Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 175 (2007) 97–117

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Formal ReSpecT in the A&A Perspective

Andrea Omicini[1]

Alma Mater Studiorum—*Università di Bologna, via Venezia 52, 47023 Cesena, Italy*

Abstract

Coordination models and languages have found a new course in the context of MAS (multiagent systems). By re-interpreting results in terms of agent-oriented abstractions, new conceptual spaces are found, which extend the reach of coordination techniques far beyond their original scope. This is for instance the case of coordination media, when recasted in terms of coordination artifacts in the MAS context.
In this paper, we take the well-established ReSpecT language for programming tuple centre behaviour, and adopt the A&A (agents and artifacts) meta-model as a perspective to reinterpret, revise, extend and complete it. A formal model of the so-called A&A ReSpecT language is presented, along with an example illustrating its use for MAS coordination.

Keywords: Tuple-based Coordination, Artifacts for MAS, A&A Meta-model, Tuple Centres, ReSpecT.

1 Introduction

In the last decade, the field of coordination models and languages has produced a wide range of results on the general issue of governing interaction in complex systems: such results are today finding their natural exploitation in hot areas of computational system research, such Web Service Orchestration, WfMS (workflow management systems), and MAS (multiagent systems). It is then seemingly appropriate to take well-established results from the coordination field, and go beyond their mere inter-disciplinary application—gearing instead toward a full trans-disciplinary approach. This means essentially that findings by coordination researchers should first be taken and used to address the issues of interaction management in complex systems, as they emerge from other research areas (inter-disciplinarity); then, they should be recast according to the new conceptual framework, suitably revised and extended along to the new lines of interpretation, and in such a new form brought back to where they came from (trans-disciplinarity) [20].

Coordination is today acknowledged as one of the key issues in the modelling and engineering of complex systems: as such, it has been the subject of numerous

[1] Email: andrea.omicini@unibo.it

doi:10.1016/j.entcs.2007.03.006

investigations in areas like Sociology, Economics and Organisational Theory [16]. There, coordination is generally conceived as a means to integrate a multiplicity of diverse activities or processes in such a way that the resulting ensemble exhibits some desired / required features. The design of coordination mechanisms is particularly challenging in the field of MAS, as they are usually embedded in highly dynamic environments, and neither the number nor the behaviour of agents are possibly known at design time.

However, conceptual foundations of the MAS area are still under impetuous development, pushed by deep and heterogeneous inputs from distributed computing, programming languages, software engineering, simulation, artificial intelligence, and other related areas that are today converging toward agent-orientedness [32]. Among the most promising approaches, the A&A meta-model [19] re-interprets MAS in terms of two fundamental abstractions: *agents* and *artifacts*. Agents are the active entities encapsulating control, which are in charge of the goals/tasks that altogether build up the whole MAS behaviour. Artifacts are instead the passive, reactive entities in charge of the services and functions that make individual agents work together in a MAS, and that shape agent environment according to the MAS needs. Altogether, the A&A meta-model has a deep impact on the way in which MAS are engineered [10], programmed [28], and simulated [11]. Along this line, coordination artifacts can be conceived as a generalisation of coordination media, as specialised artifacts encapsulating coordination services for MAS [19].

Bringing back the A&A meta-model to the coordination field suggests a number of interesting considerations. For instance, features of artifacts (like inspectability, forgeability, linkability, etc.) could be used to build up a framework for classifying coordination media, to understand and compare them along the coordination literature [18]. More generally, well-established coordination models and languages could be suitably reinterpreted within the A&A conceptual framework, and revised and extended accordingly—thus providing MAS engineers with well-known and tested technologies for the development of MAS based on the A&A meta-model. In this paper, we follow the latter line of thought.

In particular, we take the ReSpecT language for programming the behaviour of tuple centres [15,25] and its formal model [14], and discuss its re-formulation in the A&A framework. Section 2 shortly discusses the A&A meta-model, and the main features of artifacts. Section 3 briefly recalls the essentials of the TuCSoN model and of the ReSpecT language. Section 4 introduces the new, revised ReSpecT syntax, along with an example of A&A ReSpecT coordination of agents. Section 5 presents the semantics of A&A ReSpecT, obtained by largely revising and extending the original one [14] along the A&A main lines. After Section 6 discusses relations with previous work and related literature, Section 7 provides for final remarks and future lines of work.

2 The A&A Meta-Model for MAS

Our approach to MAS coordination is grounded on the A&A (agents and artifacts) meta-model, which adopts artifacts—along with agents—as the basic MAS building blocks to program and, more generally, to engineer complex software systems [19].

In the A&A meta-model, agents are the basic abstractions to represent active, task-/goal-oriented components, designed to pro-actively carry on one or more activities toward the achievement of some kind of objective, requiring different levels of skills and reasoning capabilities. On the other hand, artifacts are the basic abstractions to represent passive, function-oriented building blocks, which are constructed and used by agents, either individually or cooperatively, during their working activities.

Taking human society as a metaphor, agents play the role of humans, while artifacts coincide with the objects and tools (called artifacts in the human society, too) used by humans as either the means to support their work and achieve their goals, or the target of their activities. The role of artifacts in the context of human activities—social activities in particular—is one of the most important and investigated points of theories like Activity Theory (AT) [9] and Distributed Cognition [8], which are well-known and used in fields such as CSCW and HCI [29,12].

According to AT, any activity carried on by one or more collaborating components of a system cannot be conceived or understood without considering the tools or artifacts that mediate the actions and interactions of the components. Artifacts on the one side mediate the interaction between individual components and their environment (including the other components); on the other side they embody the portion of the environment that can be designed and controlled to support components' activities. Moreover, as an observable part of the environment, artifacts can be monitored along with the development of the activities to evaluate overall system performance and keep track of system history. In other words, mediating artifacts become first-class entities for both the analysis and synthesis of individual as well as cooperative working activities inside complex systems. Such a vision is also promoted by Distributed Cognition [8], a branch of cognitive science that claims that human cognition and knowledge representation, rather than being confined within the boundaries of an individual, are distributed across individuals, tools and artifacts in the environment.

The same complexity of activities within social systems accounted for by AT and Distributed Cognition can be found nowadays in MAS. This is why we consider the inter-disciplinary study of such conceptual frameworks as fundamental for the analysis and synthesis of social activities inside MAS, and in particular of the artifacts mediating such activities [26]. Examples range from coordination abstractions such as tuple centres [15], to pheromone infrastructure [23] in the context of stigmergy coordination, to the Institution abstraction in electronic-institution approaches [5], to cite some.

Unlike agents, artifacts are not meant to be autonomous or exhibit a pro-active behaviour, neither to have social capabilities. Among the main properties that are useful according to artifact purpose and nature [18], one could list: *(i) inspectability* and *controllability*, i.e. the capability of observing and controlling artifact structure, state and behaviour at runtime, and of supporting their on-line management, in terms of diagnosing, debugging, testing; *(ii) malleability* (or, *forgeability*), i.e. the capability of artifact function to be changed / adapted at runtime (on-the-fly) according to new requirements or unpredictable events occurring in the open environment, *(iii) linkability*, i.e. the capability of linking together distinct artifacts at runtime as a form of dynamic composition, as a means to scale up with complexity of the function to provide, and also to support dynamic reuse, *(iv) situation*, i.e. the property of being immersed in the MAS environment, and to be reactive to environment events and changes. It is worth to be remarked that most of these artifact features are not agent features: typically, agents are not inspectable, do not provide means for malleability, do not provide operations for their change, and do not compose with each other through operational links. On the other hand, agents are typically told to be situated: however, how this is realised, in particularly how pro-activity and re-activity features could be reconciled, is not an easy matter. Instead, once artifacts are situated, agent situatedness could be recasted in terms of their interaction with artifacts.

Coordination artifacts [19] are a primary example of artifacts for MAS, as artifacts designed to provide agents and MAS with specific coordination functionalities and services [31]. In human societies, coordination artifacts are as common as traffic lights, street signs, post-its on whiteboards; in computational systems, things like blackboards, event-services, shared message boxes, could be easily seen as coordination artifacts. In the context of MAS, coordination artifacts are used to both enable and govern forms of *mediated interaction*—i.e., where agents do not communicate directly but through a medium—, which is essential to support forms of communication that are uncoupled along both the time and space dimensions.

So, the overall view of MAS adopting the A&A perspective is given by agents distributed across the networks that inter-operate and coordinate both by communicating via some kind of ACL (agent communication language)—such as FIPA ACL [6]—and by sharing and (co-)utilising different kind of artifacts. Generally speaking, the A&A meta-model *recasts the space of interaction* within MAS, such that the components of a MAS can interact in three different ways: agents *speak* with agents; agents *use* artifacts; artifacts *link* with artifacts.

