ALMA MATER STUDIORUM — UNIVERSITÀ DI BOLOGNA

DISI - Dipartimento di Informatica: Scienza e Ingegneria
Dottorato in Informatica

Ciclo XXVIII
Settore Concorsuale: 09/H1
Settore Disciplinare: ING-INF/05

# COORDINATION ISSUES IN COMPLEX SOCIO-TECHNICAL SYSTEMS: SELF-ORGANISATION OF KNOWLEDGE IN $\mathcal{M}o\mathcal{K}$

*Candidato*

Dott. Ing. STEFANO MARIANI

*Supervisore*

Chiar.mo Prof. Ing. ANDREA OMICINI

*Tutor*

Chiar.mo Prof. Ing. ANDREA OMICINI

*Coordinatore*

Chiar.mo Prof. Ing. PAOLO CIACCIA

FINAL EXAMINATION YEAR 2016

*To my beloved wife, Alice*
*To our beloved daughter, Asia*

***Acknowledgements***

This thesis belongs not only to me, but also to all the researchers and professionals I had the pleasure to work with during this three-years effort.

I wish to thank Prof. Andrea Omicini for his supervision during my PhD: he is a constant and endless source of invaluably precious suggestions, criticism, and inspiration, as well as a wonderful person to share random thoughts with. A special mention goes to Prof. Di Marzo-Serugendo and Prof. Yee-King, the reviews whose comments helped in shaping the final version of this thesis, and to Prof. Michela Milano, Prof. Fabio Vitali, and Prof. Enrico Denti, the members of my internal committee which evaluated progress of the PhD research efforts.

I wish to thank Prof. Mirko Viroli and Prof. Alessandro Ricci for sharing with me many enlightening discussions, as well as a many launch breaks: they may be unaware of this, but their research work has always been a reference for me. I wish to thank Danilo Pianini, Sara Montagna, and Andrea Santi for the wonderful time we had in the APICe lab, and for all the precious discussions we shared. A special mention goes to Roberta Calegari for being so efficient in fixing tuProlog issues for the benefit of TuCSoN.

I wish to thank Prof. Schahram Dustdar, Prof. Hong-Linh Truong, Georgiana Copil, and all the distributed systems group of TU Wien, for hosting me during my period abroad: I learnt many things by working with them, and gained a broader perspective on my own work. A special mention goes to Alessio Gambi for making me feel at home while there.

I wish to thank all the brilliant students I had the pleasure to supervise for their thesis and graduate projects, from whom I have learnt as much as I have taught. A special mention goes to Michele Pratiffi, Mattia Occhiuto, Giulio Crestani, Gianluca Spadazzi, Giacomo Dradi, Luca Santonastasi, Matteo Fattori, Matteo Francia, Giovanni Ciatto, Michele Bombardi, Lorenzo Forcellini Reffi.

Last, but not least, I owe everything I have to my family: my wife Alice constantly supports me and helps me in every single important decision, and our daughter Asia has been fundamental to refresh my energies when the work to do was just too much. My parents, too, always supported me, and constantly demonstrated their care and pride about my research work, while my brother has been always available for taking a break from work.

*Stefano Mariani*, March 30, 2016

# Contents

# Chapter 1

# About this Thesis

*Knowledge-Intensive Environments* (KIE) are workplaces in which sustainability of the organisation long-term goals is influenced by, if not even dependant on, the evolution of the knowledge embodied within the organisation itself [Bha01]. Being knowledge an organised combination of data, procedures, and operations, continuously interacting and evolving according to human users practice and (learnt) experience, KIE are usually computationally supported by *Socio-Technical Systems* (STS), that is, systems in which cognitive and social interaction is mediated by information technology, rather than by the natural world alone [Whi06].

The modern IT landscape is increasingly pervaded by these kind of systems, mostly due to the astonishing amount of data available nowadays, and to the unprecedented participation of end users in the applications they use everyday—think about, e.g., social networks, crowdsourcing platforms, online collaboration tools, and the like.

By definition, both KIE and STS are heavily interaction-centred, thus, they inevitably need to deal with *coordination issues* to harness the intricacies of run-time dependencies between the data and the agents (either software or human) participating the system [MC94]. However, engineering effective coordination is far from trivial, mostly due to a few peculiarities of KIE and STS: *unpredictability* of human behaviour, *scalability* of the technological infrastructure, *size* of the amount of data, information, and knowledge to handle, *pace* of knowledge production and consumption.

For these reasons, this thesis approaches the issue of engineering knowledge-intensive STS from a *coordination perspective*, dealing with the aforementioned issues at the infrastructural level. The goal is to enable and promote *user-driven self-organisation of knowledge*, by leveraging self-organising coordination mechanisms, an architecture for situated pervasive systems, and a cognitive model of social action. Accordingly, the contribution of this thesis may be conveniently articulated as follows:

- Chapter 2 briefly reviews the literature about self-organising coordination
- Chapter 3 describes the approach to coordination in self-organising systems, dealing

with the well-known local vs. global issue by engineering coordination laws as *artificial chemical reactions*. Simulations illustrate the potentiality of the approach, while a formal definition of the coordination primitives exploited as basic implementation bricks completes the analysis—along with a study of their expressiveness

- Chapter 4 provides a distributed, *situated architecture* for coordination in pervasive Multi-Agent Systems (MAS), supporting engineering of the aforementioned coordination laws, implemented by deeply extending and refactoring an existing coordination infrastructure. The coordination language exploited by the infrastructure is accordingly extended so has to handle the novel abstractions, enabling and promoting *situated coordination*
- Chapter 5 studies the *Behavioural Implicit Communication* (BIC) model of social action, bringing its abstractions and mechanisms into the (chemical-inspired, situated) computational framework, so as to leverage the concept of tacit messages to enable *user-driven coordination*
- Chapter 6 describes the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model ($\mathcal{M}o\mathcal{K}$) for *self-organisation of knowledge* in knowledge-intensive STS
- Chapter 7 discusses the $\mathcal{M}o\mathcal{K}$ technology, describing both a prototype middleware and a complete eco-system currently under development

Chapters are further arranged in two parts: Part I includes chapters 2 to 5, Part II chapters 6 and 7. This reflects the complementary nature of the research works presented in Part I w.r.t. the $\mathcal{M}o\mathcal{K}$ model described in Part II.

There, in fact, *(i)* the chemical-inspired approach to self-organising coordination is the reference framework used to implement $\mathcal{M}o\mathcal{K}$ coordination laws as artificial chemical reactions, *(ii)* the situated architecture for pervasive MAS provides the runtime for execution of $\mathcal{M}o\mathcal{K}$ coordination laws, while the situated language is exploited to implement $\mathcal{M}o\mathcal{K}$ artificial chemical reactions, and *(iii)* BIC principles drive the self-organising coordination process according to users interactions.

## 1.1 Organisation of Chapters

The thesis is organised in three parts:

**Part I** describes the three complementary research contributions necessary to conceptually ground the $\mathcal{M}o\mathcal{K}$ model, and design & implement $\mathcal{M}o\mathcal{K}$ infrastructure

**Part II** describes the $\mathcal{M}o\mathcal{K}$ model and technology

**Part III** concludes the thesis providing final remarks and a glimpse of what could be done next

Here follows a thorough description of the Chapters within each Part.

## Part I

Part I starts with Chapter 2, providing a brief review of the literature regarding tuple-based coordination for self-organising systems, focussed on the models and technologies which mostly influenced the thesis.

Then, Chapter 3 describes the approach to self-organising coordination:

**Section 3.1** presents an interpretation of chemical-inspired coordination relying on custom kinetic rates in place of the law of mass action for artificial chemical reactions implementing coordination laws

**Section 3.2** presents uniform coordination primitives, that is, a probabilistic extension to LINDA coordination primitives, as basic mechanisms on which chemical-inspired coordination laws can be implemented, showcasing its expressiveness informally

**Section 3.3** completes uniform primitives study of expressiveness, but from the formal perspective of language embedding

**Section 3.4** provides remarks on how the research work just presented contributes to the $\mathcal{M}o\mathcal{K}$ model, and how it can be further advanced

Chapter 4 is dedicated to the issue of situatedness in pervasive systems, thus, describes the approach to deal with it:

**Section 4.1** reviews the literature to frame the issue from an historical perspective, analysing the evolution of the meta-models and architectures proposed during time, then describes the proposed architecture along with its meta-model

**Section 4.2** elaborates on how the architecture supports environmental situatedness from the infrastructural point of view

**Section 4.3** elaborates on how the architecture supports spatial situatedness from the linguistic point of view

**Section 4.4** provides remarks on how the research work just presented contributes to the $\mathcal{M}o\mathcal{K}$ model, and how it can be further advanced

Finally, Chapter 5 discusses how it is possible to take advantage of some of the peculiar traits of knowledge-intensive STS to promote coordination driven by users interactions:

**Section 5.1** presents the main challenges of such a kind of systems, and how they can be dealt with from a coordination perspective

**Section 5.2** briefly reviews the research efforts which tried to integrate socio-cognitive theories of actions within a computational framework for designing MAS

**Section 5.3** reports on a survey of the (inter-)actions means provided by a few real-world STS, to connect the aforementioned theories of action with the real world

**Section 5.4** provides remarks on how the research work just presented contributes to the $\mathcal{M}o\mathcal{K}$ model, and how it can be further advanced

## Part II

Part II starts with Chapter 6, which thoroughly describes the $\mathcal{M}o\mathcal{K}$ model for self-organisation in knowledge-intensive STS:

**Section 6.1** presents the core abstractions which the model revolves around

**Section 6.2** thoroughly describes $\mathcal{M}o\mathcal{K}$ reactions, what they are meant for, how they work, and also reports on their early evaluation through simulations

**Section 6.3** thoroughly describes the interaction model of $\mathcal{M}o\mathcal{K}$, that is, how users' interactions affect the coordination process

**Section 6.4** presents some early investigations on the issue of measuring similarity between pieces of information, for the sake of promoting semantic-driven coordination

Then, Chapter 7 describes the $\mathcal{M}o\mathcal{K}$ technology:

**Section 7.1** describes an early prototype of the $\mathcal{M}o\mathcal{K}$ middleware running on top of the TuCSoN infrastructure, and implemented in the ReSpecT language, also reporting on an early evaluation through deployment within a news management scenario

**Section 7.2** instead, reports on an ongoing effort to implement a full-fledged $\mathcal{M}o\mathcal{K}$ ecosystem, encompassing also functionalities such as information harvesting, knowledge discovery, persistency, and the like

## Part III

Part III concludes the thesis, by providing final remarks on the whole body of work, and outlining a research roadmap for further works, on each of the research contributions brought by the thesis.

## 1.2 List of Publications

Here follows a comprehensive list of the publications which directly contributed to the body of work presented in this thesis, authored, or co-authored, by the same author of this thesis:

**chapter 3**

- Franco Zambonelli, Andrea Omicini, Bernhard Anzengruber, Gabriella Castelli, Francesco L. DeAngelis, Giovanna Di Marzo Serugendo, Simon Dobson, Jose Luis Fernandez-Marquez, Alois Ferscha, Marco Mamei, Stefano Mariani, Ambra Molesini, Sara Montagna, Jussi Nieminen, Danilo Pianini, Matteo Risoldi, Alberto Rosi, Graeme Stevenson, Mirko Viroli, and Juan Ye. Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive and Mobile Computing*, 17:236–252, February 2015. Special Issue "10 years of Pervasive Computing" In Honor of Chatschik Bisdikian
- Stefano Mariani. On the "local-to-global" issue in self-organisation: Chemical reactions with custom kinetic rates. In *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014*, Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, pages 61 – 67, London, UK, September 2014. IEEE. Best student paper award
- Stefano Mariani. Parameter engineering vs. parameter tuning: the case of biochemical coordination in MoK. In Matteo Baldoni, Cristina Baroglio, Federico Bergenti, and Alfredo Garro, editors, *From Objects to Agents*, volume 1099 of *CEUR Workshop Proceedings*, pages 16–23, Turin, Italy, 2–3 December 2013. Sun SITE Central Europe, RWTH Aachen University. XIV Workshop (WOA 2013). Workshop Notes
- Stefano Mariani and Andrea Omicini. Coordination mechanisms for the modelling and simulation of stochastic systems: The case of uniform primitives. *SCS M&S Magazine*, IV:6–25, December 2014. Special Issue on "Agents and Multi-Agent Systems: From Objects to Agents"
- Stefano Mariani and Andrea Omicini. Probabilistic embedding: Experiments with tuple-based probabilistic languages. In *28th ACM Symposium on Applied Computing (SAC 2013)*, pages 1380–1382, Coimbra, Portugal, 18–22 March 2013. Poster Paper
- Stefano Mariani and Andrea Omicini. Probabilistic modular embedding for stochastic coordinated systems. In Christine Julien and Rocco De Nicola, editors, *Coordination Models and Languages*, volume 7890 of *LNCS*, pages 151–165. Springer, 2013. 15th International Conference (COORDINATION 2013), Florence, Italy, 3–6 June 2013. Proceedings

**chapter 4**

- Andrea Omicini and Stefano Mariani. Coordination for situated MAS: Towards an event-driven architecture. In Daniel Moldt and Heiko Rölke, editors, *International Workshop on Petri Nets and Software Engineering (PNSE'13)*, volume 989 of *CEUR Workshop Proceedings*, pages 17–22. Sun SITE Central Europe, RWTH Aachen University, 6 August 2013
- Stefano Mariani and Andrea Omicini. Coordinating activities and change: An event-driven architecture for situated MAS. *Engineering Applications of Artificial Intelligence*, 41:298–309, May 2015. Special Section on Agent-oriented Methods for Engineering Complex Distributed Systems
- Stefano Mariani and Andrea Omicini. TuCSoN coordination for MAS situatedness: Towards a methodology. In Corrado Santoro and Federico Bergenti, editors, *WOA 2014 – XV Workshop Nazionale "Dagli Oggetti agli Agenti"*, volume 1260 of *CEUR Workshop Proceedings*, pages 62–71, Catania, Italy, 24–26 September 2014. Sun SITE Central Europe, RWTH Aachen University
- Stefano Mariani and Andrea Omicini. Coordination in situated systems: Engineering mas environment in TuCSoN. In Giancarlo Fortino, Giuseppe Di Fatta, Wenfeng Li, Sergio Ochoa, Alfredo Cuzzocrea, and Mukaddim Pathan, editors, *Internet and Distributed Computing Systems*, volume 8729 of *Lecture Notes in Computer Science*, pages 99–110. Springer International Publishing, September 2014. 7th International Conference on Internet and Distributed Computing Systems (IDCS 2014), Calabria, Italy, 22-24 September 2014, Proceedings
- Stefano Mariani and Andrea Omicini. Space-aware coordination in ReSpecT. In Matteo Baldoni, Cristina Baroglio, Federico Bergenti, and Alfredo Garro, editors, *From Objects to Agents*, volume 1099 of *CEUR Workshop Proceedings*, pages 1–7, Turin, Italy, 2–3 December 2013. Sun SITE Central Europe, RWTH Aachen University. XIV Workshop (WOA 2013). Workshop Notes
- Stefano Mariani and Andrea Omicini. Promoting space-aware coordination: ReSpecT as a spatial-computing virtual machine. In *Spatial Computing Workshop (SCW 2013)*, AAMAS 2013, Saint Paul, Minnesota, USA, May 2013

**chapters 5, 6, 7**

- Stefano Mariani and Andrea Omicini. Molecules of Knowledge: Self-organisation in knowledge-intensive environments. In Giancarlo Fortino, Costin Bădică, Michele Malgeri, and Rainer Unland, editors, *Intelligent Distributed Computing VI*, volume 446 of *Studies in Computational Intelligence*, pages 17–22. Springer, 2013
- Stefano Mariani and Andrea Omicini. Self-organising news management: The *Molecules of Knowledge* approach. In Jeremy Pitt, editor, *Self-Adaptive and*

*Self-Organizing Systems Workshops (SASOW)*, pages 235–240. IEEE CS, 2012. 2012 IEEE Sixth International Conference (SASOW 2012), Lyon, France, 10-14 September 2012. Proceedings

- Stefano Mariani and Andrea Omicini. MoK: Stigmergy meets chemistry to exploit social actions for coordination purposes. In Harko Verhagen, Pablo Noriega, Tina Balke, and Marina de Vos, editors, *Social Coordination: Principles, Artefacts and Theories (SOCIAL.PATH)*, pages 50–57, AISB Convention 2013, University of Exeter, UK, 3–5 April 2013. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour
- Stefano Mariani and Andrea Omicini. Anticipatory coordination in socio-technical knowledge-intensive environments: Behavioural implicit communication in MoK. In Marco Gavanelli, Evelina Lamma, and Fabrizio Riguzzi, editors, *AI*IA 2015, Advances in Artificial Intelligence*, volume 9336 of *Lecture Notes in Computer Science*, chapter 8, pages 102–115. Springer International Publishing, 23–25 September 2015. XIVth International Conference of the Italian Association for Artificial Intelligence, Ferrara, Italy, September 23–25, 2015, Proceedings

# Part I

# Coordination Issues in Complex Socio-technical Systems

In the first part of this thesis the engineering challenges posed by three different but related kinds of complex systems are discussed from a coordination models & technologies perspective, then a novel approach to deal with each of them is presented. In particular, the focus is on *self-organising* (Chapters 2-3), *situated pervasive* systems (Chapter 4), finally *knowledge-intensive socio-technical* ones (Chapter 5).

Each contribution is a necessary ingredient for building the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model described in Part II of this thesis, namely: the approach to deal with self-organisation through *uniform primitives* and *artificial chemical reactions* with custom-defined kinetic rates (Chapter 3) is the ground upon which $\mathcal{M}o\mathcal{K}$ *reactions* are conceived and designed; the situated architecture and language presented in Chapter 4 is used to implement the prototype of the $\mathcal{M}o\mathcal{K}$ middleware on TuCSoN coordination infrastructure, using the extended ReSpecT language; the theory of *behavioural implicit communication* discussed in Chapter 5, in particular, the notion of *tacit message*, is used as the conceptual ground upon which to conceive and design $\mathcal{M}o\mathcal{K}$ *user-centred* self-organisation mechanisms—*enzymes*, *traces*, and *perturbation* actions in particular.

# Chapter 2

# Coordination Issues in Self-organising Systems

In this chapter is provided an overview of those approaches to self-organising coordination (Section 2.2, Section 2.3, Section 2.4) which directly motivated, inspired, and influenced the approach to coordination in self-organising systems proposed (Chapter 3). A brief argumentation on why the approaches are all tuple-based, and on which benefits this brings [RCD01], is provided as a preparatory background, describing the seminal LINDA model [Gel85] which almost all the presented approaches are built on top of (Section 2.1)—including the one proposed.

## 2.1 Where all began: LINDA

LINDA [Gel85] has been proposed as a distributed programming language exploiting the notion of *generative communication*—introduced in the same paper. Nevertheless, it is suddenly recognised by Gelernter himself as a full-fledged computational model whose implications would go beyond distributed communication. In fact, LINDA is nowadays mostly referred to as a *coordination model* [Cia96], rather than merely a distributed programming language.

The LINDA language consists of three primitives: `out`, `in`, `rd`. Respectively, they allow processes to produce, consume, and observe *tuples* stored in a *tuple space*. Tuples are named, ordered collections of heterogenous, typed values and/or variables—usually, tuples containing some variables are called *templates* (or, anti-tuples). A tuple space is the abstract computational environment in which LINDA programs are executed, thus, it includes tuples as well as processes using LINDA primitives.

The semantics of LINDA primitives is what makes the LINDA language particularly suitable to be used as a coordination model: whereas the `out` primitive always puts a tuple in the tuple space[1], `in` and `rd` *attempt* to get one—respectively, withdrawing it or not. In fact, if a tuple *matching* the variables (possibly) used in the `in` or `rd` primitive is found, that tuple is returned and the caller process may continue execution; otherwise, the caller process is *suspended* until a matching tuple becomes available. This execution semantics is often referred to as Linda *suspensive semantics*, and is one of its distinguishing features enabling coordination policies to be programmed solely on top of the Linda language.

What enables this semantics is the primary contribution of Gelernter's paper, that is, *generative communication*. In generative communication, the data items communicated – that is, the tuples – live independently w.r.t. their producers, since they are (possibly, persistently) stored by the tuple space. This also enables uncoupling in space and time: in fact, sender (receiver) processes need not to know when and where receivers (senders) will be executing in order to successfully communicate.

The last distinguishing feature of the Linda model is the *associative access* to data. In fact, interacting processes need not to know the address where a tuple is stored to access it: they simply have to know their name and content—even partially thanks to variables. This enables a third form of uncoupling: reference uncoupling (called communication orthogonality in [Gel85]), both w.r.t. tuples and to senders/receivers.

Linda appears well-suited for supporting coordination in systems featuring, e.g.: *(i)* distribution, by relying on multiple tuple spaces installed on networked hosts; *(ii)* openness, thanks to its uncoupling facilities; *(iii)* incomplete information, handled by associative access. Knowledge-intensive Socio-Technical Systems (STS) enjoy all these features, nevertheless have many more which cannot be dealt with effectively by Linda as it is; in particular, *uncertainty* and *unpredictability*, and the need for *adaptiveness*.

For this reason, the following sections briefly review some coordination models and infrastructures, either inspired to Linda or implementing Linda in interesting ways, all having in common the goal of dealing with the aforementioned shortcomings—some by looking at natural metaphors for implementation, some by extending the model.

## 2.2 Leveraging Stochasticity

A natural solution to account for *unpredictability* and *uncertainty* is to embrace *stochasticity* by tolerating probabilistic rather than deterministic computations and decision making. Among the many existing stochastic approaches to coordination reviewed – e.g., digital pheromones [PBS02], biochemical tuple spaces [VC09], SAPERE [ZCF$^+$11], probabilistic pi-calculus [HP01], stochastic Klaim [DNLKM06] – the focus is on describing the one which proven to be particularly influential for this thesis: SwarmLinda [TM04].

[1]In his paper Gelernter does not further specifies `out` semantics. Much more recently, [BGZ00] distinguishes two admissible semantics regarding actual tuple insertion in the tuple space w.r.t. to the producer process, deeply affecting the computational expressiveness of the Linda model. In the following, is always assumed the *ordered semantics* of `out` primitive—as well as of any other insertion primitive which could be mentioned.

### 2.2.1 SwarmLinda

SwarmLinda [TM04] is the proposal of a scalable implementation of the Linda model based on *swarm intelligence* techniques, drawing inspiration from ant colonies [Par97].

One can understand the world of SwarmLinda as a terrain (network of tuple spaces) in which ants (tuple templates) search for food (tuples), leaving pheromone trails upon successful matches, indicating the *likelihood* that further matches for that template are available. Ants look for food in the proximity of the anthill (the caller process); when found, the food is brought to the anthill[2] and a trail is left so that other ants can know where the food is. The digital ants behave according to the following rules:

1. spread the scent of the caller process in the node it is interacting with and its neighbourhood, to represent the anthill
2. check for a matching tuple: if a match is found, return to the anthill leaving scent for the template matched at each step (tuple space traversed); if no match is found, check the 1-hop neighbourhood for traces of the desired scent
3. if no desired scent is found, *randomly* choose a neighbour space to continue search
4. if a desired scent is found, move one step toward that scent and start over

The key to scalability lies in the *local* nature of ants perceptions: they carry out local searches solely, and inquire direct neighbours only. Furthermore, *probabilistic non-determinism* – necessary for supporting adaptiveness and fault-tolerance [TM04] – is achieved by adding a small random factor to each scent: this enables paths other that the one with the strongest scent to be chosen.

This results in the *emergence* of application specific paths between tuple producers and consumers. Moreover, given that scents are *volatile* thanks to an evaporation mechanism, the paths found can dynamically adapt to changes in the system—e.g., when consumers or producers join, leave or move, or in case of failures in nodes hosting tuple spaces.

Besides tuples searching, ant colonies inspiration is also used to partition tuple spaces dynamically, that is, in brood sorting [Par97]. In nature, ants are able to sort different kinds of things kept in the anthill such as food, larvae, eggs, etc., and do so in spite of the amount of each type, thus being very scalable. In SwarmLinda, tuples are the things to sort and the ant is the active process representing an `out`, thus:

1. upon execution of an `out`, start visiting the nodes
2. observe the *kind* of tuples the nodes store, that is, the template they match
3. store the tuple in the node if nearby nodes store tuples matching the same template, considering a small random factor

[2]The ants know the way back to the anthill because *(i)* they have a short memory of the last few steps they took and *(i)* the anthill has a distinctive scent.

4. if nearby nodes do not contain similar tuples, randomly choose whether to drop or continue to carry the tuple to another node

To guarantee convergence to meaningful partitions, certain conditions must be satisfied: *(i)* for each time the process decides not to store the tuple, the random factor will tend to $\epsilon \approx 0$ so as to increase the chance of storing the tuple in next steps; *(ii)* the likelihood of locally storing the tuple is calculated probabilistically, based on the kinds of objects in the ant's memory, that is, if most of them are similar to the one being carried, the likelihood to deposit the tuple increases.

Summing up, SwarmLinda is a nature-inspired implementation of the LINDA model, accounting for *uncertainty* and *unpredictability* – e.g., of tuples' location and of agents' interactions – by embracing probability in the mechanisms supporting tuples searching and storage. *Adaptiveness* is enabled in turn, by leveraging *stochasticity* in both resource-to-consumer paths formation and tuples partitioning.

## 2.3 Leveraging Programmability

As far as adaptiveness is a main concern, a natural solution to support it is to allow some degree of *programmability* of the coordination machinery, e.g., allowing interacting agents to change the coordination laws, or even the coordination medium to change them itself. Among the many existing approaches to programmable coordination reviewed – e.g., TOTA [MZ09], MARS [CLZ00], LGI [MU00] – the focus is on describing the one which proven to be particularly influential for this thesis: ReSpecT [Omi07].

### 2.3.1 ReSpecT

ReSpecT (Reaction Specification Tuples) is a *logic-based* language for the coordination of complex software systems [Omi07]. ReSpecT promotes a coordination model providing *tuple centres* [OD01b] as programmable, general-purpose coordination media [Cia96]. The behaviour of ReSpecT tuple centres is programmed through the ReSpecT first-order logic language.

A tuple centre is a tuple space enhanced with the possibility to program its behaviour in response to interactions. First of all, coordinated entities (ReSpecT agents, henceforth, or simply agents) can operate on a ReSpecT tuple centre in the same way as on a standard LINDA tuple space: by exchanging tuples – which are first-order logic terms, in the case of ReSpecT – through a simple set of coordination primitives. Accordingly, a tuple centre enjoys all the many features of a tuple space mentioned in Section 2.1, that is, generative communication, associative access, and suspensive semantics.

Then, while the basic tuple centre model is independent of the type of tuple, ReSpecT tuple centres adopt logic tuples – both tuples and tuple templates are essentially Prolog facts – and *logic unification* is used as the tuple-matching mechanism. Since the overall content of a tuple centre is a multi-set 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.

Finally, a tuple centre is a *programmable* tuple space, so as to add programmability of the coordination medium as a new dimension of coordination. While the behaviour of a tuple space in response to interaction events is *fixed* – so, the effects of coordination primitives are fixed –, the behaviour of a tuple centre can be *tailored* to the system 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, ReSpecT tuple centres are obviously programmed through the ReSpecT logic-based specification language.

