Initial roles are generally used to denote those processes in charge of managing the coordination medium, either preparing its initial state or controlling its state as interactions with coordinated entities occur. On the other hand, the set $\mathcal{P}_0$ can be understood as the set of all the *open roles* of the system, which can be played by coordinated entities unpredictably entering the system to exploit its services. For instance, by imposing $\mathcal{P}_0 = \{P_1, P_2\}$, both roles P_1 and P_2 defined above can be played by new clients entering the system.

It is important to point out that the choice of a given well-quasi-order is particularly crucial, for it should accommodate the trade-off between decidability of properties and usefulness of properties. In our framework, the well-quasi-ordering obtained by composing multiset inclusion for coordination media and coordinated processes allows for dealing with the following three kinds of unsafe states, whose usefulness is shown in the next section.

- **Medium inconsistency.** If M_U is an unsafe state for the coordination medium – e.g. denoting an inconsistency –, then $M_U \otimes 0$ can be used as basis for the corresponding unsafe system configurations.
- **Unsafe accesses.** Say $P_1|| \ldots ||P_n$ is a configuration for the coordinated space denoting an unsafe state – such as a combination of process states that invalidates a mutual exclusive access to some resource –, then $0 \otimes P_1|| \ldots ||P_n$ can be used as basis for the corresponding unsafe system configurations.
- **Unsafe configurations.** Sometimes, unsafe states can be characterised only in terms of two conditions concurrently occurring in the coordination

medium and the coordinated space – e.g. when a process state does not correspond to the expected medium state. In this general case, the upward-closure of $M \otimes C$, where M denotes the condition in the medium and C that in the coordinate space, denotes the corresponding set of unsafe configurations.

5 An Application to Linda

LINDA is the paradigmatic example of coordination model [29]. It is not only one of the main models from which the whole coordination field has been developed, but is also the basic framework above which a number of extensions/adaptations have been devised, featuring e.g. programmable behaviour (ReSpecT) [35], multiple access to tuple spaces (LOGOP) [41], transactions and expiring tuples (JavaSpaces) [27], and Prolog-like features (Shared Prolog) [7] just to mention a few of them. So, we consider here the LINDA coordination model, both because of its simplicity, and also because of the potential extension of our results to other coordination models such as those mentioned above.

5.1 The Well-Structure of Linda

The traditional approach to the formalisation of a coordination model is based on expressing a coordinated system in terms of a process algebra [3], describing its admissible evolutions by a TS semantics. As outlined in the previous section, the distributed state of a coordinated system is seen as a parallel composition of *agents* – interacting through coordination primitives – and *items* of the shared dataspace. Execution of coordination primitives is modelled similarly to synchronous communications as for instance in CCS [33].

This approach is traditional for the LINDA case as well, see e.g. [9,8,46,4]. The dataspace is represented as a multiset of data items $t \in T$ (where T is supposed to be finite), representing the tuples occurring in the tuple space. We consider the three basic coordination primitives `in`, `rd`, and `out` without matching – that is, specifying a concrete tuple instead of a tuple-template as e.g. in [9]. We have the syntax:

$$P ::= 0 \mid \texttt{in}(t) \mid \texttt{rd}(t) \mid \texttt{out}(t) \mid P; P \mid P + P$$

In particular, the coordination medium representing a LINDA tuple space can be described by the transition system $\langle M_0, \mathcal{M}, \rightarrow_{\mathcal{M}}, A \rangle$, where $M_0 = 0$, $\mathcal{M}$ is any multiset of tuples, $A = \{\texttt{in}(t), \texttt{rd}(t), \texttt{out}(t) : t \in T\}$ is the set of invoca-

tions for the three primitives, and $\rightarrow_{\mathcal{M}}$ is defined by the three simple rules:

$$t \xrightarrow{\texttt{rd}(t)}_{\mathcal{M}} t \qquad t \xrightarrow{\texttt{in}(t)}_{\mathcal{M}} 0 \qquad 0 \xrightarrow{\texttt{out}(t)}_{\mathcal{M}} t$$