Dealing with the management of interaction, coordination models and infrastructures like TuCSoN [22] represent the most natural technologies upon which the A&A approach can be put to test. Therefore, revising TuCSoN models and languages under the A&A viewpoint is seemingly appropriate. In particular, in this paper we recast the ReSpecT language for programming TuCSoN tuple centres: we reinterpret, revise, extend and complete it so as to make it fit the A&A meta-model for MAS. The goal of this operation is twofold. On the one hand, we aim at showing the modelling power of A&A when applied to (MAS) coordination. On the other hand, we aim at providing MAS engineers with reliable and tested technologies like ReSpecT and TuCSoN to build MAS artifacts according to the A&A meta-model.

3 TuCSoN & ReSpecT

TuCSoN (Tuple Centres Spread over the Network[2]) is a general-purpose agent-oriented model and infrastructure for MAS coordination [22]. TuCSoN is based on a coordination model providing *tuple centres* as first-class abstractions to design and develop general-purpose coordination artifacts [15]. TuCSoN tuple centres are programmed through the ReSpecT logic-based specification language. In the remainder of this section, we first recall the essentials of tuple centre coordination in TuCSoN (Subsection 3.1); then, we resume the main features of the original ReSpecT language for programming the behaviour of TuCSoN tuple centres (Subsection 3.2).

3.1 The TuCSoN *Tuple Centre Coordination Model*

A tuple centre is a tuple space enhanced with the possibility to program its behaviour in response to interactions.

So, first of all, agents can operate on a TuCSoN tuple centre in the same way as on a Linda tuple space [7]: by exchanging *tuples* (which are ordered collection of knowledge chunks) through a simple set of coordination primitive. An agent can write a tuple in a tuple centre with an `out` primitive; or read a tuple from a tuple centre with primitives such as `in`, `rd`, `inp`, `rdp` specifying a *tuple template*—that is, an identifier for a set of tuples, according to some *tuple matching* mechanism. Reading tuples can be destructive (`in`, `inp` remove the matching tuple) or non-destructive (`rd`, `rdp` simply read the matching tuple), suspensive (`in`, `rd` wait until a matching tuple is found) or non-suspensive (`inp`, `rdp` immediately return either the matching tuple or a failure result)—but is anyway non-deterministic: when more than one tuple in a tuple centre are found that match a tuple template, one is non-deterministically chosen among them.

Accordingly, a tuple centre enjoys all the many features of a tuple space, which can be classified along three different dimensions: generative communication, associative access, and suspensive semantics. The main features of generative communication (where information generated has an independent life with respect to the generator) are the forms of uncoupling (space, time, name) based on mediated interaction: sender and receiver do not need to know each other, to coexist in the same space or at the same time in order to communicate (to exchange a tuple, in particular). Associative access (access based on structure and content of information exchanged, rather than on location, or on name) based on tuple matching promotes synchronisation based on tuple structure and content: thus, coordination is data-driven, and allows for knowledge-based coordination patterns. Finally, suspensive semantics promotes coordination patterns based on knowledge availability, and couples well with incomplete or partial knowledge.

Even more, while the basic tuple centre model is independent of the type of tuple [15], TuCSoN tuple centres adopt logic tuples—both tuples and tuple templates are essentially Prolog facts—and unification is used as the tuple-matching mecha-

[2] The TuCSoN technology is available as an open source project at the TuCSoN web site [30]

nism. So, for instance, an agent `ag1` performing operation `we?in(activity(ag1,CaseID))` on tuple centre `we` containing tuples `activity(ag1,c16)` and `activity(ag2,c22)` will be returned tuple `activity(ag1,c16)` (the one unifying with the template) removed from `we`. Since the overall content of a tuple centre is a multiset of logic facts, it has a twofold interpretation as either a collection of messages, or a (logic) theory of communication among agents—thus promoting in principle forms of reasoning about communication.

The TuCSoN infrastructure makes it possible to exploit tuple centres as coordination services distributed over the network [22]. In particular, TuCSoN overall coordination space is constituted by an open set of TuCSoN *nodes*, which correspond to Internet hosts or servers connected by the network. Each node can contain any number of tuple centres, each identified by a unique (inside the node) logic name (e.g. `message_board`). An agent can refer tuple centres either specifying their *full name*, that is, their logic name plus the address of the node hosting the tuple centre (e.g. `message_board@acme.org`), or their *local name*, for tuple centres located on the same host where the agent is situated. As a result, agents can exploit either the local or the global coordination space by adopting either the local or the full name.

Finally, a tuple centre is a programmable tuple space—thus adding programmability of the coordination medium as a new dimension of coordination. While the behaviour of a tuple space in response to communication events is fixed (so, the effects of coordination primitives is fixed), the behaviour of a tuple centre can be tailored to the application needs by defining a set of specification tuples, or reactions, which determine how a tuple centre should react to incoming / outgoing events.

While the basic tuple centre model is not bound to any specific language to define reactions [15], TuCSoN adopts the logic-based language ReSpecT (Reaction Specification Tuples) to program tuple centres.

3.2 ReSpecT *as a Core Coordination Language*

The original ReSpecT [14] is a logic-based language for the specification of the behaviour of tuple centre adopted by TuCSoN. As a behaviour specification language, ReSpecT:

- enables the definition of computations within a tuple centre, called *reactions*, and
- makes it possible to associate reactions to events occurring in a tuple centre.

So, ReSpecT has both a declarative and a procedural part. As a *specification language*, it allows events to be declaratively associated to reactions by means of specific logic tuples, called *specification tuples*, whose form is `reaction(E,R)`. In short, given a event Ev, a specification tuple `reaction(E,R)` associates a reaction $R\theta$ to Ev if $\theta = mgu(E,Ev)$.[3] As a reaction language, ReSpecT enables reactions to be procedurally defined in terms of sequences of logic reaction goals, each one either succeeding or failing. A reaction as a whole succeeds if all its reaction goals suc-

[3] *mgu* is the most general unifier, as defined in logic programming.

ceed, and fails otherwise. Each reaction is executed sequentially with a transactional semantics: so, a failed reaction has no effect on the state of a logic tuple centre.

All the reactions triggered by a communication event are executed before serving any other event: so, agents perceive the result of serving the communication event and executing all the associated reactions altogether as a single transition of the tuple centre state. As a result, the effect of a communication primitive on a logic tuple centre can be made as complex as needed by the coordination requirements of a MAS. Generally speaking, since ReSpecT has been shown to be Turing-equivalent [3], any computable coordination law could be in principle encapsulated into a ReSpecT tuple centre. This is why ReSpecT can be assumed as a general-purpose core language for coordination: a language that could then be used to represent and enact policies and rules of any sort for collaboration support systems.

Adopting the declarative interpretation of ReSpecT tuples, a TuCSoN tuple centre has then a twofold nature [14]: a theory of communication (the set of the ordinary tuples) and a theory of coordination (the set of the specification tuples). This allows in principle intelligent agents to reason about the state of collaboration activities, and to possibly affect their dynamics. Furthermore, the twofold interpretation of ReSpecT specification tuples (either declarative or procedural) allows knowledge and control to be represented uniformly (as Prolog-like facts) and encapsulated within the same coordination artifact.

3.3 TuCSoN *&* ReSpecT *in the A&A Perspective*

In the A&A perspective, TuCSoN provides agents with a multiplicity of distributed artifacts (the tuple centres) containing both shared knowledge and the logic of coordination expressed in terms of logic tuples. ReSpecT tuple centres are inspectable artifacts—they are not controllable, however. Also, they are malleable, since their behaviour can be affected at run-time by changing their behaviour specification.

While the original ReSpecT specification did not encompass neither linkability nor situatedness [15], two extensions were already introduced that moved along such directions. First, a first extension was proposed in [27], which introduced the first linkability primitive for tuple-centre composition, that is, `out_tc`. Then, Timed ReSpecT was defined in [17], which first proposed the notions of timed artifact and timed tuple centre, and allowed for the specification of time-dependent coordination policies, encapsulated within Timed ReSpecT tuple centres.