The ReSpecT coordination language is a logic-based language for the specification of the behaviour of tuple centres. As a behaviour specification language, ReSpecT *(i)* enables the definition of computations within a tuple centre, called *reactions*, and *(ii)* 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 $\theta = mgu(E, Ev)$—where *mgu* is the most general unifier, in Prolog terminology. 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 succeed, and fails otherwise. Each reaction is executed sequentially with a *transactional semantics*: so, a failed reaction has no effect on the state of a tuple centre.

All the reactions triggered by an event are executed before serving any other event: so, agents perceive the result of serving the event and executing all the associated reactions altogether as a single transition of the tuple centre state. As a result, the effect of a coordination primitive on a tuple centre can be made as complex as needed by the coordination requirements of a system.

Generally speaking, since ReSpecT has been shown to be Turing-equivalent [DNO98], 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 be used to represent and enact policies and rules (laws) for coordination systems of any sort.

Summing up, ReSpecT is a coordination language supporting programmability of coordination primitives and laws, allowing run-time modifications of the semantics of primitives, as well as run-time adjustment of coordination laws, thus enabling and promoting *adaptiveness* of the overall coordination logic.

## 2.4 Putting all together

It appears natural to combine the aforementioned approaches into a single, comprehensive and coherent approach leveraging both stochasticity and programmability of the coordination machinery, so as to better cope with adaptiveness and unpredictability/uncertainty. The following sections report on the two most promising approaches reviewed, which also influenced the coordination approach proposed in Chapter 3, as well as the overall contribution of this thesis: biochemical tuple spaces [VC09] and SAPERE [ZOA+15].

### 2.4.1 Biochemical Tuple Spaces

Biochemical Tuple Spaces (henceforth BTS) are a *stochastic*, *chemical-inspired* extension of LINDA tuple spaces first, then of the whole LINDA model [VC09]. Chemical inspiration stems mainly from two considerations: on the one hand, coordination models are increasingly required to tackle *self-** properties as inherent systemic properties, rather than peculiar aspects of individual coordinated components [ZV08]; on the other hand, among the many plausible natural metaphors available, the chemical one appears particularly interesting for the complexity of the stochastic behaviours it enables, despite the simplicity of its foundation [VCO09].

The essential idea behind the BTS model is to equip each tuple with a *concentration* value, seen as a measure of the pertinency/activity of the tuple: the higher it is, the more *likely* and *frequently* the tuple will influence system coordination. Concentration of tuples is dynamic, and evolves driven by programmable *chemical-like rules* installed into the tuple space, affecting concentrations over time *exactly* as chemical substances evolve within chemical solutions [Gil77].

Accordingly, primitive `out` now injects a given concentration of a tuple, possibly increasing the concentration of identical tuples already in the space. Primitive `in` either entirely removes a tuple, if no concentration is specified, or decreases the concentration of an existing tuple of the given amount. Primitive `rd` just reads tuples instead of removing them. Also the matching function $\mu$ changes w.r.t. LINDA, so as to return 0 if tuple $\tau$ does not match template $\tau'$, 1 if they completely match, and any value in between to represent partial matching.

From an infrastructural standpoint, a BTS-coordinated system is a set of tuple spaces networked in some *topological* structure, defining the neighbourhood of each space. Interaction between tuple spaces is mediated by a chemical-like law that fires some tuples to a tuple space in the neighbourhood *picked probabilistically.* Links ($\rightsquigarrow$) between tuple spaces ($\sigma$ and $\sigma'$) allow tuples movement according to a *markovian rate* $r$, representing the average movement frequency: $\sigma \overset{r}{\rightsquigarrow} \sigma'$. Markovian rates follow the argument of [Gil77] stating that chemical reactions can be simulated as Continuous-Time Markov Chains (CTMC)[3]

[3]Actually, the BTS model is an *hybrid* CTMC/DTMC model, since instantaneous transitions are allowed; please, refer to [VC09] for a thorough explanation.

[Car08], and are also exploited for both primitives and chemical laws execution.

As regards execution of chemical-like reactions, expressed as rewriting rules of the form $[T_i \xrightarrow{r} T_o]$ – meaning reactants $T_i$ should be transformed into products $T_o$ with a markovian rate $r$ –, the following is worth noticing:

- in the general case where a tuple set $T$ is found that is not equal to $T_i$, but it rather matches $T_i$, $\{T/T_i\}$ could provide more solutions
- being one transition allowed *for each* different solution of substitutions $\{T_i/T\}$, one is to be chosen *probabilistically* depending on the matching function
- accordingly, the actual markovian rate of reaction execution is given by $\mu(T_i, T) * G(r, T, T|S)$, which computes the transition rate according to Gillespie's algorithm [Gil77]

The $G(\cdot,\cdot,\cdot)$ function is $r * \mathit{count}(T, T|S)$, where function $\mathit{count}(T, S)$ counts how many different combinations of tuples in $T$ actually occur in $S$, namely:

$$\mathit{count}(o, S) = 1, \quad \mathit{count}(\tau\langle n\rangle \oplus T, \tau\langle m\rangle \oplus S) = \frac{m(m-1)\ldots(m-n+1)}{n!} * \mathit{count}(T, S)$$

Summing up, the rate of execution is influenced by three factors: its intrinsic markovian rate $r$, the matching function $\mu$, the relative concentration of involved reactants $G$.

As regards Linda primitives extension, BTS operations semantics states that:

- primitive `out` is pretty much similar to Linda `out`
- primitive `rd` reads a matching tuple with greater concentration than the one specified, where the *likelihood* of choosing a particular tuple among the matching ones depends on the *ratio* between concentrations of $\tau'$ and $\tau$—namely, the higher the concentration of a tuple, the more it is likely to read it
- primitive `in` functioning is identical to `rd` but for the destructive semantics

In conclusion, the BTS model promotes self-organisation in multi-agent systems by leveraging a (bio-)chemical metaphor to extend the Linda model toward a fully stochastic and programmable coordination model.

### 2.4.2 SAPERE

SAPERE (Self-aware Pervasive Service Ecosystems) [ZCF+11] was a EU STREP Project funded within the EU 7 FP[4]. SAPERE takes as ground the Linda model, then follows the pioneer work of approaches like ReSpecT [Omi07] regarding supporting programmability of coordination laws, but taking a different stance inspired to the laws of nature observed

[4]FP7-ICT-2009.8.5: Self-awareness in Autonomic Systems.

in *natural ecosystems*, to promote self-organisation of services in pervasive computing scenarios. SAPERE is inspired to the Biochemical Tuple Space model presented in Subsection 2.4.1, although with some notable differences—e.g., the chemical concentration mechanisms exactly mimicking chemical dynamics is not mandatory in SAPERE.

SAPERE considers modelling and architecting a *pervasive* service environment as a non-layered *spatial* substrate, made up of networked SAPERE nodes, laid above the actual network infrastructure. This substrate embeds the basic laws of nature (*Eco-Laws* in SAPERE terminology) that rule the activities of the system.

Users can access the ecology in a decentralised way to consume and produce data and services. Any individual (e.g., devices, users, software services) has an associated semantic representation inside the ecosystem called *Live Semantic Annotation* (LSA). LSA are handled as living, *dynamic* entities, capable of reflecting the current situation and context of the component they describe. LSA my be used to encapsulate data relevant to the ecology, *reify* events, act as *observable* interfaces of components, and ultimately be the basis for enforcing *semantic* and *self-aware* forms of dynamic interaction—both for service aggregation/composition and for data/knowledge management.

Eco-laws define the basic policies to rule sorts of *virtual chemical reactions* among LSA, thus enforcing dynamic concept-based (e.g., semantic and goal-oriented) networking, composition, and *coordination* of data and services in the ecosystem, to establish bonds between entities, produce new LSA, and diffuse LSA in the network world.

More specifically, each SAPERE node embeds a so-called *LSA-space*, in which self-adaptive coordination mechanisms take place so as to mediate the interaction between components. Whenever a component approaches a node, its own LSA is automatically injected into the LSA-space of that node, making the component part of that space and of its *local coordination dynamics*.

LSA are semantic annotations with same expressiveness as standard frameworks like RDF. An LSA *pattern* $P$ is essentially an LSA with some variables in place of some values – similarly to Linda templates –, and an LSA $L$ is said to match the pattern $P$ if there exists a substitution of variables to values that applied to $P$ gives $L$. Differently from Linda, the *matching* mechanism here is *semantic* and *fuzzy* [NOV11].

Besides the LSA-space, each node embeds the set of eco-laws ruling the activities of the ecosystem. An eco-law is of the kind $P_1 + \ldots + P_n \xrightarrow{r} P'_1 + \ldots + P'_m$ where: *(i)* the left-hand side (reagents) specifies patterns that should match the LSA $L_1, \ldots, L_n$ to be extracted from the space; *(ii)* the right-hand side (products) specifies patterns of LSA which are to be inserted back in the space; and *(iii)* rate expression $r$ is a numerical positive value indicating the *average frequency* at which the eco-law is to be fired.

In other words, SAPERE models execution of the eco-laws as a CTMC (Continuous Time Markov Chain) transition with Markovian rate $r$. An eco-law is applied as follows: *(i)* iteratively, one reagent pattern $P_i$ is *non-deterministically* extracted from the eco-law, a matching LSA $L_i$ is found, and the resulting substitution is applied to the remainder of the eco-law; *(ii)* when (and if) iteration is over, products form the set of LSA to be inserted back in the space. As far as *topology* is concerned, the framework imposes that an eco-law applies to LSA belonging to the same space, and constrains products to be inserted in that space or in a *neighbouring* one (to realise space-space interaction).

Summing up, the SAPERE project proposed a model, a middleware, and a methodology for the engineering of pervasive services ecosystems, in which the *stochasticity* of interactions and computations, the *programmability* of the coordination laws, and aspects related to *semantic matching* and *context awareness* are of prominent relevance to enable and promote self-organisation and adaptiveness.

## 2.5 Remarks & Outlook

All the models and technologies presented in this chapter influences the whole thesis, not only the way in which the *M*olecules *o*f *K*nowledge model is conceived and designed.

Linda is taken as the reference model for designing data-driven, fully uncoupled coordination mechanisms. The whole chemical-inspired machinery described in Section 1 of next chapter is rooted on a tuple space based setting, with tuples playing the role of chemicals and tuple spaces of chemical solutions. Uniform primitives presented in Section 2 of next chapter are a specialisation of Linda primitives, born to overcome some limitations of the original model.

The infrastructure refactored and extended in Chapter 4, along with its language, is tuple based. Within *MoK*, besides information items to be managed, also users actions, as well as their side-effects, are represented as tuples in a tuple space based setting.

From SwarmLinda is taken the idea of using pheromone-based coordination to cluster tuples and find optimal routing paths between who searches for a tuple and who owns it—although this feature is included in *MoK* ecosystem only, and still under development (see Section 7.2 in Chapter 7).

Among the many differences w.r.t. the approach just described is the fact that the approach proposed in this thesis delegates pheromone handling to the infrastructure, in the form of chemical-inspired coordination laws, whereas SwarmLinda relies on dedicated processes. Also, in SwarmLinda tuple clustering is driven by tuples structure and content solely, while *MoK* also considers semantic aspects as well as the epistemic nature of users interactions—to some extent.

ReSpecT is the language used to implement the chemical machinery necessary to deploy the artificial chemical reactions described in Section 1 of next chapter, as well as the chemical reactions themselves. It is also the language used to program tuple centres within the TuCSoN infrastructure, thus the language extended as part of the research contribution described in Chapter 4.

Finally, the *MoK* prototype on TuCSoN infrastructure is implemented as a combination of Java (the language used to implement TuCSoN) and ReSpecT, with the latter dedicated, in particular, to implement *MoK* reactions and *MoK* compartments' chemical engine (see Section 7.1).

The chemical metaphor behind $\mathcal{M}o\mathcal{K}$ is similar to the Biochemical Tuple Spaces (BTS) model, although with some notable difference. Similarities encompasses tuple spaces being seen as chemical solutions simulators, coordination laws being modelled as chemical reactions, the notion of concentration to resemble tuples relevance, LINDA matching function being revisited to also consider partial matching (potentially, semantic, too) and influence reactions application rate.

Despite the many similarities, there are substantial differences to be pointed out, affecting, e.g., expressiveness of the model. In $\mathcal{M}o\mathcal{K}$ compartments are still chemical solutions simulators, but reactions are not obliged to follow the law of mass action, that is, to have a perfect match with how chemical solutions evolve in the natural world. This holds true also for the approach to chemical inspired coordination presented in Section 1 of next chapter. Also, in $\mathcal{M}o\mathcal{K}$ the pool of reactions driving system evolution is fixed: although reactions may be tuned, no reactions may be disabled, nor new ones added to the system. Communication among compartments is not stochastic, but only probabilistic – that is, timing aspects are not modelled – and primitives at agents disposal have the usual LINDA semantics—no stochasticity here, too.

The SAPERE project, too, similarly to the BTS model, influenced of the $\mathcal{M}o\mathcal{K}$ model and the whole stance to coordination taken in this thesis: actually, the two models are born in the same context and almost at the same time, and for a one-year research grant the author of this thesis worked on both at the same time. SAPERE in turn, is deeply rooted in the BTS model, too, so many of the similarities seen among BTS and the work in this thesis still hold, as well as many of the differences.

One of the most notable differences lies on the target scenario of the models: SAPERE is mostly conceived for engineering self-organisation within pervasive service-oriented architectures, while $\mathcal{M}o\mathcal{K}$ targets the issue of information and knowledge management in STS. In SAPERE, any service component interacts with others through programmable eco-laws involving their representative LSA. LSA are meant to reflect at anytime the state of the corresponding service component, enabling others to observe certain portions of it, so as to promote some form of awareness. In $\mathcal{M}o\mathcal{K}$, instead, nothing similar exists: interacting agents are not statically represented, but their actions within the system are dynamically reified and exploited for the benefit of the coordination process.

# Chapter 3

# Re-thinking Stochastic, Programmable Coordination

In this chapter a novel approach to coordination in self-organising systems is described, which re-thinks the basis of chemically inspired coordination, both from the engineering standpoint of coordination laws and primitives design, and from the scientific standpoint of relative linguistic expressiveness.

Accordingly, firstly the well-known local vs. global issue in self-organising systems is dealt with by engineering coordination laws as artificial chemical reactions with custom kinetic rates (Section 3.1), then, the impact of uniform coordination primitives on self-organising systems is discussed, experiments on their applicability are reported (Section 3.2), their semantics is defined, and their linguistic expressiveness studied (Section 3.3).

## 3.1 Chemical Reactions as Coordination Laws

A foremost issue while engineering *self-organising systems* is the *local vs. global* issue: how to link the *local* mechanisms, through which the components of the system interact, to the *emergent, global* behaviour exhibited by the system as a whole [BB06].

Existing approaches to face the issue are based on simulation [GVO06], parameter tuning [GVO09], (approximate) model checking [CV13], or bio-inspired design patterns [FMMSM$^+$12]. Simulation investigates the behaviour of the system prior to real-world deployment; parameter tuning helps improving the performance (according to whatever definition) of the single mechanisms; (approximate) model checking enables formal verification of expected global behaviours; design patterns assist programmers in deploying the correct self-organising solutions according to the problem to solve.

Nevertheless, these approaches may be not enough, especially if used separately: simulation may not be able to accurately reproduce real world contingencies; parameter tuning may lead to sub-optimal settings; model checking may be impractical for the complexity of the problem at hand; design patterns give no guarantees about quality of solutions.

For these reasons, an integrated approach to deal with the local vs. global issue in self-organising systems is proposed: *(i)* rely on design patterns to design the local mechanisms by implementing them as *artificial chemical reactions*; *(ii)* go beyond the *law of mass action* [Car08] by engineering *custom kinetic rates* reflecting the emergent, global behaviour desired; *(iii)* simulate-then-tune loop to adjust the dynamics of the (artificial) chemical system obtained so as to achieve the emergent, global behaviour desired.

This body of work is used as the ground upon which the custom kinetic rates of the chemical-like coordination laws in the $\mathcal{M}o\mathcal{K}$ model discussed in Part II of this thesis are designed—in particular, see Section 2 of Chapter 6.

## 3.1.1 Self-organisation Patterns

Among the possible approaches to face the local vs. global issue, one is to rely on *bio-inspired design patterns*. As the name suggests, the concept is similar to design patterns in OO programming: recurrent solutions to recurrent problems are modelled and encapsulated as architectural best practices to promote design reuse—and code reuse, ultimately. As explicitly stated in [FMMSM$^+$12] (e.g., Fig. 4 therein), patterns are often decomposable into atomic, non further decomposable patterns: the focus is precisely on these.

Thus, a survey of the state-of-art literature regarding design patterns and basic mechanisms enabling self-organisation has been undertaken – considering [Nag04, DWH07, FMSM12, FMDMSA11, TRDMS11, VCMZ11] –, which led to identification of the following set of basic patterns—in what follows, the term "information" is used, but can be replaced with "process", "component", anything which can be subject of self-organisation:

**decay** the decay pattern (aka evaporation, or cleaning) destroys information. Usually, it destroys a finite amount of information as time passes (e.g., evaporation of pheromone scent)

**feed** the feed pattern (aka reinforcement) increases information relevance. Usually, this means increasing information quantity (e.g., pheromone deposit over other pheromone)

**activation/inhibition** the activation pattern changes information status depending on external stimuli—the same holds for the dual inhibition pattern. Usually, this means some sort of stimulus triggers some sort of response (e.g., neurones activation in the brain), possibly changing some of the information attributes

**aggregation** the aggregation pattern fuses information together. Fusion can be filtering (e.g., new info replaces old), merging (e.g., related news synthesised into a single story), composing (e.g., building a list from separate values), transforming (e.g., summing, averaging, etc.), etc.

**diffusion** the diffusion pattern (aka spreading, or propagation) moves information within a topology. Diffusion can destroy moved information, resembling some sort of migration, or not, resembling replication

**repulsion/attraction** the repulsion pattern drifts apart information—dually, attraction approaches information instead. Usually, repulsion considers topological information to spread information fairly

Each of these patterns[1] is modelled as a chemical reaction, then engineered in different ways so as to obtain different global behaviours, finally simulated with the BioPEPA tool, presented in next paragraph, to show the effects of custom rates on the emergent, global self-organising behaviour obtained.

**On simulation** Among the existing frameworks and tools allowing simulation of chemical reactions, either born in biochemistry [AAS06] or in the multi-agent systems community [MAC+07], BioPEPA[2] [CH09] is used.

BioPEPA [CH09] is a language and tool for the simulation of biochemical processes. It is based on PEPA [GH94], a process algebra aimed at performance analysis, extended to deal with the typical features of biochemical networks, such as stoichiometry, compartments and kinetic laws. The most appealing features of BioPEPA are:

- the possibility to define custom kinetic laws by designing *functional rate expressions*
- the possibility to define stoichiometry (how many molecules of a given kind participate) and role played by the species (reactant, product, enzyme, etc.) involved a given reaction
- the possibility to define topologies of compartments among which reactants may move
- its theoretical roots in CTMC semantics [Her02]—behind any BioPEPA specification lies a stochastic labelled transition system modeling a CTMC

Rate expressions are defined as mathematical functions involving reactants concentrations (denoted with the reactant name and dynamically computed at run-time) and supporting:

- mathematical operators, e.g., `exp` and `log` functions
- built-in common kinetic laws, e.g., the law of mass action (denoted with the keyword `fMA`)
- time dependency through variable `time`, increasing according to the current simulation time step

[1]Except repulsion/attraction, that has been left-out since it can be engineered on top of diffusion.

[2]Eclipse plugin at `http://groups.inf.ed.ac.uk/pepa/update/` update site.

In summary, BioPEPA allows to: model self-organisation primitives as chemical reactions, simulate them using different stochastic simulation algorithms – the choice is Gillespie's algorithm [Gil77] –, tune rates according to simulation results.

### 3.1.2 Custom Kinetic Rates

The self-organisation patterns listed in Subsection 3.1.1 are now modelled as artificial chemical reactions representing the local mechanisms, then encoded in the BioPEPA language to finally simulate the emergent, global behaviour achievable. While doing so, custom kinetic rates are engineered in different ways, and BioPEPA plots resulting from simulations are compared to highlight how changes in the local mechanisms affect the global behaviour.

**Technical notes on experiments** Each of the following experiments is performed using the Gillespie stochastic simulation algorithm provided by BioPEPA. Each of the following plots is directly generated from BioPEPA as a result of the correspondent experiment, consisting of 100 Gillespie runs.

Nothing changes from run to run, they are necessary due to the stochastic aspects embedded in the simulation algorithm, and aims at showing more regular trends—it should be noticed, however, that single runs plots are really similar. In each chart, the $x$-axis plots the time steps of the simulation, whereas the $y$-axis plots the concentration level of the reactants expressed as units of molecules.

The codebase tracking the BioPEPA specifications used in the examples is publicly available under LGPL license at `http://bitbucket.org/smariani/mok-biopepa`.

**Decay** The decay pattern can be represented as an artificial chemical reaction as follows:

$$data \xrightarrow{r_{decay}} \perp$$

meaning that a certain unit of *data* disappears at a pace given by $r_{decay}$—this is the local mechanism. The reaction can be encoded in BioPEPA as follows:

```
DECAY_CONSTANT = 0.5;
r_decay = [fMA(DECAY_CONSTANT)]; // kinetic rate
data = (r_decay, 1) <<;          // chemical reaction
```

meaning that species *data* participates in decay chemical reaction as a reactant (<<), thus being consumed, with stoichiometry 1, thus one unit of *data* is involved. The rate ($r\_decay$) follows the usual law of mass action (fMA) [Car08].

Simulations of the above BioPEPA specification lead to the plots depicted in Figure 3.1, showing the global behaviour achieved—that is, the behaviour of the whole population of *data* items.

As shown, implementing the decay self-organisation pattern (the local mechanism) as an artificial chemical reaction with a kinetic rate following the law of mass action, leads

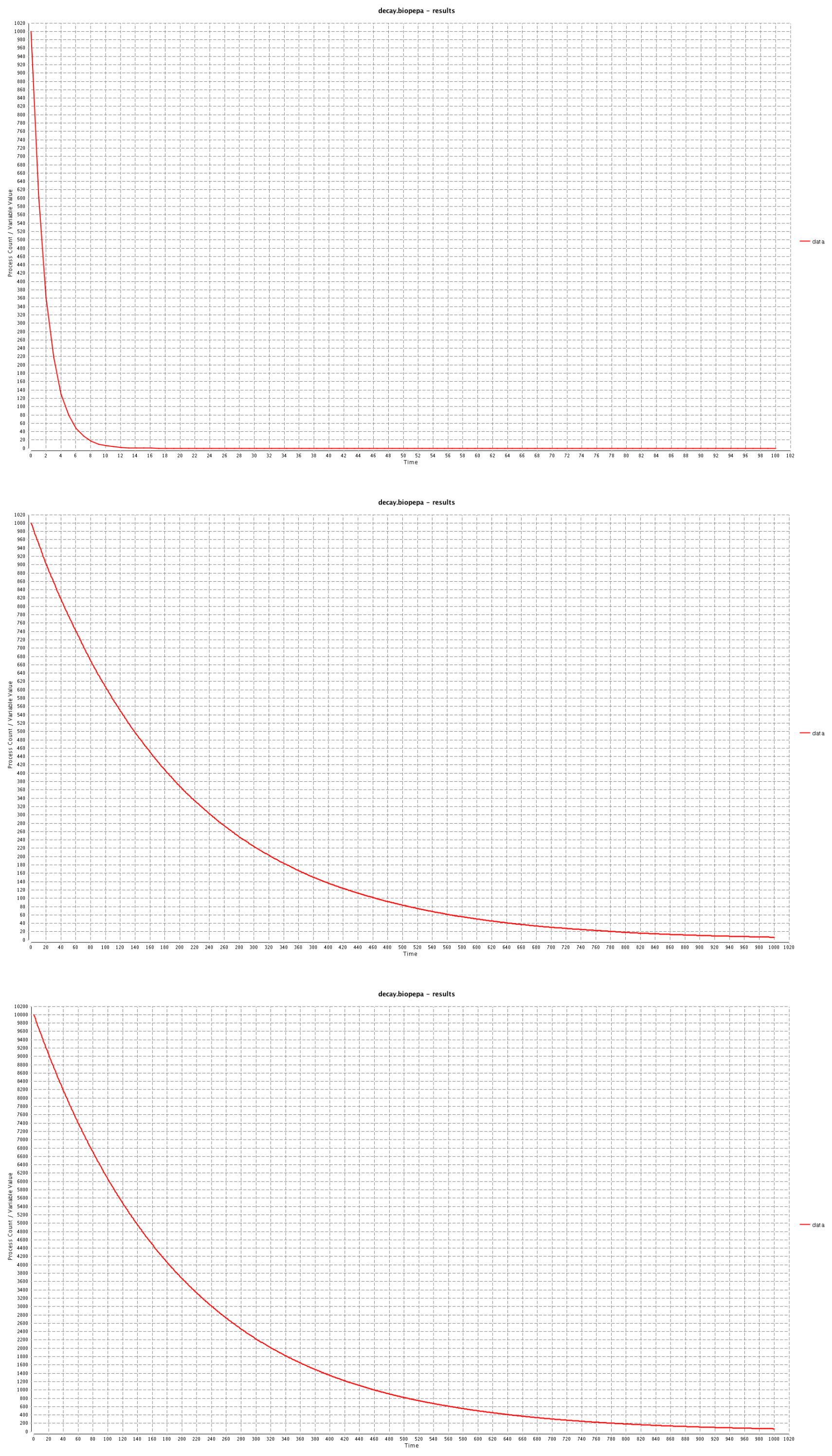


Figure 3.1: Decay chemical reaction with usual law of mass action rate [Mar14]. In the middle plot the rate constant is decreased w.r.t. the top plot (from 0.5 to 0.005), thus time needed to complete decay increases accordingly (from 100 simulation steps of top plot to 1, 000 steps of middle plot). In the bottom plot data quantity is increased w.r.t. previous plots (from 1, 000 units to 10, 000) but decay time remains the same (still 1, 000 time steps).

to a fast-then-slow decay (the emergent, global behaviour), which is independent of the quantity of data to decay (compare middle plot to bottom plot), and whose timing can be tuned by changing the rate constant (compare top plot to middle plot).

But: what if the aforementioned trend is not the best to suit the needs of the application at hand? What if the self-organising system should display a different trend, e.g., an opposite slow-then-fast decay? Maybe also sensitive to the quantity of information to decay? A time-dependant custom kinetic rate can be enconded as follows:

```
DECAY_CONSTANT = 0.5;
r_decay = [fMA(DECAY_CONSTANT) + H(data) * time/data];
data = (r_decay, 1) <<;
```

where `time` is the BioPEPA variable tracking simulation time steps and `H(·)` is the Heaviside step function[3], whose value is 0 for negative arguments and 1 for positive ones (0 arguments are usually associated to 0)—useful to avoid meaningless negative rates.

Simulations of the above custom kinetic rate are depicted in Figure 3.2. As shown, decay has now a completely different trend w.r.t. the plain fMA-driven rate of Figure 3.1: actually, the opposite, slow-then-fast trend. Furthermore, whereas decreasing the rate constant still leads to a delay in decay completion (compare top plot with middle plot), changing the quantity of *data* to decay now affects decay time proportionally (compare middle plot to bottom plot).