Since the set of actions A must be finite, we suppose that the set of tuples is finite as well. The first rule defines the semantics of the `rd` primitive: when the requested tuple occurs in the space, then the action is allowed. The second rule handles the case of the `in` primitive: as for the `rd` primitive the action is allowed when the requested tuple occurs, but then such a tuple is also removed. The third rule defines the (ordered) semantics of the `out` primitive: the tuple is inserted in the space as the action is executed. It is easy to recognise that this specification is strict and strong compatible w.r.t multiset inclusion, as its rules adhere to the structure of basic Petri nets. Hence, for Theorem 4.2 we have that the LINDA coordinated system is a WSTS with strict and strong compatibility, where inevitability, termination, boundedness and covering are decidable. Notice that this result is not affected by the fact that processes have finite behaviour: even by adding the ability to recursively define processes the compatibility properties remain unchanged – the ability of indefinitely spawning processes being conceptually equivalent to such an addition. The main reason is that the well-quasi-order exploited for processes never works directly at the level of processes' states, but simply relates multiset of processes by the inclusion relation.

It worth noting that compatibility does not hold when considering the LINDA extension with predicative queries `rdp` and `inp`. Consider e.g. also the primitive $\texttt{inp}(t)$ and the two corresponding informal rules, expressing global evolutions:

$$M||t \xrightarrow{\texttt{inp}(t)}_{\mathcal{M}} M \qquad\qquad M \xrightarrow{\texttt{inp}(t)}_{\mathcal{M}} M \quad \text{if } \{t\} \not\leq M$$

The second rule, in particular, can be understood as a transition in a Petri net with inhibitory arcs, which makes the system losing the upward compatibility property.

5.2 The Dining Philosophers Example

The dining philosophers example is a traditional case study for the expressiveness of coordination models [24,22,28], whose main idea is here considered for testing the applicability of WSTSs and their verification to coordinated systems.

The dataspace is used as a means to rule the access of 4 (kinds of) processes to 4 resources, by exploiting 4 tickets – figuratively, 4 hungry philosophers

accessing 4 dishes by 4 forks. Each resource is associated to two tickets, each ticket is shared between two resources – figuratively, the dishes being over a circular table separated by forks. In order to use a specific resource, the process must first get the two associated tickets, which should be released when the resource is no longer used. This problem is solved in LINDA by considering a tuple space where the tuples $t1, t2, t3, t4$ – representing the tickets – initially occur, and with the set of open roles $\mathcal{P}_0$ including the four processes:

$$P1 = \texttt{in}(t1).\texttt{in}(t2).\texttt{out}(t1).\texttt{out}(t2)$$
$$P2 = \texttt{in}(t2).\texttt{in}(t3).\texttt{out}(t2).\texttt{out}(t3)$$
$$P3 = \texttt{in}(t3).\texttt{in}(t4).\texttt{out}(t3).\texttt{out}(t4)$$
$$P4 = \texttt{in}(t4).\texttt{in}(t1).\texttt{out}(t4).\texttt{out}(t1)$$

Notice that the version of the dining philosophers considered here deals with opennes: instead of just four entities we suppose to have an unbounded number of entities each playing any of the four roles.

Coordinated systems can be modelled in terms of a number of Petri net-like rules by: *(i)* considering a bootstrap rule preparing the initial system configuration, *(ii)* adding one spawning rule for each open role, *(iii)* adding one rule for each combination of (corresponding) interactions in the coordination medium and in a process. The resulting specification is reported in Figure 4. Notice that we substituted any process state by a corresponding atomic item, e.g. writing *in1-out4-out1* instead of $\texttt{in}(t1);\texttt{out}(t4);\texttt{out}(t1)$. Whereas this is only a syntactic change, it allows us to exploit the MSR(C) tool to automatically verify the safety properties of interest.

In the general case, the system can be considered safe when two adjacent resources are not exploited concurrently, that is, when no couple of adjacent processes are in the state where the two tickets have been consumed. Therefore, we consider the set of four unsafe states:

$$\{\ \textit{out1-out2} \mid \textit{out2-out3},\ \textit{out2-out3} \mid \textit{out3-out4},\ \textit{out3-out4} \mid \textit{out4-out1},\ \textit{out4-out1} \mid \textit{out1-out2}\ \}$$

Just to report about efficiency, the MSR(C) tool verifies this system's safety in 17 steps of backward analysis and about 1 minute of computation [5]. Another safety condition of interest is deadlock prevention: the system should prevent the case where four processes have got one ticket and are waiting for the other.