4 Introducing A&A ReSpecT

4.1 Adopting the A&A Perspective

Adopting the A&A perspective promotes a more articulated view over the space of MAS interaction. First of all, a more general notion of event is required. Since artifacts are passive entities, the only real sources of events in a MAS are agents and the environment. So, whatever happens in a MAS has its "prime cause" either in an agent action, or in an environment phenomenon. However, artifacts are reactive, and link with each other—so, they can affect one each other. As a first consequence, the direct cause of any artifact event may also be some link invocation from another artifact—not the prime cause, anyway. So, a general event descriptor should include both the original cause of an event, and the most direct one—thus allowing the chain of the events to be fully observed, and artifact coordinative behaviours to be properly defined.

As a meta-model for distributed computing, A&A also promote uncoupling of control: so, *(i)* linked artifacts should be fully uncoupled, *(ii)* agents should be left free to autonomously choose either synchronous or asynchronous primitives, while the behaviour of target artifacts remains unchanged and unaffected. As a result, every operation (or link) on an artifact should have a request / response structure: any *invocation* (request), once served, always implies a message of *completion* (response)—along with the result, if needed—to be handled by the "operator" according to its nature. In case of a link invoked by another artifact, completion should be handled in a completely asynchronous fashion—to ensure full uncoupling of artifact control; in case of an operation invoked by an agent, completion should be dealt with in either a synchronous or an asynchronous way according to the agent autonomous choice.

In general, operations for *usage* of artifacts by agents, and links for *composition* between artifacts are not necessarily related—if not by the artifact structure and behaviour, of course. However, conceptual integrity in the engineering of artifacts and MAS would clearly benefit from uniformity between artifact operations and links. So, it might be desirable that primitives for artifact operations are available for exploitation to both agents and artifacts—with no *a priori* assumptions on the nature and behaviour of the invoker of a primitive, but rather with the artifact ability to discern the nature of the invoker by observing the invocation whenever useful or required.

4.2 A&A ReSpecT*: The News*

Along the lines above, the original ReSpecT language has been revised and extended to follow the A&A perspective. The resulting core syntax of the newly-defined A&A ReSpecT is reported in Table 1.[4]

The first apparent extension concerns the specification part of A&A ReSpecT: the `reaction` specification tuple has been extended to include a *guard specification*. Then, the behaviour of an A&A ReSpecT tuple centre is defined in terms of specification tuples of the form `reaction(E,G,R)`:[5] such a tuple associates a reaction $R\theta$ to *Ev* if $\theta = mgu(E,Ev)$ and guard `G` is true. A guard is a sequence of guard

[4] The A&A ReSpecT language is obviously enriched with aliases and shortcuts for reasons of expressiveness. In particular, as far as ⟨*EventInformation*⟩ identifiers are concerned, aliases for `predicate` are `pred`, `call`, and (deprecated) `operation` and `op`; an alias for `tuple` is `arg`; an alias for `source` is `from`; an alias for `target` is `to`. As far as ⟨*GuardPredicate*⟩ identifiers are concerned, `invocation`, `inv`, `req`, and `pre` are aliases for `request`; `completion`, `compl`, `resp`, and `post` are aliases for `response`; `between(Time,Time')` is an alias for `(before(Time),after(Time'))`; `operation` is an alias for `(from_agent,to_tc)`; `link_out` is an alias for `(from_tc,to_tc,endo,inter)`; `link_in` is an alias for `(from_tc,to_tc,exo,intra)`; `internal` is an alias for `(from_tc,to_tc,endo,intra)`.

[5] The original ReSpecT form `reaction(E,R)` is maintained to represent reactions with an empty guard.

Table 1
Core syntax of A&A ReSpecT

⟨*TCSpecification*⟩	::=	{⟨*SpecificationTuple*⟩ `.`}
⟨*SpecificationTuple*⟩	::=	`reaction(` ⟨*SimpleTCEvent*⟩ `,` [⟨*Guard*⟩ `,`] ⟨*Reaction*⟩ `)`
⟨*SimpleTCEvent*⟩	::=	⟨*SimpleTCPredicate*⟩ `(` ⟨*Tuple*⟩ `)` | `time(` ⟨*Time*⟩ `)`
⟨*Guard*⟩	::=	⟨*GuardPredicate*⟩ | `(` ⟨*GuardPredicate*⟩ {`,` ⟨*GuardPredicate*⟩} `)`
⟨*Reaction*⟩	::=	⟨*ReactionGoal*⟩ | `(` ⟨*ReactionGoal*⟩ {`,` ⟨*ReactionGoal*⟩} `)`
⟨*ReactionGoal*⟩	::=	⟨*TCPredicate*⟩ `(` ⟨*Tuple*⟩ `)` | ⟨*ObservationPredicate*⟩ `(` ⟨*Tuple*⟩ `)` | ⟨*Computation*⟩ | `(` ⟨*ReactionGoal*⟩ `;` ⟨*ReactionGoal*⟩ `)`
⟨*TCPredicate*⟩	::=	⟨*SimpleTCPredicate*⟩ | ⟨*TCLinkPredicate*⟩
⟨*TCLinkPredicate*⟩	::=	⟨*TCIdentifier*⟩ `?` ⟨*SimpleTCPredicate*⟩
⟨*SimpleTCPredicate*⟩	::=	⟨*TCStatePredicate*⟩ | ⟨*TCForgePredicate*⟩
⟨*TCStatePredicate*⟩	::=	`in` | `inp` | `rd` | `rdp` | `out` | `no` | `get` | `set`
⟨*TCForgePredicate*⟩	::=	⟨*TCStatePredicate*⟩`_s`
⟨*ObservationPredicate*⟩	::=	⟨*EventView*⟩`_`⟨*EventInformation*⟩
⟨*EventView*⟩	::=	`current` | `event` | `start`
⟨*EventInformation*⟩	::=	`predicate` | `tuple` | `source` | `target` | `time`
⟨*GuardPredicate*⟩	::=	`request` | `response` | `success` | `failure` | `endo` | `exo` | `intra` | `inter` | `from_agent` | `to_agent` | `from_tc` | `to_tc` | `before(` ⟨*Time*⟩ `)` | `after(` ⟨*Time*⟩ `)`
⟨*Time*⟩	is	a non-negative integer
⟨*Tuple*⟩	is	Prolog term
⟨*Computation*⟩	is	a Prolog-like goal performing arithmetic / logic computations
⟨*TCIdentifier*⟩	::=	⟨*TCName*⟩ `@` ⟨*NetworkLocation*⟩
⟨*TCName*⟩	is	a Prolog ground term
⟨*NetworkLocation*⟩	is	a Prolog string representing either an IP name or a DNS entry

predicates as defined by ⟨*GuardPredicate*⟩ in Table 1, whose semantics is defined in Table 5. A wide number of conditions over an event can now be checked before a reaction is triggered in a tuple centre: the event status, its source, its target, its time.

Along the same line, observation predicates have been generalised according to the new event model. Since an A&A ReSpecT event is defined according to the structure in Table 4, ⟨*ObservationPredicate*⟩ predicates have now the form defined in Table 1: in particular, `event_` and `start_` predicates refer to the direct and "prime" cause of an event, respectively.

Another fundamental extension concerns uniformity of operations, links and internal operations on tuple centres. Admissible primitives on a A&A ReSpecT tuple centres—⟨*TCStatePredicate*⟩ in Table 1—can be invoked by an agent, but can also be used within reactions for a tuple centre to act on its state, or to act upon another tuple centre state through a link invocation. Also the semantics is essentially the same—with the only exception of the `in` and `rd` primitives, whose suspensive semantics is not preserved inside a reaction, if not in a link invocation.[6]

[6] This is basically due to the fact that reaction execution cannot suspend, given its transactional nature. As a consequence, the semantics of `in` and `rd` essentially "collapses" to `inp` and `rdp`, respectively, within

Even more, the same class of predicates used for ordinary tuples can be used for specification tuples as well—⟨*TCForgePredicate*⟩ in Table 1—, by simply adding the _s postfix—thus adding another dimension to uniformity.

Finally, A&A ReSpecT includes the first extension toward situatedness of coordination artifacts. In fact, following Timed ReSpecT [17], it includes time events (Table 4), timed reactions, as well as predicates to handle time (Table 1). More generally, further "situation" events could be envisioned, handling topology, or other environment issues: however, as shown in [17], handling time is one of the first, essential features for any real-world coordination model.

4.3 Distributed Dining Philosophers in A&A ReSpecT

In the classical Dining Philosopher problem, N philosopher agents share N chopsticks and a spaghetti bowl [4]. Each philosopher needs two chopsticks to eat, but each chopstick is shared by two adjacent philosophers: so, the two chopsticks have to be acquired atomically to avoid deadlock, and released atomically to ensure fairness.

