Adaptive Coordination Models for Pervasive Environments

Andrea Omicini
andrea.omicini@unibo.it

Alma Mater Studiorum—Università di Bologna a Cesena

PerAda Summer School @ Rimini, Italy, 6/9/2008

Next in Line...

Disclaimer I

Invited talk in a doctorate school

- When you are preparing an invited talk, you are aiming at being stimulating, entertaining, exciting, brilliant, shattering, enlightening,
  - fast & furious, more or less
- When you give a lecture to a Doctorate School, you are aiming at being plain, clear, comprehensive, complete, exhaustive, at most slightly humorous – but boring is essentially ok – maybe brilliant and enlightening, again
  - slow & calm, anyway
- An invited talk at a Doctorate School is then expected to be slow & fast, calm & furious
  - please, let apart brilliant and enlightening, be serious...

Disclaimer II

The proposed algorithm

As a result, I cordially invite you to adopt the following algorithm

- when you understand what I say, please consider I am giving a Doctorate School lecture
- when you do not understand but enjoy the talk anyway, please consider I am giving an invited talk
- when you do not understand, and also do not enjoy the talk, please consider it is Sunday morning...

Disclaimer III

Technical disclaimer

Since you are here, I assume that

- most of you have an interest and some sort of understanding of what *adaptive systems* and *pervasive computing* mean
- most of the things about these topics – and related ones – will be covered by my esteemed colleagues giving *real lectures* in the rest of the week
- *some* of you have already some strong opinions about those matters, which you are not to change anyway no matter what I am going to say

As a consequence, I will try to avoid teasing you with my own personal interpretations of the two terms, and possibly waste your time (as well as mine) discussing them here

What is the Vision?

The vision behing this talk

- Structured computational environments...
- ...immersed in dynamic and unpredictable physical environments...
- ...built around a vast and typically growing number of computational resources...
- ...providing widespread, fine-grained coordination services...
- ...to a huge number of heterogeneous and autonomous computational processes...
- ...and promoting the engineering of adaptive systems based on principles for self-organisation

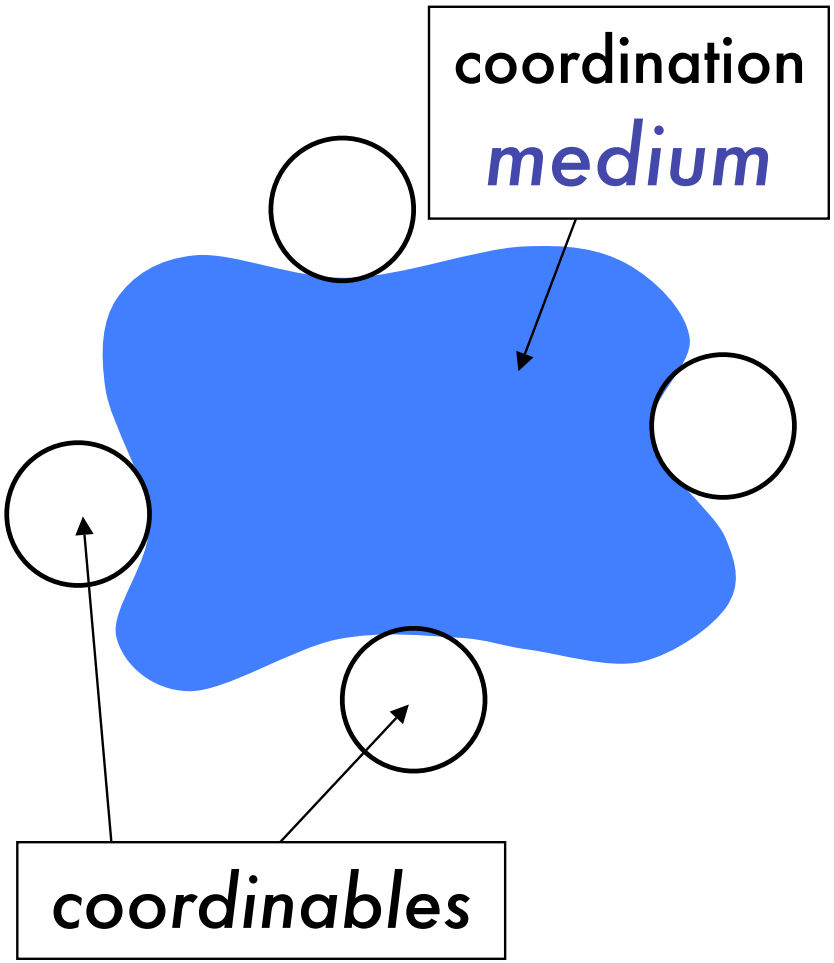
Next in Line...

Next in Line...

Coordination: Sketching a Meta-model

The *medium of coordination*

- "fills" the interaction space
- enables / promotes / governs the admissible / desirable / required interactions among the interacting entities
- according to some *coordination laws*
  - enacted by the behaviour of the medium
  - defining the semantics of coordination



Coordination in Concurrent & Distributed Programming

Coordination model as a glue

A coordination model is the glue that binds separate activities into an ensemble [Gelernter and Carriero, 1992]

Coordination model as an agent interaction framework

A coordination model provides a framework in which the interaction of active and independent entities called agents can be expressed [Ciancarini, 1996]

Issues for a coordination model

A coordination model should cover the issues of $creation$ and $destruction$ of agents, $communication$ among agents, and spatial $distribution$ of agents, as well as $synchronization$ and $distribution$ of their $actions$ over time [Ciancarini, 1996]

Coordination: A Meta-model [Ciancarini, 1996]

A constructive approach

Which are the components of a coordination system?