The reason for this behaviour twist lies in the new factors added to the kinetic rate expression: direct proportionality to `time` and inverse proportionality to *data*. Thus, while time passes decay slows down, whereas while quantity of data to decay decreases decay becomes faster. It should be noted that, in order to keep independency of data quantity, factor $\frac{1}{data}$ can be removed from the rate expression.

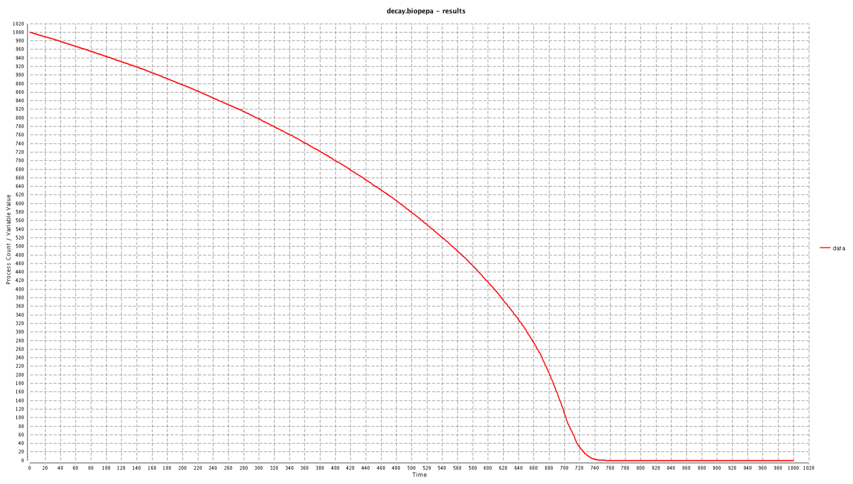
This demonstrates the extreme flexibility and accurate controllability that custom kinetic rates provide to engineers of self-organising systems: adding/removing factors to the local mechanism (the reaction manipulating each *data* item) leads to a well-defined change in the emergent, global behaviour achieved (the evolution of the whole population of *data* items), tackling the local vs. global issue.

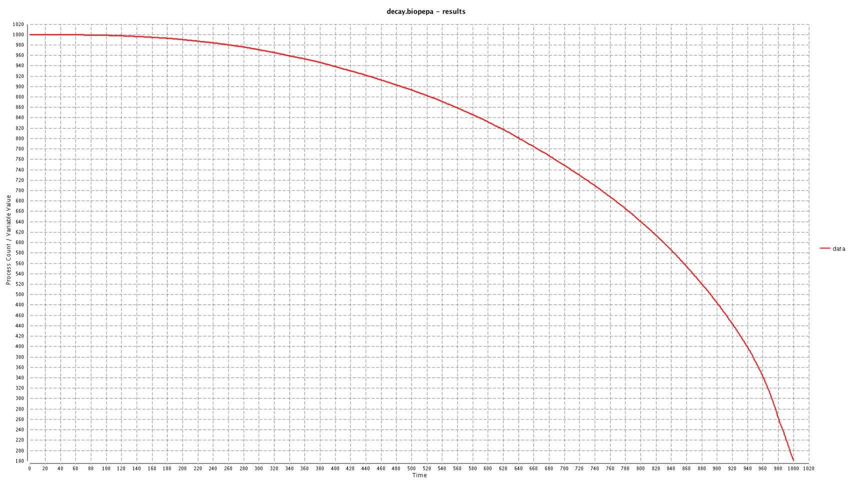
Discussing why it could be useful to switch from the trend depicted in Figure 3.1 to that of Figure 3.2 in a real-world application is out of the scope of this section, and is done, although in a more specific scenario, in Part II of this thesis—specifically, in Section 2 of Chapter 6. Nevertheless, giving some clues is undoubtedly useful.

Imagine to exploit self-organisation patterns in, e.g., an information management application, in which novel data can be produced and consumed anytime, without knowing *when* a given piece of information may become interesting for some of the co-workers. There, shifting to the slow-then-fast trend exhibited by the custom, time-dependant kinetic rate is better.

The reason is that, exactly because it is not known in advance when information will become relevant, it is unreasonable to decay it as soon as it is put in the shared

[3]`http://en.wikipedia.org/wiki/Heaviside_step_function`





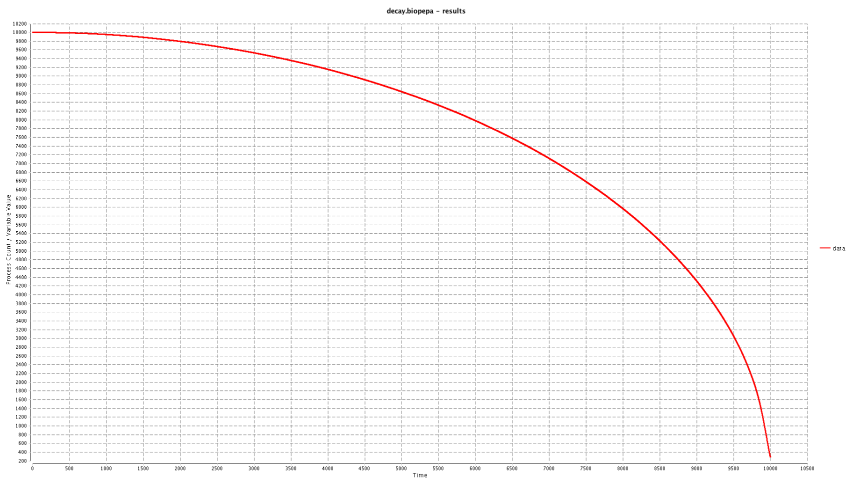


Figure 3.2: Decay chemical reaction with custom kinetic rate [Mar14]. In the middle plot the rate constant is decreased w.r.t. the top plot (from 0.5 to 0.005), thus time needed to complete decay increases accordingly (from $\approx$ 700 simulation steps of top plot to over $1,000$ steps of middle plot). In the bottom plot data quantity is increased w.r.t. previous plots (from $1,000$ units to $10,000$) and decay time increases proportionally (over $10,000$ time steps).

collaboration space: it is better to start with a slow decay so as to give co-workers the opportunity to find the information, then, only if after a while nobody manifested interest, the decay process can be safely quickened to get rid of (potentially) useless information.

**Feed** The feed pattern for self-organisation can be represented as an artificial chemical reaction as follows:

$$data + food \xrightarrow{r_{feed}} data + data$$

meaning that if *food* is present then *data* can be increased at a pace given by $r_{feed}$—the local mechanism. The reaction can be encoded in BioPEPA as follows, if the law of mass action only should influence the kinetic rate:

```
FEED_CONSTANT = 0.5;
r_feed = [fMA(FEED_CONSTANT)];
// same rate name ==> same chemical reaction
data = (r_feed, 1) >>; // product (1 unit produced)
food = (r_feed, 1) <<; // reactant (1 unit consumed)
```

Or as follows, if a time-dependant kinetic rate is preferred (only changes are reported):

```
r_feed = [fMA(FEED_CONSTANT) + H(food) * time/food];
```

Simulations focussing on the above BioPEPA specifications lead to the plots depicted in Figure 3.3. As shown, similarly to what happened in the case of decay, shifting to a custom kinetic rate twists the global behaviour achieved, that is, the growing trend exhibited by the whole population of *data* items: basically, with the usual law of mass action increase is fast-then-slow, whereas for the $\frac{time}{food}$-dependant rate, the trend is the opposite.

But, conversely to decay, acting on the rate constant or on the quantity of data to feed has the same effect both in the case of the fMA-driven kinetic rate and for the custom one: a lower rate leads to a slower feeding process, whereas a higher quantity of data does not affect time taken to complete the process.

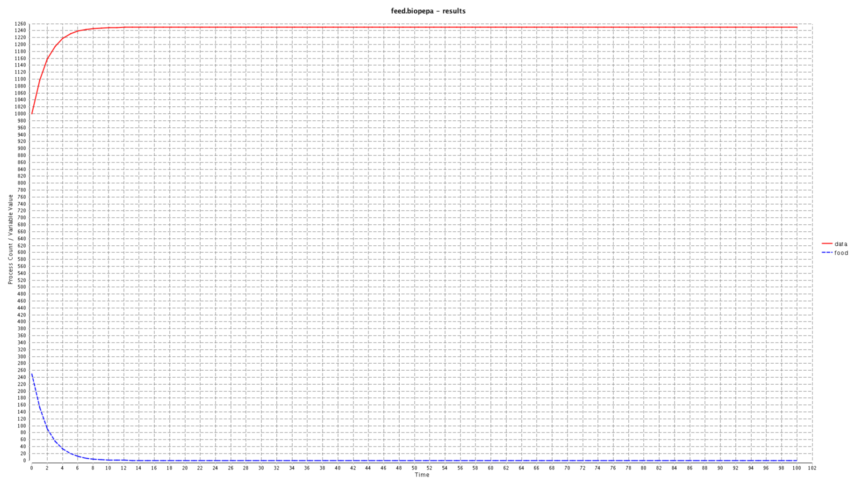
The reason for this difference at the level of emergent behaviour – that is, the change in the population dynamics –, can be once again explained in terms of the local mechanisms level—changing the reaction rate: whereas for the custom decay proportionality of the kinetic rate was to *data*, for the custom feed primitive proportionality is to *food*.

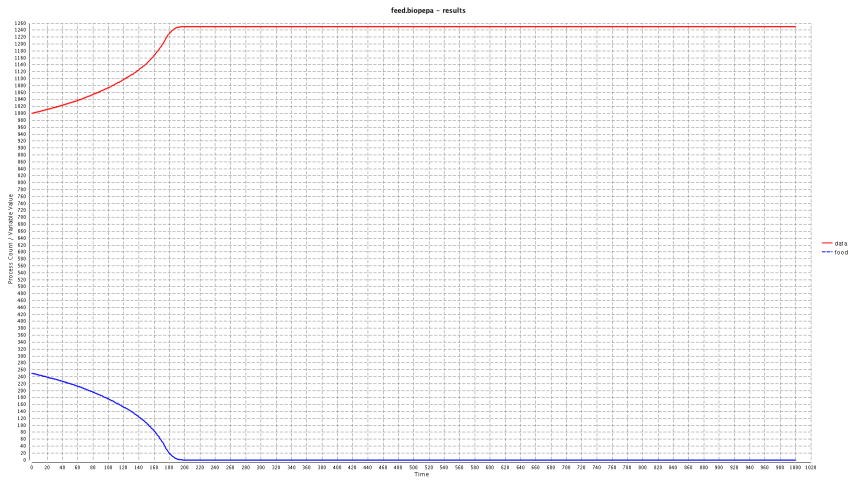
Once again, the global behaviour can be directly put in a causal relationship with the local mechanisms.

**Activation/Inhibition** The activation pattern can be represented as an artificial chemical reaction as follows:

$$data + on \xrightarrow{r_{activation}} on + data\_on$$

meaning that if *on* is present then *data* becomes *data_on*, due to activation, at a pace given by $r_{activation}$—the local mechanism. The dual inhibition pattern is similar:





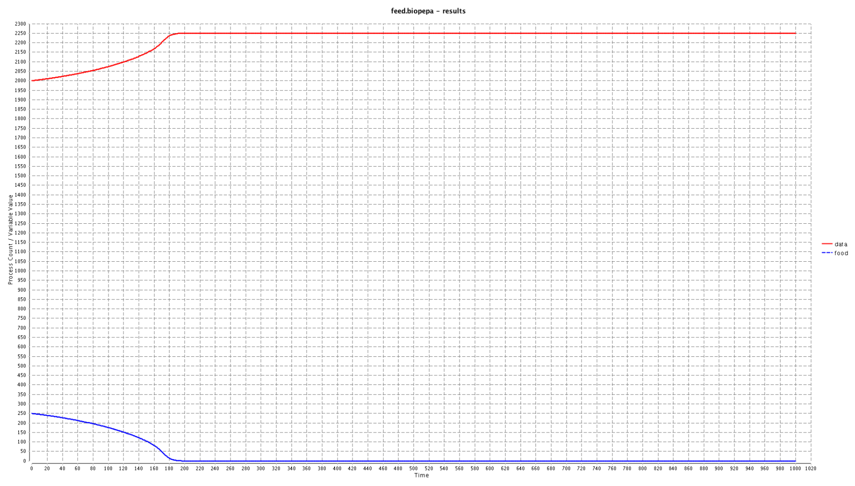


Figure 3.3: Feed chemical reaction with fMA-only (top plot) and custom kinetic rate (middle and bottom plots) [Mar14]. fMA-only rate is sensitive to rate constant change and to data quantity likewise decay; custom kinetic rate, too (bottom plot has twice the data of middle plot but saturation time is the same), due to its inverse proportionality to *food*, not *data*. Changing *food* affects saturation time of both rate expressions.

$$data + off \xrightarrow{r_{inhibition}} off + data\_off$$

Since also the trends exhibited by the two complementary patterns are dual, the focus is on either one: activation. The corresponding reaction is then encoded in BioPEPA according to three different kinetic rates – usual fMA-only, *data_on*-dependant and *data*-dependant –, as follows:

```
ACTIVATION_CONSTANT = 0.5;
r_activation = [fMA(ACTIVATION_CONSTANT)];
data = (r_activation, 1) <<;
on = (r_activation, 1) (+); // activator enzyme (not consumed)
data_on = (r_activation, 1) >>;
// inhibition ==> replace (+) with (-)
```

```
r_activation = [fMA(ACTIVATION_CONSTANT) +
                 H(data) * time/data_on];
```

```
r_activation = [fMA(ACTIVATION_CONSTANT) +
                 H(data) * time/data];
```

Simulations of the above BioPEPA specifications are depicted, respectively, in Figure 3.4, Figure 3.5, and Figure 3.6. As shown, three different emergent, global behaviours arise, that is, the dynamics exhibited by the whole population of *data* and *data_on* items, both in terms of the exhibited trend and regarding how involved parameters affect it.

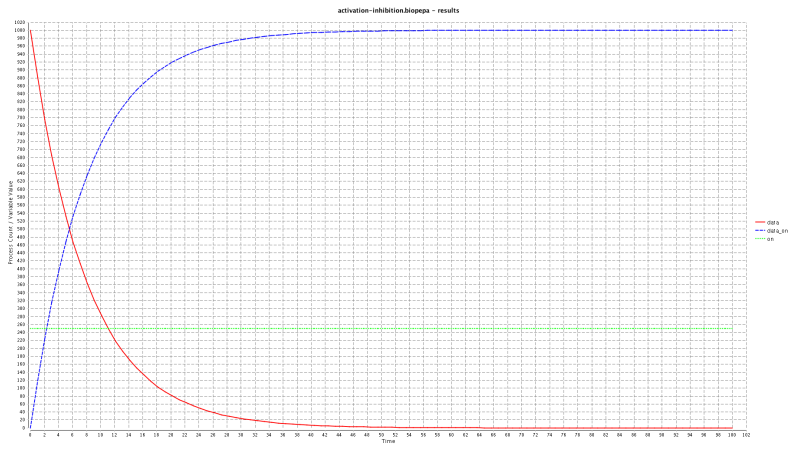
First of all, Figure 3.4 shows that a fMA-driven activation reaction is dependant on the rate constant as well as on the quantity of the activator enzyme (*on*), but not on the quantity of the reactant to activate (*data*). This is in contrast to what happened for the feed pattern, in which changing the reactant (*food*) affected the time taken to complete the feeding process.

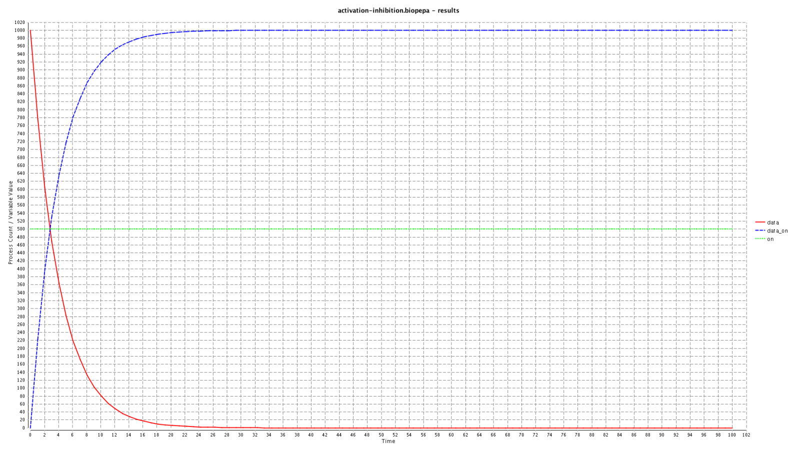
It should be noted that comparison is between *food* and *data*, not with *on* as could be done intuitively. The reason is that, although *food* and *on* play a similar role in the artificial chemical reaction, that is, that of activators, they have a different chemical nature: *food* is a *reactant*, being consumed by the reaction, whereas *on* is an enzyme, not altered by it. This difference is taken into account by the simulation algorithm provided by BioPEPA, and affects the outcome of the simulation accordingly.

In Figure 3.5, not only the emergent, global behaviour changes, but also sensitivity to the parameters involved in kinetic rate computation: dependency on the quantity of the activator enzyme is lost (*on*), while direct proportionality to the quantity of data undergoing the activation process is experienced.

This is the first time that a *product* of the artificial chemical reaction in process is put in its kinetic rate computation. This is something impossible to find in chemistry as it is in the natural world, but something to take advantage of in *artificial* chemical reactions. As a side note, this is the true power of custom kinetic rates: to be free from the natural world's constraints.

Finally, as far as Figure 3.6 is concerned, whereas the trend is completely different from both previous plots, sensitivity to parameters is the same as that of Figure 3.5: independency of the enzyme, direct proportionality to *data*.





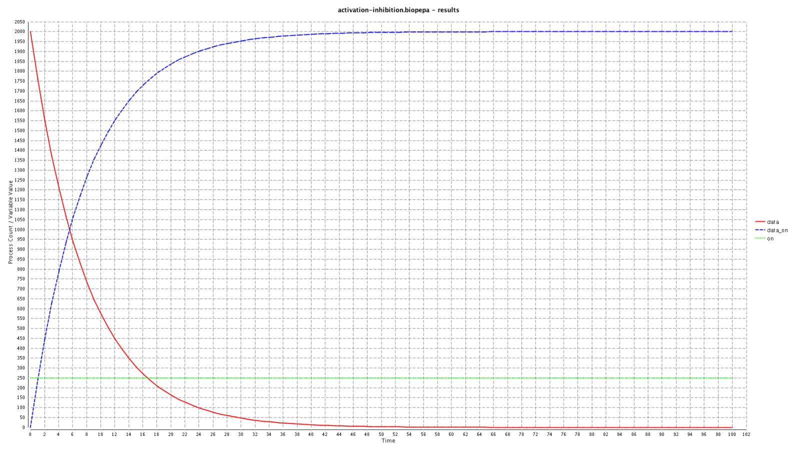
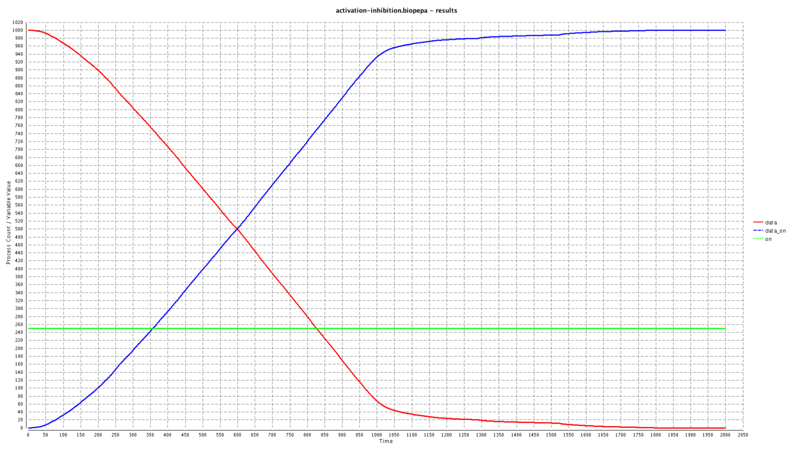
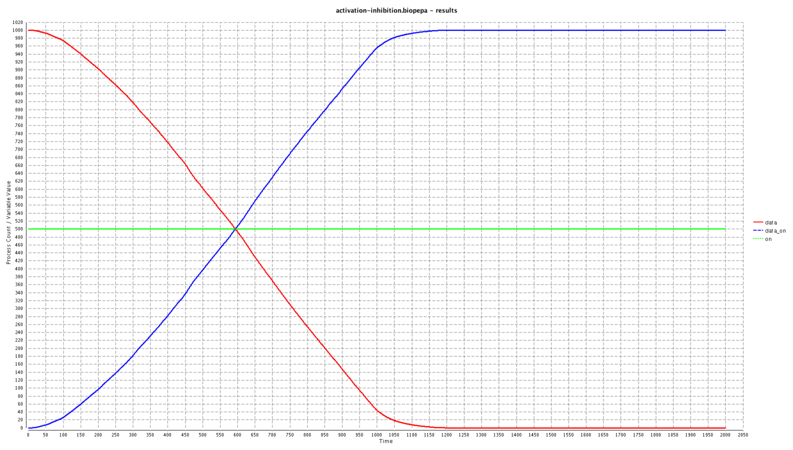


Figure 3.4: Activation chemical reaction with usual law of mass action rate [Mar14]. In the middle plot the activator enzyme quantity (*on*) is twice that of the top plot, causing a faster activation process (almost time step 6 in top plot, around time step 3 in middle plot). In the bottom plot *data* quantity is twice that of the other plots, but activation time is the same as that of top plot: thus, conversely w.r.t. reactant *food* in the feed pattern, the quantity of reactant *data* here does not affect timing. Acting on the rate constant speeds up or slows down the process as usual.





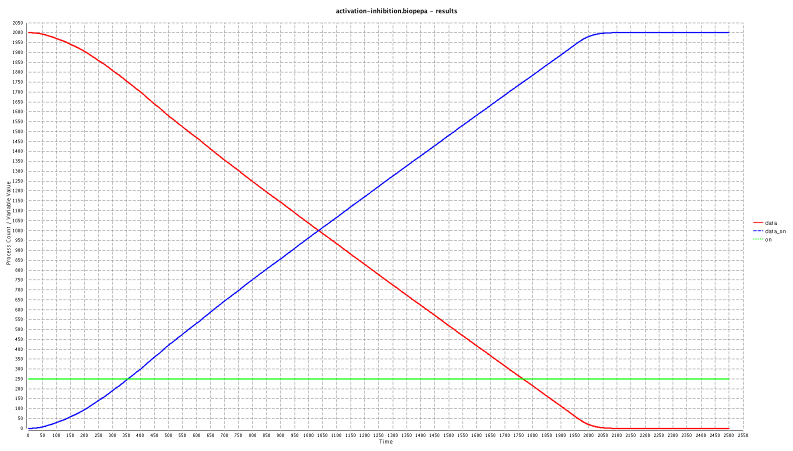
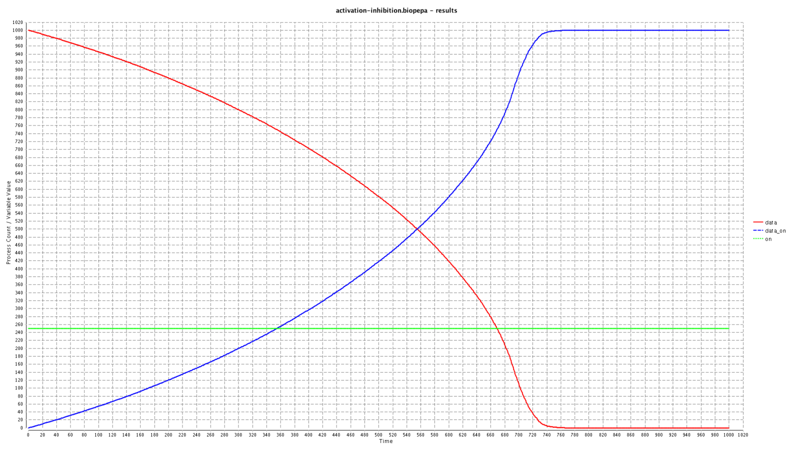
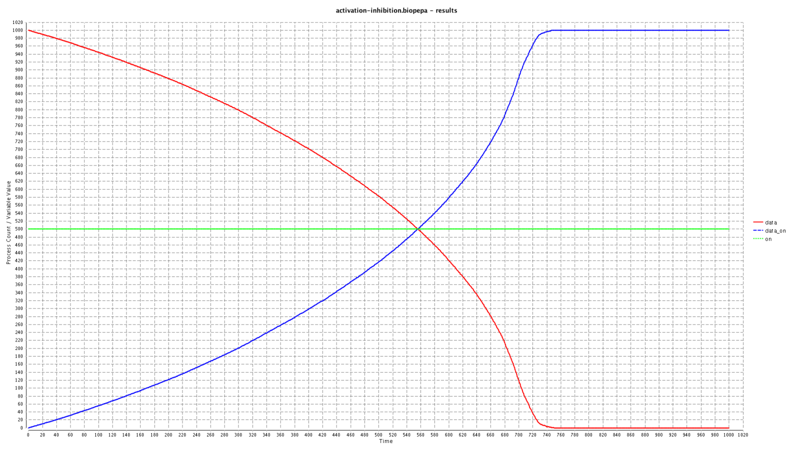


Figure 3.5: Activation chemical reaction with time and *data_on* -dependant kinetic rate [Mar14]. Besides the exhibited trend being completely different w.r.t. that of Figure 3.4, also sensitivity to parameters changes: increasing the activator enzyme quantity no longer affects the activation process time (middle plot has twice *on* than top plot, but crossing time step is still ≈ 600), whereas increasing the quantity of data to activate does (bottom plot has twice *data* than top plot, thus time taken until the crossing point increases to ≈ 1050). Acting on the rate constant speeds up or slows down the process as usual.





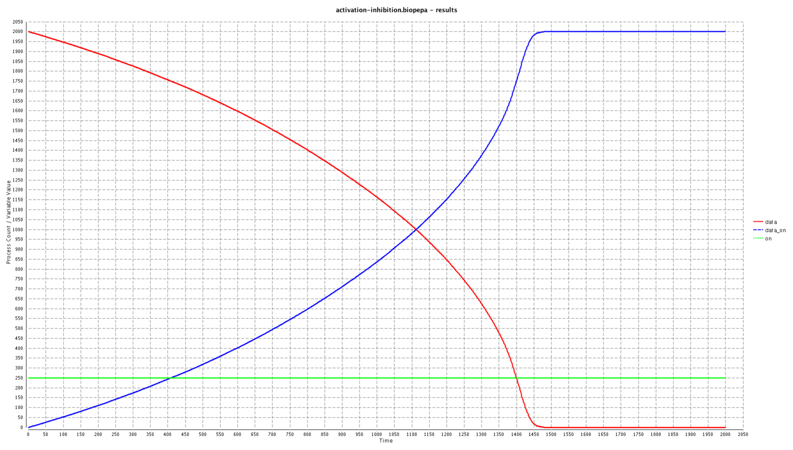


Figure 3.6: Activation chemical reaction with time and *data* -dependant kinetic rate [Mar14]. Whereas the emergent, global behaviour achieved is different from both previous ones, sensitivity to parameters is the same as for that in Figure 3.5: increasing the activator enzyme quantity does not affect the activation process time (middle plot has twice *on* than top plot, but crossing time step is still $\approx 560$), whereas increasing the quantity of data to activate does (bottom plot has twice *data* than top plot, thus time taken until the crossing point increases to over 1100). Acting on the rate constant speeds up or slows down the process as usual.