In [14], a ReSpecT-based implementation of the Dining Philosophers problem was presented, where

- each philosopher agent acquires / releases his chopstick pairs as a tuple `chops(`i`,`j`)`: a philosopher willing to eat acquires the pair he needs from the `table` tuple centre by means of a single `in(chops(`i`,`j`))` operation, and releases it by means of a single `out(chops(`i`,`j`))` operation.
- individual chopsticks are represented as tuples of the kind `chop/1`: the result of philosopher's operations is the atomic removal / insertion of both `chop(`i`)` and `chop(`j`)` tuples from / in the `table` tuple centre.
- the `table` tuple centre works both as the knowledge repository for the table state—as a logic tuple space—, and as the mediator between the two discrepant representations—as a programmable coordination artifact—through a suitable ReSpecT behaviour specification.

Here, we exploit some of the new features of A&A ReSpecT in order to implement a distributed version of the problem. The basic idea is to move the classical problem, which models multiple concurrent accesses to shared resources, to the distributed context, exploiting the intrinsic distribution promoted by the A&A meta-model in terms of agents and artifacts.

In the Distributed Dining Philosophers problem, N philosopher agents are supposed to be distributed around the network: each philosopher is assigned a *seat*, which is represented by a coordination artifact (a `seat(`i`,`j`)` tuple centre—meaning that `chops(`i`,`j`)` is the chopstick pair assigned to the philosopher) located in the same TuCSoN node where the agent is. When a philosopher intends to eat / think, he just expresses his intention by emitting a tuple `wanna_eat` / `wanna_think` in his `seat(`i`,`j`)` tuple centre. In turn, each `seat(`i`,`j`)` tuple centre is in charge to handle

A&A ReSpecT reactions—where `in` and `rd` are then included mainly in order to preserve uniformity of language.

Table 2
Distributed Dining Philosophers: A&A ReSpecT code for `seat(i,j)` tuple centres.

```
reaction( out(wanna_eat), (operation, invocation), (          % (1)
    in(philosopher(thinking)), out(philosopher(waiting_to_eat)),
    current_target(seat(C1,C2)),
    table@node ? in(chops(C1,C2)) )
).
reaction( out(wanna_eat), (operation, completion),            % (2)
    in(wanna_eat)
).
reaction( in(chops(C1,C2)), (link_out, completion), (         % (3)
    in(philosopher(waiting_to_eat)), out(philosopher(eating)),
    out(chops(C1,C2)) )
).
reaction( out(wanna_think), (operation, invocation), (        % (4)
    in(philosopher(eating)), out(philosopher(waiting_to_think)),
    current_target(seat(C1,C2)), in(chops(C1,C2)),
    table@node ? out(chops(C1,C2)) )
).
reaction( out(wanna_think), (operation, completion),          % (5)
    in(wanna_think)
).
reaction( out(chops(C1,C2)), (link_out, completion), (        % (6)
    in(philosopher(waiting_to_think)), out(philosopher(thinking)) )
).
```

its own agent intentions (to eat and to think), recording both the philosopher state (thinking, waiting to eat, eating, waiting to think) and the availability of chopsticks (`chops(i,j)` tuple), and interacting with the single `table` tuple centre (located in the `node` node), which holds and manages the `chop/1` tuples representing individual chopsticks on the table.

In all, the Distributed Dining Philosophers problem requires $N + 1$ artifacts: in particular, one *social artifact*, shared among all the philosophers and representing the table as well as the coordination rules regulating philosopher's interactions, and N *individual artifacts*, each one associated to one philosopher agent, and handling interaction between its associated philosopher and the table [18]. The overall artifact architecture results in a star network with the `table` tuple centre in the middle, and the N `seat(i,j)` tuple centres around, mediating between the philosophers and the table. Connections between distributed tuple centres—required to maintain consistency of the global system behaviour—are based on linkability predicates introduced in A&A ReSpecT—$\langle$*TCLinkPredicate*$\rangle$ in Table 1.

In particular, the A&A ReSpecT code in Table 2 is the same for all the `seat(i,j)` tuple centres—the specific chopstick pair is recorded in the tuple centre name, and retrieved (reaction 1 in Table 2) via one of the predicates for event observation extended in A&A ReSpecT—$\langle$*ObservationPredicate*$\rangle$ in Table 1. Reactions 1 and 3 in Table 2 deal with the expressions of philosopher's intentions, either retrieving or restoring the proper `chops(i,j)` tuple from / to the `table` tuple centre at node `node`.

The semantics of linkability predicates (all of them have their completion, but they are asynchronous) allows for a 4-state representation of philosopher agents: *thinking*, *waiting to eat*, *eating*, *waiting to think*—where the transition states (*waiting to eat* / *waiting to think*) can be easily handled by reacting to the completion of `table@node?in(chops(i,j))` and `table@node?out(chops(i,j))` links (reactions

Table 3
Distributed Dining Philosophers: A&A ReSpecT code for the `table` tuple centre.

```
reaction( out(chops(C1,C2)), (link_in, completion), (        % (1)
    in(chops(C1,C2)),
    out(chop(C1)), out(chop(C2)) )
).
reaction( in(chops(C1,C2)), (link_in, invocation), (         % (2)
    out(required(C1,C2)) )
).
reaction( in(chops(C1,C2)), (link_in, completion), (         % (3)
    in(required(C1,C2)) )
).
reaction( out(required(C1,C2)), internal, (                  % (4)
    in(chop(C1)), in(chop(C2)),
    out(chops(C1,C2)) )
).
reaction( out(chop(C)), internal, (                          % (5)
    rd(required(C,C2))
    in(chop(C)), in(chop(C2)),
    out(chops(C,C2)) )
).
reaction( out(chop(C)), internal, (                          % (5')
    rd(required(C1,C))
    in(chop(C1)), in(chop(C)),
    out(chops(C1,C)) )
).
```

3 and 6 in Table 2).

Even though not discussed here, the availability of situation predicates like the time predicates in A&A ReSpecT would allow for more complex coordination patterns, including for instance fault tolerance schemes. For instance, it would be easy to associate timeouts to the different states of agent philosophers—which is of paramount importance in a distributed, non-reliable environment, where even a simple `out` invocation could easily fail, making chopsticks disappear in the vacuum. The association of timed reactions to the philosopher's transition states could then permit the recovery from faulty situations, as well as the introduction of timed coordination policies—as the ones discussed in [17].

The code in Table 3 is the behaviour specification for the `table` tuple centre, and is more or less the translation in A&A ReSpecT of the code discussed in [14]. Worth to note, then, is just the uniform syntax for all tuple centre operations, links and internal operations, as well as the introduction of the notion of guard (along with guard predicates) that makes A&A ReSpecT reactions more general and expressive.

5 A&A ReSpecT: The Semantics

5.1 A&A ReSpecT*: Informal Semantics*

Before introducing the formal specification of A&A ReSpecT, an informal description of the inner architecture and behaviour may help understanding how A&A ReSpecT tuple centres actually work.

The main cycle of a tuple centre work as follow. Whenever the invocation of a tuple centre primitive by either an agent (operation) or an artifact (link) is performed, an (admissible) A&A ReSpecT event is generated, and reaches its target tuple centre, where it is automatically and orderly inserted in its *InQ* queue. When

Table 4
Events in A&A ReSpecT

$\langle \mathit{GeneralTCEvent} \rangle$	$::=$	$\langle \mathit{StartCause} \rangle$ **,** $\langle \mathit{Cause} \rangle$ **,** $\langle \mathit{TCCycleResult} \rangle$
$\langle \mathit{StartCause} \rangle$, $\langle \mathit{Cause} \rangle$	$::=$	$\langle \mathit{SimpleTCEvent} \rangle$ **,** $\langle \mathit{Source} \rangle$ **,** $\langle \mathit{Target} \rangle$ **,** $\langle \mathit{Time} \rangle$
$\langle \mathit{Source} \rangle$, $\langle \mathit{Target} \rangle$	$::=$	$\langle \mathit{AgentIdentifier} \rangle \mid \langle \mathit{TCIdentifier} \rangle$
$\langle \mathit{AgentIdentifier} \rangle$	$::=$	$\langle \mathit{AgentName} \rangle$ `@` $\langle \mathit{NetworkLocation} \rangle$
$\langle \mathit{AgentName} \rangle$		is a Prolog ground term
$\langle \mathit{TCCycleResult} \rangle$	$::=$	$\bot \mid \{\langle \mathit{Tuple} \rangle\}$

the tuple centre is idle (that is, no reaction is currently being executed), the first event ϵ in *InQ* (according to a FIFO policy) is logged (Table 9), and moved to the multiset *Op* of the requests to be served: this stage is called the *request* phase of the event ϵ. Consequently, reactions to the request phase of ϵ are triggered ($\mathsf{Z}_\Sigma(\epsilon) \cup \mathsf{Z}_\Sigma(n)$, according to Definitions 5.2, 5.3) by adding them to the multiset *Re* of the triggered reactions waiting to be executed.