Coordination entities Entities whose mutual interaction is ruled by the model, also called the *coordinables*

Coordination media Abstractions enabling and ruling agent interactions

Coordination laws Rules defining the behaviour of the coordination media in response to interaction

Coordinables

Original definition [Ciancarini, 1996]

These are the entity types that are coordinated. These could be Unix-like processes, threads, concurrent objects and the like, and even users.

examples Processes, threads, objects, human users, agents, ...

focus Observable behaviour of the coordinables

remark We are noy anyhow concernd here with the internal machinery / functioning of the coordinable, in principle

Coordination Media

Original definition [Ciancarini, 1996]

These are the media making communication among the agents possible. Moreover, a coordination medium can serve to aggregate agents that should be manipulated as a whole. Examples are classic media such as semaphores, monitors, or channels, or more complex media such as tuple spaces, blackboards, pipelines, and the like.

examples	Semaphors, monitors, channels, tuple spaces, blackboards, pipes, ...
focus	The core around which the components of the system are organised
question	Which are the possible computational models for coordination media?

Coordination Laws

Original definition [Ciancarini, 1996]

A coordination model should dictate a number of laws to describe how agents coordinate themselves through the given coordination media and using a number of coordination primitives. Examples are laws that enact either synchronous or asynchronous behaviors or exploit explicit or implicit naming schemes for coordination entities.

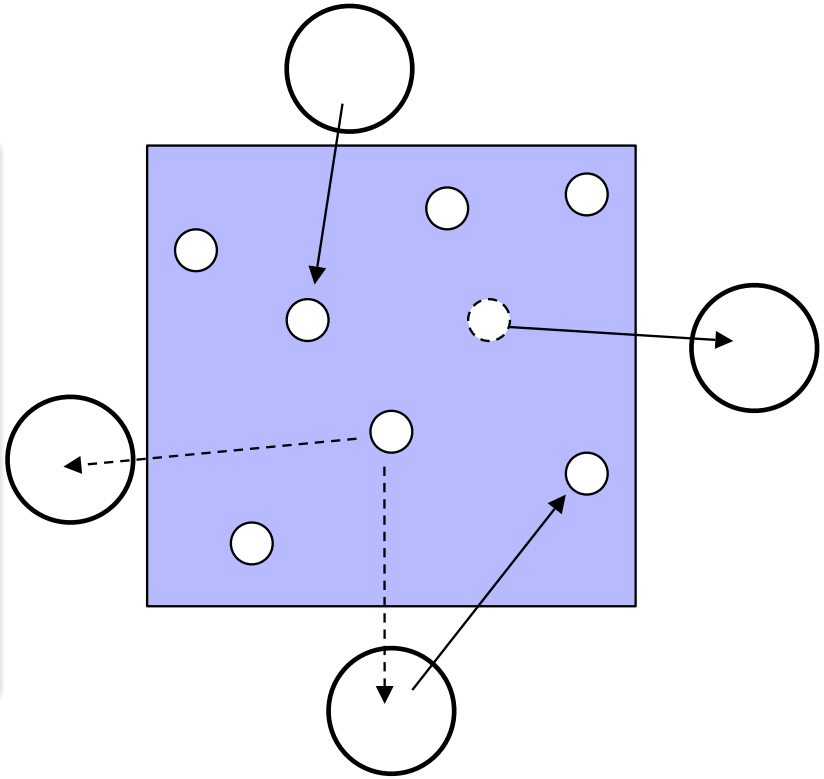
- Coordination laws define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model

Next in Line...

Space-based Meta-models

The basics [Rossi et al., 2001]

- *Coordinables* synchronise, cooperate, compete
  - based on *tuples*
  - available in the *tuple space*
  - by *associatively* accessing, consuming and producing tuples



Space-based Coordination Systems

Adopting the constructive coordination meta-model [Ciancarini, 1996]

coordination media tuple spaces

- as multiset / bag of data objects / structures called *tuples*

communication language tuples

- as ordered collections of (possibly heterogeneous) information items

coordination language tuple space primitives

- as a set of operations to put, browse and retrieve tuples to/from the space

Linda / Space-based Communication

Communication Language

tuples ordered collections of possibly heterogeneous information chunks

- examples: p(1), printer('HP',dpi(300)), [0,0.5], matrix(m0,3,3,0.5), tree_node(node00,value(13),left(_),right(node01)), ...

templates / anti-tuples specifications of set / classes of tuples

- examples: p(X), [?int,?int], tree_node(N), ...

tuple matching the mechanism matching tuples and templates

- examples: pattern matching, unification, ...

First and main example: Linda [Gelernter, 1985]

Linda / Space-based Coordination [Gelernter, 1985] I

out(T)

- out(T) puts tuple T in to the tuple space

  examples `out(p(1))`, `out(0,0.5)`, `out(course('Denti Enrico','Poetry',hours(150))` ...

Linda / Space-based Coordination [Gelernter, 1985] II

`in(TT)`

- `in(TT)` retrieves a tuple matching template `TT` from to the tuple space

  destructive reading the tuple retrieved is removed from the tuple centre

  non-determinism if more than one tuple matches the template, one is chosen non-deterministically

  suspensive semantics if no matching tuples are found in the tuple space, operation execution is suspended, and woken when a matching tuple is finally found

  examples `in(p(X))`, `in(0,0.5)`, `in(course('Denti Enrico',Title,hours(X))` ...