It should be noted that direct proportionality is w.r.t. time taken to complete the activation process, not to the kinetic rate dynamically computed. In fact, being *data* the denominator of the rate expression, starting the process with a higher value implies term `time` is divided by a greater number, thus the factor added to the fMA factor is smaller, likewise the whole kinetic rate.

Furthermore, denominator *data* is slowly decreasing due to conversion to *data_on*, while `time` is increasing: this leads to a numerator increasing and a denominator decreasing, which explains the faster activation after some time has passed.

**Aggregation** The aggregation pattern for self-organisation can be represented as an artificial chemical reaction as follows:

$$part^1 + part^2 \xrightarrow{r_{aggregation}} whole$$

meaning that if both parts composing a whole are available, those parts ($part^1$ and $part^2$) should be consumed to produce *whole*, at a pace given by $r_{aggregation}$—the local mechanism.

Aggregation reaction can then be encoded in BioPEPA according to three different kinetic rates – usual fMA-only, time and $\frac{1}{whole}$ dependency, time and $\frac{1}{part^1}$ dependency ($part^2$ will be identical) –, as follows:

```
AGGREGATION_CONSTANT = 0.0005;
r_aggregation = [fMA(AGGREGATION_CONSTANT)];
part1 = (r_aggregation, 1) <<;
part2 = (r_aggregation, 1) <<;
whole = (r_aggregation, 1) >>;
```

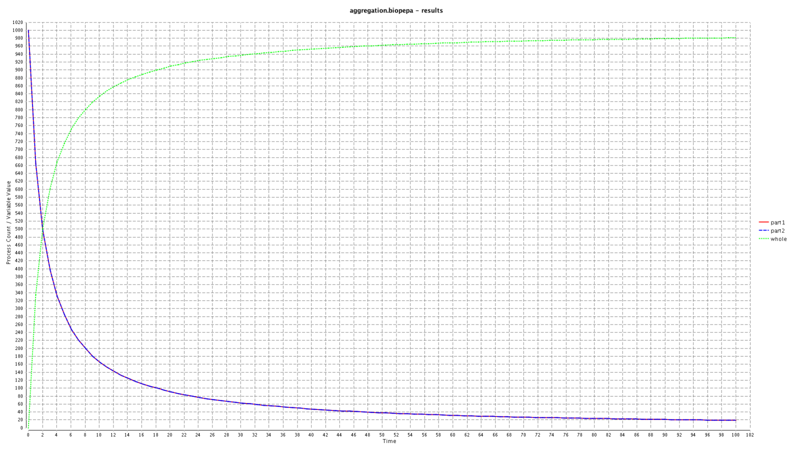
```
r_aggregation = [fMA(AGGREGATION_CONSTANT) +
                   H(part1) * H(part2) * time/whole];
```

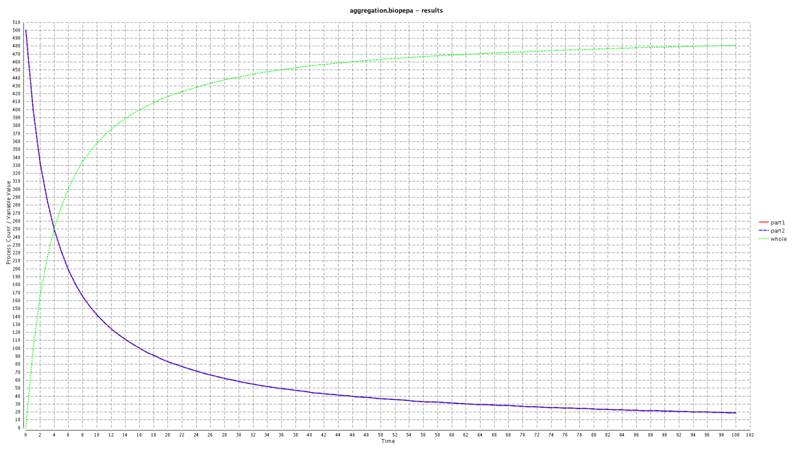
```
r_aggregation = [fMA(AGGREGATION_CONSTANT) +
                   H(part1) * H(part2) * time/part1];
```

Simulations of the above custom kinetic rates under different settings are depicted, respectively, in Figure 3.7, Figure 3.8, and Figure 3.9. As shown, the exhibited trends are almost identical to those of the activation pattern; nevertheless, sensitivity to reactant change is not always in line with those results.

Figure 3.7 shows the same trend as Figure 3.4, but the effect of increasing (decreasing) quantity of reactant *data* is counter-intuitive: while in Figure 3.4 timing was independent of *data*, and in subsequent plots of Figure 3.5 and Figure 3.6 dependency had the form of a direct proportionality, here increasing (decreasing) *data* decreases (increases) time taken to aggregate parts into *whole*. This is the first time that inverse proportionality of time w.r.t. the quantity of the reactant involved in the reaction is experienced.

As far as Figure 3.8 and Figure 3.9 are concerned, both the exhibited trend and the local mechanisms' parameters effect on the emergent, global behaviour achieved is the same as that of Figure 3.5 and Figure 3.6, respectively.





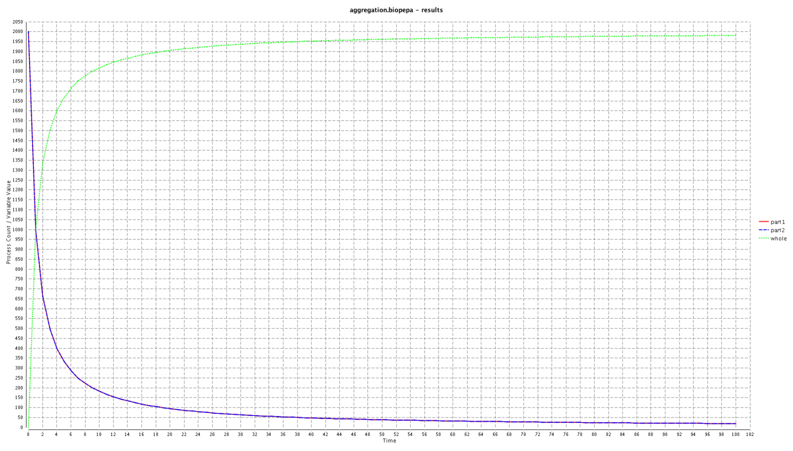


Figure 3.7: Aggregation chemical reaction with usual law of mass action rate [Mar14]. In the middle plot the two reactants $part^1$ and $part^2$ are half w.r.t. the top plot, causing a slower aggregation process (time step 2 in top plot, time step 4 in middle plot). In the bottom plot instead, their quantity is twice that of the top plot (2000 units), thus aggregation time faster (half time w.r.t. top plot). Acting on the rate constant speeds up or slows down the process as usual.

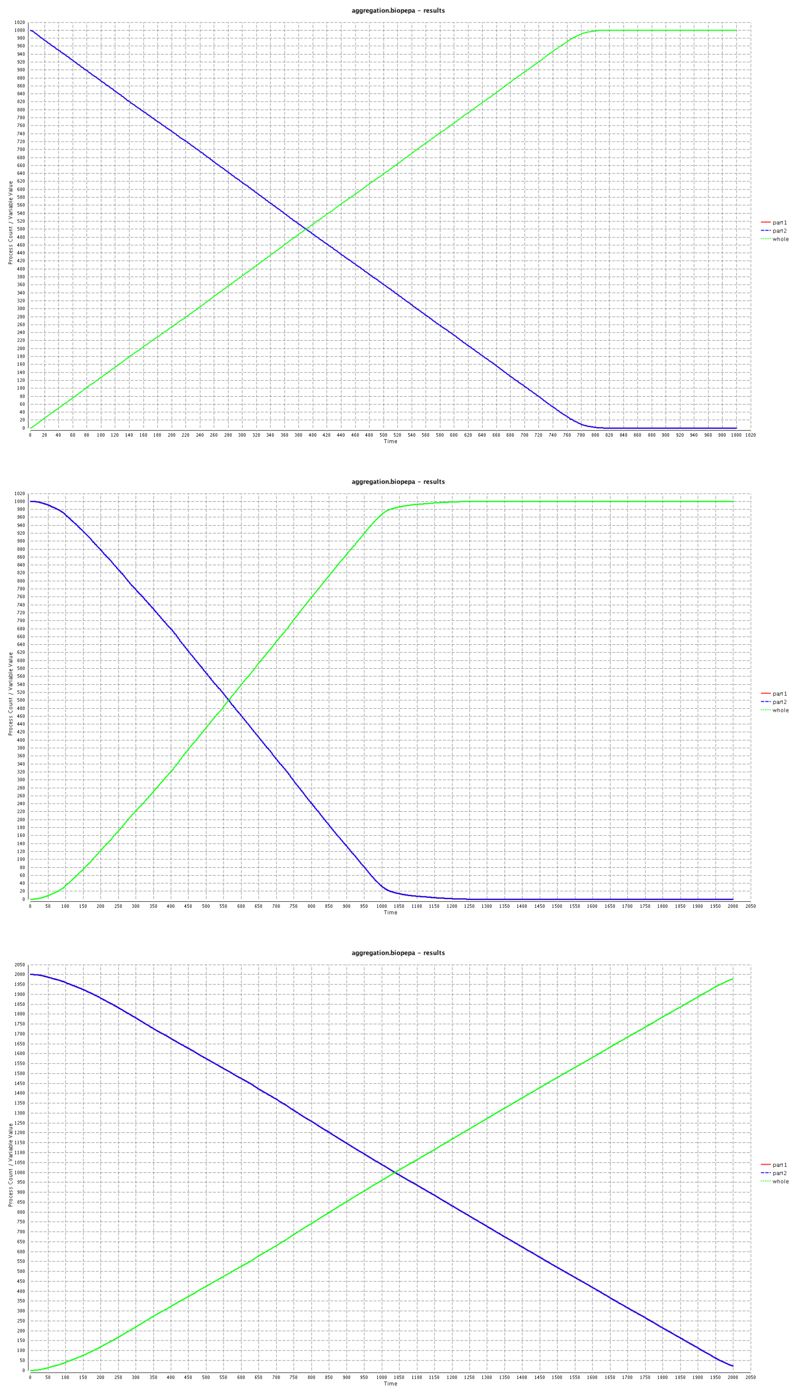


Figure 3.8: Aggregation chemical reaction with $\frac{time}{whole}$-dependant kinetic rate [Mar14]. Besides the emergent, global trend exhibited, also sensitivity to parameters is very different w.r.t. Figure 3.7: now increasing the reactants quantity (bottom plot has twice parts than middle plot) increases time taken to complete aggregation (from slightly more than 550 time steps in middle plot, to almost 1050 in bottom plot). Instead, acting on the rate constant speeds up or slows down the process as usual (top plot has rate constant 0.5 with crossing point at $\approx$ 390 time steps, whereas middle plot rate constant is 0.005 thus crossing point gets delayed to over 550 time steps).

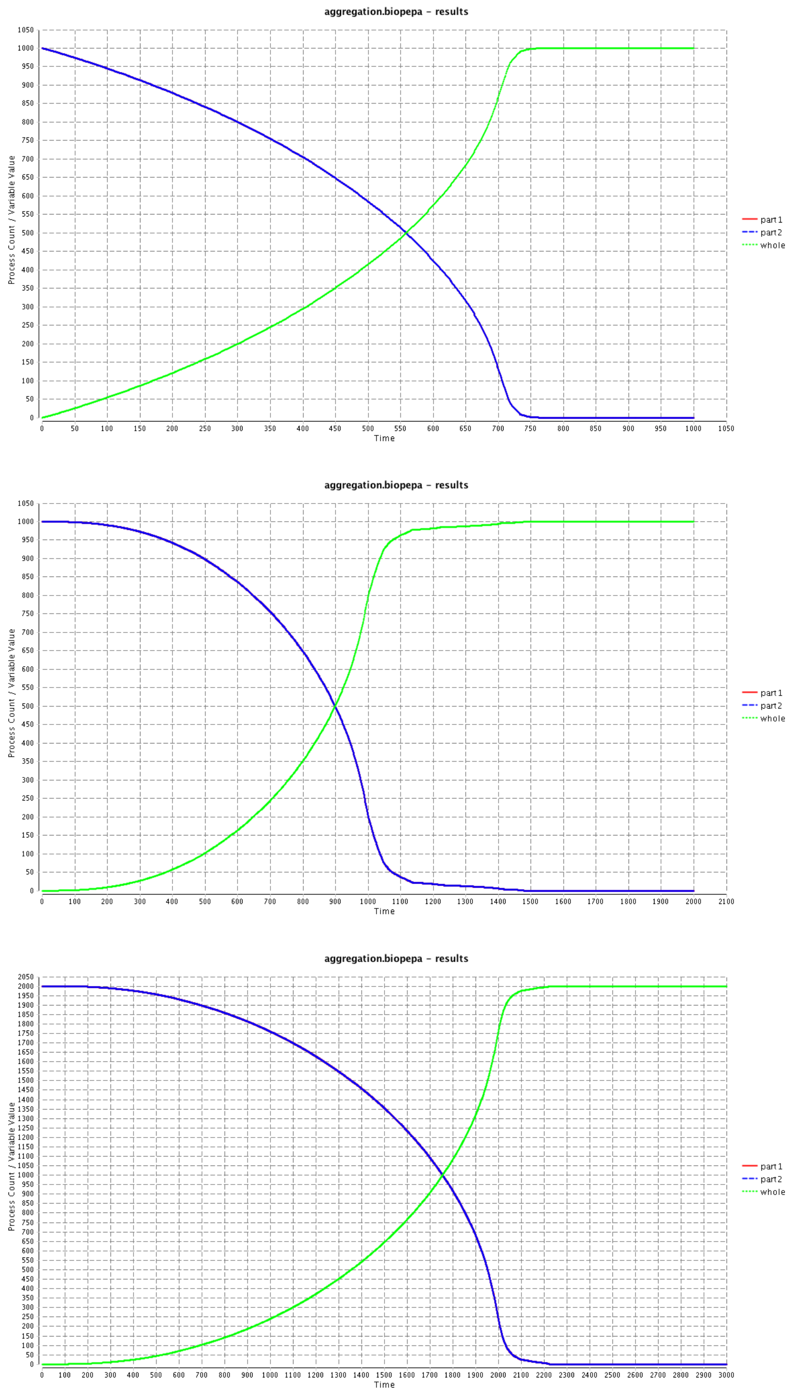


Figure 3.9: Aggregation chemical reaction with $\frac{time}{part^1}$-dependant kinetic rate [Mar14]. Although the behaviour shown is yet again different from both previous ones, parameters of the local mechanism (the aggregation reaction) have the same effect on the emergent, self-organising behaviour achieved, as described for Figure 3.8. In particular, middle plot has a slower rate than top plot, thus time scale almost doubles, and bottom plot has twice parts than middle plot, thus, again, time scale almost doubles.

This is another consequence of the nice controllability property of custom kinetic rates in artificial chemical reactions: being the role played by *whole* in the aggregation pattern the same as that played by *data_on* in activation – that is, *products* of the reaction – as well as the role played by $part^1$ and $part^2$ in aggregation the same as that played by *data* in activation – that is, *reactants* – it is reasonable to expect for them both the same global behaviour to emerge and the same sensitivity to parameters.

**Diffusion** The diffusion pattern for self-organisation can be represented as an artificial chemical reaction as follows:

$$data_{c'} \xrightarrow{r_{diffusion}} data_{c''} \mid data_{c'''}$$

meaning that one piece of *data* is withdrawn from compartment $c'$ and put into either compartment $c''$ or $c'''$, chosen probabilistically[4], at a pace given by $r_{diffusion}$—the local mechanism.

Diffusion reaction can be encoded in BioPEPA according, e.g., to the following kinetic rates – usual fMA, equal-distribution driven by source compartment, dual to fMA, and equal-distribution driven by destination compartment –, as follows:

```
// topology definition (c2<---c1--->c3)
location c1 : size = 1, type = compartment;
location c2 : size = 1, type = compartment;
location c3 : size = 1, type = compartment;

diff_const = 0.0005;
// usual fMA
r_diffusion = [fMA(diff_const)];
data = (r_diffusion[c1->c2], 1) (.) data + (r_diffusion[c1->c3], 1) (.) data;
```

```
// equal-distribution driven by source compartment
r_diffusion = [fMA(diff_const) * (data@c1 - data@c2)];
```

```
// dual to fMA
r_diffusion = [fMA(diff_const) * (data@c2 + 1)];
```

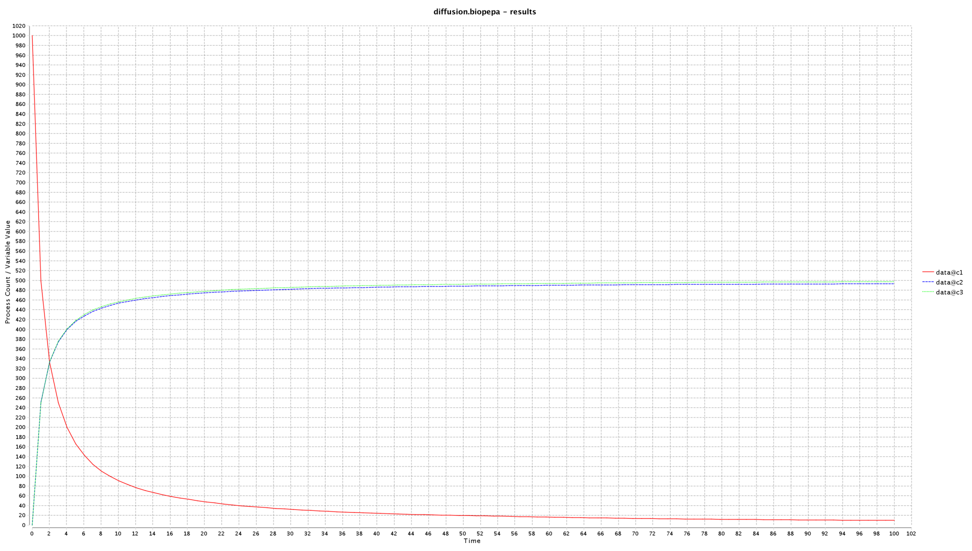
```
// equal-distribution driven by destination compartment
r_diffusion = [fMA(diff_const) * H(data@c1 - data@c2) * (data@c2 + 1)];
```

Simulations of the above custom kinetic rates are depicted in Figure 3.10 and Figure 3.11.

As shown, the fMA kinetic rate (Figure 3.10, top) exhibits a fast-then-slow global (emergent) trend, where *data* from its origin compartment is eventually depleted. The equal-distribution custom rate, instead, (Figure 3.10, bottom) does not deplete the concentration of *data* in its origin compartment, but asymptotically tends to balance the distribution of *data* items among the neighbouring compartments.

It should be noted that this emergent behaviour is driven by the source compartment in the sense that the process is faster the more *data* is concentrated in the origin compartment—in fact, it is fast-then-slow.

[4]How the destination compartment is chosen is not relevant here.



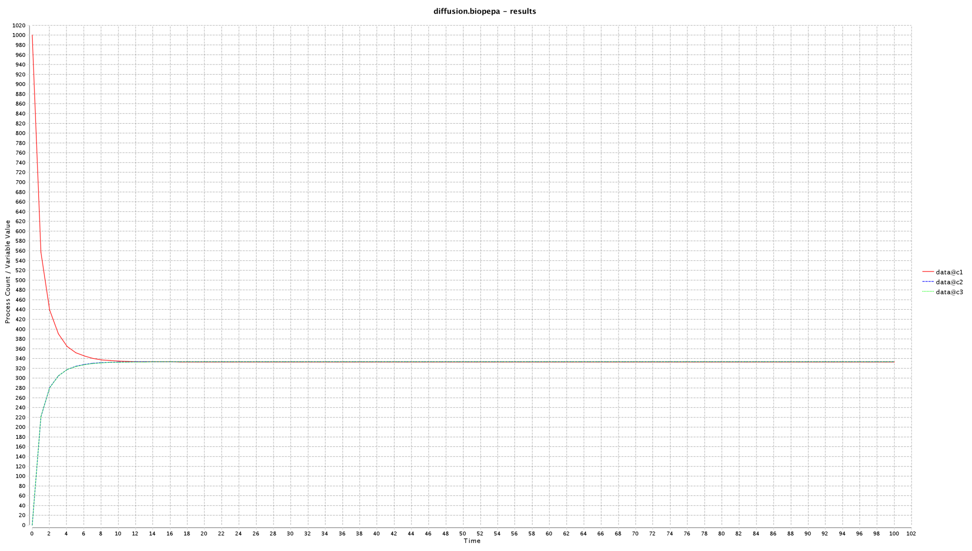
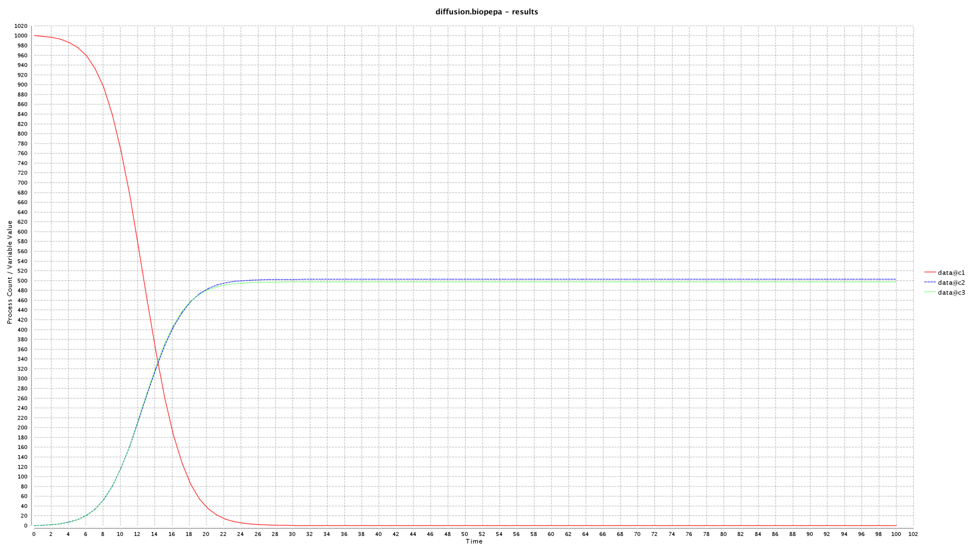


Figure 3.10: At the top, usual fMA diffusion. At the bottom, equal-distribution driven by source compartment.



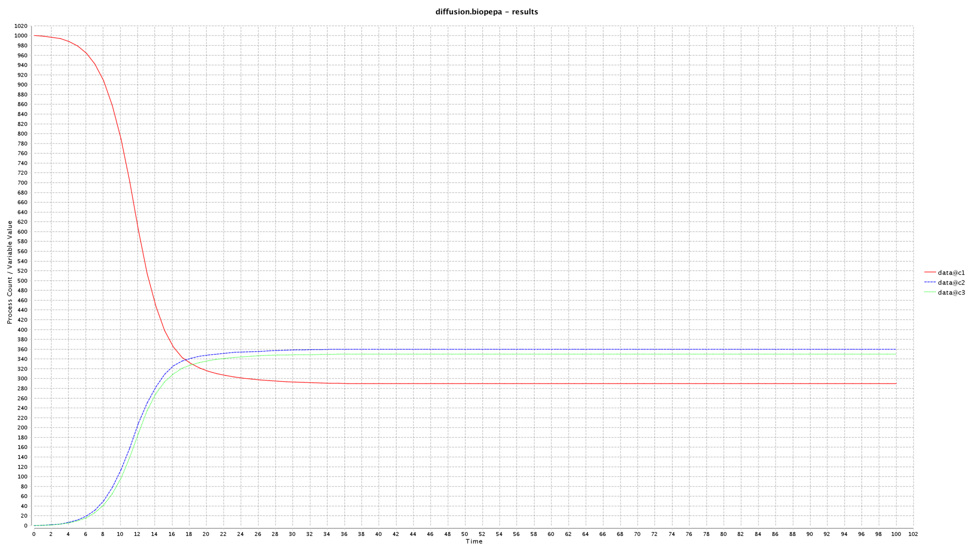


Figure 3.11: At the top, the exhibited trend is dual w.r.t. the fMA one. At the bottom, the exhibited trend is dual w.r.t. the one with equal distribution, and driven by concentration of *data* in the source compartment concentration.

In Figure 3.11, at the bottom, the same (almost) uniform distribution is achieved with the dual trend – that is, slow-then-fast – where what drives the emergent process is the concentration of *data* items in the destination compartment.

Finally, Figure 3.11, at the top, shows the dual trend w.r.t. to the fMA-only one (seen in Figure 3.10, top), that is, a slow-then-fast global behaviour.

### 3.1.3 Discussion of Results

In this section, it has been shown that simply imitating nature *as is* may be not the optimal approach while engineering nature-inspired self-organising systems. Indeed, once a suitable natural metaphor is found, system designers should ask themselves if the natural system's parameters are the optimal ones also for the artificial system they aim to build.

If it is not the case, they should clearly state which goals their system is pursuing, then detect, preferably within the system itself, which parameters better suit their needs as well as which (if any) functional dependencies between parameters better cope with the problem their system aims to solve.

Accordingly, presented a novel approach in dealing with the local vs. global issue in engineering self-organising systems was presented, composed of three ingredients: model self-organisation primitives as *artificial chemical reactions*, design *custom kinetic rates*, adjust rates' parameters according to the emergent, global behaviour desired.

In particular, w.r.t. to the latter ingredient,it has been shown that the *law of mass action* [Car08] may be not enough to effectively engineer kinetic rates for biochemical coordination, and how the factors chosen for the custom kinetic rate expressions as well as the value of the parameters involved have a well-defined, *controllable* effect on the global behaviour achieved. This is made possible by adoption of the chemical reaction metaphor to implement self-organisation primitives, and helps facing the local vs. global issue—ultimately leading to a better engineering of self-organising behaviours.

The work starts with a rather simple observation: the fact that a given natural system – in this case, chemical solutions – works properly by relying on a given set of parameters – e.g., concentration, rate, stoichiometry, etc. –, each of which having a given set of functional dependencies with others – e.g., the law of mass action –, does not necessarily mean that the same sets of parameters and functional dependencies will work for an artificial system drawing inspiration from the natural one.

Similar considerations can be done for other natural metaphors: most notably, the ant colony optimisation approach to distributed optimisation, in which the original ant colony metaphor is indeed just a metaphor, not the actual implementation [DB10].

In order to do so, many experiments were done: *(i)* which are the desiderata for the self-organising system behaviour is defined, *(ii)* rates are *engineered* by designing functional dependencies likely to pursue the chosen goal, *(iii)* a pure parameter tuning stage is included to fine-tune the system behaviour (when needed). All of this has been done one reaction (aka coordination law) at a time, thus one functional rate at a time, incrementally accumulated until composing the whole self-organising system behaviour.