All triggered reactions in *Re* are then executed in a non-deterministic order. Each reaction is executed sequentially, with a transactional semantics (according to Tables 6, 7), and may trigger further reactions, again to be added to *Re*, as well as new output events representing link invocations: such events are added to the multiset *Out* of the outgoing events, and then moved to the tuple-centre outgoing queue *OutQ* at the end of the reaction execution—if successful.

Only when *Re* is finally empty, requests waiting to be served in *Op* are possibly executed by the tuple centre (according to Table 8), and operation / link completions are sent back to invokers. This may give raise to further reactions, associated to the *response* phase of the original invocation, and executed again with the same semantics specified above for the request phase. Thus, the main cycle of an A&A ReSpecT tuple centre is finally concluded.

5.2 *A&A* ReSpecT*: Formal Semantics*

According to the framework defined in [13], a coordination medium is suitable for an operational characterisation in terms of an interactive transition system, where the state of communication is the system state, some transitions are triggered by interaction events, and some transitions generate output events. So, in order to formally denote the behaviour of a coordination artifact like a A&A ReSpecT tuple centre, we should first define its notion of *admissible tuple centre event*, then define its behaviour in terms of a transition system.

Definition 5.1 [A&A ReSpecT Event] An *admissible tuple centre event* for A&A ReSpecT (A&A ReSpecT event in short) is defined according to the structure in Table 4. Such a structure also defines implicitly the way in which an A&A ReSpecT event is denoted: if ϵ is an A&A ReSpecT event, then $\epsilon.\mathit{Cause}.\mathit{Source}$ denotes the entity whose activity directly caused the event, $\epsilon.\mathit{TCCycleResult}$ denotes the result of the tuple centre computation triggered by the event, and so on.

5.2.1 *Semantics of A&A* ReSpecT *Reactions*

An A&A ReSpecT tuple centre is basically a logic tuple space enhanced with a behaviour specification that defines how the tuple centre reacts to events. Then, once A&A ReSpecT events have been defined, the reaction model can be given, in terms of the reactions triggered by an A&A ReSpecT event ϵ.

Definition 5.2 [A&A ReSpecT Triggered Reaction Multiset] Given a tuple centre c and its behaviour specification Σ, if ϵ is an A&A ReSpecT event, then the multiset of the ϵ triggered reactions is defined as

$$\mathsf{Z}_\Sigma(\epsilon) ::= \biguplus_{\texttt{reaction(e,G,R)}\in\Sigma} (\epsilon, \texttt{R}\theta) \mid \theta = \mathit{Unify}(\epsilon, \texttt{e}) \neq \bot, \mathit{Guard}(\epsilon, \texttt{G})$$

There,

$$\mathit{Unify}(\epsilon, \texttt{e}) ::= \mathit{mgu}(\texttt{e}, \epsilon.\mathit{Cause}.\mathit{SimpleTCEvent})$$

while the truth value of $\mathit{Guard}(\epsilon, \texttt{G})$ is defined according to Table 5.

Definition 5.3 [A&A ReSpecT Time-Triggered Reaction Multiset] Given a tuple centre c and its behaviour specification Σ, if nc is the local tuple centre time, then the multiset of the nc time-triggered reactions is defined as

$$\mathsf{Z}_\Sigma(nc) ::= \biguplus_{\texttt{reaction(time(}t\texttt{),G,R)}\in\mathsf{timed}(nc,\Sigma)} (\epsilon_t, \texttt{R}) \mid \mathit{Guard}(\epsilon_t, \texttt{G})$$

There,

$$\mathsf{timed}(nc, \Sigma) ::= \{\texttt{reaction}(\texttt{time}(t), \texttt{G}, \texttt{R}) \in \Sigma \mid t \leq nc\}$$

while $\epsilon_t ::= \langle \texttt{time}(t), c, c, t, \texttt{time}(t), c, c, t, t\rangle$, according to the event structure defined in Table 4.

So, given a tuple centre c at time nc with behaviour specification Σ, and an event ϵ, $\mathsf{Z}_\Sigma(\epsilon)$ denotes the multiset of triggered reactions caused by ϵ, while $\mathsf{Z}_\Sigma(nc)$ denotes the multiset of time-triggered reactions at time nc.

Once defined which reactions are triggered and when, the effects of reaction execution should be accounted for. This is encapsulated in the *reaction execution function.*

Definition 5.4 [Reaction Execution Function] Let R, R' be sequences of reaction goals, Tu, Tu' multi-sets of (ordinary) logic tuples, Σ, Σ' multi-sets of specification tuples, Re, Re' multi-sets of triggered reactions, and ϵ an A&A ReSpecT event, Out, Out' sequences of A&A ReSpecT events. A *reaction execution state* is then defined as a (labelled) quintuple $\langle R, Tu, \Sigma, Re, Out\rangle_\epsilon$, whereas a *reaction execution step* is a transition

$$\langle R, Tu, \Sigma, Re, Out\rangle_\epsilon \longrightarrow_e \langle R', Tu', \Sigma', Re', Out'\rangle_\epsilon$$

following the rules of Table 6 and Table 7. If a *reaction execution sequence* is a sequence of reaction execution steps, then

$$\langle R, Tu, \Sigma, Re, Out\rangle_\epsilon^*$$

Table 5
Guard Predicates in A&A ReSpecT

Guard atom	True if
$Guard(\epsilon, (g, G))$	$Guard(\epsilon, g) \wedge Guard(\epsilon, G)$
$Guard(\epsilon, \texttt{endo})$	$\epsilon.Cause.Source = c$
$Guard(\epsilon, \texttt{exo})$	$\epsilon.Cause.Source \neq c$
$Guard(\epsilon, \texttt{intra})$	$\epsilon.Cause.Target = c$
$Guard(\epsilon, \texttt{inter})$	$\epsilon.Cause.Target \neq c$
$Guard(\epsilon, \texttt{from\_agent})$	$\epsilon.Cause.Source$ *is an agent*
$Guard(\epsilon, \texttt{to\_agent})$	$\epsilon.Cause.Target$ *is an agent*
$Guard(\epsilon, \texttt{from\_tc})$	$\epsilon.Cause.Source$ *is a tuple centre*
$Guard(\epsilon, \texttt{to\_tc})$	$\epsilon.Cause.Target$ *is a tuple centre*
$Guard(\epsilon, \texttt{before}(t))$	$\epsilon.Cause.Time < t$
$Guard(\epsilon, \texttt{after}(t))$	$\epsilon.Cause.Time > t$
$Guard(\epsilon, \texttt{request})$	$\epsilon.TCCycleResult$ *is undefined*
$Guard(\epsilon, \texttt{response})$	$\epsilon.TCCycleResult$ *is defined*
$Guard(\epsilon, \texttt{success})$	$\epsilon.TCCycleResult \neq \bot$
$Guard(\epsilon, \texttt{failure})$	$\epsilon.TCCycleResult = \bot$

Hypotheses: c is the reacting tuple centre; ϵ is an admissible A&A ReSpecT event; g is an atomic guard predicate; G is a sequence of atomic guard predicates; t is a non-negative integer.

denotes the *final state* of the reaction execution sequence whose initial state is $\langle R, Tu, \Sigma, Re, Out\rangle_\epsilon$, that is, the first state of the sequence for which no applicable rule exists in Table 6 and Table 7. Finally, if $\langle R, Tu, \Sigma, Re, Out\rangle^*_\epsilon = \langle R', Tu', \Sigma', Re', Out'\rangle_\epsilon$, then the *reaction execution function* E is defined as follows:

$$\mathsf{E}((\epsilon, R), Tu, \Sigma) ::= \begin{cases} (Tu', \Sigma', Re', Out') & \textit{if } R' = \emptyset \\ (Tu, \Sigma, \emptyset, \emptyset) & \textit{if } R' \neq \emptyset \end{cases}$$

To help intuition, at any step of a reaction execution sequence, R represents the reaction goals yet to be executed, Tu the current state of the space of ordinary tuples, Σ the current state of the space of specification tuples, Re the set of the reactions triggered by reaction goals already executed, Out the sequence of events to be emitted at the end of the execution, whereas ϵ is the event initially triggering reaction execution. Correspondingly, the execution of a triggered reaction (ϵ, R) in an A&A ReSpecT tuple centre whose tuple space is Tu and whose behaviour specification is Σ is represented by a sequence whose initial state is $\langle R, Tu, \Sigma, \emptyset, \emptyset\rangle_\epsilon$.