Linda / Space-based Coordination [Gelernter, 1985] III

`rd(TT)`

- `rd(TT)` retrieves a tuple matching template `TT` from to the tuple space

  non-destructive reading the tuple retrieved is left untouched in the tuple centre

  non-determinism if more than one tuple matches the template, one is chosen non-deterministically

  suspensive semantics if no matching tuples are found in the tuple space, operation execution is suspended, and awakened when a matching tuple is finally found

  examples `rd(p(X))`, `rd(0,0.5)`, `rd(course('Ricci Alessandro','Operating Systems',hours(X))` ...

Linda Extensions: Predicative Primitives

`inp(TT)`, `rdp(TT)`

- both `inp(TT)` and `rdp(TT)` retrieve tuple `T` matching template `TT` from the tuple space

  $=$ `in(TT)`, `rp(TT)` (non-)destructive reading, non-determinism, and syntax structure is maintained

  $\neq$`in(TT)`, `rp(TT)` suspensive semantics is lost: this *predicative* versions primitives just fail when no tuple matching `TT` is found in the tuple space

  success / failure predicative primitives introduce *success / failure semantics*: when a matching tuple is found, it is returned with a success result; when it is not, a failure is reported

Linda Extensions: Bulk Primitives

`in_all(TT)`, `rd_all(TT)`

- Linda primitives (including predicative ones) deal with a tuple at a time
  - some coordination problems require more than one tuple to be handled by a single primitive
- `rd_all(TT)`, `in_all(TT)` get all tuples in the tuple space matching with `TT`, and returns them all
  - no suspensive semantics: if no matching tuple is found, an empty collection is returned
  - no success / failure semantics: a collection of tuple is always successfully returned—possibly, an empty one
  - in case of logic-based primitives / tuples, the form of the primitive are `rd_all(TT,LT)`, `in_all(TT,LT)` (or equivalent), where the (possibly empty) list of tuples unifying with `TT` is unified with `LT`
  - (non-)destructive reading: `in_all(TT)` consumes all matching tuples in the tuple space; `rd_all(TT)` leaves the tuple space untouched
- Many other bulk primitives have been proposed and implemented to address particular classes of problems

Linda Extensions: Multiple Tuple Spaces

`ts ? out(T)`

- Linda tuple space might be a bottleneck for coordination
- Many extensions have focussed on making a multiplicity of tuple spaces available to agents
  - each of them encapsulating a portion of the coordination load
  - either hosted by a single machine, or distributed across the network
- Syntax required, and dependent on particular models and implementations
  - a space for tuple space names, possibly including network location
  - operators to associate Linda operators to tuple spaces
- For instance, `ts@node ? out(p)` may denote the invocation of operation `out(p)` over tuple space `ts` on node `node`

Main Features of Space-based Coordination

Main features of Linda & space-based models

tuples A tuple is an ordered collection of knowledge chunks, possibly heterogeneous in sort

generative communication until explicitly withdrawn, the tuples generated by coordinables have an independent existence in the tuple space; a tuple is equally accessible to all the coordinables, but is bound to none

associative access tuples in the tuple space are accessed through their content & structure, rather than by name, address, or location

suspensive semantics operations may be suspended based on unavailability of matching tuples, and be woken up when such tuples become available

Features of Space-based Coordination: Tuples

- A tuple is an ordered collection of knowledge chunks, possibly heterogeneous in sort
  - a record-like structure
  - with no need of field names
  - easy aggregation of knowledge
  - semantic interpretation: a tuple contains all information concerning an given item
- Tuple structure based on
  - arity
  - type
  - position
  - information content
- Anti-tuples / Tuple templates
  - to describe / define sets of tuples
- Matching mechanism
  - to define belongingness to a set

Features of Space-based Coordination: Generative Communication

- *Communication orthogonality*: both senders and the receivers can interact even without having prior knowledge about each others
  - space uncoupling (also called distributed naming): no need to coexist in space for two agents to interact
  - time uncoupling : no need for simultaneity for two agents to interact
  - name uncoupling: no need for names for agents to interact

Features of Space-based Coordination: Associative Access

- Content-based coordination: synchronisation based on tuple content & structure
  - absence / presence of tuples with some content / structure determines the overall behaviour of the coordinables, and of the coordinated system in the overall
  - based on tuple templates & matching mechanism
- Information-driven coordination
  - patterns of coordination based on data / information availability
  - based on tuple templates & matching mechanism
- Reification
  - making events become tuples
  - grouping classes of events with tuple syntax, and accessing them via tuple templates

Features of Space-based Coordination: Suspensive Semantics

- `in` & `rd` primitives in Linda have a suspensive semantics
  - the coordination medium makes the primitives waiting in case a matching tuple is not found, and wakes it up when such a tuple is found
  - the coordinable invoking the suspensive primitive is expected to wait for its successful completion
- Twofold wait

  in the coordination medium the operation is first (possibly) suspended, then (possibly) served: coordination based on absence / presence of tuples belonging to a given set

  in the coordination entity the invocation may cause a wait-state in the invoker: hypothesis on the internal behaviour of the coordinable

Dining Philosophers in Linda

- The spaghetti bowl, or, more easily, the table where the bowl and the chopstick are, and the philosophers are seated, are represented by the tuple space
- Chopsticks are represented as tuples `chop(`i`)`, that represents the left chopstick for the $i-th$ philosopher
  - philosopher i needs chopsticks i (left) and $(i+1) mod N$ (right)
- Philosophers try to eat by getting their chopstick pairs from the tuple space as a pair of tuples `chop(`i`)` `chop(`i`+1 mod N)`
- Philosophers start to think by releasing their own chopstick pairs to the tuple space as a pair of tuples `chop(`i`)` `chop(`i`+1 mod N)`

Dining Philosophers in Linda: A Simple Philosopher Protocol

Philosopher using ins and outs

```
philosopher(I,J) :-
    think,                          % thinking
    in(chop(I)), in(chop(J)),       % waiting to eat
    eat,                            % eating
    out(chop(I)), out(chop(J)),     % waiting to think
!,  philosopher(I,J).
```

Issues

+ shared resources handled correctly
– starvation, deadlock and unfairness possible

Next in Line...