## 3.2 Uniform Primitives as Coordination Primitives

A foremost feature common to all the tuple-based coordination models mentioned in Chapter 1 is *non-determinism*, stemming from their roots in the LINDA model [Gel85]. LINDA features *don't know* non-determinism in the access to tuples in tuple spaces, handled with a *don't care* approach: (i) a tuple space is a multiset of tuples where multiple tuples possibly match a given template; (ii) which tuple among the matching ones is actually retrieved by a getter operation (e.g., `in`, `rd`) can be neither specified nor predicted (*don't know*); (iii) nonetheless, the coordinated system is designed so as to keep on working safely whichever is the matching tuple returned (*don't care*).

The latter assumption requires that when a process uses a template matching multiple tuples, which specific tuple is actually retrieved is not relevant for that process. This is not the case, however, in many of today adaptive and self-organising systems, where processes may need to implement *stochastic behaviours* – like "most of the time do this" – which obviously do not cope well with don't know non-determinism [OV11].

For instance, all the nature-inspired models and systems emerged in the last decade – such as chemical, biochemical, stigmergic, and field-based – are examples of the broad class of self-organising systems that precisely require such a sort of behaviour [Omi13a], which by no means can be enabled by the canonical LINDA model and its direct derivatives.

To this end, in the following *uniform coordination primitives* (`uin`, `urd`) – first mentioned in [GVCO07] – are discussed, as the specialisation of LINDA getter primitives featuring *probabilistic non-determinism* instead of don't know non-determinism.

Roughly speaking, uniform primitives make it possible to both *specify* and (*statistically*) *predict* the probability to retrieve one specific tuple in a multiset of matching ones, thus making it possible to statistically control non-deterministic systems. This simple mechanism extends the reach of tuple-based coordination towards nature-inspired systems, allowing, e.g., coordination-based simulation of complex stochastic behaviours.

It should be noted that uniform primitives are the basic mechanism upon which the $\mathcal{MoK}$ middleware prototype discussed in Part II of this thesis is designed, in particular, as regards artificial chemical reactions scheduling, and reactants consumption.

### 3.2.1 Related Approaches

In [POS13], the authors adopt a simulation approach based on *biochemical tuple spaces* [VC09]. Technically, biochemical tuple spaces are built as ReSpecT *tuple centres* [OD01b], distributed across the TuCSoN coordination infrastructure [OZ99]. Tuples are logic-based tuples, while biochemical laws are implemented as ReSpecT *specification tuples*, so that they can be inserted, modified, and removed from the biochemical compartment (the tuple centre) via ReSpecT coordination primitives.

A *chemical solution simulator* is successfully built as a ReSpecT tuple centre, programmed to implement the Gillespie's algorithm for *exact* chemical solutions simulation [Gil77]. Then, the Ras-mitogen-activated protein kinase (MAPK) signaling pathway is simulated, but uniform primitives are not exploited: thus, e.g., the peculiar feature of probabilistic, concentration-driven selection of reactants is lacking. In Subsection 3.2.3 the implementation is re-designed on top of uniform primitives, to demonstrate their impact on system behaviour in spite of their simplicity—namely, real probabilistic selection of reactants based on an approximation of concentration values.

The idea of engineering nature-inspired self-organising systems attracted many researchers in the last decade: [Nag04, MMTZ06, GVCO07, DWH07] all proposed either bio-inspired coordination primitives or full-fledged design patterns aimed at easing the understanding and engineering of nature-inspired computational systems.

However, a step beyond primitives and patterns definition is taken in [FMMSM$^+$12], by classifying patterns into *layers*, and by describing the *relationships* between patterns in different layers, so as to effectively outline a pattern composition schema suitable to be reused in different scenarios. Figure 3.12 below depicts the aforementioned layers:

**basic patterns** *repulsion*, *evaporation*, *aggregation*, and *spreading*, are interpreted as the most primitive nature-inspired patterns which all other patterns should be programmed upon, according to the composition represented by the arrows (dashed for optional composition, solid for mandatory)

**composed patterns** *digital pheromone*, *gradients*, and *gossip*, are in the middle layer, because (i) being programmable as a composition of other patterns, they cannot be basic, and (ii) being exploitable to engineer other patterns by composition, they are not high level either

**high level patterns** *flocking*, *foraging*, *quorum sensing*, *chemotaxis*, and *morphogene-*

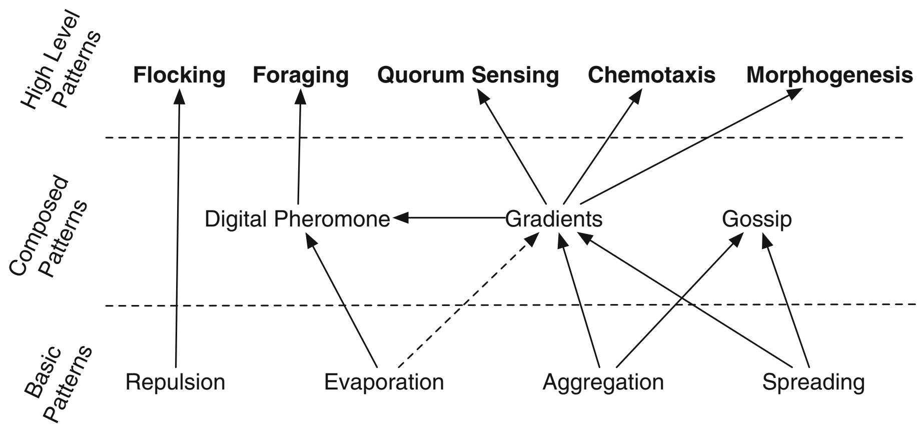


Figure 3.12: Patterns layering and relationships (image taken from [FMMSM$^+$12]).

*sis*, are all defined as high-level patterns: thus, as patterns directly usable at the application level, e.g., to implement self-organising coordination policies

In Subsection 3.2.3 it is shown how uniform primitives could play a fundamental role in implementing digital pheromones and foraging.

Uniform primitives are used in [CVG09] as a tool for solving a specific coordination problem, called *collective sort*. In [VC09] similar primitives are presented and formally defined to forge the *biochemical tuple space* notion.

The main difference w.r.t. the approach with uniform primitives is that: (i) they rely on tuples multiplicity to model probability, leaving Linda tuples' structure untouched, (ii) they are executed as Linda getter primitives, whereas in [VC09] they are executed according to a *stochastic rate*.

**Differentiation** To the best of the author knowledge, probabilistic extensions to Linda follow two main approaches [DPHW05]:

- *data-driven*, where the quantitative information required to model probability is associated with the data items (the tuples) in the form of weights, then the matching mechanism is extended so as to take into account *weights*—transforming them into probabilities by normalisation. This approach is adopted, e.g., in `ProbLinCa` [BGLZ05], the probabilistic version of a Linda-based process calculus

- *scheduler-driven*, where quantitative information is added to the processes using special probabilistic schedulers, ascribing different probabilities to different active processes—independently of the tuples involved. This is the approach taken by, e.g., [DPHW04], to define a probabilistic extension of the Klaim model named pKlaim

Instead, the approach just described belongs to a third, novel category, called *interaction-driven*, where probabilistic behaviour is (i) associated to communication primitives – thus, neither to processes (or schedulers), nor to tuples – and (ii) enacted during the interaction between a process and the coordination medium—that is, solely through primitives.

In fact, whereas `ProbLinCa` requires the modification of the tuple structure and the matching mechanism so as to support probabilistic operations, and pKlaim requires special, probabilistic schedulers for processes, uniform primitives extend Linda by *specialising* standard Linda primitives, without changing neither the tuple structure nor the scheduling policy.

Furthermore, uniform primitives could be used to emulate both approaches: tuple weights could be reified by their multiplicity in the space, whereas probabilistic scheduling could be obtained by properly synchronising processes upon probabilistic consumption of shared tuples.

Moreover, uniform coordination primitives could be used in place of Linda standard ones without affecting the model, simply refining don't care non-determinism as probabilistic non-determinism: as a result, all the expressiveness results and all the applications based on the canonical Linda model would still hold.

More complex coordination models exist where uniform primitives could potentially play a key role in providing the probabilistic mechanisms required for the engineering of stochastic systems like adaptive and self-organising ones.

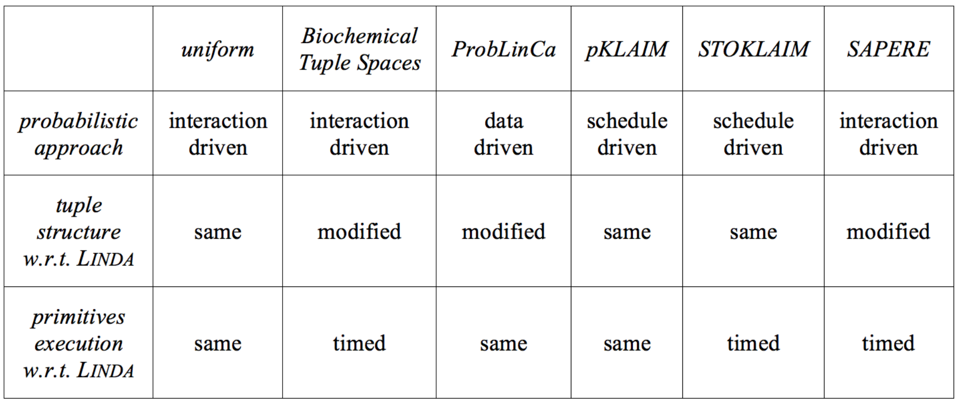
StoKlaim [DNLKM06] is an extension to Klaim where process actions are equipped with *rates* affecting execution probability, and execution *delays* as well—that is, time needed to carry out an action. By reifying action rates as tuples in the space, with multiplicity proportional to rates, uniform-reading tuples would allow action execution to be probabilistically scheduled *à la* StoKlaim. Furthermore, delays could be emulated, too, by uniform-reading a set of time tuples, where a higher value corresponds to a lower action rate.

In SAPERE [ZCF+11], tuples are managed through *eco-laws*, which are a sort of chemical-like rules, scheduled according to their rates. Hence, uniform primitives could play in SAPERE the same role as in StoKlaim—once eco-laws are reified as tuples with a multiplicity proportional to execution rate. Furthermore, from the pool of all the tuples which can participate in a eco-law, the ones actually consumed by the law – as *chemical reactants* – are selected probabilistically. Once again, this kind of behaviour could be enabled by uniform consumption of reactant tuples in eco-laws.

Figure 3.13 below summarises the main differences between uniform primitives and the other aforementioned primitives.

### 3.2.2 Informal Definition

Linda getter primitives, that is, data-retrieval primitives `in` and `rd`, are shared by all tuple-based coordination models, and provide them with *don't know non-determinism*: when one or more tuples in a tuple space match a given template, any of the matching



| | *uniform* | *Biochemical Tuple Spaces* | *ProbLinCa* | *pKLAIM* | *STOKLAIM* | *SAPERE* |
|---|---|---|---|---|---|---|
| *probabilistic approach* | interaction driven | interaction driven | data driven | schedule driven | schedule driven | interaction driven |
| *tuple structure w.r.t. LINDA* | same | modified | modified | same | same | modified |
| *primitives execution w.r.t. LINDA* | same | timed | same | same | timed | timed |

Figure 3.13: Comparison of probabilistic coordination models [MO14b].

tuples can be non-deterministically returned.

Therefore, in a single getter operation, only a *point-wise property* affects tuple retrieval: that is, the conformance of a tuple to the template, independently of the *spatial context*—namely, the other tuples in the same space.

Furthermore, in a sequence of getter operations, don't know non-determinism makes any prediction of the overall behaviour impossible: e.g., reading one thousand times with the same template in a tuple space with ten matching tuples could possibly lead to retrieve the same tuple all times, or one hundred times each, or whatever admissible combination one could think of—no prediction possible, according to the model.

Again, then, only a point-wise property can be ensured even in *time*: that is, only the mere compliance to the model of each individual operation in the sequence.

*Uniform primitives* instead, enrich tuple-based coordination models with the ability of performing operations that ensure *global system properties* instead of point-wise ones, both in space and in time.

More precisely, uniform primitives replace don't know non-determinism with *probabilistic non-determinism* to *situate* a primitive invocation in *space* – the tuple actually retrieved depends on the other tuples in the space – and to *predict* its behaviour in *time*—*statistically*, the distribution of the tuples retrieved will tend to be *uniform*.

The main motivation behind the formal definition and expressiveness study of uniform primitives, is that of introducing a *simple* yet *expressive* probabilistic mechanism in tuple-based coordination: simple enough to work as a specialisation of standard Linda operations, expressive enough to allow modelling of the most relevant probabilistic behaviours exhibited by nature-inspired complex computational systems.

Whereas *expressiveness* is discussed in Section 4, *simplicity* is achieved by defining uniform primitives as *specialised* versions of standard Linda primitives: so, first of all, `uin` and `urd` are compliant with the standard semantics of `in` and `rd`.

In the same way as `in` and `rd`, `uin` and `urd` ask tuple spaces for one tuple matching a given template, possibly suspend when no matching tuple is available, and finally return a matching tuple chosen non-deterministically when one or more matching tuples are available in the tuple space. As a straightforward consequence, any tuple-based coordination system using `in` and `rd` would also work by exploiting instead `uin` and `urd`, respectively—and any process using `in` and `rd` could adopt `uin` and `urd` instead without any further change.

On the other hand, the nature of the specialisation lays precisely in the way in which a tuple is non-deterministically chosen among the (possibly) many tuples matching the template. While in standard Linda the choice is performed based on don't know non-determinism, uniform primitives exploit instead *probabilistic non-determinism* with *uniform distribution*. So, if a standard getter primitive requires a tuple with template $T$, and $m$ tuples $t_1, \ldots, t_m$ matching $T$ are in the tuple space when the request is executed, any tuple $t_{i \in 1 \ldots m}$ could be retrieved, but nothing more could be said—no other assertion is possible about the result of the getter operation.

Instead, when a uniform getter primitive requires a tuple with a template $T$, and $m$ tuples $t_1, \ldots, t_m$ matching $T$ are available in the tuple space when the request is served, one assertion is possible about the result of the getter operation: each of the $m$ matching tuples $t_1, \ldots, t_m$ has exactly the *same probability* $\frac{1}{m}$ to be returned. So, for instance, if 2 `colour(blue)` and 3 `colour(red)` tuples occur in the tuple space when a `urd(colour(X))` is executed, the probability of the tuple retrieved to be `colour(blue)` or `colour(red)` is exactly 40% or 60%, respectively.

Operationally, uniform primitives behave as follows. When executed, a uniform primitive takes a *snapshot* of the tuple space, freezing its state at a certain point in time—and space, being a single tuple space the target of basic Linda primitives. The snapshot is then exploited to assign a probability value $p_i \in [0, 1]$ to any tuple $t_{i \in 1 \ldots n}$ in the space—where $n$ is the number of tuples in the space. There, non-matching tuples have value $p = 0$, matching tuples have value $p = \frac{1}{m}$ (where $m \leq n$ is the number of matching tuples), and the sum of probability values is $\sum_{i \in 1 \ldots n} p_i = 1$. The matching tuple returned is *statistically* chosen based on the probability values computed.

As a consequence, whereas standard getter primitives exhibit point-wise properties only, uniform primitives feature *global properties*, both in space and time.

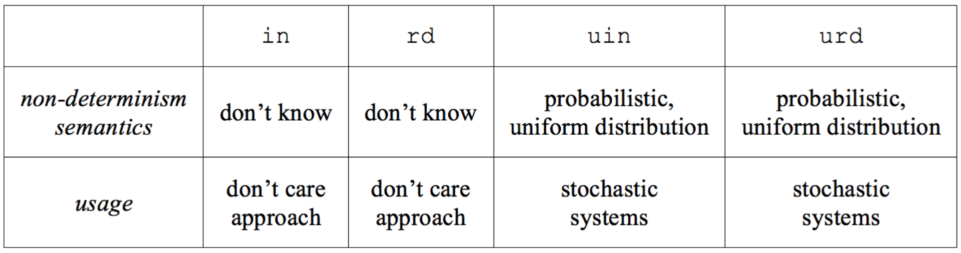
In terms of spatial context, in fact, standard getter primitives return a matching tuple independently of the other tuples currently in the same space—so, they are *context unaware*. Instead, uniform getter primitives return matching tuples based on the overall state of the tuple space—so, their behavior is *context aware*.

In terms of time, too, sequences of standard getter operations feature no meaningful properties. Instead, by definition, sequences of uniform getter operations tend to globally exhibit a *uniform distribution* over time. So, for instance, performing $N$ `urd(colour(X))` operations over a tuple space containing 10 `colour(white)` and 100 `colour(black)` tuples, would lead to a sequence of returned tuples which, while growing, tends to contain 10 times more `colour(black)` tuples than `colour(white)` ones.

In principle, the snapshot behaviour above described may be computationally expensive, and potentially represent an implementation bottleneck: given the tuple space target of the uniform primitive, the set of matching tuples should be found, then each different matching tuple should be counted, finally one of those matching tuples chosen probabilistically. Furthermore, in the meanwhile no other primitive can be served by the tuple space, according to Linda semantics.

Nevertheless, many techniques can be used to effectively deal with the issue: for instance, (i) using suitable data structures—e.g., an hashmap to conveniently store together identical copies of the same tuple; (ii) distributing tuples among several networked tuple spaces, so as to lower the load of tuples to snapshot for each one; (iii) tracking the number of tuples internally, so as to avoid counting.

Figure 3.14 below compares execution semantics and expected usage of uniform primitives w.r.t. Linda primitives—`out` is left out because its semantics is unchanged.



| | `in` | `rd` | `uin` | `urd` |
|---|---|---|---|---|
| *non-determinism semantics* | don't know | don't know | probabilistic, uniform distribution | probabilistic, uniform distribution |
| *usage* | don't care approach | don't care approach | stochastic systems | stochastic systems |

Figure 3.14: Comparison of LINDA and uLINDA primitives [MO14b].

### 3.2.3 Informal Expressiveness

In [BGLZ05] the authors demonstrate that LINDA-based languages cannot implement probabilistic models: a LINDA process calculus, although Turing-complete, is not expressive enough to express probabilistic choice.

Since formally asserting a gap in expressiveness does not necessarily make it easy to fully appreciate how much this can make the difference when programming, e.g., adaptive and self-organising systems, in the remainder of this section a few examples are discussed, showing which kind of behaviours are straightforwardly enabled by uniform primitives.

**Load balancing** Two service providers are both offering the same service to clients through proper advertising tuples. The first is slower than the second, that is, it needs more time to process a request—modelling differences in, e.g., computational power.

Their working cycle is simple: a worker thread gets requests from a shared tuple space, then puts them in the bounded queue of the master thread (the actual service provider). The master thread continuously queries the queue looking for requests to serve: when one is found, it is served, then the master emits another advertising tuple; if none is found, the master does something else, then re-queries the queue—no advertising.

Decoupling enforced by the queue is useful to model the fact that service providers should not block on the space waiting for incoming requests, so as to be free of performing other jobs in the meanwhile—e.g., reporting, resource clean-up, etc. The queue is bounded to model, e.g., memory constraints.

It should be noted that this toy scenario is a simplified version of a multi-client/multi-server deployment that is quite common in web applications, distributed computing, service-oriented architectures, etc.

In this setting, clients (whose Java code is listed in Figure 3.15) search for available services first via `rd` primitive (Figure 3.16), then via `urd` (Figure 3.17) [5].

By using the `rd` primitive, clients *blindly commit* to the actual implementation of the LINDA model currently at hand. For instance, Figure 3.16 gives some hints about

[5]All charts values are the average of several runs of the scenario—e.g., value plotted at time step 60 is the average of the number of requests observable at time step 60 in a number of runs (actually, 100).

```java
LogicTuple templ;
while(!die){
  templ = LogicTuple.parse("ad(S)");
  // Pick a server probabilistically
  op = acc.urd(tid, templ, null);
  // Plain Linda version
  // op = acc.rd(tid, templ, null);
  if (op.isResultSuccess()) {
    service = op.getLogicTupleResult();
    // Submit request
    req = LogicTuple.parse(
      "req("+service.getArg(0)+","+reqID+")"
      );
    acc.out(tid, req, null);
  }
}
```

Figure 3.15: Java code of clients looking for services [MO14b].

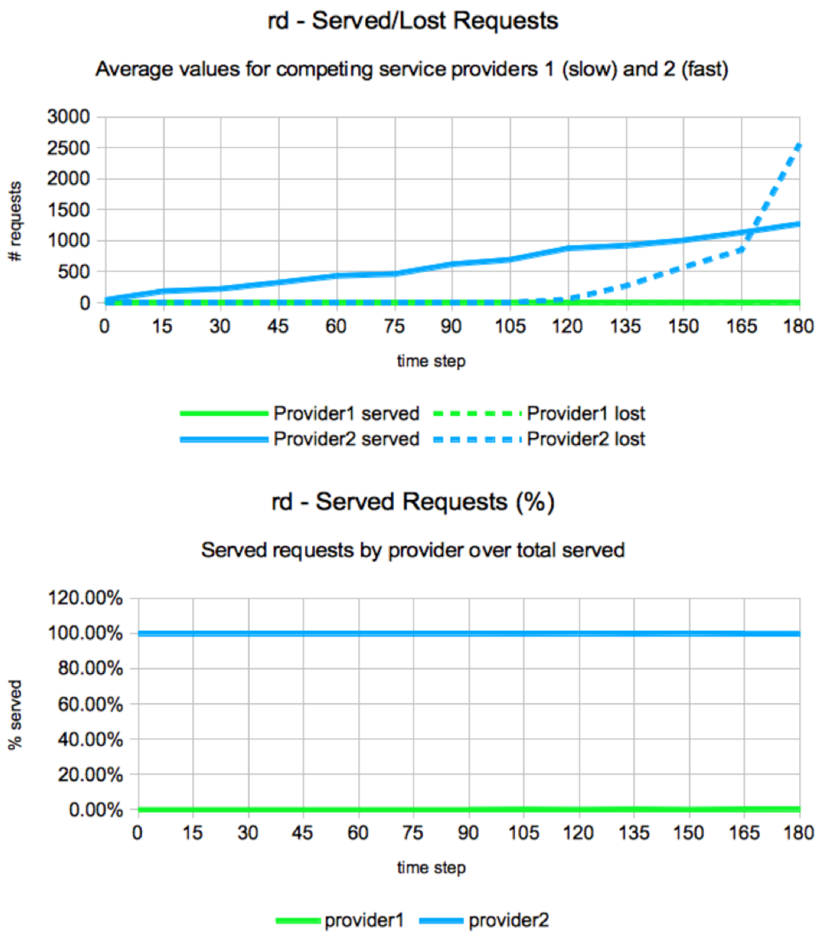


Figure 3.16: Clients using `rd` primitive: service provider 1 is under-exploited [MO14b].

the implementation used in this scenario—the TuCSoN coordination middleware [OZ99]: since provider 1 is almost unused, it may be inferred that `rd` is implemented as a FIFO queue, always matching the first tuple among many ones—provider 2 advertising tuple, in this case. The point here is that prediction is not possible before actually deploying the system, and with no information on the actual LINDA implementation used.

By using primitive `urd` instead (Figure 3.17), it is known – and *predictable* – how much each service provider will be exploited by clients: since it is known by design that after successfully serving a request a provider emits an advertising tuple, and that tuples are those looked up by clients, it is known that the faster provider will produce more tuples, hence it will be more frequently found, than the slower one.

Figure 3.17, in fact, shows how competing service providers self-organise by sharing in-

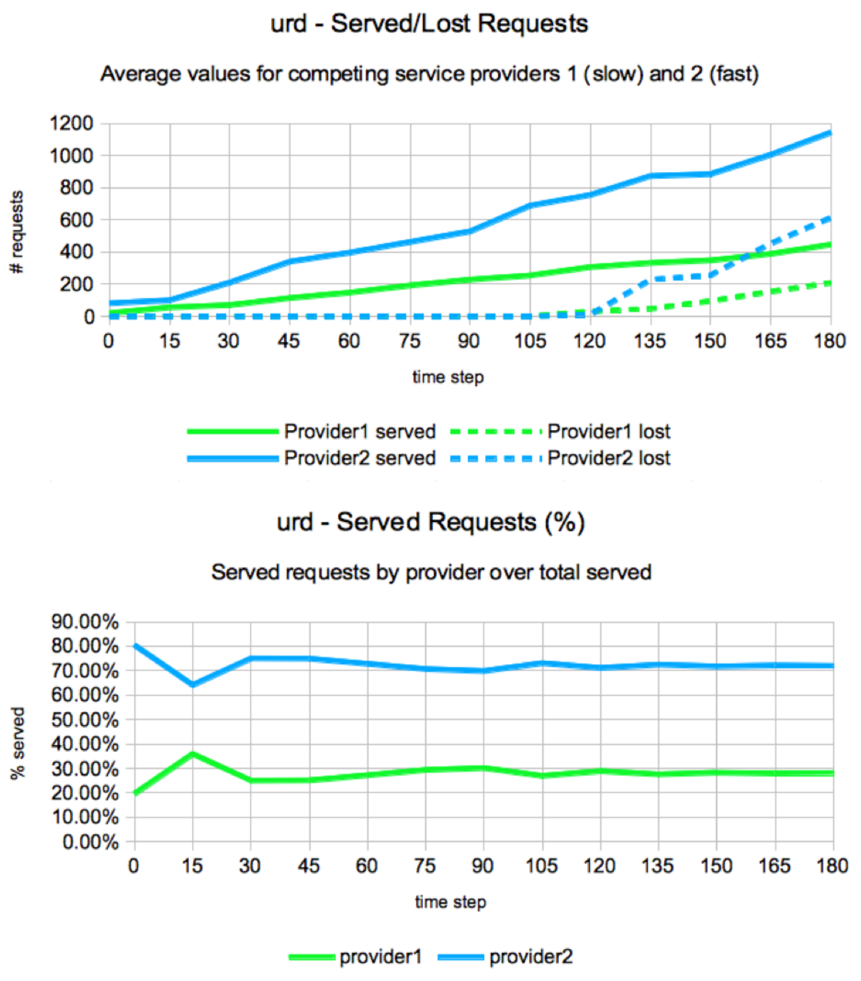


Figure 3.17: Clients using `urd` primitive: a certain degree of fairness is guaranteed, based on self-organisation [MO14b].

coming requests. Furthermore, this behaviour is not statically designed or superimposed, but results by *emergence* from a number of run-time factors, such as clients interactions, service providers computational load, computational power, and memory.

It should also be noted that this form of *load balancing* is not the only benefit gained when using `urd`: actually, the `urd` scenario successfully serves $\approx$ 1600 requests – distributed among providers 1 and 2 – losing $\approx$ 600, whereas the `rd` scenario serves successfully $\approx$ 1250 – leaving provider 1 unused – losing over 2500.

Although quite simple, the load balancing scenario just described can be taken as an example of that kind of complex systems where probabilistic mechanisms and feedback loops altogether make self-organisation appear by emergence [Omi13a].

Furthermore, even more traditional real-world scenarios – such as the aforementioned service-oriented and distributed ones –, could benefit from uniform primitives features to increase throughput, reliability, availability, and efficiency, as demonstrated by Figure 3.16 and Figure 3.17.