The above definition of E also accounts for the success/failure transactional semantics of A&A ReSpecT reactions. If the sequence of the reaction goals R' to be executed is empty, then reaction R triggered by event ϵ has been executed successfully, and a new ordinary-tuple multiset Tu', a new specification-tuple multiset Σ', along with the newly-triggered reaction set Re' and the sequence of events to be emitted Out' are provided for updating the tuple centre state. Otherwise ($R' \neq \emptyset$ and no further reaction execution steps possible), the old multisets Tu and Σ are

Table 6
Tuple-centre predicate execution in A&A ReSpecT

Execution transition						
$\langle (r,R), Tu, \Sigma, Re, Out \rangle_\epsilon \longrightarrow_e \langle R', Tu', \Sigma', Re \cup \mathsf{Z}_\Sigma(\epsilon'), Out' \rangle_\epsilon$						
r	Tu'	Σ'	R'	$\epsilon'.Cause, \epsilon''.Cause$	Out'	where
$op(\_T)?\mathtt{c}'$	Tu	Σ	R	$\langle op(\_T), c, c', nc \rangle$	$Out \cup \epsilon''$	
`out(T)`	$Tu \cup \mathtt{T}$	Σ	R	$\langle \mathtt{out(T)}, c, c, nc \rangle$	Out	
`in(T)`	Tu/tu	Σ	$R\theta$	$\langle \mathtt{in(T)}, c, c, nc \rangle$	Out	$\theta = mgu(tu, \mathtt{T})$
`inp(T)`	Tu/tu	Σ	$R\theta$	$\langle \mathtt{inp(T)}, c, c, nc \rangle$	Out	$\theta = mgu(tu, \mathtt{T})$
`rd(T)`	Tu	Σ	$R\theta$	$\langle \mathtt{rd(T)}, c, c, nc \rangle$	Out	$\theta = mgu(tu, \mathtt{T})$
`rdp(T)`	Tu	Σ	$R\theta$	$\langle \mathtt{rdp(T)}, c, c, nc \rangle$	Out	$\theta = mgu(tu, \mathtt{T})$
`no(T)`	Tu	Σ	R	$\langle \mathtt{no(T)}, c, c, nc \rangle$	Out	$\bot = mgu(tu, \mathtt{T})$
`set(TT)`	`TT`	Σ	R	$\langle \mathtt{set(TT)}, c, c, nc \rangle$	Out	
`get(TT)`	Tu	Σ	$R\theta$	$\langle \mathtt{get(TT)}, c, c, nc \rangle$	Out	$\theta = mgu(Tu, \mathtt{TT})$
`out_s(S)`	Tu	$\Sigma \cup \mathtt{S}$	R	$\langle \mathtt{out\_s(S)}, c, c, nc \rangle$	Out	
`in_s(S)`	Tu	Σ/σ	$R\theta$	$\langle \mathtt{in\_s(S)}, c, c, nc \rangle$	Out	$\theta = mgu(\sigma, \mathtt{S})$
`inp_s(S)`	Tu	Σ/σ	$R\theta$	$\langle \mathtt{inp\_s(S)}, c, c, nc \rangle$	Out	$\theta = mgu(\sigma, \mathtt{S})$
`rd_s(S)`	Tu	Σ	$R\theta$	$\langle \mathtt{rd\_s(S)}, c, c, nc \rangle$	Out	$\theta = mgu(\sigma, \mathtt{S})$
`rdp_s(S)`	Tu	Σ	$R\theta$	$\langle \mathtt{rdp\_s(S)}, c, c, nc \rangle$	Out	$\theta = mgu(\sigma, \mathtt{S})$
`no_s(S)`	Tu	Σ	R	$\langle \mathtt{no\_s(S)}, c, c, nc \rangle$	Out	$\bot = mgu(\sigma, \mathtt{S})$
`set_s(SS)`	Tu	`SS`	R	$\langle \mathtt{set\_s(SS)}, c, c, nc \rangle$	Out	
`get_s(SS)`	Tu	Σ	$R\theta$	$\langle \mathtt{get\_s(SS)}, c, c, nc \rangle$	Out	$\theta = mgu(\Sigma, \mathtt{SS})$

Hypotheses: $\epsilon, \epsilon', \epsilon''$ are A&A ReSpecT events, such that $\epsilon.StartCause = \epsilon'.StartCause = \epsilon''.StartCause$, $\epsilon'.TCCycleResult = \epsilon.TCCycleResult$, and $\epsilon''.TCCycleResult = \bot$; $tu \in Tu$ is a tuple; $\sigma \in \Sigma$ is a specification tuple; r is a reaction goal, R is a sequence of reaction goals; c, c' are tuple centres (c denotes the tuple centre currently in charge of the computation); nc is the time local to c when the execution takes place.

Table 7
Observation predicate execution in A&A ReSpecT

Execution transition	
$\langle (r,R), Tu, \Sigma, Re, Out \rangle_\epsilon \longrightarrow_e \langle R\theta, Tu, \Sigma, Re, Out \rangle_\epsilon$	
r	where
`event_predicate(Obs)`	$\theta = mgu(\epsilon.Cause.SimpleTCEvent.SimpleTCPredicate, \mathtt{Obs})$
`event_tuple(Obs)`	$\theta = mgu(\epsilon.Cause.SimpleTCEvent.Tuple, \mathtt{Obs})$
`event_source(Obs)`	$\theta = mgu(\epsilon.Cause.Source, \mathtt{Obs})$
`event_target(Obs)`	$\theta = mgu(\epsilon.Cause.Target, \mathtt{Obs})$
`event_time(Obs)`	$\theta = mgu(\epsilon.Cause.Time, \mathtt{Obs})$
`start_predicate(Obs)`	$\theta = mgu(\epsilon.StartCause.SimpleTCEvent.SimpleTCPredicate, \mathtt{Obs})$
`start_tuple(Obs)`	$\theta = mgu(\epsilon.StartCause.SimpleTCEvent.Tuple, \mathtt{Obs})$
`start_source(Obs)`	$\theta = mgu(\epsilon.StartCause.Source, \mathtt{Obs})$
`start_target(Obs)`	$\theta = mgu(\epsilon.StartCause.Target, \mathtt{Obs})$
`start_time(Obs)`	$\theta = mgu(\epsilon.StartCause.Time, \mathtt{Obs})$
`current_predicate(Obs)`	$\theta = mgu(\mathtt{current\_predicate}, \mathtt{Obs})$
`current_tuple(Obs)`	$\theta = mgu(\mathtt{Obs}, \mathtt{Obs}) = \{\}$
`current_source(Obs)`	$\theta = mgu(c, \mathtt{Obs})$
`current_target(Obs)`	$\theta = mgu(c, \mathtt{Obs})$
`current_time(Obs)`	$\theta = mgu(nc, \mathtt{Obs})$

Hypotheses: ϵ is an A&A ReSpecT event; r is a reaction goal, R is a sequence of reaction goals; c denotes the tuple centre currently in charge of the computation; nc is the time local to c when the execution takes place.

returned, no new reactions are triggered, and no events to be emitted are added—so that no further changes occur in the tuple centre state.

Transitions occur according to the rules of Table 6 and Table 7, where all the symbols retain their usual meanings. The final state of a sequence is reached whenever either no reaction goals are still to be executed, or there is no applicable rule available. Since each step actually deletes one goal from a reaction, and the number of reaction goals is finite for any reaction, each reaction is guaranteed to be executed in a finite number of steps.