Objective vs. Subjective coordination [Omicini and Ossowski, 2003] I

Subjective coordination

- Local / individual / agent's viewpoint over coordination
  - agents as *coordinating* entities
  - coordination *within* the agents

Objective coordination

- Global / social / engineer's viewpoint over coordination
  - agents as *coordinated* entities
  - coordination *outside* the agents

Objective vs. Subjective coordination [Omicini and Ossowski, 2003] II

Objective vs. subjective coordination in space-based models

- Tuple spaces in charge of objective aspects
- Limited expressive power
  - any coordination problem that can be expressed in terms of individual tuples occurring in a shared space can be in principle well-separated
  - any other more complex coordination issue requires patches and compromises

Coordination as a Language vs. as a Service I

Coordination as a Language (CaL)

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

$$P ::= 0 \mid \mathtt{in}(a).P \mid \mathtt{rd}(a).P \mid \mathtt{out}(a).P$$

- Operational semantics through a transition system

Coordination as a Language vs. as a Service II

Coordination as a Service (CaS)

- Open systems require coordination to be imposed through powerful abstractions that
  - persist through the whole engineering process – from design to execution time – and
  - provide coordination services to applications in the form of coordination media, possibly by a shared infrastructure
- Coordination media are then characterised in terms of their interactive behaviour, and are seen as primary abstractions amenable of formal investigation, promoting their exploitation at every step of the engineering process
- Foundation, definitions and the whole formal framework in [Viroli and Omicini, 2006]

Coordination as a Service for Autonomic and Pervasive Scenarios

Coordination as a service for autonomous agents

- *Coordination services* are provided by coordination media to autonomous agents that deliberately commit to use them for coordinating their activities
- This preserves *agents autonomy*, *global rules*, and in principle some degree of separation between objective and subjective coordination

Environment-based Coordination I

Environment in coordination [Omicini et al., 2004]

- In the context of human organisations, environment plays a fundamental role for supporting cooperative work and, more generally, complex coordination activities
- Support is realised through services, tools, artifacts shared and exploited by the collectivity of individuals for achieving individual as well as global objectives

Environment-based Coordination II

Coordination artifacts [Ricci et al., 2005]

- Coordination artifacts are the entities used to instrument the environment so as to fruitfully support cooperative and social activities
- Infrastructures play a key role by providing services for artifact use and management

Environment engineering with coordination artifacts

- Environment should be treated as a first-class entity in the engineering of complex distributed systems [Weyns et al., 2007]
- Coordination artifacts for shaping the environment / engineering the environment [Ricci and Viroli, 2005]

Stigmergy & Coordination

Stigmergic coordination [Ricci et al., 2007]

- More articulated forms of *environment-based coordination* are possible, where artifacts give structure to the environment by encapsulating and promoting the mechanisms for stigmergic coordination
- For instance, when signals (e.g., pheromones) are interpreted as *signs* at a symbolic level by rational agents
- Stigmergy & *cognitive stigmergy* for emergent coordination...
- ...where both reactive and intelligent agents can fruitfully participate in an emergently-coordinated activity...
- ...even though with different level of understanding of the coordinating environment

Next in Line...

ReSpecT in One Slide [Omicini and Denti, 2001]

Reaction **Spec**ification **T**uples

`reaction(Event, Guard, Body)`

- Coordination as *reactive behaviour*
  - of coordination abstractions
  - in response to events / actions

  $\Rightarrow$ When an event ϵ matching `Event` occurs in the tuple centre, and the `Guard` succeeds over ϵ properties, then reaction $(\epsilon, \texttt{Body})$ is triggered and executed

- Coordination specified via (FOL) specification tuples
  - tuple space + specification space = tuple centre
  - theory of communication + theory of coordination = theory of interaction
  - declarative vs. procedural language

ReSpecT Features

1. A **coordination** language
   - supporting clean separation between subjective & objective issues of coordination
2. **Space-based** coordination
   - growing a control-driven layer upon the data-driven basic model
3. **Logic-based** language
   - representing communication & coordination spaces as FOL theories

Coordination Artifacts in ReSpecT I

From coordination media to coordination artifacts [Omicini et al., 2004]

⇒ ReSpecT tuple centres can be used as coordination artifacts for MAS, encapsulating the rules for MAS coordination expressed in terms of ReSpecT reactions

Coordination as a service [Viroli and Omicini, 2006]

⇒ ReSpecT tuple centres can be used to provide MAS with coordination services, with coordination policies possibly inspectable by agents as FOL theories

Stigmergic coordination [Ricci et al., 2007]

⇒ ReSpecT tuple centres can be used to build up structured environments for self-organising MAS based on cognitive stigmergy

ReSpecT I

A&A meta-model: artifacts in ReSpecT

- The A&A meta-model conceives MAS as built out of two basic abstractions: *agents* and *artifacts* [Omicini et al., 2008]
- Even though general actions cannot be operated on tuple centres, information tuples can represent in principle any action, and ReSpecT is Turing-equivalent