**Pheromone-based coordination** In *pheromone-based coordination*, used by ants to find optimal paths – as well as by many ant-inspired computational systems, such as network routing algorithms [DS04] and autonomous unmanned vehicles [PBS02] – each agent basically wanders *randomly* through a network of locations until it finds a pheromone trail, which the agent is *likely* to follow based on the trail strength.

Here, aspects such as pheromone release, scent, and evaporation are not relevant: instead, the above mentioned requirements of randomness and likelihood are, on the one hand, *essential* for pheromone-based coordination, while, on the other hand, *require* uniform primitives to be designed using a tuple-based coordination model.

In particular, consider a network of $n$ nodes representing places $p_i$, with $i = 1 \dots n$, through which ant agents walk. The default tuple space in node $p_i$ contains at least one *neighbour* tuple `n(`$p_j$`)` for each neighbour node $p_j$. The neighbourhood relation is symmetric—so, if node $p_i$ and $p_j$ are neighbours, tuple space $p_i$ contains tuple `n(`$p_j$`)` and tuple space $p_j$ contains tuple `n(`$p_i$`)`. Pheromone deposit in nodes is modelled by the insertion of a new tuple `n(`$p_i$`)` in every neighbour node $p_j$.

It should be noted that the described setting is common to a plethora of scenarios in computer science: essentially, any problem that can be represented by a graph-based structure. For instance, in the field of computer networks management, the following experiment could be taken as a reference for building much more complex applications, aimed at, e.g., routing of message packets, finding optimal paths from a source node to a destination one, reconfiguration of links between hubs, etc.

Engineering solutions to this kind of problems by using uniform primitives, instead of classical Linda operations, would bring all the benefits of self-organisation highlighted in the following discussion.

In the scenario just depicted, ants wandering through places and ants following trails can both be easily modelled using uniform primitives: ant agents just need to look locally for neighbour tuples through a `urd(n(P))`. If no pheromone trail is detected nearby, every neighbour place is represented by a single tuple, so all neighbour places have the same probability to be chosen—leading to random wandering of ants.

In case some of the neighbours contain a detectable trail, the corresponding neighbour tuple occurs more than once in the local tuple space: so, by using uniform primitives, the tuple corresponding to a neighbour place with a pheromone trail has a greater probability to be chosen than others.

The experiment was conducted in a toy scenario involving digital ants and pheromones programmed in ReSpecT [OD01a] upon the TuCSoN coordination middleware [OZ99]. The experiment involves ten digital ants starting from the anthill with the goal of finding food, and follows the canonical assumptions of ant systems [DS04].

So, at the beginning, any path has the same probability of being chosen, thus modelling random walking of ants in absence of pheromone. As ants begin to wander around, they eventually find food, and release pheromone on their path while coming back home. As a consequence, the shortest path finally gets most pheromone since using it takes less time w.r.t. any other path.

Pheromones as well as connections between tuple centres are modelled as described above, with neighbour tuples: the more neighbour tuples of a certain type, the more likely ants will move to that neighbour tuple centre next.

Figure 3.18 below depicts a few screenshots of the experiment: there, five distributed tuple centres (the large boxes) model a topology connecting the anthill to a food source: the leftmost path is longer – modelled as a 2-hop step – whereas the rightmost is shorter. The green spray-like effect on paths (black lines) models the strength of the pheromone scent: the greater and greener the path, the more pheromone on it.

By plotting pheromones strength evolution over time, Figure 3.19 shows how expectations about digital ants behaviour are met: in fact, despite starting from the situation in which any path is equiprobable (the amount of pheromones on either path is the same), the system eventually detects the shortest path, which becomes the most exploited—and contains in fact more pheromone units. In the Java code describing the behaviour of ants (Figure 3.20), in particular in method `smellPheromone()` (line 10), the usage of the uniform primitive `urd` is visible on line 27, whereas line 29 shows the tuple template given as its argument, that is, `n(NBR)`: at runtime, `NBR` unifies with a TuCSoN tuple centre identifier, making it possible for the ant to move there.

Quite obviously, the idea here is not just showing a new way to model ant-like systems. Instead, the example above is meant to point out how a non-trivial behaviour – that is, dynamically solving a shortest path problem – can be achieved by simply substituting uniform primitives to traditional LINDA getter primitives—which instead would not allow the system to work as required. Furthermore, the solution is adaptive, fully distributed, based upon local information solely – thus, it appears by *emergence* – and robust against topology changes—a ReSpecT specification implementing evaporation was used.

These are the main reasons why similar algorithms inspired by ants behaviour are so

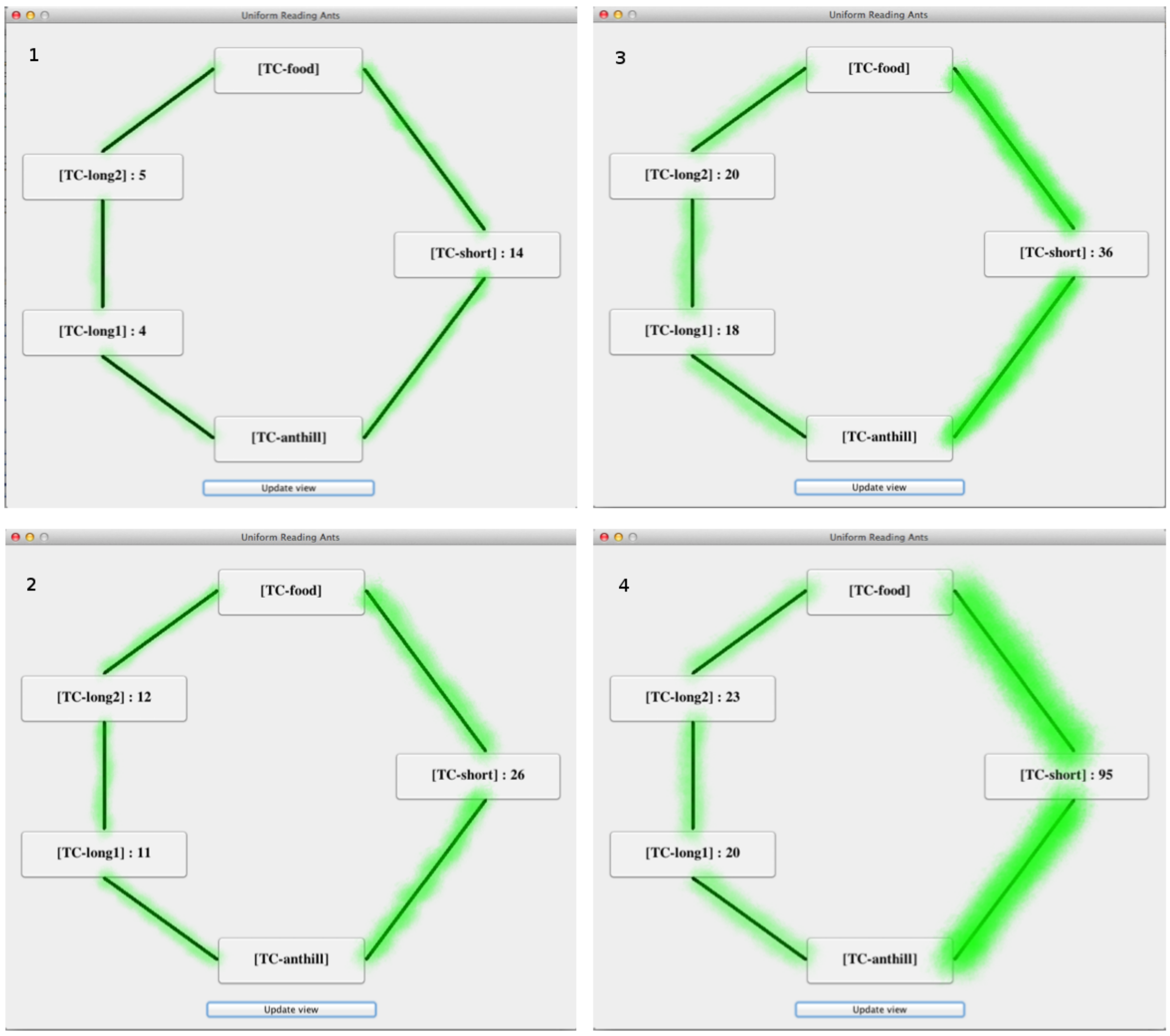


Figure 3.18: Ants search food ("TC-food" box) wandering randomly from their anthill ("TC-anthill" box). By `urd`-ing digital pheromones left while carrying food, ants stochastically find the optimal path toward the food source [MO14b]—numbers next to tuple centre names denote pheromone strength, that is, the number of `n(NBR)` tuples.

popular in computer science, e.g., in the computer networks management field: a simple yet expressive probabilistic mechanism alone – here, uniform primitives – is able to impact the output of the algorithm, as well as to support new features.

In other words, given a non-deterministic, distributed routing algorithm based on classical LINDA operations, a simple shift of calls from `in` and `rd` to `uin` and `urd` can dramatically impact both the output and the non-functional properties of the algorithm—e.g., robustness to topology change, adaptiveness to run-time bottlenecks formation, etc.

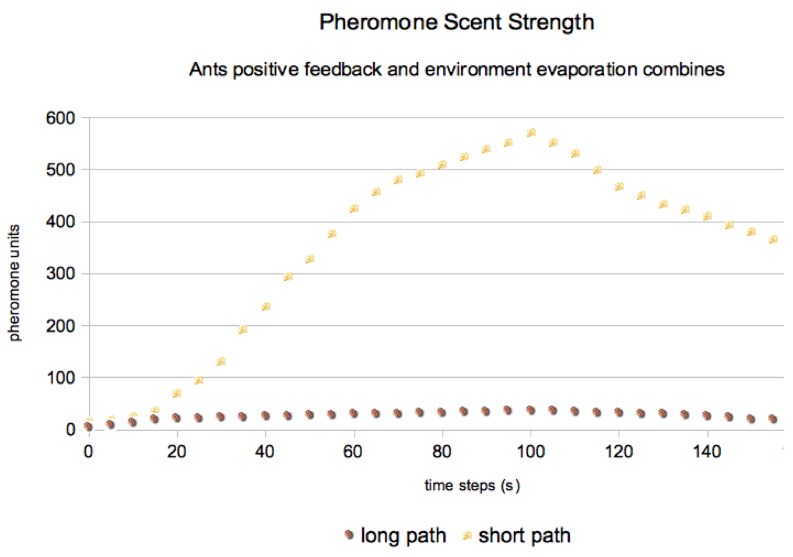


Figure 3.19: Pheromone strength over time [MO14b]. Descending phase corresponds to evaporation of pheromones due to food depletion.

```
while (!stopped) {
  if (!carryingFood) {
    // If not carrying food
    isFood = smellFood();
    if (isFood) {
      // pick up food if any
      pickFood();
    } else {
      // or stochastically follow pheromone
      direction = smellPheromone();
      move(direction);
    }
  } else {
    // If carrying food
    if (isAnthill()) {
      // drop food if in anthill
      dropFood();
    } else {
      // or move toward anthill
      direction = smellAnthill();
      move(direction);
    }
  }
}

private LogicTuple smellPheromone() {
  ITucsonOperation op = acc.urd(
    tcid,
    LogicTuple.parse("n(NBR)"),
    TIMEOUT
  );
  if (op.isResultSuccess()) {
    return op.getLogicTupleResult();
  }
}
```

Figure 3.20: Java code for ants [MO14b].

It should be noted that, w.r.t. the classification represented by Figure 3.12 in Subsection 4.2.5, coordination between ants and the environment, especially, the feedback loop created by ants depositing pheromone and evaporation depleting it, actually implements some of the patterns therein defined [FMMSM$^+$12].

In particular: (i) the digital pheromone composite pattern and the foraging high-level pattern, but also, to some extent, (ii) the gradient pattern, by interpreting pheromone trails as ant-steering gradients, and the chemotaxis, by interpreting pheromone trails as carriers of ants, driving them from the anthill to the food source and back.

**Gillespie's chemical solution simulation algorithm** A number of extensions to tuple-based coordination models exploit programmability of tuple spaces [DNO97]: there, while the tuple space interface is left unchanged (coordination primitives are the usual LINDA ones), the tuple space behavior in response to coordination events can be programmed so as to embed coordination laws.

The approach is adopted by LGI [MU00], MARS [CLZ00], TuCSoN [OZ99] among the many. For instance, the TuCSoN coordination model provides ReSpecT [Omi07] as a Turing-complete language for programming TuCSoN tuple centres, therefore allowing tuple spaces to embed any computable coordination law.

In particular, ReSpecT coordination is obtained by means of reactions, that is, logic-based computational activities – triggered in response to coordination events – which include, among the operations available, the basic LINDA primitives.

However, Turing equivalence within a coordination medium is still not enough to embed probabilistic behaviours in tuple centres: uniform primitives are required. This can be seen by building a chemical solution dynamics simulator based on Gillespie's well-known stochastic algorithm [Gil77] upon a TuCSoN tuple centre programmed in ReSpecT. The simulator depends on the availability of uniform primitives in ReSpecT—that is, available for use *within* ReSpecT reactions.

Given: (i) a multi-set of chemical laws of the form `law(RTs,Rate,RTs')`, where *RTs, RTs'* ::= $\perp$ | *RT* | *RT, RTs* and *RT* is a template for chemical reactants and products; and (ii) a multi set of reactants of the form `reactant(R,C)`, representing concentration `C` for reactant `R`; then, Gillespie's chemical solution simulation algorithm could be summarised by the following steps—to be executed in loop:

1. collect only the *triggerable laws*, that is, those for which at least one `R` for each `RT` exists, then, for each of them, consider any possible combination of $\mu$(`R, RT`) – where $\mu$ is a given matching function – and define *actual laws*

2. for each actual law, compute its *effective rate* `ERate` as the product between the given `Rate` and each $\mathtt{R}_i$'s concentration $\mathtt{C}_i = \frac{|\mathtt{R}_i|}{\sum_j |\mathtt{R}_j|}$ (using ReSpecT, the value could be automatically kept up-to-date)

3. compute the corresponding law *execution probability* as $p_i = \frac{\texttt{ERate}_i}{\sum_j \texttt{ERate}_j}$, according to which one law is selected for execution with a probabilistic choice

All steps but the third are easily computed using standard ReSpecT—namely, no uniform primitives are needed. In particular, actual laws can be represented as logic tuples of the form `actual(IL,OL)` – where `IL, OL` represent reactants and products, respectively – whose multiplicity in the space should be kept as desired to represent its execution probability. What cannot be done without uniform primitives is last part of step 3: there, in fact, a `urd(actual(IL,OL))` is to be used so as to probabilistically select the chemical law to be executed.

As shown by the ReSpecT code snippet listed in Figure 3.21, using a uniform primitive (line 17) to probabilistically *sample* the space of chemical laws allows step 3 of the Gillespie simulation algorithm to be faithfully implemented. In particular, given that step 2 inserts in the tuple centre a number of actual law tuples proportional to the effective rate determined in steps $1 - 2$ – so that higher rates correspond to higher multiplicity of tuples – then predicate `choose/2` effectively selects for execution the most probable law, since, according to uniform primitives semantics, the higher the multiplicity of a tuple, the higher the probability it will be selected for matching.

Enhancing the ReSpecT language with uniform primitives makes it possible for a TuCSoN tuple centre to work as a *biochemical tuple space* [VC09], thus, to play a key role in the design of complex computational systems, such as adaptive and self-organising ones. In particular, by replacing the implementation of Gillespie's chemical solution simulation algorithm of [POS13] with the uniform primitive-based one sketched in Figure 3.21, the probabilistic, concentration-driven mechanism for the selection of reactants to be con-

```
 1  % Invoked by timed ReSpecT reaction.
 2  gillespie():-
 3    % Steps 1-2
 4    collectTriggerable(-Trig),
 5    buildActual(+Trig, -Actual),
 6    % Step 2: higher rates => higher multiplicity
 7    computeRates(+Actual),
 8    % Step 3
 9    choose(-ALaw, -DeltaT),
10    % Law execution & scheduling of simulator.
11    execute(+ALaw, +DeltaT),
12    schedule(+DeltaT).
13  [...]
14  % Probabilistically, pick a law.
15  choose(AL, D):-
16    % higher multiplicity => higher probability
17    urd(actual(IL, OL)),
18    AL = actual(IL, OL),
19    rd_all(AL, L),
20    length(L, D).
21  [...]
```

Figure 3.21: Snippet of ReSpecT code implementing Gillespie's algorithm [MO14b].

sumed in reactions can be effectively and easily implemented—leading to a more accurate simulation, more faithfully following Gillespie's algorithm.

It should be noted that building accurate chemical-like mechanisms in a distributed setting – like the tuple-based one here proposed – is interesting not just for the obvious benefits it can bring to the computational chemistry community – which is focussed, e.g., on reliable simulation of biochemical networks and intra-cellular pathways – but also for its impact on the engineering of nature-inspired self-organising systems.

A chemical-like middleware is well suited for supporting self-* features, e.g., self-configuration, self-management, self-healing, self-optimisation, etc., and to inject these features in many different computational scenarios, such as pervasive systems [ZCF$^{+}$11], service-oriented architectures [VC09], and distributed algorithms [CVG09].

### 3.2.4 Discussion of Results

In this section *uniform primitives* were discussed, as well as their potential of impacting self-organising systems modelling and simulation. Starting from the central role of interaction in self-organising systems and, consequently, the foremost importance of coordination, the limits of purely non-deterministic approaches such as LINDA were recognised, and the benefits brought by specialising LINDA primitives as *probabilistic* primitives following a *uniform distribution* were shown—namely, uniform primitives.

Accordingly, a number of different examples were proposed and analysed to showcase the possible implications of uniform primitives on the observable behaviour of different self-organising coordinated systems.

In particular: how uniform primitives can directly enrich systems with self-organising behaviours, such as the self-optimisation provided by *load balancing*; how uniform primitives can play a central role in nature-inspired systems engineering and simulation, by modelling ant-like agents *random walks* and *foraging*; how uniform primitives can improve a *chemical solution simulator* based on Gillespie's algorithm.

Altogether, the results and examples suggest that uniform primitives can play the role of the *fundamental mechanisms* required to faithfully model, simulate, and design self-organising systems.

## 3.3 Formal Expressiveness of Uniform Primitives

A core issue for computer science since the early days, expressiveness of computational languages is still essential nowadays, in particular for coordination languages, which, by focussing on interaction, deal with the most relevant source of complexity in computational systems [Weg97].

Unsurprisingly, the area of coordination models and languages has produced a long stream of ideas and results on the subject, both adopting/adapting traditional approaches

– such as Turing equivalence for coordination languages [DNO98, BGZ00] – and inventing its own original techniques [WG03].

Comparing languages based on either their structural properties or the observable behaviour of the systems built upon them is seemingly a good way to classify their expressiveness. Among the many available approaches, the notion of *modular embedding* [dBP94], refinement of Shapiro's *embedding* [Sha91], is particularly effective in capturing the expressiveness of concurrent and coordination languages.

However, the emergence of classes of systems featuring new sorts of behaviours – pervasive, adaptive, self-organising systems [OV11, Omi13b] – is pushing computational languages beyond their previous limits, and asks for new models and techniques to observe, model and, measure their expressiveness. In particular, modular embedding fails in telling probabilistic languages apart from non-probabilistic ones.

### 3.3.1 Formal Definition of Uniform Primitives

To define the semantics of (getter) uniform primitives, a simplified version of the process-algebraic framework in [Bra08] is exploited, dropping multi-level priority probabilities.

Therein, the proposed formalism aims at dealing with the issue of open transition systems specification, requiring *quantitative information* to be attached to synchronisation actions at run-time—that is, based on the *environment* state during the computation.

The idea is that of *partially closing* labelled transition systems via a process-algebraic *closure* operator ($\uparrow$), which associates quantitative values – e.g., probabilities – to admissible transition actions based upon a set of *handles* defined in an application-specific manner, dictating which quantity should be attached to which action. More precisely:

1. actions labelling open transitions are equipped with handles
2. the operator $\uparrow$ is exploited to compose a system to a specification $G$, associating at run-time each handle to a given value—e.g., a value $\in \mathbb{N}$
3. quantitative informations with which to equip actions – e.g., probabilities $\in [0, 1]$ summing up to 1 – are computed from handle values for each enabled action, possibly based on the action context (environment)
4. quantitatively-labelled actions turn an open transition into a reduction, which then executes according to the quantitative information

Here, closure operator $\uparrow$, handles $h$, and closure term $G$, are exploited as follows:

- handles coupled to actions (open transition labels) represent tuple templates associated to corresponding primitives
- handles listed in restriction term $G$ represent tuples offered (as synchronisation items) by the tuple space

- restriction term $G$ associates handles (tuples) to their weight in the tuple space
- restriction operator $\uparrow$ *(i)* matches admissible synchronisations between processes and the tuple space, cutting out unavailable actions, and *(ii)* computes their associated probability distribution based upon handle-associated values

It is worth to note that closure operator $\uparrow$ could be seen as following the statistical interpretation of a uniform primitive: it takes a snapshot of the tuple space state – *matching*, step *(i)* – then samples it probabilistically—*sampling*, step *(ii)*.

**Semantics of `uin` (uniform consumption)** Three transition rules define the operational semantics of the `uin` primitive for *uniform consumption*:

Synch-C open transition representing the request for process-space synchronisation upon template $T$, which leads to the snapshot:

$$\texttt{uin}(T).P \mid \langle t_1, .., t_n \rangle \xrightarrow{T} \texttt{uin}(T).P \mid \langle t_1, .., t_n \rangle \uparrow \{(t_1, v_1), .., (t_n, v_n)\}$$

where $v_{i=1..n} = \mu(T, t_i)$, and $\mu(\cdot, \cdot)$ is the standard matching function of Linda, hence $\forall i, v_i ::= 1 \mid 0$

Close-C closed unlabelled transition (reduction) representing the internal computation assigning probabilities to synchronisation items (uniform distribution computation):

$$\texttt{uin}(T).P \mid \langle t_1, .., t_n \rangle \uparrow \{(t_1, v_1), .., (t_n, v_n)\} \hookrightarrow \texttt{uin}(T).P \mid \langle t_1, .., t_n \rangle \uparrow \{(t_1, p_1), .., (t_n, p_n)\}$$

where $p_j = \frac{v_j}{\sum_{i=1}^{n} v_i}$ is the absolute probability of retrieving tuple $t_j$, with $j = 1..n$

Exec-C open transition representing the probabilistic response to the requested synchronisation (the sampling):

$$\texttt{uin}(T).P \mid \langle t_1, .., t_n \rangle \uparrow \{.., (t_j, p_j), ..\} \xrightarrow{t_j}_{p_j} P[t_j/T] \mid \langle t_1, .., t_n \rangle \backslash t_j$$

where $[\cdot/\cdot]$ represents term substitution in process $P$ continuation, and $\backslash$ is multiset difference, expressing removal of tuple $t_j$ from the tuple space

**Semantics of `urd` (uniform reading)** As for standard Linda getter primitives, the only difference between *uniform reading* (`urd`) and uniform consumption (`uin`) is the non-destructive semantics of the reading primitive `urd`. This is reflected by Exec-R open transition:

Exec-R he same as Exec-C, except for the fact that it does not remove matching tuple

$$\texttt{urd}(T).P \mid \langle t_1, .., t_n \rangle \uparrow \{.., (t_j, p_j), ..\} \xrightarrow{t_j}_{p_j} P[t_j/T] \mid \langle t_1, .., t_n \rangle$$

whereas other transitions are left unchanged.

**Reduction example** As an example, in the following system state

$$\mathtt{uin}(T).P \mid \langle ta, ta, tb, tc \rangle$$

where $\mu(T, tx)$ holds for $x = a, b, c$, the following synchronisation transitions are enabled:

*(a)* $\mathtt{uin}(T).P \mid \langle ta, ta, tb, tc \rangle \xrightarrow{ta}_{0.5} P[ta/T] \mid \langle ta, tb, tc \rangle$

*(b)* $\mathtt{uin}(T).P \mid \langle ta, ta, tb, tc \rangle \xrightarrow{tb}_{0.25} P[tb/T] \mid \langle ta, ta, tc \rangle$

*(c)* $\mathtt{uin}(T).P \mid \langle ta, ta, tb, tc \rangle \xrightarrow{tc}_{0.25} P[tc/T] \mid \langle ta, ta, tb \rangle$

For instance, if transition *(a)* wins the probabilistic selection, then the system evolves according to the following trace—simplified by summing up cardinalities and probabilities in order to enhance readability:

$$\begin{gathered}
\mathtt{uin}(T).P \mid \langle ta, ta, tb, tc \rangle \\
\xrightarrow{T} \\
\mathtt{uin}(T).P \mid \langle ta, ta, tb, tc \rangle \uparrow \{(ta, 2), (tb, 1), (tc, 1)\} \\
\hookrightarrow \\
\mathtt{uin}(T).P \mid \langle ta, ta, tb, tc \rangle \uparrow \{(ta, \tfrac{1}{2}), (tb, \tfrac{1}{4}), (tc, \tfrac{1}{4})\} \\
\xrightarrow{ta}_{\frac{1}{2}} \\
P[ta/T] \mid \langle ta, tb, tc \rangle
\end{gathered}$$

### 3.3.2 From Modular Embedding to PME

**Sequential and modular embedding** The informal definition of *embedding* assumes that a language could be *easily* and *equivalently* translated in another one. “Easily” is usually interpreted as “without the need for a global reorganisation of the program”; whereas “equivalently” typically means “without affecting the program's *observable behaviour*”, according to some well-defined *observation criteria*—usually to be specified for the application at hand.

This intuitive definition was formalised by Shapiro [Sha91] for *sequential languages*. Given two languages $L, L'$, their program sets $Prog_L, Prog_{L'}$, and the powersets of their observable behaviours $Obs$, $Obs'$, assumption is that two *observation criteria* $\Psi, \Psi'$ hold:

$$\Psi : Prog_L \to Obs \qquad \Psi' : Prog_{L'} \to Obs'$$

Then, $L$ *embeds* $L'$ (written $L \succeq L'$) *iff* there exist a *compiler* $C : Prog_{L'} \to Prog_L$ and a *decoder* $D : Obs \to Obs'$ such that for every program $W \in L'$

$$D(\Psi[C(W)]) = \Psi'[W]$$

Subsequently, De Boer and Palamidessi [dBP94] argued the definition to be too weak to be applied proficiently, because any pair of Turing-complete languages would embed each other. Moreover, *concurrent languages* need at least *(i)* a novel notion of *termination* w.r.t. sequential ones, so as to handle deadlock and computation failure, and *(ii)* a different definition for the compiler, so as to consider also a priori unknown *run-time interactions* between concurrent processes.