Table 8
Service transition in A&A ReSpecT

Service transition				
${}^{InQ}\langle Tu, \Sigma, \emptyset, Op \cup \epsilon\rangle_n^{OutQ} \longrightarrow_s {}^{InQ}\langle Tu', \Sigma', \mathsf{Z}_\Sigma(\epsilon') \cup \mathsf{Z}_\Sigma(n), Op\rangle_{n'}^{OutQ,\epsilon'}$				
$\epsilon.Cause.SimpleTCEvent$	Tu'	Σ'	res	where
`out(T)`	$Tu \cup \mathtt{T}$	Σ	`T`	
`in(T)` `inp(T)`	Tu/tu	Σ	$\mathtt{T}\theta$	$\theta = mgu(tu, \mathtt{T})$
`rd(T)` `rdp(T)`	Tu	Σ	$\mathtt{T}\theta$	$\theta = mgu(tu, \mathtt{T})$
`inp(T)` `rdp(T)`	Tu	Σ	$\perp$	$\perp = mgu(tu, \mathtt{T})$
`no(T)`	Tu	Σ	`T`	$\perp = mgu(tu, \mathtt{T})$
`no(T)`	Tu	Σ	$\perp$	$\theta = mgu(tu, \mathtt{T})$
`set(TT)`	`TT`	Σ	`TT`	
`get(TT)`	Tu	Σ	$\mathtt{TT}\theta$	$\theta = mgu(Tu, \mathtt{TT})$
`get(TT)`	Tu	Σ	$\perp$	$\perp = mgu(Tu, \mathtt{TT})$
`out_s(S)`	Tu	$\Sigma \cup \mathtt{S}$	`S`	
`in_s(S)` `inp_s(S)`	Tu	Σ/σ	$\mathtt{S}\theta$	$\theta = mgu(\sigma, \mathtt{S})$
`rd_s(S)` `rdp_s(S)`	Tu	Σ	$\mathtt{S}\theta$	$\theta = mgu(\sigma, \mathtt{S})$
`inp_s(S)` `rdp_s(S)`	Tu	Σ	$\perp$	$\perp = mgu(\sigma, \mathtt{S})$
`no_s(S)`	Tu	Σ	`S`	$\perp = mgu(\sigma, \mathtt{S})$
`no_s(S)`	Tu	Σ	$\perp$	$\theta = mgu(\sigma, \mathtt{S})$
`set_s(SS)`	Tu	`SS`	`SS`	
`get_s(SS)`	Tu	Σ	$\mathtt{SS}\theta$	$\theta = mgu(\Sigma, \mathtt{SS})$
`get_s(SS)`	Tu	Σ	$\perp$	$\perp = mgu(\Sigma, \mathtt{SS})$

Hypotheses: ϵ, ϵ' are A&A ReSpecT events: $\epsilon \in \mathsf{sat}(Op, Tu, \Sigma)$, ϵ' is such that $\epsilon'.StartCause = \epsilon.StartCause$, $\epsilon'.Cause = \epsilon.Cause$ and $\epsilon'.TCCycleResult = res$; $tu \in Tu$ is a tuple; $\sigma \in \Sigma$ is a specification tuple.

5.2.2 Behaviour of A&A ReSpecT Tuple Centres

The state of a A&A ReSpecT tuple centre is expressed as a labelled quadruple ${}^{InQ}\langle Tu, \Sigma, Re, Op\rangle_n^{OutQ}$. There, Tu and Σ are the multisets of the ordinary and specification tuples in the tuple centre, respectively; Re is the multiset of the triggered reactions waiting to be executed; Op is the multiset of the requests waiting for a response; InQ and $OutQ$ are the incoming and outgoing event queues, respectively; finally, n is the local tuple centre time.[7] InQ is a queue that is automatically extended whenever incoming events affect a tuple centre—so no special transitions are required for incoming events. Dually, $OutQ$ is automatically emptied by emitting the outgoing events, with no need again of special transitions.

The operational behaviour of an A&A ReSpecT tuple centre whose state is ${}^{InQ}\langle Tu, \Sigma, Re, Op\rangle_n^{OutQ}$ can now be modelled in terms of a transition system with four kind of different transitions—below, in order of decreasing priority:

reaction When $Re \neq \emptyset$, triggered reactions in Re are executed through a *reaction* transition ($\longrightarrow_r$).

time When $Re = \emptyset$ and $\mathsf{timed}(n, \Sigma) \neq \emptyset$, timed reactions can trigger new reactions through a *time* transition ($\longrightarrow_t$).

service When $Re = \mathsf{timed}(n, \Sigma) = \emptyset$, and $\mathsf{sat}(Op, Tu, \Sigma) \neq \emptyset$, requests waiting for a response can be served through a *service* transition ($\longrightarrow_s$).[8]

log When $Re = \mathsf{timed}(n, \Sigma) = \mathsf{sat}(Op, Tu, \Sigma) = \emptyset$, and $InQ \neq \emptyset$, requests queued in InQ can be "logged" by a tuple centre through a *log* transition ($\longrightarrow_l$)

[7] Whenever not needed by the context, InQ, $OutQ$ and n could be dropped from the representation of a tuple centre state.

[8] $\mathsf{sat}(Op, Tu, \Sigma) \subseteq Op$ is the subset of the Op requests waiting for a response that can be actually served given the current state of the tuple centre.

Table 9
Log transition in A&A ReSpecT

Log transition		
${}^{\epsilon,InQ}\langle Tu, \Sigma, \emptyset, Op\rangle_{n}^{OutQ} \longrightarrow_{l} {}^{InQ}\langle Tu, \Sigma, \mathsf{Z}_{\Sigma}(\epsilon) \cup \mathsf{Z}_{\Sigma}(n), Op'\rangle_{n'}^{OutQ}$		
$\epsilon.Cause.SimpleTCEvent$	$\epsilon.TCCycleResult$	Op'
$op(\_)$	undefined	$Op \cup \epsilon$
$op(\_)$	defined	Op
Hypotheses: ϵ is an A&A ReSpecT event.		

Reaction transition works as follows:

$${}^{InQ}\langle Tu, \Sigma, Re \cup re, Op\rangle_{n}^{OutQ} \longrightarrow_{r} {}^{InQ}\langle Tu', \Sigma', Re \cup Re', Op\rangle_{n'}^{OutQ,Out'}$$

where $\mathsf{E}(re, Tu, \Sigma) = (Tu', \Sigma', Re', Out')$—thus computing according to the semantics of reaction predicates presented in the previous subsection.

Time transition takes instead the following form:

$${}^{InQ}\langle Tu, \Sigma, \emptyset, Op\rangle_{n}^{OutQ} \longrightarrow_{t} {}^{InQ}\langle Tu, \Sigma/\mathsf{timed}(n, \Sigma), \mathsf{Z}_{\Sigma}(n), Op\rangle_{n'}^{OutQ}$$

where past timed reactions ($\mathsf{timed}(n, \Sigma)$) are evaluated and then discarded ($\Sigma/\mathsf{timed}(n, \Sigma)$), and possibly generate some time-triggered reactions ($\mathsf{Z}_{\Sigma}(n)$).

The more articulated service and log transitions are regulated according to Table 8 and Table 9, respectively.

6 Related Works

With respect to the original formulation [15], A&A ReSpecT, as presented in this paper, is largely changed and extended. First, the main things left unchanged are *(i)* the basic reaction model, with the two-level atomicity of reaction execution (system level and agent level), and *(i)* the logic-based syntax, with event descriptions unifying with reaction heads, and the reaction bodies built as sequences of Prolog-like atoms.

Some syntax modifications have addressed known limitations in the original ReSpecT. The introduction of guards, for instance, has improved on the expressiveness of reactions, allowing programmers to minimise the number of unnecessarily-triggered reactions; a number of expressive guard predicates are introduced in A&A ReSpecT for this purpose. Also, the syntax of primitives is now uniform: the same primitive invocations can be used by agents on a tuple centre, and by a programmer within a reaction, either to access and change the internal tuple centre state, or to compose tuple centres; even the primitives for accessing and changing a tuple centre behaviour specification have essentially the same form. Even more, semantics of any agent invocation is now in a sense uniform: all the coordination primitives that an agent can invoke on a tuple centre have the same request / response behaviour (not only `ins` and `rds`, but also `outs`), and reactions can then be designed having in mind the same conceptual structure for any of the primitives involved.

More generally, the adoption of A&A as the underlying meta-model has led to a number of new features in the novel A&A ReSpecT. First of all, an extended event model is defined for A&A ReSpecT, which encompasses the A&A meta-model and its general event model. Then, linkability of artifacts is developed and extended up to its maximum reach: any tuple centre operation can now be invoked in a reaction to be executed either within the tuple centre itself (as an internal operation), or on any other tuple centre (as a link). [9] Finally, situatedness of artifacts is recognised here as a general issue, which encompasses timed artifacts and their computational model. With respect to the original formulation of Timed ReSpecT [17], the model is then made more general, and also fully formalised within the overall A&A ReSpecT framework.

Linkability in its most general acceptation is not strictly a new idea in the field of coordination models and languages. The most prominent example is Reo [1], where channel composition is one of the most important and relevant features. Also, Reo has been recently experimented explicitly in the MAS field [2]. However, Linda-based approaches better cope with agent autonomy, since coordination is not forced upon the agents participating to the workflow, but is instead provided them as a service [31].