$\Rightarrow$ ReSpecT tuple centres can be used to build up generic artifacts for MAS, expressing artifact's behaviour in terms of ReSpecT reactions

ReSpecT II

ReSpecT: Extensions

- The whole ReSpecT language has been revised and extended according to A&A meta-model [Omicini, 2007]

$\Rightarrow$ artifact interaction model has been completed: full artifact linkability, and ability to manifest events to agents

$\Rightarrow$ synchronous operations with asynchronous interaction model extended uniformly to all primitives

Timed ReSpecT I

ReSpecT artifacts in the (spatio-)temporal fabric

- Artifacts are immersed into the spatio-temporal continuum, and should be reactive to events of such a sort
- Also, most elementary MAS techniques require timed coordination—any commitment of any sort, in practice

$\Rightarrow$ ReSpecT tuple centres should be able to capture the passage of time, and express timed coordination policies

Timed ReSpecT II

Timed ReSpecT: Extensions

- The ReSpecT language has been revised and extended so as to capture and express time in coordination [Omicini et al., 2005]

$\Rightarrow$ ReSpecT VM captures time and generates time events

$\Rightarrow$ the ReSpecT language has been extended to react to and observe time events, and to express timed coordination policies

Situated ReSpecT I

ReSpecT artifacts for environment engineering

- Artifacts are immersed into the MAS environment, and should be reactive to events of *any* sort
- Also, artifacts should mediate any agent activity toward the environment, allowing for a fruitful interaction

$\Rightarrow$ ReSpecT tuple centres should be able to capture general environment events, and to generally mediate agent-environment interaction

Situated ReSpecT II

Situated ReSpecT: Extensions

- The ReSpecT language has been revised and extended so as to capture environment events, and express general MAS-environment interactions [Casadei and Omicini, 2008]

$\Rightarrow$ ReSpecT captures, reacts to, and observes general environment events

$\Rightarrow$ ReSpecT can explicitly interact with the environment

Dining Philosophers in ReSpecT

- The spaghetti bowl, or, more easily, the table where the bowl and the chopstick are, and the philosophers are seated, are represented by tuple centre `table`
- Chopsticks are represented as tuples `chop(`i`)`, that represents the left chopstick for the $i-th$ philosopher
  - philosopher i needs chopsticks i (left) and $(i+1) mod N$ (right)
- An agent philosopher tries to eat by getting his chopstick pair from the tuple centre by means of a `in(chops(`i`,`i`+1 mod N)` invocation
- A philosopher starts to think by releasing his own chopstick pair to the tuple centre by means of a `out(chops(`i`,`i`+1 mod N)` invocation

Dining Philosophers in ReSpecT: Philosopher Protocol

```
philosopher(I,J) :-
    think,                          % thinking
    table ? in(chops(I,J)),         % waiting to eat
    eat,                            % eating
    table ? out(chops(I,J)),        % waiting to think
!,  philosopher(I,J).
```

Results

- \+ fairness, no deadlock
- \+ trivial philosopher's interaction protocol
- ? shared resources handled properly?
- ? starvation still possible?

Dining Philosophers in ReSpecT: `table` Behaviour Specification

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, 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)) )).
```

Dining Philosophers in ReSpecT: Results

Results

protocol	no deadlock
protocol	fairness
protocol	trivial philosopher's interaction protocol
tuple centre	shared resources handled properly
-	starvation still possible

Distributed Dining Philosophers

Dining Philosophers in a distributed setting

- N philosopher agents are distributed along the network
  - each philosopher is assigned a seat, represented by the tuple centre `seat(i,j)`
  - `seat(i,j)` denotes that the associated philosopher needs chopstick pair chops(i,j) so as to eat
- each chopstick `i` is represented as a tuple `chop(i)` in the `table` tuple centre
- each philosopher expresses his intention to eat / think by emitting a tuple `wanna_eat` / `wanna_think` in his `seat(i,j)` tuple centre
  - everything else is handled automatically in ReSpecT, embedded in the tuple centre / artifact behaviour
- N individual artifacts (`seat(i,j)`) + 1 social artifact (`table`) connected in a star network

Distributed Dining Philosophers: Individual Interaction

Philosopher–seat interaction (*use*)

- four states, represented by tuple `philosopher(_)`
  - `thinking`, `waiting_to_eat`, `eating`, `waiting_to_think`
- determined by
  - the `out(wanna_eat)` / `out(wanna_think)` invocations, expressing the philosopher's intentions
  - the interaction with the `table` tuple centre, expressing the availability of `chop` resources
- tuple `chops(i,j)` only occurs in tuple centre `seat(i,j)` in the `philosopher(eating)` state
- state transitions only occur when they are safe
  - from `waiting_to_think` to `thinking` only when chopsticks are safely back on the `table`
  - from `waiting_to_eat` to `eating` only when chopsticks are actually at the `seat`

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)) )).
```

Distributed Dining Philosophers: Social Interaction

Seat–table interaction (*link*)

- tuple centre `seat(i,j)` requires / returns tuple `chops(i,j)` from / to `table` tuple centre
- tuple centre `table` transforms tuple `chops(i,j)` into a tuple pair `chop(i)`, `chop(j)` whenever required, and back `chop(i)`, `chop(j)` into `chops(i,j)` whenever required and possible