[5] The reference architecture for these measures is a 2GHz Intel Pentium III PC.

$$init \longrightarrow t1 \,|\, t2 \,|\, t3 \,|\, t4 \qquad [\text{start}]$$

$$\epsilon \longrightarrow in1\text{-}in2\text{-}out1\text{-}out2 \qquad [\text{spawn1}]$$
$$\epsilon \longrightarrow in2\text{-}in3\text{-}out2\text{-}out3 \qquad [\text{spawn2}]$$
$$\epsilon \longrightarrow in3\text{-}in4\text{-}out3\text{-}out4 \qquad [\text{spawn3}]$$
$$\epsilon \longrightarrow in4\text{-}in1\text{-}out4\text{-}out1 \qquad [\text{spawn4}]$$

$$in1\text{-}in2\text{-}out1\text{-}out2 \,|\, t1 \longrightarrow in2\text{-}out1\text{-}out2 \qquad [\text{getL1}]$$
$$in2\text{-}out1\text{-}out2 \,|\, t2 \longrightarrow out1\text{-}out2 \qquad [\text{getR1}]$$
$$out1\text{-}out2 \longrightarrow out2 \,|\, t1 \qquad [\text{putL1}]$$
$$out2 \longrightarrow t2 \qquad [\text{putR1}]$$

$$in2\text{-}in3\text{-}out2\text{-}out3 \,|\, t2 \longrightarrow in3\text{-}out2\text{-}out3 \qquad [\text{getL2}]$$
$$in3\text{-}out2\text{-}out3 \,|\, t3 \longrightarrow out2\text{-}out3 \qquad [\text{getR2}]$$
$$out2\text{-}out3 \longrightarrow out3 \,|\, t2 \qquad [\text{putL2}]$$
$$out3 \longrightarrow t3 \qquad [\text{putR2}]$$

$$in3\text{-}in4\text{-}out3\text{-}out4 \,|\, t3 \longrightarrow in4\text{-}out3\text{-}out4 \qquad [\text{getL3}]$$
$$in4\text{-}out3\text{-}out4 \,|\, t4 \longrightarrow out3\text{-}out4 \qquad [\text{getR3}]$$
$$out3\text{-}out4 \longrightarrow out4 \,|\, t3 \qquad [\text{putL3}]$$
$$out4 \longrightarrow t4 \qquad [\text{putR3}]$$

$$in4\text{-}in1\text{-}out4\text{-}out1 \,|\, t4 \longrightarrow in1\text{-}out4\text{-}out1 \qquad [\text{getL4}]$$
$$in1\text{-}out4\text{-}out1 \,|\, t1 \longrightarrow out4\text{-}out1 \qquad [\text{getR4}]$$
$$out4\text{-}out1 \longrightarrow out1 \,|\, t4 \qquad [\text{putL4}]$$
$$out1 \longrightarrow t1 \qquad [\text{putR4}]$$

Fig. 4. The Dining Philosophers Example

This can be expressed by the unsafe state:

$$in2\text{-}out1\text{-}out2 \,|\, in3\text{-}out2\text{-}out3 \,|\, in4\text{-}out3\text{-}out4 \,|\, in1\text{-}out4\text{-}out1$$

The specification reported in Figure 4 is clearly not safe from this point of

view, since four processes can be spawned each accessing the first ticket. The MSR(C) tool in this case discovers the problem and shows a trace from *init* to the unsafe state in 9 steps of backward analysis and 1 minute of computation. It is well known (see e.g. [28]) that the problem can be avoided by reversing the order of ticket access in one process, e.g. by making $P1$ first accessing $t2$ and then $t1$: the resulting system is proved safe in 16 steps and about 3 minutes of computation.

According to the terminology introduced in Section 4.3, all the unsafe states seen so far are unsafe accesses, that is, are about a wrong state in the coordinated space. An example of unsafe states due to a medium inconsistency is the following:

$$\{t1 \,|\, t1, t2 \,|\, t2, t3 \,|\, t3, t4 \,|\, t4\}$$

which is used to ensure that the coordination medium never holds more copy of a ticket – which is proved unfeasible in 11 steps and few seconds. An example of unsafe configuration – with a condition on both the coordination medium and the process – is instead:

$$\begin{array}{l} \{\ \mathit{in2\text{-}out1\text{-}out2} \,|\, t1,\ \mathit{in3\text{-}out2\text{-}out3} \,|\, t2, \\ \quad \mathit{in4\text{-}out3\text{-}out4} \,|\, t3,\ \mathit{in1\text{-}out4\text{-}out1} \,|\, t4 \quad \} \end{array}$$

used to prevent the case where a ticket occurs in the space also after a process is expected to have removed it – proved unfeasible in 8 steps and few seconds.