In the context of Linda-based models, to the best of our knowledge, only Lime [24] could exhibit some sort of mechanism for tuple-space composition. However, such a mechanism is essentially implicit, and does not allow for the explicit control allowed instead by A&A ReSpecT linkability primitives. At the same time, Lime mechanisms for tuple-space automatic composition provide for a sort of spatial situatedness that A&A ReSpecT does not feature in its present form.

7 Conclusions & Future Work

In this paper, we adopt the A&A (agents & artifacts) meta-model for MAS, and recast the ReSpecT language for programming the behaviour of tuple centres, and its formal model as well, according to the A&A perspective. The resulting model and language, called A&A ReSpecT, is introduced: the new syntax is defined, an example (the Distributed Dining Philosophers) is discussed, and the formal semantics of A&A ReSpecT is provided.

While implementation of the new A&A ReSpecT is underway, along with the new version of the TuCSoN infrastructure for MAS coordination, in the future we plan to experiment with A&A ReSpecT in the many domains where the original ReSpecT is already used—from e-learning to workflow management systems and case-handling, from simulation to self-organising systems. Meanwhile, we mean to further explore the issue of *situatedness* of coordination artifacts, by extending the ability of tuple centres to react to environment events. In the version of A&A ReSpecT presented here, in fact, only time events are accounted for and treated: more general forms of environment events (like topological ones, for instance) should be instead made available and properly manageable.

[9] The first element of linkability in ReSpecT was introduced in [27], in the limited form of an `out_tc` predicate, and used in [21] for distributed workflow.

8 Acknowledgements

This paper owes a lot to a number of brilliant people that have worked with me on ReSpecT in the last decade: mostly, Marco Venuti, Antonio Natali, Enrico Denti, and Alessandro Ricci. Mirko Viroli also deserves credit if for no other reason than his ability to kindly but constantly pressing me so as to make me finally write this article. Finally, the enthusiasm and the many gentle but precise remarks by Matteo Casadei have been very important to correct and improve this paper up to its present shape.

References

[1] Farhad Arbab. Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14:329–366, 2004.

[2] Mehdi Dastani and OSGi-Open Services Gateway Initiatives Consortium. *Coordination and Composition of Multi-Agent Systems*. Invited talk, 1st International Workshop on Coordination and Organisation (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 23 April 2005.

[3] Enrico Denti, Antonio Natali, and Andrea Omicini. On the expressive power of a language for programming coordination media. In *1998 ACM Symposium on Applied Computing (SAC'98)*, pages 169–177, Atlanta, GA, USA, 27 February– 1 March 1998. ACM. Special Track on Coordination Models, Languages and Applications.

[4] Edsger Wybe Dijkstra. Co-operating sequential processes. In Per Brinch Hansen, editor, *The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls*, chapter 2, pages 65–138. Springer, 2002. Reprinted. 1st edition: 1965.

[5] Marc Esteva, Bruno Rosell, Juan Antonio Rodríguez-Aguilar, and Josep Lluís Arcos. Ameli: An agent-based middleware for electronic institutions. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 236–243, New York, USA, 19–23July 2004. ACM.

[6] Foundation for Intelligent Physical Agents. FIPA home page. http://www.fipa.org.

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

[8] David Kirsh. Distributed cognition, coordination and environment design. In Sebastiano Bagnara, editor, *3rd European Conference on Cognitive Science (ECCS'99)*, pages 1–11, Certosa di Pontignano, Siena, Italy, 1999. Istituto di Psicologia, Consiglio Nazionale delle Ricerche.

[9] Aleksie Nikolaevich Leontjev. *Activity, Consciousness, and Personality.* Prentice Hall, 1978.

[10] Ambra Molesini, Andrea Omicini, Enrico Denti, and Alessandro Ricci. SODA: A roadmap to artefacts. In Oğuz Dikenelli, Marie-Pierre Gleizes, and Alessandro Ricci, editors, *Engineering Societies in the Agents World VI*, volume 3963 of *LNAI*, pages 49–62. Springer, June 2006. 6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised, Selected & Invited Papers.

[11] Sara Montagna, Alessandro Ricci, and Andrea Omicini. Agents & Artifacts for Systems Biology: Toward a framework based on TuCSoN. In Alessandro Genco, Antonio Gentile, and Salvatore Sorce, editors, *Industrial Simulation Conference 2006 (ISC 2006)*, pages 25–32, Palermo, Italy, 5–7 June 2006. EUROSIS (The European Simulation Society) & ETI (The European Technology Institute).

[12] Bonnie Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction.* MIT Press, 1996.

[13] Andrea Omicini. On the semantics of tuple-based coordination models. In *1999 ACM Symposium on Applied Computing (SAC'99)*, pages 175–182, San Antonio, TX, USA, 28 February– 2March 1999. ACM. Special Track on Coordination Models, Languages and Applications.

[14] Andrea Omicini and Enrico Denti. Formal ReSpecT. *Electronic Notes in Theoretical Computer Science*, 48:179–196, June 2001. Declarative Programming – Selected Papers from AGP 2000, La Habana, Cuba, 4–6 December2000.

[15] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.

[16] Andrea Omicini, Sascha Ossowski, and Alessandro Ricci. Coordination infrastructures in the engineering of multiagent systems. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter 14, pages 273–296. Kluwer Academic Publishers, June 2004.

[17] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Time-aware coordination in ReSpecT. In Jean-Marie Jacquet and Gian Pietro Picco, editors, *Coordination Models and Languages*, volume 3454 of *LNCS*, pages 268–282. Springer-Verlag, April 2005. 7th International Conference (COORDINATION 2005), Namur, Belgium, 20–23 April 2005. Proceedings.

[18] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.

[19] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In Alessandro F. Garcia, Ricardo Choren, Carlos Lucena, Paolo Giorgini, Tom Holvoet, and Alexander Romanovsky, editors, *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*, volume 3914 of *LNAI*, pages 71–90. Springer, April 2006. Invited Paper.

[20] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. The multidisciplinary patterns of interaction from sciences to Computer Science. In Dina Q. Goldin, Scott A. Smolka, and Peter Wegner, editors, *Interactive Computation: The New Paradigm*, pages 395–414. Springer, September 2006.

[21] Andrea Omicini, Alessandro Ricci, and Nicola Zaghini. Distributed workflow upon linkable coordination artifacts. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages*, volume 4038 of *LNCS*, pages 228–246. Springer, June 2006. 8th International Conference (COORDINATION 2006), Bologna, Italy, 14–16 June 2006. Proceedings.

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

[23] H. Van Dyke Parunak, Sven Brueckner, and John Sauter. Digital pheromone mechanisms for coordination of unmanned vehicles. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, volume 1, pages 449–450, Bologna, Italy, 15–19 July 2002. ACM Press.

[24] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Lime: Linda Meets Mobility. In David Garlan, editor, *21st International Conference on Software Engineering (ICSE'99)*, pages 368–377, Los Angeles, CA, USA, May 1999. ACM Press.

[25] ReSpecT home page. http://respect.alice.unibo.it.

[26] Alessandro Ricci, Andrea Omicini, Enrico Denti, and Marco Cadoli. Activity Theory as a framework for MAS coordination. In Paolo Petta, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents World III*, volume 2577 of *LNCS*, pages 96–110. Springer-Verlag, April 2003. 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17 September 2002. Revised Papers.

[27] Alessandro Ricci, Andrea Omicini, and Mirko Viroli. Extending ReSpecT for multiple coordination flows. In Hamid R. Arabnia, editor, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, volume III, pages 1407–1413, Las Vegas, NV, USA, 24–27 July 2002. CSREA Press.

[28] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Programming MAS with artifacts. In Rafael P. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3862 of *LNAI*, pages 206–221. Springer, March 2006. 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers.

[29] Tarja Susi and Tom Ziemke. Social cognition, artefacts, and stigmergy: A comparative analysis of theoretical frameworks for the understanding of artefact-mediated collaborative activity. *Cognitive Systems Research*, 2(4):273–290, December 2001.

[30] TuCSoN home page. http://tucson.alice.unibo.it.

[31] Mirko Viroli and Andrea Omicini. Coordination as a service. *Fundamenta Informaticae*, 73(4):507–534, 2006. Special Issue: Best papers of FOCLASA 2002.

[32] Franco Zambonelli and Andrea Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, November 2004. Special Issue: Challenges for Agent-Based Computing.