ReSpecT Code for `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)) )).
```

Distributed Dining Philosophers: Features

- Full separation of concerns
  - philosopher agents just express their intentions, in terms of simple tuples
  - individual artifacts (`seat(i,j)` tuple centres) handle individual behaviours and state, and mediate interaction of individuals with social artifacts (`table` tuple centre)
  - the social artifact (`table` tuple centre) deals with shared resources (`chop` tuples) and ensures global system properties, like fairness and deadlock avoidance
- At any time, one could look at the coordination artifacts, and find exactly the consistent representation of the current distributed state
  - properly distributed, suitably encapsulated
    - the state of shared resources is in the shared distributed abstraction, the state of single agents is into individual local abstractions
  - accessible, represented in a declarative way
    - the state of individual philosophers is exposed through accessible artifacts as far as the portion representing their social interaction is concerned

Timed Dining Philosophers

- An example for situatedness in the spatio-temporal fabric
- `table` tuple centre stores the maximum amount of time for any agent (philosopher) to use the resource (to eat using chops)
  - in terms of a tuple `max_eating_time(@Time)`
  - if this time expires the locks are automatically released—chopsticks are re-inserted by the table tuple centre
  - late releases (by agents through `seat` tuple centres) are to be ignored—linkability used to make `seat` tuple centres consistent
- With a very simple extension using timed reactions, Distributed Timed Dining Philosophers are done
  - see [Omicini et al., 2005]

Timed Dining Philosophers: Philosopher

```
philosopher(I,J) :-
    think,                          % thinking
    table ? in(chops(I,J)),         % waiting to eat
    eat,                            % eating
    table ? out(chops(I,J)),        % waiting to think
!,  philosopher(I,J).
```

With respect to Dining Philosopher's protocol...

...this is left unchanged

Timed Dining Philosophers: `table` ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (     % (1)
    in(used(C1,C2,_)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (      % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, 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(C1,C)), in(chop(C1)), in(chop(C)), out(chops(C1,C)) )).
reaction( in(chops(C1,C2)), (operation, completion), (      % (6)
    current_time(T), rd(max_eating_time(Max)), T1 is T + Max,
    out(used(C1,C2,T)),
    out_s(time(T1),(in(used(C1,C2,T)), out(chop(C1)), out(chop(C2)))) )).
```

Timed Dining Philosophers in ReSpecT: Results

Results

protocol	no deadlock
protocol	fairness
protocol	trivial philosopher's interaction protocol
tuple centre	shared resources handled properly
tuple centre	no starvation

Next in Line...

Next in Line...

Composing Adaptive Software – The Classical Way I

Three basic techniques [McKinley et al., 2004]

- Separation of concerns
- Computational reflection
- Component-based design

Separation of concerns

Separating functional and non-functional

- Non-fuctional properties like reliability, performance, security, ..., should be faced separatedly
- *Aspect-oriented programming* and *aspect-oriented software development* deals with cross-cutting concerns

Composing Adaptive Software – The Classical Way II

Computational reflection

The ability to inspect oneself and possibly self-adapt behaviour

- Reflection if at the core of modern programming language like Java
- Observing the state of a program by the program itself
- Intercession is changing the state of the program after reflection
- Observing oneself state related with the environment makes it possible to change behaviour adaptively

Composing Adaptive Software – The Classical Way III

Component-based design

Adaptation through composition

- Once an architecture is open—e.g., hot-pluggable
- A new behaviour may be added by adding a component on the fly
- Once an architecture for open systems is available, the point is how to select a component that may add the required behaviour to the system

Composing Adaptive Software – The Coordination Way I

Separation of concerns

Separating objective and subjective coordination

- Objective and subjective coordination should be engineered separately, and encapsulated within different abstractions
- A ReSpecT tuple centre could be changed to reflect new needs in social coordination policies, while an individual agent could adjust its individual coordination to the general coordination setting

Composing Adaptive Software – The Coordination Way II

Computational reflection

The ability to inspect the state of coordination and possibly self-adapt behaviour

- Observing the state of coordination by inspecting the coordination media
- Changing the system's behaviour by either adapting the individual behaviour, or changing the social policies in the coordination medium
- A ReSpecT tuple centre could be inspected to observe the state of both communication and coordination, and its behaviour adapted by modifying its specification space
- A possible path toward adaptation by self-organisation

Composing Adaptive Software – The Coordination Way III

Component-based design

Adaptation by openness of architecture

- Coordination as a service is in principle open
- A new behaviour is added to the system when a new agent enters and commits to a certain coordination service, ...
- ...or when a new coordination artifact is linked to the existing ones

Next in Line...

Re-considering Environment-based Coordination I

An observation

- In a coordinated system, the environment is filled with coordination media – e.g. tuple spaces or ReSpecT tuple centres – enacting coordination laws that are typically *reactive*, *deterministic*, and *global*—in most models...
- In self-organising systems – as well as in few emerging works in coordination –, coordination services with interesting global properties appear by emergence from probabilistic coordination laws, based on local criteria, and time-reactive

Re-considering Environment-based Coordination II

Most coordination models are old-style

- Deterministic in most essential meanings, typically time-independent, usually global in effect
- TOTA, SwarmLinda, stoKLAIM are few examples of attempting new paths
- A good mixture of old and new features is required for self-organising coordination

A Framework for Self-organising Coordination I

Essential features of self-organising coordination [Casadei and Viroli, 2008]

- Topology
- Locality
- On-line charachter
- Time
- Probability

Topology

- The application is deployed over a topologically structured distributed system
- Coordination media and agents are deployed over locations