Following their intuitions, De Boer and Palamidessi proposed a novel definition of embedding for which $C$ and $D$ should satisfy three properties:

**independent observation** elements $O \in \mathit{Obs}$ are sets representing all the possible outcomes of all the possible computations of a given system, hence they will be typically observed independently one from the other—since they are different systems. Thus, $D$ can be defined to be *elementwise*, that is:

$$\forall O \in \mathit{Obs} : D(O) = \{d(o) \mid o \in O\} \text{ (for some } d)$$

**compositionality of $C$** in a concurrent setting, it is difficult to predict the behaviour of all the processes in the environment, due to run-time non-deterministic interactions. Therefore, it is reasonable to require compositionality of the compiler $C$ both w.r.t. the parallel composition operator ($||$) and to the exclusive choice ($+$). Formally:

$$C(A \;||'\; B) = C(A) \;||\; C(B) \quad \text{and} \quad C(A \;+'\; B) = C(A) \;+\; C(B)$$

for every pair of programs $A, B \in L'$, where $'$ denotes symbols of $L'$

**deadlock invariance** unlike sequential languages, where only successful computations do matter – basically because unsuccessful ones could be supposed to backtrack –, in a concurrent setting it is needed to consider at least deadlocks, interpreting failure as a special case of deadlock, which should then be preserved by the decoder $D$:

$$\forall O \in \mathit{Obs}, \forall o \in O : \mathit{tm}'(D_{d(o)}) = \mathit{tm}(o)$$

where $\mathit{tm}$ and $\mathit{tm}'$ refer to termination modes of $L$ and $L'$ respectively

If an embedding satisfies all the three properties above, then it is called *modular*. In the following, symbol $\succeq$ is used for this notion embedding, since it is assumed to be the default reference embedding.

**Expressiveness of modular embedding** Suppose the following uLinda process $P$ – that is, a process using uniform specialisation of Linda primitives – and Linda process $Q$ are acting on tuple space $S$:

$$P = \texttt{uin}(T).\emptyset + \texttt{uin}(T).\texttt{urd}_p(T').\emptyset \qquad Q = \texttt{in}(T).\emptyset + \texttt{in}(T).\texttt{rd}(T').\emptyset$$
$$S = \langle \texttt{t}_\texttt{l}[20], \texttt{t}_\texttt{r}[10] \rangle$$

where: $T$ is a Linda template matching both tuples $\texttt{t}_\texttt{l}$ and $\texttt{t}_\texttt{r}$, $T'$ matches $\texttt{t}_\texttt{r}$ solely, and square brackets denote the multiplicity of each tuple. Suppose also that both processes have the following non-deterministic branching policy: branch left if consumption primitive (`uin` and `in`) returns $\texttt{t}_\texttt{l}$, branch right if consumption primitive returns $\texttt{t}_\texttt{r}$—as subscript suggests.

From the *modular observable behaviour* viewpoint exploited in modular embedding (ME), $P$ and $Q$ are *not* distinguishable. In fact, according to any observation function $\Psi$ defined based on [dBP94], $\Psi[P] = \Psi[Q]$, that is, $P$ and $Q$ can reach the same final states:

$$\Psi[P] = (\texttt{success}, \langle \texttt{t}_\texttt{r}[10] \rangle) \text{ OR } (\texttt{deadlock}, \langle \texttt{t}_\texttt{l}[20] \rangle)$$
$$\Psi[Q] = (\texttt{success}, \langle \texttt{t}_\texttt{r}[10] \rangle) \text{ OR } (\texttt{deadlock}, \langle \texttt{t}_\texttt{l}[20] \rangle)$$

The main point here is that while $P$ and $Q$ are *qualitatively* equivalent, they are not *quantitatively* equivalent. Notwithstanding, by no means ME can distinguish between their behaviours: since ME cannot tell apart the probabilistic information conveyed by, e.g., a uLinda primitive w.r.t. a Linda one.

For the don't know non-deterministic process $Q$, no *probability value* is available that could measure the chance to reach one state over the other, hence ME does not capture this property—quite obviously, since it is not meant to.

In fact, a two-way *modular encoding* can be trivially established between the two languages used above – say, uLinda (`out`, `urd`, `uin`) vs. Linda (`out`, `rd`, `in`) – by defining compilers $C$ as

$$C_{Linda} = \begin{cases} \texttt{out} & \longmapsto \texttt{out} \\ \texttt{rd} & \longmapsto \texttt{urd} \\ \texttt{in} & \longmapsto \texttt{uin} \end{cases} \quad C_{uLinda} = \begin{cases} \texttt{out} & \longmapsto \texttt{out} \\ \texttt{urd} & \longmapsto \texttt{rd} \\ \texttt{uin} & \longmapsto \texttt{in} \end{cases}$$

and letting decoder $D$ be compliant to the concurrent notion of observables given in [dBP94]. Given $C$ and $D$, it can be stated that uLinda *(modularly) embeds* Linda and also that Linda (modularly) embeds uLinda, hence they are *(observational) equivalent* ($\equiv_\Psi$). Formally:

$$\text{uLinda} \succeq \text{Linda} \ \wedge \ \text{Linda} \succeq \text{uLinda} \quad \Longrightarrow \quad \text{uLinda} \equiv_\Psi \text{Linda}$$

However, process $P$ *becomes* a probabilistic process due to the weighted-probability feature of `urd`, hence a probabilistic measure for $P$ behaviour would be potentially available:

however, it is not captured by ME. In the above example, for instance, such an additional bit of information would make it possible to assess that "one state is twice as probable than the other". This is exactly the purpose of *probabilistic modular embedding* (PME), defined in the remainder of this section.

**Probabilistic setting requirements** In order to define the notion of *probabilistic modular embedding* (PME), is is needed to elaborate on the informal definition of embedding, thus giving a precise characterisation to both the words "easily" and "equivalently".

Although the definition of "easily" given in [dBP94] could be rather satisfactory in general, it is preferable to strengthen its meaning by narrowing its scope to asynchronous coordination languages and calculi: without limiting the generality of the approach, this allows to make precise assumptions on the structure of programs.

A process can be said to be *easily* mappable into another if it requires:

1. no extra-computations to mimic complex coordination operators
2. no extra-coordinators (neither coordinated processes nor coordination medium) to handle suspensive semantics
3. no unbounded extra-interactions to perform additional coordination

Requirement 1 ensures absence of internal protocols in-between process-medium interactions, to emulate complex interaction primitives or behaviours—e.g., $\texttt{in}_p$ probabilistic selection simulated by processes drawing random numbers.

Requirement 2 avoids proliferation of processes and media while translating a program into another, constraining mappings to have the same number of processes and media.

The last requirement complements the first in ensuring absence of complex interaction patterns to mimic complex coordination operators, such as the `in_all` global primitive as a composition of multiple `inp` (`in` predicative version)—which could be obtained by forbidding unbounded *replication* and *recursion* algebraic operators in compiler $C$.

Altogether, the three requirements above represent a necessary constraint since the goal here is to focus precisely on coordination expressiveness, that is, on the sole expressiveness of coordination primitives, while abstracting away from the *algorithmic expressiveness* of processes and media.

The refined notion of "equivalently" is a bit more involved due to the very nature of a probabilistic process, that is, its intrinsic *randomness*. The notions of *observable behaviour* and *termination* are affected by randomness, thus they need to be re-casted in the probabilistic setting.

Probabilistic processes, in fact, have their actions conditioned by probabilities, hence their observable transitions between reachable states are probabilistic, too—so, their execution is possible but never guaranteed. Therefore, also final states are reached by chance only, following a certain *probability path*, hence termination, too, should be equipped with its own associated probability—also in case of deadlock.

In order to address the above issues, PME improves ME by making the following properties about observable behaviour and termination available:

**probabilistic observation** observable actions performed by processes – e.g., uLinda coordination primitives – should be associated with their *execution probability*. The aforementioned probability should depend on their run-time context, that is, synchronisation opportunities offered by the coordination medium. Then, compiler $C$ should preserve transition probabilities and properly aggregate them along any *probabilistic trace*—that is, a sequence of probabilistic actions

**probabilistic termination** final states of processes and media should be first defined as those states for which all outgoing transitions have probability 0. Then, they should be refined with a *probabilistic reachability value*, that is, the probability of reaching that state from a given initial one. Finally, decoder $D$ should preserve probabilities and determine how to compute them

**Probabilistic observation** A single probabilistic observable transition step, deriving from the synchronisation between a uLinda process and a uLinda space – e.g., by using a `uin` –, can be formally defined, according to Subsection 3.3.2, as follows:

$$\mathtt{uin}(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \quad \xrightarrow{\mu(T,t_j)}_{p_j} \quad P[t_j/T] \mid \langle t_1[w_1], .., t_n[w_n] \rangle \backslash t_j$$

where operator $\mu(T,t)$ denotes Linda matching function, symbol $[\cdot/\cdot]$ stands for template substitution in process continuation, and operator $\backslash$ represents multiset difference, there expressing removal of tuple $t_j$ from the tuple space.

By expanding the observable transition in its embedded reduction steps – that is, non-observable, silent transitions – the probabilistic semantics can precisely characterised, thanks to the $\uparrow$ operator:

$$\begin{gathered}
\mathtt{uin}(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \\
\xrightarrow{T} \\
\mathtt{uin}(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \uparrow \{(t_1, w_1), .., (t_n, w_n)\} \\
\hookrightarrow \\
\mathtt{uin}(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \uparrow \{(t_1, p_1), .., (t_j, p_j), .., (t_n, p_n)\} \\
\xrightarrow{t_j}_{p_j} \\
P[t_j/T] \mid \langle t_1[w_1], .., t_n[w_n] \rangle \backslash t_j
\end{gathered}$$

where $p_j = \frac{w_j}{\sum_{i=1}^{n} w_i}$ is the absolute probability of retrieving tuple $t_j$ (with $j = 1..n$) assuming for the sake of simplicity that all tuples match template $T$.

The $\uparrow$ operator implicitly enforces a re-normalisation of probabilities based on available synchronisations offered by the tuple space – that is, which tuples in the space match the given template –, in the spirit of PCCS restriction operator used in [VSS95] for the generative model of probabilistic processes. For instance, given the following *probabilistic* process $P$ acting on uLinda space $S$

$$P = \frac{1}{6}\mathtt{uin}(T).P + \frac{1}{2}\mathtt{uin}(T).P + \frac{1}{3}\mathtt{urd}(T).P$$
$$S = \langle \mathtt{t_l}[w], \mathtt{t_r}[w] \rangle$$

where template $T$ matches with both tuples $\mathtt{t_l}$, $\mathtt{t_r}$, it can be observed that an experiment `uin` could succeed with probability $\frac{2}{3} = \frac{1}{6} + \frac{1}{2}$, and that `urd` could do so with probability $\frac{1}{3}$. Furthermore, suppose the branching choice after experiment $\mathtt{in}_p$ to depend upon the consumed tuple.Then, the aforementioned re-normalisation can be seen by computing the probability to branch left ($\mathtt{t_l}$ is returned), which is 0.25, and that of branching right ($\mathtt{t_r}$ is returned), which is instead 0.75.

This addresses the first issue of probabilistic observation: *observable actions* are defined as all those actions requiring synchronisation with the medium. Then, they are equipped with a probability of execution driven by available run-time synchronisation opportunities, and normalised according to the generative model interpretation enforced by $\uparrow$. Formally, the *probabilistic observation function* ($\Theta$), mapping a process ($W$) into observables, is defined as follows:

$$\Theta[W] = \Big\{ (\rho, W[\bar{\mu}]) \mid \quad (W, \langle\sigma\rangle) \longrightarrow^{*} (\rho, W[\bar{\mu}]) \Big\}$$

where $\rho$ is a probability value $\in [0,1]$, $\bar{\mu}$ is a sequence of actual synchronisations – e.g., $\bar{\mu} = \mu(T_1, t_1), \ldots, \mu(T_n, t_n)$ – and $\sigma$ is the space state—e.g., $\sigma = t_1, \ldots, t_n$.

The definition is partial in the sense that it is only known how to compute $\rho$ for single-step transitions – that is, according to the $\uparrow$-dependent generative semantics –, in fact, it tells nothing about how to compute $\rho$ also for *observable traces*—that is, for sequences of observable actions. Nothing more than standard probability theory is needed here, stating that [Dra67]:

- the cumulative probability of a *sequence* – that is, a list separated by symbol '$\cdot$' – of probabilistic actions is the *product* of the probabilities of actions
- the cumulative probability of a *choice* – that is, a list separated by symbol '+' – of probabilistic actions is the *sum* of the probabilities of actions

Formally, *sequence probability aggregation function* ($\bar{\nu}$) and *choice probability aggregation function* ($\nu^{+}$), mapping multiple probability values to a single one, are defined as:

$$\bar{\nu} : W \times \langle\sigma\rangle \mapsto \rho \ \text{ where } \ \rho = \textstyle\prod_{j=0}^{n} \{ p_j \mid (p_j, \mu_{\bar{\ell}}) \in \Theta[W = \bar{\ell}.W'] \} \}$$
$$\nu^{+} : W \times \langle\sigma\rangle \mapsto \rho \ \text{ where } \ \rho = \textstyle\sum_{j=0}^{n} \{ p_j \mid (p_j, \mu_{\ell^{+}}) \in \Theta[W = \ell^{+}.W'] \} \}$$

where $\bar{\ell}$ is a *sequence* of synchronisation actions – e.g., $\bar{\ell} = \mathtt{uin}(T_1).\mathtt{urd}(T_2). \ldots$ – and $\ell^{+}$ is a *choice* between synchronisation actions—e.g., $\bar{\ell}^{+} = \mathtt{uin}(T_1) + \mathtt{urd}(T_2) + \ldots$. By properly composing aggregation functions, it is possible to compute $\Theta[W]$ for any process $W$ and for any transition sequence $\longrightarrow^{*}$.

An example may help clarifying the above definitions. Consider the following process $P$ and space $S$ (sequence operator has priority w.r.t. choice):

$$P = \mathtt{uin}(T).\big(\mathtt{urd}(T').P' + \mathtt{urd}(T').P''\big) + \mathtt{uin}(T).P'$$
$$S = \langle \mathtt{t_{l1}}[40], \mathtt{t_{l2}}[30], \mathtt{t_{r1}}[20], \mathtt{t_{r2}}[10] \rangle$$

where template $T$ may match either $\mathtt{t_{l1}}$ or $\mathtt{t_{r1}}$ whereas $T'$ may match either $\mathtt{t_{l2}}$ or $\mathtt{t_{r2}}$—and branching structure is based on returned tuple as usual, that is, $\mathtt{t_{l1}}$,$\mathtt{t_{l2}}$ for left, $\mathtt{t_{r1}}$,$\mathtt{t_{r2}}$ for right. Applying function $\Theta$ to process $P$ could lead to the following *observable states*:

$$\Theta[P] = (0.5, P'[\mu(T, \mathtt{t_{l1}}), \mu(T', \mathtt{t_{l2}})])$$
$$\Theta[P] = (0.1\bar{6}, P''[\mu(T, \mathtt{t_{l1}}), \mu(T', \mathtt{t_{r2}})])$$
$$\Theta[P] = (0.\bar{3}, P'[\mu(T, \mathtt{t_{r1}})])$$

According to $\Theta$, and using both aggregation functions $\bar{\nu}$ and $\nu^+$, it can be stated that process $P$ will eventually behave like $P'$ – although with different substitutions – with a probability of $\simeq 0.83$, and like $P''$ with a probability of $\simeq 0.17$.

As a last note, one may consider that sequence probability aggregation function $\bar{\nu}$ asymptotically tends to 0 as the length of the sequence $\bar{l}$ tends to infinity. This is unavoidable according to the basic probability theory framework adopted throughout this paper. One way to fix this aspect could be that of considering only the prefix sequence executed in loop by a process, then to associate that process not with the probability of $n$ iterations of the loop, but with the probability of the looping prefix sequence solely—that is, with only 1 iteration of the loop.

**Probabilistic termination** In order to define *probabilistic termination*, the classical notion of termination should be adapted to the probabilistic setting. For this purpose, *ending states* are defined as all those states for which either no more transitions are possible or all outgoing transitions have probability 0 to occur.

Other than that, termination states can be enumerated as usual [dBP94] to be $\tau = \mathtt{success}, \mathtt{failure}, \mathtt{deadlock}$, plus the `undefined` state, which could be useful to distinguish *absorbing states* – that is, those states for which the probability of performing a self-loop transition is 1 – from deadlocks. Furthermore, termination states have to be equipped with a *probabilistic reachability value*.

Formally, the reachability value $\rho_\perp$ and the *probabilistic termination state function* $\Phi$ are defined as follows:

$$\Phi[W] = \Big\{ (\rho_\perp, \tau) \mid \quad (W, \langle\sigma\rangle) \longrightarrow^*_\perp (\rho_\perp, \tau) \Big\}$$

where subscript $\perp$ denotes a sequence of finite transitions leading to termination $\tau$. By comparing this function with the observation function $\Theta$, it can be noted that $\Phi$ abstracts away from computation *traces*, that is, it does not keep track of synchronisations, hence substitutions, in term $W[\bar{\mu}]$, focussing solely on termination states $\tau$. However, when computing the value of $\rho_\perp$, the same aggregation functions $\bar{\nu}$ and $\nu^+$ have to be used.

For instance, recalling process $P$ used to test observation function $\Theta$, changing space $S$ configuration as follows:

$$P = \texttt{uin}(T).\big(\texttt{urd}(T').P' + \texttt{urd}(T').P''\big) + \texttt{uin}(T).P'$$
$$S = \langle \texttt{t}_{\texttt{l1}}[40], \texttt{t}_{\texttt{r1}}[20] \rangle$$

where $P' \equiv P'' = \emptyset$ so as to reach termination, the application of the probabilistic termination state function just defined leads to the following *observable termination states*:

$$\Phi[P] = (0.\bar{6}, \texttt{deadlock}) \quad \vee \quad \Phi[P] = (0.\bar{3}, \texttt{success})$$

In particular, $P$ deadlocks with probability $\frac{2}{3}$ if tuple $\texttt{t}_{\texttt{l1}}$ is consumed, whereas succeeds with probability $\frac{1}{3}$ if tuple $\texttt{t}_{\texttt{r1}}$ is consumed in its stead.

It should be noted that here absorbing states cause no harm for the sequence probability aggregation function $\bar{\nu}$, since the probability value aggregated until reaching the absorbing state will be from now on always multiplied by 1—in the very end, making each iteration of the self-loop indistinguishable from the others.

### 3.3.3 Relative Expressiveness Results

**uLinda vs. Linda** Recall the two processes $P$ and $Q$ acting on space $S$ introduced in the example of Subsection 3.3.2:

$$P = \texttt{uin}(T).\emptyset + \texttt{uin}(T).\texttt{urd}(T').\emptyset \qquad Q = \texttt{in}(T).\emptyset + \texttt{in}(T).\texttt{rd}(T').\emptyset$$
$$S = \langle \texttt{t}_{\texttt{l}}[20], \texttt{t}_{\texttt{r}}[10] \rangle$$

Embedding observation is now repeated under the assumptions of PME. As expected, the *behaviour* of process $P$ can now be distinguished from that of process $Q$. In fact, applying function $\Phi$ to both $P$ and $Q$ leads to:

$$\Phi[P] = (0.\bar{6}, \texttt{success}) \text{ OR } (0.\bar{3}, \texttt{deadlock})$$
$$\Phi[Q] = (\bullet, \texttt{success}) \text{ OR } (\bullet, \texttt{deadlock})$$

where symbol $\bullet$ denotes absence of information.

Therefore, only a one-way *encoding* can be now established between the languages – again, uLinda (`out`, `urd`, `uin`) vs. Linda (`out`, `rd`, `in`) – by defining compiler $C_{Linda}$ as

$$C_{Linda} = \begin{cases} \texttt{out} & \longmapsto & \texttt{out} \\ \texttt{rd} & \longmapsto & \texttt{urd} \\ \texttt{in} & \longmapsto & \texttt{uin} \end{cases}$$

and making decoder $D$ rely on observation function $\Theta$ and termination function $\Phi$. Then, it can be stated that uLinda *probabilistically embeds* ($\succeq_p$) Linda—but not the other way around. Formally, according to PME:

$$\text{uLinda} \succeq_p \text{Linda} \;\wedge\; \text{Linda} \not\succeq_p \text{uLinda} \quad \Longrightarrow \quad \text{uLinda} \not\equiv_p \text{Linda}$$

In the end, PME succeeds in telling uLinda apart from Linda (classifying uLinda as *more expressive* than Linda), whereas ME fails.

### 3.3.4 Similar Approaches

To the best of the author knowledge, no other research extends the work by De Boer and Palamidessi [dBP94] towards a probabilistic setting with a focus on coordination languages, in the same way as no other work tries to connect the concept of linguistic embedding with any of the probability models defined in [VSS95].

In [BLY01], the authors try to answer questions such as how to formalise probabilistic transition system, and how to extend non-probabilistic process algebras operators to the probabilistic setting. In particular, they focus on reactive models of probability – hence, models where pure non-determinism and probability coexist – and provide the notions of *probabilistic bisimulation*, *probabilistic simulation* (the asymmetric version of bisimulation), and *probabilistic testing preorders* (testing-based observation of equivalence), again applied to PCCS.

Although targeted to the PCCS reactive model, their work is related to the one here described in the attempt to find a way to *compare* the relative expressiveness of different probabilistic languages. On the other hand, the approach described is quite different because it adopts a *linguistic embedding* perspective rather than a *process bisimulation* viewpoint. Whereas probabilistic bisimulation can prove the *observational equivalence* of different probabilistic models, it cannot detect which is the most expressive among them.

However, it cannot be excluded that a two-way probabilistic embedding relationship may correspond to a probabilistic bisimulation according to [BLY01] definition of bisimulation—at least for reactive models.

In [BDPW02] the notion of *linear embedding* is introduced. Starting from the definition of ME in [dBP94], the authors aim at *quantifying* how much a language embeds another one, that is, how much a given language is more expressive than another. To do so, they *(i)* take *linear vector spaces* as a semantic domain for a subset of LINDA-like languages – that is, considering `tell, get, ask, nask` primitives –; *(ii)* define an observation criteria associating to each program a linear algebra operator acting on the aforementioned vector spaces; then *(iii)* quantify the difference in expressive power by computing the *dimension* of the linear algebras associated to each language.

Although the possibility to *quantify* the relative expressive power of a set of languages is appealing, the work in [BDPW02] do consider neither probabilistic languages nor probabilistic processes, hence cannot be directly compared to the one described here. However, it still remains an interesting path to follow for further developments of the probabilistic embedding here proposed.

Last but not least, in [DPHW03] the authors apply the *Probabilistic Abstract Interpretation* (PAI) theory and its techniques to probabilistic transition systems, in order to formally define the notion of *approximate process equivalence*—that is, two probabilistic processes are equivalent up to an error $\varepsilon$. As in [BDPW02], Di Pierro, Hankin, and Wiklicky adopt linear algebras to represent some semantical domain, but they consider probabilistic transition systems instead of deterministic ones. Therefore, they allow matrices representing algebraic operators to specify probability values $v \in [0, 1]$ instead of binary values $b = 0 \mid 1$.

Then, by using the PAI framework and drawing inspiration from *statistical testing* approaches, they define the notion of *$\varepsilon$-bisimilarity*, which allows the minimum number of tests needed to accept the bisimilarity relation between two processes to be quantified with a given confidence. By examining this value, a quantitative idea of the *statistical distance* between two given sets of (processes) admissible behaviours can be inferred.

Although quite different from the work here described, theirs can be considered nevertheless as another opportunity for further improvement of PME: for instance, an enhanced version of PME may be able to detect some notion of approximate process equivalence.

## 3.4 Remarks & Outlook

The material presented in this chapter is at the very core of the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model described in Part II of this thesis. In particular:

- some of the bio-inspired basic patterns extensively discussed in Section 3.1 are used to implement $\mathcal{M}o\mathcal{K}$ reactions—e.g., decay, aggregation, and diffusion for their homonym reactions, and feed for reinforcement
- the whole approach to artificial chemical reactions engineering is extremely valuable for identifying the core set of $\mathcal{M}o\mathcal{K}$ reactions, as well as their kinetic rates
- uniform primitives are exploited to implement the $\mathcal{M}o\mathcal{K}$ prototype on TuCSoN, as described in Section 7.1 of Chapter 7, both as regards the chemical machinery working as $\mathcal{M}o\mathcal{K}$ compartment, and $\mathcal{M}o\mathcal{K}$ reactions themselves

Next chapter is strongly related, too, being the infrastructure and language there described conceived and designed to better support the approach here described to engineering self-organising coordination, within pervasive computing scenarios.

Nevertheless, there is still much work which can be done to further advance the field of chemical-inspired self-organising coordination, e.g.:

- as basic patterns may be composed into higher level patterns depicted in Figure 3.12, it is interesting to investigate if their encodings into artificial chemical reactions may be similarly composed, to obtain the same higher level patterns; then, it is interesting to undertake simulations to see which degree of controllability composite patterns exhibit, e.g., w.r.t. the tuning of the custom kinetic rates of the basic patterns they are built on top of
- also, studying the interplay between different artificial chemical reactions coexisting in the same coordination space is undoubtedly something worth doing, to understand, e.g., if the whole approach is composable or not, and if not, which kind of

issues arise from composition—it should be noted that a similar study has been done for those patterns used to implement $\mathcal{MoK}$ reactions (see Subsection 6.2.2)

- as far as uniform primitives are concerned, an interesting further work is to generalise the probabilistic distribution exploited, possibly allowing application-specific distributions to be dynamically provided
- besides, a very interesting effort would be that of further investigating formally the expressiveness leap provided by uniform primitives, e.g., trying to build a correlation with the hierarchy of probability models described in [VSS95]

# Chapter 4

# Coordination Issues in Situated Pervasive Systems

In this chapter a novel approach to coordination in situated Multi-Agent Systems (MAS) is described, by proposing a meta-model for situated MAS engineering, and an architecture for the design of shared space -based middleware for situated coordination. Accordingly, a review of the TuCSoN architecture is done, and an extension to the ReSpecT coordination language provided so as to deal with situatedness-related issues directly at the level of the coordination language.

Thus, in what follows a review of meta-models and architectures is performed from an historical perspective, discussing their evolution, to propose the reference meta-model and abstract architecture (Section 4.1); then, instantiation of the proposed architecture on the TuCSoN middleware is described, discussing the methodology it implicitly suggests to situated MAS designers, and focussing on the issue of environmental situatedness (Section 4.2); finally, spatial situatedness is focussed, discussing the proposed extension to the ReSpecT language (Section 4.3).

## 4.1 The Quest toward Situatedness in MAS

Nowadays, *activities* in computational systems are typically modelled by means of *agents* in MAS. Modelling activities through agents basically means representing *actions* along with their motivations—namely, the *goals* that determine and explain the agent's course of actions [Cas12]. Handling *dependencies* among activities in MAS is in essence a *coordination* problem [MC94].

Accordingly, *coordination models* are the most suitable conceptual tools to harness complexity in MAS [COZ00]. Through the notion of *social action* [Cas98], dependencies are modelled in MAS in terms of agent *societies*, in turn represented computationally by means of *coordination artefacts* [ORV$^+$04a], also called *coordination media* [Cia96].

However, agents and societies are not the only main bricks for MAS: *environment* is an essential abstraction for MAS modelling and engineering [WOO07], which needs to be suitably represented, and related to agents. This is the core of the notion of *situated action*, as the realisation that coordinated, social, intelligent action arises from strict interaction with the environment, rather than from rational practical reasoning [Suc87].

The need for *situatedness* of agents and MAS is often translated into the requirement of being sensitive to *environment change* [FM96]. This basically means dependency, again: so, agent activity should be affected by environment change.

In all, this means that *(i)* things happen in a MAS because of either agent activities or environment change – the only two sources of *events* in MAS –, *(ii)* complexity arises from both agent-agent and agent-environment dependencies—roughly speaking, from both *social* and *situated* interaction. Also, this suggests that coordination – in charge of *managing dependencies* [MC94] – could be used to deal with both forms of dependencies in a uniform way; so, also, that coordination artefacts could be exploited to handle both social and situated interaction [OM13].