Other than safety, the compatibility properties of our formalisation entail decidability of termination, boundedness, and inevitability – in particular, exploiting the finite reachability tree methodology [6]. For instance, it is easy to see that the system admits some infinite evolution – due to the rules to spawn processes inserting subsumed nodes in the tree – and that it is not bounded – again due to the spawning rules inserting nodes subsumed by strictly smaller nodes. Concerning liveness properties, any upward-closed set can be shown to be either inevitably escaped, or conversely, to be a maintainable control state. For instance, the state $t1$ is a maintainable control state – suffices it to keep spawning processes and serving only processes $P2$ and $P3$.

6 Towards New Applications

The example of the four dining philosophers shows that the framework of WSTSs holds good promises in terms of verifying the safety properties of coordinated systems. For a wide range of coordination media, including LINDA and

[6] This methodology, however, is not implemented by the MSR(C) tool, nor [26] mentioned any existing implementation.

all the others whose coordination laws can be described as rules in Petri-nets (possibly with reset and transfer arcs), the decidability results of WSTSs are satisfied. In particular, a number of interesting safety properties are decidable in a quite effective and efficient way, including properties of the coordination medium and of the agent roles.

Still, in spite of the effectiveness and efficiency we experienced in the automatic verification by backward analysis, one might also observe that as far as modelling coordinated systems is concerned, increasing the expressiveness of the notation – not only making specifications more compact but also allowing to model more cases – is an important challenge, that the work on multiset rewriting with constraints by Delzanno has started addressing [18,20]. In particular, the trade-off between enhanced expressiveness and decidability/efficiency is clearly crucial, and is the subject of state-of-the-art research in this context. It is interesting to analyse the impact of this issue in the verification of safety properties for coordinated systems, which can also highlight the potential of current formal studies for the coordination field.

In particular, we leave out the particular case of LINDA coordination model for generality, and devise an interesting extension to the dining philosophers example featuring constraints of MSR(C) [7] . In this new version the number of resources/tickets is not limited to a specific value – 4 in our previous case – but can dynamically increase inside the coordination medium without any bound. In exchange of the expected increase of complexity for proving safety, verifying properties in this example also ensures their validity in the traditional example of dining philosophers, independently of the number of forks. The key idea is to exploit variables and constraints on variables to define generic tickets and processes, with a daemon activity inside the coordination medium in charge of adding new tickets and control the requests coming from processes. The system reported in Figure 5 implements this idea.

The coordination medium includes a set of terms of the kind

$$fst(M_0) \mid r(M_0, M_1) \mid \ldots \mid r(M_{n-1}, M_n) \mid lst(M_n),$$

where each $r(M_i, M_{i+1})$ denotes a resource regulated by the two tickets $t(M_i)$, $t(M_{i+1})$ occurring in the medium. This configuration is initialised with only one resource by rule [init], expanded with one more resource each time by rule [inc], and finalised by rule [last], which closes the loop by creating the resource $r(M_n, M_0)$ and dropping the term $lst(M_n)$. Rule [spawn] allows for any process entering the system, independently of the pair of tickets it wants to exploit

[7] This new example is actually directly realisable by an extension of tuple spaces with programmed behaviour, such as e.g. by ReSpecT tuple centres [35].

$$init \longrightarrow fst(M) \,|\, r(M,N) \,|\, lst(N) \,|\, t(M) \,|\, t(N) : N > M \qquad \text{[start]}$$

$$r(M,N) \,|\, lst(N) \longrightarrow r(M,N) \,|\, r(N,O) \,|\, lst(O) \,|\, t(O) : O > N \qquad \text{[inc]}$$

$$fst(M) \,|\, r(N,O) \,|\, last(O) \longrightarrow fst(M) \,|\, r(N,O) \,|\, r(N,M) : O > M \qquad \text{[last]}$$

$$\epsilon \longrightarrow wait(M,N) : true \qquad \text{[spawn]}$$

$$r(M,N) \,|\, wait(M,N) \,|\, t(M) \longrightarrow r(M,N) \,|\, wait2(M,N) : true \qquad \text{[first]}$$

$$r(M,N) \,|\, wait2(M,N) \,|\, t(N) \longrightarrow r(M,N) \,|\, use(M,N) : true \qquad \text{[second]}$$

$$use(M,N) \longrightarrow t(M) \,|\, t(N) : true \qquad \text{[release]}$$

Fig. 5. The Dining Philosophers Example in the Unbounded Case

(which may not yet being created or may not properly identify any resource). Rules [first] and [second] handle a process consuming the two tickets, which is allowed only if the requested tickets conform to a term $r(M,N)$. Finally, rule [release] models the process releasing the tickets.