A Framework for Self-organising Coordination II

Locality

- Topology is strictly tight with the *scope* of interactions
- A coordinated system can feature two kinds of interaction, between an agent and a coordination medium, and between coordination media
- Both kinds of interaction should occur *locally*, that is, across the same location or across two neighboring locations as defined by topology

A Framework for Self-organising Coordination III

On-line charachter

- Coordination media should not be merely reactive—reactive to interaction, affecting interaction
- Instead, they should behave with an on-line coordination behaviour, enacted as an *always-running service*
- For instance, in order to work properly, the fading mechanism should not be precisely defined at design time, but rather adapt on-line to the rate at which agents move

Time

- Coordination policies should depend on time
  - a self-organising coordination service should be given at a certain "rate"
  - some coordination primitive could be time-dependent—e.g., timeout

A Framework for Self-organising Coordination IV

Probability

- Space-based non-determinism is essentially delegation to implementation
- It is not the same as "natural" non-determinism...
- Effects of actions are not deterministic in nature: stochastic distribution is typical
- It should then be possible to express stochastic (coordination) behaviours within coordination media

Self-organising Coordination

A definition [Casadei and Viroli, 2008]

- Self-organising coordination is the management of system interactions featuring self-organising properties, namely, where interactions are local, and *global desired effects of coordination appear by emergence*
- Constructively, self-organising coordination is achieved through *coordination media spread over the topological environment*, enacting *probabilistic and time-dependent coordination rules*

Self-organising Coordination with ReSpecT & TuCSoN I

What the heck is TuCSoN? [Omicini and Zambonelli, 1999]

- A coordination infrastructure providing ReSpecT tuple centres as its coordination media
- Here, just the TuCSoN distributed topology is required

Topology

- Assuming the network is organised in a topologically structured distributed system, TuCSoN allows one or more tuple centres to be created locally to a specific node
- Here, coordinating (Java) agents, too, are supposed to be localised in a node of the network

Self-organising Coordination with ReSpecT & TuCSoN II

Locality

- TuCSoN agents and tuple centres should be aware of locality: they should just know the list of tuple centre identifiers in the neighbourhood
- For a tuple centre, e.g., this simply means a tuple `neighbour(tc)` occurs in the space if tuple centre `tc` is in the neighbourhood

Self-organising Coordination with ReSpecT & TuCSoN III

On-line charachter & Time

- ReSpecT support timed reactions: when the tuple centre time (expressed as Java milliseconds) reaches `T`, the corresponding reaction is fired
- Moreover, a reaction goal can be of the kind `out_s(reaction(time(T),G,R))`, which inserts tuple `reaction(time(T),G,R)` in the space, thus triggering a new reaction—and essentially, self-modifying itself
- These mechanisms can be used to realise either an on-line service that keeps transforming tuples as time passes, or time-dependent coordination primitives

Self-organising Coordination with ReSpecT & TuCSoN IV

Probability

- Probability of coordination rules is supported in TuCSoN in two ways [Casadei et al., 2007]
  - On the one hand, it is possible to draw random numbers and use them to drive the reaction firing process, that is, tuple transformation by reactions can be intrinsically probabilistic
  - On the other hand, tuple retrieval can be moved from the non-deterministic version to a *probabilistic* version by using a variant of the usual primitives, i.e. `urd` instead of `rd` This is called *uniform read* operation (and analogously for `uin`, `uinp`, and `urdp`): its semantics is such that among all tuples that match the template, one is chosen equiprobabilistically. As an example, if the space has 100 tuples `t(red)`, and 50 tuples `t(blue)`, then operation `rdp(t(X))` yields `t(red)` with probability 66%.

Self-organising Coordination: Examples

Two examples [Viroli et al., 2008]

- Adaptive tuple distribution – self-organisation through interaction *between* coordination media
- Chemical coordination – self-organisation through interaction (of tuples) *inside* a coordination medium

Adaptive Tuple Distribution I

Tuple clustering

- Like corpse clustering by ants
- Tuples carrying information of the same class should be aggregated...
- ...thus forming clusters of similar tuples across the network
- Implemented in TuCSoN with 20 lines of ReSpecT code
- ReSpecT tuple centres react and interact with each other to adaptively distribute tuples

Adaptive Tuple Distribution II

An algorithm

- Meta-data: tuples are either *moving* or *still*
- Let t be a tuple observed by a `urd` operation; then
  - if t is *still*, by probability P_{mobile} it moves to state *moving* (storing the local value of concentration)
  - if t is *moving*, by probability P_{still} it moves to state *still*
  - if t is *moving*, by probability $1 - P_{still}$ it moves to a neighboring tuple space R randomly chosen

P_{mobile} is a fixed value—0.2 in our experiments, though this value like depends on the size of the network and the number of tuples to be clustered. $P_{still} = e^{-(conc(t)/localconc)}$, where $conc(t)/localconc$ ($\in [0, +\infty]$) is the relative concentration of tuples perceived by t during the last movement, with respect to the local concentration.