The reader may enjoy the expressiveness of this specification with respect to the bounded one of Figure 4. Simplicity also holds when expressing the mutual exclusion unsafe state, which is now simply $\{use(N,M) \,|\, use(M,O)\}$. Unfortunately, dealing with constraints in this way makes it hard to denote whole configurations, hence the deadlock unsafe state cannot be represented by an upward-closed set with finite basis. As discussed in Section 3.2, this happens since in general the MSR(C) formalism does not allow for a well-quasi-order, hence it is unable to guarantee upward-closed sets to be finite, as well as the backward analysis to converge. In fact, the MSR(C) tool is unable to directly verify the safety with respect to $\{use(N,M) \,|\, use(M,O)\}$. [8]

However, we found that the problem can be solved by somehow guiding the backward analyser towards the solution, by adding to the unsafe state to be verified other unsafe states, from very easy ones to increasingly more complex ones. The system of Figure 5 is proved to be safe in 5 steps and few

[8] Notice that this is not a problem of this specific CLP implementation, since the system of Figure 5 not even satisfies the sufficient properties for convergence reported in [5]. At this time, we are unaware of any work defining a well-quasi-order for it.

seconds with respect to the greater set of unsafe states:

$$
\begin{array}{ll}
\{\ t(N) \mid t(N), & \text{(duplicated tickets)} \\
\quad wait2(N,M) \mid wait2(N,O), & \text{(double consumption)} \\
\quad t(N) \mid wait2(N,O), & \text{(false consumption)} \\
\quad use(N,M) \mid use(M,O) & \text{(concurrent access)} \\
\}
\end{array}
$$

The three "structural" unsafe states added follow the tree kinds of unsafe states in our framework of well-structured coordination: tickets cannot be duplicated (medium inconsistency), two processes may not have consumed the same ticket (unsafe access), and the tickets must disappear after their consumption (unsafe configuration).

Finally, notice that this example of coordination not only takes into account open systems and unbound data, but also considers aspects related to communication: the modelled system can be interpreted as if the coordination medium allowed processes to exploit resources only as far as they provide right information about the name of tickets. We believe that this example provides useful insights about the potential of this tool for proving the safety of complex and full-featured coordinated systems – see e.g. also [6] –, which will be the subject of our future research in this field.

7 Conclusions

The coordination field promotes a rigorous approach to the engineering of systems featuring complex interaction patterns. Most notably, formal models have often been used to describe syntax and semantics of coordination languages, and to characterise the dynamics of coordination architectures and infrastructures. Expected advantages of formal approaches include abstracting away from unnecessary details [45], avoiding the underspecification of relevant aspects [11], and intercepting design errors [12]. However, we believe that a main application of formal models, the automatic verification of system properties, has not received the deserved interest in the context of coordination models.

The main goal of this paper, then, is to try to fill the gap between the recent results obtained in the context of verification and their applicability to the nowadays coordination models and systems. From the work on parameterized verification and the framework of well-structured transition systems, we introduced the notion of well-structured coordination, a framework for

coordination where interesting properties concerning safety, liveness, and termination are decidable. Most notably, we applied this result to the LINDA coordination model, and show how a paradigmatic example of application – namely, the dining philosophers problem – can be viewed as well-structured, and how its safety properties can be proved by existing automatic tools. Also, the current frontier of infinite verification is discussed, and the potential applicability to the new and more complex coordination models and systems is outlined.

We believe that this work may open the way to a number of interesting researches and future works. First of all, the well-structure properties of other coordination models has to be investigated, so as to deepen the actual applicability of the existing results in the verification of properties. In particular, it would be interesting to exploit the framework of well-structured transition systems to analyse properties of programmable coordination media – such as ReSpecT [35] –, which are meant to provide a uniform means to express different kinds of coordination laws. Also, we aim at extending the framework of well-structured coordination to include other relevant aspects such as communication, exploiting the current results obtained in the context of multiset rewriting with constraints. Finally, it is interesting also to deepen the comparison of our approach with other techniques in the infinite verification field, such as the work on the μCLR tool.

References

[1] Abdulla, P. A., K. Cerans, B. Jonsson and Y.-K. Tsay, *General decidability theorems for infinite-state systems*, in: *11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)* (1996), pp. 313–321.

[2] Bergstra, J. A. and J. Klop, *Algebra of communicating processes with abstraction*, Theoretical Computer Science **37** (1985), pp. 77–121.

[3] Bergstra, J. A., A. Ponse and S. A. Smolka, editors, "Handbook of Process Algebra," North-Holland, 2001.

[4] Bonsangue, M. M., J. N. Kok and G. Zavattaro, *Comparing coordination models and architectures using embeddings*, Science of Computer Programming **46** (2003).

[5] Bozzano, M. and G. Delzanno, *Beyond parameterized verification*, in: *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS **2280**, Springer-Verlag, 2002 pp. 221–235.

[6] Brim, L., D. Gilber, J.-M. Jacquet and M. Kretínský, *Multi-agent systems as concurrent constraint processes*, in: *SOFSEM 2001: Theory and Practice of Informatics: 28th Conference on Current Trends in Theory and Practice of Informatics*, LNCS **2234**, Springer-Verlag, 2001 pp. 201–211.

[7] Brogi, A. and P. Ciancarini, *The concurrent language Shared Prolog*, Transactions on Programming Languages and Systems **13** (1991), pp. 99–123.

[8] Busi, N., P. Ciancarini, R. Gorrieri and G. Zavattaro, *Coordination models: A guided tour*, in: A. Omicini, F. Zambonelli, M. Klusch and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer-Verlag, 2001 pp. 6–24.

[9] Busi, N., R. Gorrieri and G. Zavattaro, *A process algebraic view of Linda coordination primitives*, Theoretical Computer Science **192** (1998), pp. 167–199.

[10] Busi, N. and G. Zavattaro, *Event notification in data-driven coordination languages: Comparing the ordered and unordered interpretations*, in: *The 2000 ACM Symposium on Applied Computing (SAC 2000)* (2000), pp. 233–239.

[11] Busi, N. and G. Zavattaro, *Event notification in data-driven coordination languages: Comparing the ordered and unordered interpretations*, in: *The 2000 ACM Symposium on Applied Computing (SAC 2000)* (2000), pp. 233–239, track on Coordination Models, Languages and Applications.

[12] Busi, N. and G. Zavattaro, *On the serializability of transactions in JavaSpaces*, in: U. Montanari and V. Sassone, editors, *ConCoord: International Workshop on Concurrency and Coordination*, Electronic Notes in Theoretical Computer Science **54** (2001).

[13] Christensen, S., Y. Hirshfeld and F.Moller, *Decidable subsets of CCS*, The Computer Journal **37** (1969), pp. 233–242.

[14] Ciancarini, P., F. Franzé and C. Mascolo, *Using a coordination language to specify and analyze systems containing mobile components*, ACM Transaction on Software Engineering **9** (2000), pp. 167–198.

[15] Ciardo, G., *Petri nets with marking-dependent arc cardinality: Properties and analysis*, in: *Applications and Theory of Petri Nets*, LNCS **815**, Springer-Verlag, 1994 pp. 179–198.

[16] Clarke, Jr., E. M., O. Grumberg and D. A. Peled, "Model Checking," The MIT Press, 2001.

[17] Dechering, P. and I. van Langevelde, *On the verification of coordination*, in: A. Porto and G.-C. Roman, editors, *Coordination Languages and Models*, LNCS **1906**, Springer-Verlag, 2000 pp. 335–340.

[18] Delzanno, G., *Automatic verification of parameterized cache coherence protocols*, in: *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, LNCS **1855**, Springer-Verlag, 2000 pp. 53–68.

[19] Delzanno,
G., *A constraint-based framework for the automated verification of infinite-state concurrent systems* (2002), prototype available at: http://www.disi.unige.it/person/DelzannoG/MSR/.

[20] Delzanno, G., *An overview of MSR(C): A CLP-based framework for the symbolic verification of parameterized concurrent systems*, in: M. Comini and M. Falaschi, editors, *Electronic Notes in Theoretical Computer Science*, Electronic Notes in Theoretical Computer Science **76**, Elsevier Science Publishers, 2002 .