Chemical-like Coordination I

A biologically-inspired coordination model

- Chemical-like coordination laws embedded in the coordination medium
- In ReSpecT tuple centres, built as ReSpecT reactions

A sketch

- Tuples in a tuple centre behave like chemical components
- Chemical laws are expressed as ReSpecT reactions

More in [Viroli et al., 2008]

- Where coordination laws like decay, Lotka reactions, Oregonator, . . . , are described and discussed
- as well as implemented in ReSpecT and experimented

The ReSpecT Code

A true engine for chemical systems

```
reaction( time(Time), endo, out(engine_trigger) ).
reaction( out(engine_trigger), endo, (
    in(engine_trigger),
    chooseOne(Law,Dt),
    reaction_time(Time),Time2 is Time + Dt,
    out_s(reaction(time(Time2),endo,out(engine_trigger))),
    out(execution(Law))
)).
reaction( out(execution(law([C|T],_,O)), endo, (
    in(execution(law([C|T],_,O)),
    in(reactant(C,N)), N2 is N-1,
    out(reactant(C,N2)),out(execution(law(T,_,O))
)).
reaction( out(execution(law([],_,[C|T])), endo, (
    in(execution(law([],_,[C|T])),
    in(reactant(C,N)), N2 is N+1,
    out(reactant(C,N2)),out(execution(law([],_,T))
)).
reaction( out(execution(law([],_,[])), endo, (
    in(execution(law([],_,[]))
)).
```

Self-organising Coordination: Future Work

Still at the beginning...

- A well-founded methoology would be needed
- Some hints already from the present work
- A lot of work to be done...

Next in Line...

Thanks...

...to the aliCE group in Cesena

- Antonio Natali & Enrico Denti, who followed me on the first ReSpecT trail
- Alessandro Ricci & Mirko Viroli, who I have followed several times, yet
- Matteo Casadei & Marco Fabbri, currently doing the hard work on ReSpecT
- Matteo Casadei & Mirko Viroli (both, again), leading the group in self-organising coordination

...to the PerAda 2008 Summer School organisers in Rimini

- for inviting me to such a nice place
- in particular, thanks to Franco Zambonelli for inventing the Invited Doctored School Talk

Bibliography I

Casadei, M., Gardelli, L., and Viroli, M. (2007).
Simulating emergent properties of coordination in Maude: the collective sort case.
Electronic Notes in Theoretical Computer Sciences, 175(2):59–80.
5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.

Casadei, M. and Omicini, A. (2008).
Situated tuple centres in ReSpecT.
In *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, Honolulu, Hawai'i, USA.
Submitted.

Casadei, M. and Viroli, M. (2008).
Applying self-organising coordination to the emergent tuple organization in distributed networks.
In Brueckner, S., Robertson, P., and van Steen, M., editors, *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, Venice, Italy. IEEE CS.

Ciancarini, P. (1996).
Coordination models and languages as software integrators.
ACM Computing Surveys, 28(2):300–302.

Bibliography II


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


Gelernter, D. and Carriero, N. (1992).
Coordination languages and their significance.
Communications of the ACM, 35(2):97–107.


McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. (2004).
Composing adaptive software.
IEEE Computer, 37(7):56–64.


Omicini, A. (2007).
Formal ReSpecT in the A&A perspective.
Electronic Notes in Theoretical Computer Sciences, 175(2):97–117.
5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.


Omicini, A. and Denti, E. (2001).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.

Bibliography III

Omicini, A. and Ossowski, S. (2003).
Objective versus subjective coordination in the engineering of agent systems.
In Klusch, M., Bergamaschi, S., Edwards, P., and Petta, P., editors, *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*, pages 179–202. Springer.

Omicini, A., Ricci, A., and Viroli, M. (2005).
Time-aware coordination in ReSpecT.
In Jacquet, J.-M. and Picco, G. P., editors, *Coordination Models and Languages*, volume 3454 of *LNCS*, pages 268–282. Springer-Verlag.
7th International Conference (COORDINATION 2005), Namur, Belgium, 20–23 April 2005. Proceedings.

Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17(3).
Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

Bibliography IV

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004).
Coordination artifacts: Environment-based coordination for intelligent agents.
In Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 286–293, New York, USA. ACM.

Omicini, A. and Zambonelli, F. (1999).
Coordination for Internet application development.
Autonomous Agents and Multi-Agent Systems, 2(3):251–269.
Special Issue: Coordination Mechanisms for Web Agents.

Ricci, A., Omicini, A., Viroli, M., Gardelli, L., and Oliva, E. (2007).
Cognitive stigmergy: Towards a framework based on agents and artifacts.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 124–140. Springer.
3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

Ricci, A. and Viroli, M. (2005).
Coordination artifacts: A unifying abstraction for engineering environment-mediated coordination in MAS.
Informatica, 29(4):433–443.

Bibliography V

Ricci, A., Viroli, M., and Omicini, A. (2005).
Environment-based coordination through coordination artifacts.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for Multi-Agent Systems*, volume 3374 of *LNAI*, pages 190–214. Springer.
1st International Workshop (E4MAS 2004), New York, NY, USA, 19 July 2004. Revised Selected Papers.

Rossi, D., Cabri, G., and Denti, E. (2001).
Tuple-based technologies for coordination.
In Omicini, A., Zambonelli, F., Klusch, M., and Tolksdorf, R., editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 4, pages 83–109. Springer-Verlag.

Viroli, M., Casadei, M., and Omicini, A. (2008).
A framework for modelling and implementing self-organising coordination.
In *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, Honolulu, Hawai'i, USA.
Submitted.

Bibliography VI

Viroli, M. and Omicini, A. (2006).
Coordination as a service.
Fundamenta Informaticae, 73(4):507–534.
Special Issue: Best papers of FOCLASA 2002.

Weyns, D., Omicini, A., and Odell, J. (2007).
Environment as a first-class abstraction in multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):5–30.
Special Issue on Environments for Multi-agent Systems.

Adaptive Coordination Models for Pervasive Environments

Andrea Omicini
andrea.omicini@unibo.it

Alma Mater Studiorum—Università di Bologna a Cesena

PerAda Summer School @ Rimini, Italy, 6/9/2008