[21] Delzanno, G., *Automatic verification of security protocols via constraint-based symbolic model checking*, Technical Report DISI, DISI, Genova (2003).

[22] Denti, E., A. Natali and A. Omicini, *On the expressive power of a language for programming coordination media*, in: *1998 ACM Symposium on Applied Computing (SAC'98)*, Atlanta (GA), 1998, pp. 169–177.

[23] Dickson, L. E., *Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors*, American Journal of Mathematics **35** (1913), pp. 413–422.

[24] Dijkstra, E., "Co-operating Sequential Processes," Academic Press, London, 1965.

[25] Esparza, J., A. Finkel and R. Mayr, *On the verification of broadcast protocols*, in: *14th Annual IEEE Symposium on Logic in Computer Science (LICS 1999)* (1999), pp. 352–359.

[26] Finkel, A. and Ph. Schnoebelen, *Well-structured transition systems everywhere!*, Theoretical Computer Science **256** (2001), pp. 63–92.

[27] Freeman, E., S. Hupfer and K. Arnold, "JavaSpaces: Principles, Patterns, and Practice," Addison-Wesley, 1999.

[28] Gaspari, M. and G. Zavattaro, *An actor algebra for specifying distributed systems: the hurried philosophers case study*, in: G. R. G. Agha, F. de Cindio, editor, *Concurrent Object-oriented Programming and Petri Nets*, Lecture Notes in Computer Science **2001** (2001), pp. 428–444.

[29] Gelernter, D., *Generative communication in Linda*, ACM Transactions on Programming Languages and Systems **7** (1985), pp. 80–112.

[30] Glabbeek, R. v., *The linear time – branching time spectrum I. The semantics of concrete, sequential processes*, in: Bergstra et al. [3] pp. 3–100.

[31] Karp, R. and R. Miller, *Parallel programming schemata*, Journal on Computer and System Sciences **3** (1969), pp. 147–195.

[32] Kruskal, J., *The theory of well-quasi-ordering: A frequently discovered concept*, Journal on Combinatorial Theory, Series A **13** (1972), pp. 297–305.

[33] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

[34] Omicini, A., *Towards a notion of agent coordination context*, in: D. C. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, CRC Press, 2002 pp. 187–200.

[35] Omicini, A. and E. Denti, *From tuple spaces to tuple centres*, Science of Computer Programming **41(3)** (2001).

[36] Omicini, A. and A. Ricci, *Reasoning about organisation: Shaping the infrastructure*, AI*IA Notizie **XVI** (2003), pp. 7–16.

[37] Orzan, S. and J. van de Pol, *Distribution of a simple shared dataspace architecture*, in: A. Brogi and J.-M. Jacquet, editors, *1st International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA '02)*, Electronic Notes in Theoretical Computer Science **68(3)**, Elsevier Science B. V., 2003 .

[38] Parrow, J., *An introduction to the π-calculus*, in: Bergstra et al. [3] pp. 479–544.

[39] Pierce, B. C., "Basic Category Theory for Computer Scientists," The MIT Press, 1998.

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

[41] Snyder, J. and R. Menezes, *Using logical operators as an extended coordination mechanism in Linda*, in: *Coordination Languages and Models*, LNCS **2315**, Springer-Verlag, 2002 pp. 317–331.

[42] Srba, J., *A roadmap of infinite results*, Bulletin of EATCS **78** (2002), pp. 163–175.

[43] van de Pol, J. and M. V. Espada, *Formal specification of javaspaces architecture using μCRL*, in: F. Arbab and C. Talcott, editors, *Coordination Languages and Models*, LNCS **2315**, Springer-Verlag, 2002 pp. 274–290.

[44] Viroli, M., *Comparing semantic frameworks for coordination: on the conformance issue for coordination media*, in: *18th ACM Symposium on Applied Computing (SAC 2003)* (2003), to appear.

[45] Viroli, M. and A. Omicini, *Tuple-based models in the observation framework*, in: *Coordination Languages and Models*, LNCS **2315**, Springer-Verlag, 2002 pp. 364–379.

[46] Viroli, M. and A. Omicini, *Coordination as a service: Ontological and formal foundation*, in: A. Brogi and J.-M. Jacquet, editors, *1st International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA '02)*, Electronic Notes in Theoretical Computer Science **68(3)**, Elsevier Science B. V., 2003 .