Accordingly, an agent-oriented, *event-driven architecture* for situated pervasive systems is proposed, that exploits coordination to handle both social and situated interaction. By focussing on situatedness, is first observed the evolution of the notion of environment in MAS meta-models (Subsection 4.1.1), then the approach to situatedness by some well-known agent frameworks (Subsection 4.1.2) is explored. The proposed architecture is presented and detailed (Subsection 4.1.3), then implemented within the TuCSoN middleware for MAS coordination [OZ99] in next section (Section 4.2).

### 4.1.1 Review of Meta-models

A linear account of the evolution of the concepts and ideas within any field of human knowledge is likely to be at the same time artificial and essential. Artificial, in that evolution of concepts is actually never linear, so that for instance several contrasting ideas typically coexist altogether at the same time on the same subject in the same field. Essential, in that understanding history of ideas typically requires some linearisation, for instance when trying to explain the emergence of new concepts, or, to foresee the next steps of their evolution.

In this section a short linear account of the evolution of agent-oriented *meta-models* along the years is provided, by focussing on the abstractions adopted to address the issues of *situatedness* and *coordination* in MAS engineering.

**Activities by agents** While the essential role of the environment was made clear since the early days of research on MAS [FM96] – mostly based on the work on reactive agents [Bro86, DM91] –, all the focus then was on activities, with no abstractions devoted to model either environment change or dependencies. Figure 4.1 roughly depicts the corresponding (implicit) meta-model. There, first of all, agents' only means of interaction is by

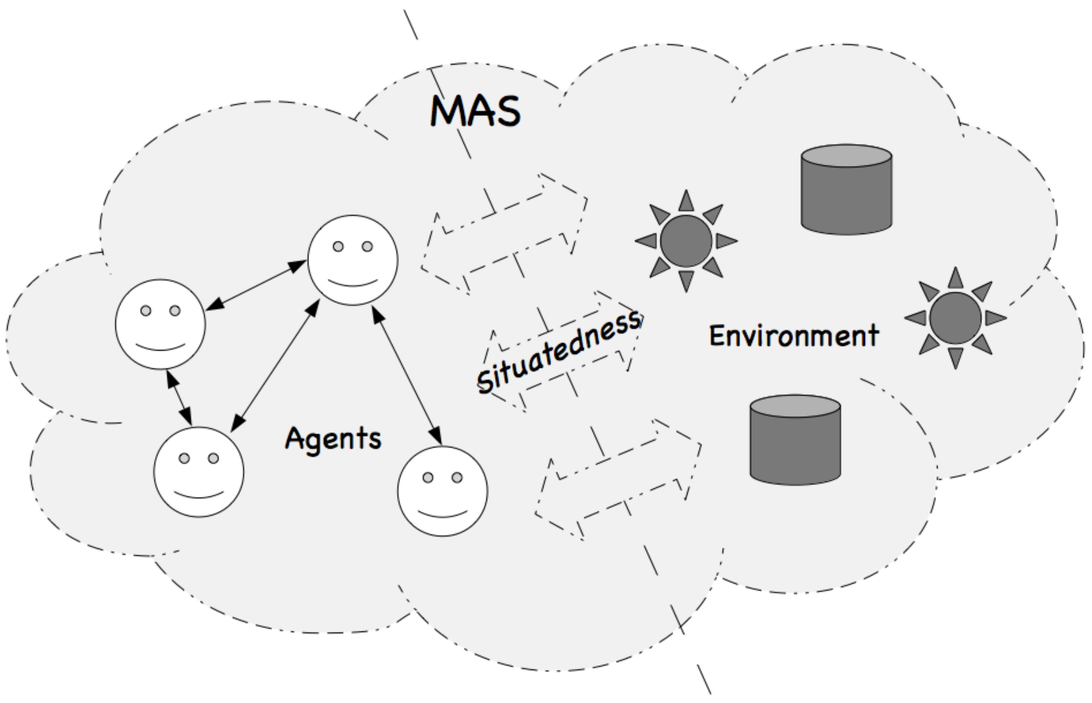


Figure 4.1: MAS meta-model in the early days of MAS history: only activities are explicitly accounted, modelled as agents, and the only way to handle their mutual dependencies is essentially by *plain communication* [MO15b].

messages exchange. This means that inter-agent dependencies are basically dealt with via inter-agent communication, handling all coordination issues at the individual level—using the so-called *subjective* approach to coordination [Sch01, OO03]. Furthermore, no specific abstraction is devoted to *environment engineering* [WOO07]: every agent is basically supposed to directly deal with environment resources, thus providing no specific support to agent-environment interaction.

So, on the one hand, the only thing that makes things happen in a MAS are activities, modelled through agents. On the other hand, any sort of situatedness requires ad-hoc solutions – such as bridges to lower-level languages –, with no general-purpose abstraction to directly support environment engineering.

**Environment change by agents** When the notion of environment was not yet recognised as a first-class MAS abstraction – as the one of agent – but as a sort of technical missing link between agent technology and real-world applications, the easiest approach to environment modelling in MAS was quite obvious: environment resources or properties are represented and manipulated by *environment agents*, acting as wrappers—as in [CM01], for instance.

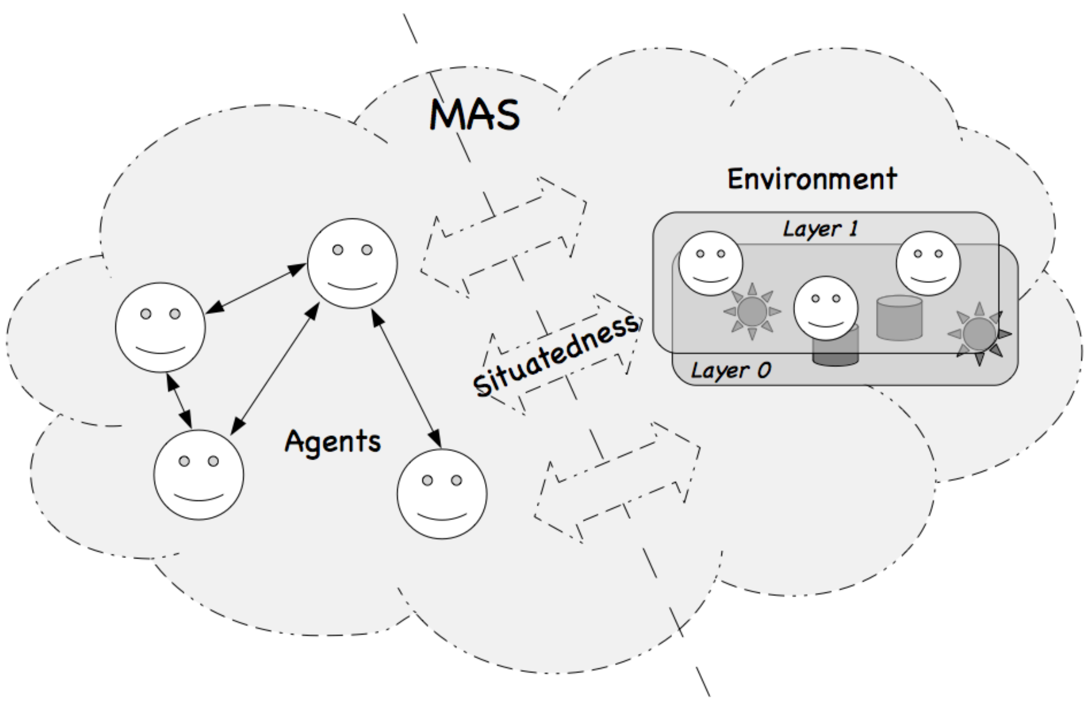


Figure 4.2: MAS environment recognised as a technical issue in MAS design: environment activities are delegated to wrapper environment agents; both coordination and situatedness are correspondingly reduced to message-passing issues [MO15b].

Thus, no novel, specific abstraction is provided for capturing environment change: simply, the agent abstraction is somehow abused, exploiting autonomy just to reproduce unpredictable behaviour. Correspondingly, situatedness is (poorly) handled merely as inter-agent communication.

Figure 4.2 depicts the corresponding MAS scenario: a software layer of environment agents is built upon the lower level of environment resources—the one of low-level languages and legacy API. In this way, situatedness of proper agents (those on the left side of Figure 4.2) is just a consequence of their communication with their reactive siblings (those within the *"Layer 1"* frame). As far as dependencies are concerned, nothing new happens: agents in a MAS are still a bunch of threads exchanging messages, and the reach of subjective coordination is extended to cover all dependencies among activities, including agent-environment interactions.

**Coordination for social dependencies** Since MAS became the reference paradigm for complex computational systems [OZ04], direct inter-agent communication turned out to be a feeble solution for handling non-trivial dependencies. The recognition that an *interaction space* exists in MAS, and is a primary source of complexity, made the need of specific abstractions emerge, devoted to the management of social interaction.

Along this line – as depicted in Figure 4.3 – the importance of governing the interaction space *outside* the agents was recognised, and the shift towards *objective coordination* began [Sch01, OO03], that is, coordination provided to agents *as a service* [VO06], rather than cooperatively built by agent themselves through communication protocols. Starting from the simplest attempts, such as inter-agent communication protocols and individual agent mailboxes, a number of general-purpose *coordination models* were defined [OZKT01] – such as TuCSoN [OZ99], Lime [PMR99], and MARS [CLZ00] –, allowing MAS engineers to manage MAS interaction space via coordination artefacts—handling social dependencies by ruling agent-agent interaction [COZ00].

However, no specific attention is still devoted here to the interaction with MAS environment, since environment representation lacks suitable abstractions supporting agent-environment dependencies. Thus, even though clearly a coordination issue, situatedness here is still conceived as a separated problem independent of coordination, and consequently managed just at the technology level—that is, according to the custom solutions provided by the MAS development framework at hand.

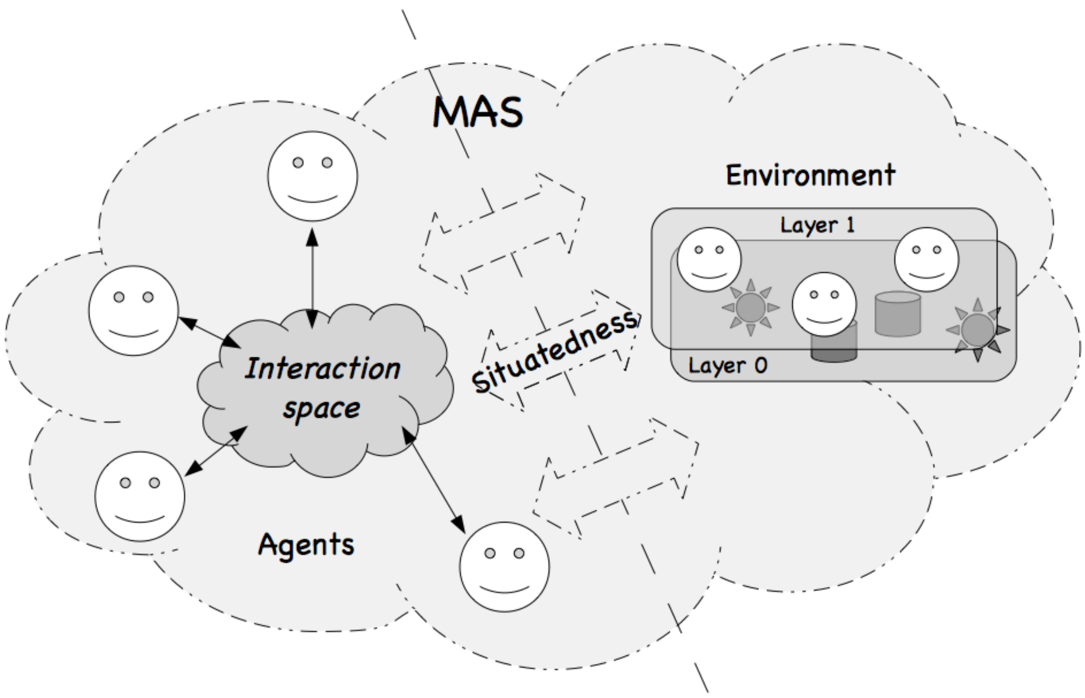


Figure 4.3: The existence of an *interaction space* outside the agents is recognised, requiring suitable abstractions to be modelled and governed; agent-environment interactions are not considered as part of the space [MO15b].

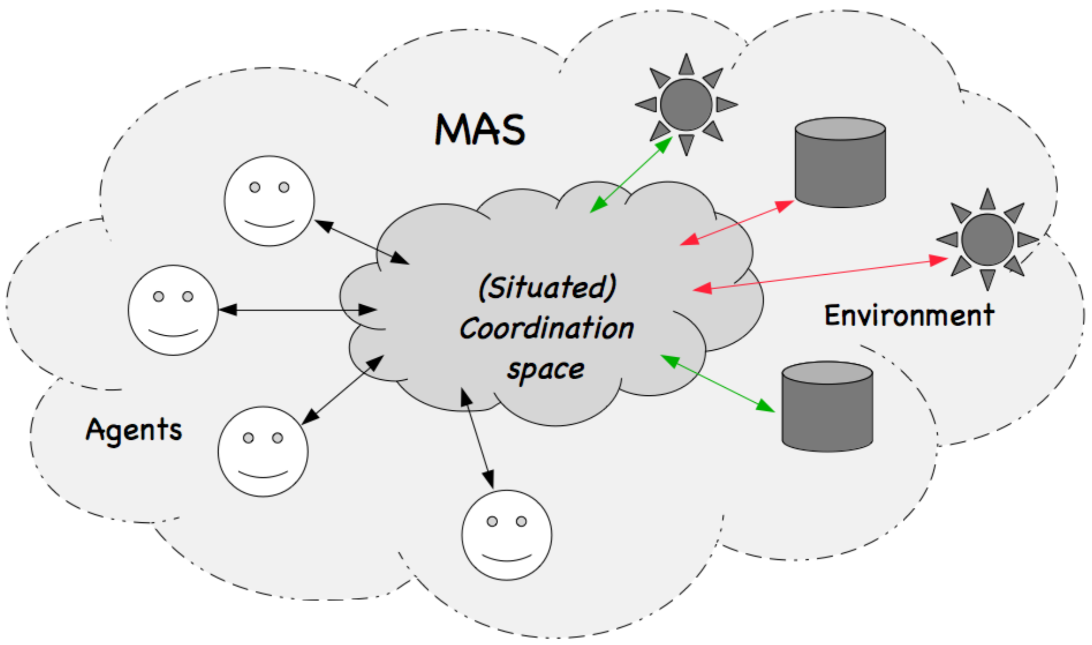


Figure 4.4: The fundamental role of *situated coordination* in supporting situatedness in MAS is recognised: thus, a uniform, coherent, and comprehensive *situated coordination service* is provided to promote a system-level view of MAS coordination [MO15b].

**Coordination for all dependencies** As soon as situatedness is considered as a proper coordination issue, since it deals with agent-environment interaction, the chance of a uniform meta-model emerges, which could handle both social and situated dependencies in a coherent way. This is for instance the view promoted by the TuCSoN architecture [MO13h], where ReSpecT tuple centres work as coordination artefacts handling both agent-agent and agent-environment interaction [MO13a].

The corresponding meta-model is depicted in Figure 4.4 above, where also interaction with MAS environment is considered to be part of the MAS interaction space – now, a *situated coordination space* –, promoting a view of MAS in which all dependencies are uniformly handled by coordination abstractions [MO13h]. In this scenario, the property of being situated can be interpreted both at the individual *agent level* and at the overall *system level*, as a property belonging to the whole coordinated MAS—since agent-environment interaction can be handled both at the individual and at the social level by coordination artefacts [ORV+04a].

### 4.1.2 Review of Architectures

Agent-oriented programming frameworks bring agent abstractions to life, by reifying agent (meta-)models, and making them available to MAS engineers. It is then easy to understand how the conceptual evolution devised in Subsection 4.1.1 has led to different frameworks for MAS development over the years.

Accordingly, in this section agent-oriented *architectures* are focussed, as they are adopted by some well-known agent-oriented programming frameworks, and a discussion about how they address the issues of situatedness and coordination in the engineering of MAS is provided. In particular, three MAS development frameworks are analysed, that is, JADE, Jason, and CArtAgO, chosen for their relevance and wide-spread adoption, as well as for their belonging to different periods of MAS research.

**Activities, dependencies, environment change in JADE** JADE (Java Agent DEvelopment framework)[1] [BPR99] is a Java-based framework providing an API and a run-time middleware to develop and deploy agent-based applications in compliance with FIPA[2] standards for MAS. Generally speaking, JADE most notable features are: transparent message-passing based on FIPA ACL (agent communication language), white and yellow pages services, support for strong mobility, and built-in FIPA protocols.

The most relevant architectural components of the JADE middleware are:

**agents** JADE agents are Java objects executed by a single thread of control. The ability to pursue different goals at the same time is provided by JADE *behaviour model*: each agent has a set of behaviours (again, Java objects) representing task-achieving jobs, which are executed by a non-preemptive round-robin scheduler internal to the agent (thus hidden to JADE programmers). The only means agents have to communicate (and coordinate) is by exchanging FIPA ACL messages

**ACC** the Agent Communication Channel is the run-time facility in charge of *asynchronous* delivery of messages: each agent has its own mailbox, and is notified upon reception of any message, whereas if and when to process the message is left to the agent's own deliberation—for the sake of preserving its autonomy

**FIPA protocols** JADE supports FIPA standard *communication protocols* (built on top of ACC services) by providing built-in behaviours that the programmer extends by specifying what to do in each step of the protocol: upon reception of the expected message according to the protocol state, the proper callback method is automatically called by JADE run-time

This short middleware description is enough to relate JADE to the pictures in Subsection 4.1.1: *(i)* agents are the abstractions to handle activities in JADE; *(ii)* JADE provides no specific environment abstraction, so that JADE agents have to be used to capture environment change, too; and *(iii)* the only JADE abstraction to handle dependencies is the FIPA protocol, governing agent-agent communication.

[1]`http://jade.tilab.com/`
[2]`http://www.fipa.org/`

Accordingly, Figure 4.5 relates JADE architectural model with the meta-model of Figure 4.3, emphasising how JADE agents are abused to represent environment properties and resources.

As a result, the implicit JADE coordination model is somehow a hybrid one, borrowing from both subjective and objective coordination: on the one hand, most is charged upon subjective coordination, since coordination is based on protocols, and protocols are implemented as agent behaviours; on the other hand, the fact that protocols are behaviour – hence, Java objects different from agents – makes it possible to separate to some extent the coordination (social) logic (indeed, inside the protocols) from the agent (individual) logic (inside non-protocol agents' behaviours).

Also, as depicted in Figure 4.5, situatedness in JADE is merely handled as an inter-agent communication issue.

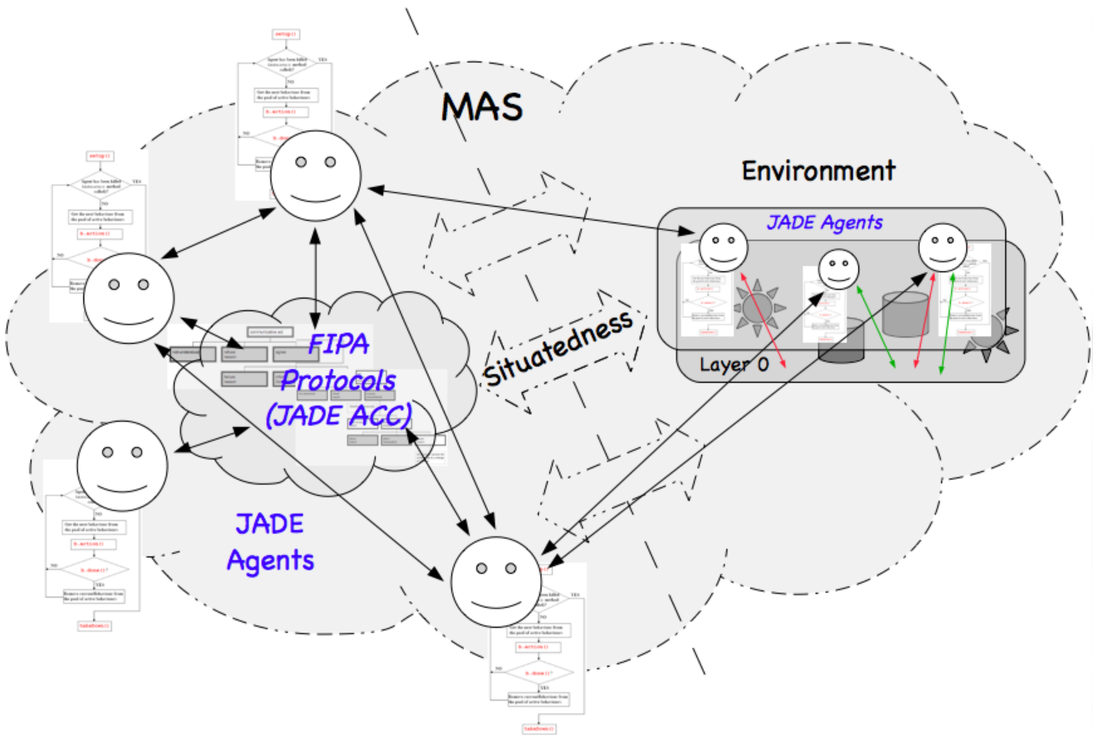


Figure 4.5: In JADE, the only abstraction given to handle dependencies is that of *protocol*, whereas no specific abstraction exists for the environment; agent-agent interaction can be governed by FIPA communication protocols, and situatedness is achieved through *environment agents* [MO15b]—that is, agents deployed ad-hoc to model the environment and represent its change.

**Activities, dependencies, environment change in Jason** Jason[3] [BHW07] is both a programming language, an agent development framework, and a run-time system. As a language, it implements a dialect of AgentSpeak [Rao96]; as a development framework, it comes with an API to design agents and MAS; as a run-time system, it provides the infrastructure needed to execute a MAS—in a non-distributed setting; to enable distribution, an integration with Jade exists.

Although Jason is entirely programmed in Java, it features BDI agents, so that a higher-level language (the Jason language) is used to program Jason agents using BDI architectural abstractions. Jason natively supports the notion of MAS environment and situatedness: in particular, Jason agents are said to be *situated* in an environment since they can *sense* it through sensors, and *act* upon it through actuators. Jason sensors and actuators have to be implemented directly using Java—along with all the environment resources needed to model the MAS environment.

As a result, Jason architectural components that are worth to be mentioned here are:

**agents** Jason agents are BDI architectures whose reasoning cycle encompasses environment perception, beliefs update, message exchange, plan selection/execution, action. Different plans can be executed concurrently as expected by a goal-directed agent architecture. Jason agents can also interact *(i)* directly by sharing goals/plans/beliefs (again via messages) as well as *(i)* indirectly by acting upon the environment—knowing that other agents will perceive any environment change

**messages** by exploiting message passing, Jason agents can exchange data structures belonging to the BDI architecture, such as goals, plans, and beliefs. Neither explicit protocols nor specific abstractions to objectively govern the interaction space are provided by Jason

**environment** unlike Jade, Jason supports the environment as a first-class abstraction in MAS modelling, by automatically updating agent beliefs based upon *perception*, and by providing an environment layer offering an *acting API*. Nevertheless, the environment layer has to be programmed in Java, since no high-level, specific architectural component is provided by Jason

**events** somehow hidden inside Jason agent architecture, the notion of *event* is what drives agent internal reasoner: almost everything is an event in Jason, from beliefs addition/removal to plan triggering. However, events are *internal to agents*—Subsection 4.1.3 describes the (possibly more general) notion of event

Figure 4.6 depicts how the Jason architectural model can be mapped upon the architecture in Figure 4.2: the environment layer is simply based on Java, with no higher-level abstraction provided, furthermore no explicit abstraction to govern the interaction space is given neither—other than message passing and goal/plan/beliefs sharing.

[3]`http://jason.sourceforge.net/`

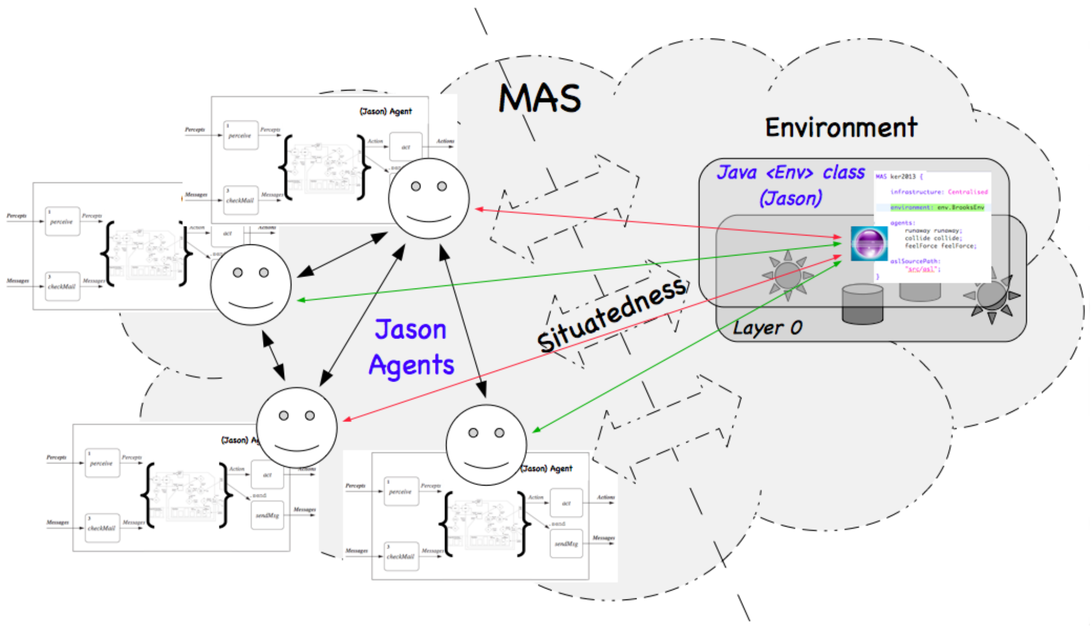


Figure 4.6: In Jason, there is no abstraction for the interaction space whereas a (limited) layer of abstraction for the environment is given through Java objects. Social dependencies are mostly governed by *goal/plan/beliefs sharing*, whereas situatedness is achieved through a *Java-programmed environment layer* featuring individual agents' (simulated) sensors and actuators [MO15b].

Thus, Jason coordination model is in principle a *subjective* one, since agents themselves are in charge of properly coordinate: however, coordination may occur either directly, by means of message passing, or indirectly, that is, mediated by environment perception and action, as in stigmergy-coordinated MAS [Omi12].

Situatedness is still a feature belonging to individual agents, but, unlike JADE, it is achieved through the environment abstractions of sensors ($\rightarrow$ *perception*) and actuators ($\rightarrow$ *action*)—although no specialised architectural component is given except for an abstract Java class.

**Activities, dependencies, environment change in CArtAgO** CArtAgO[4] [RVO07] is a Java-based framework and infrastructure based on the A&A (agents & artefacts) meta-model [ORV08]. The A&A meta-model introduces *artefacts* as the tools that agents use to enhance their own capabilities, for achieving their own goals [OPRV09]—as human beings do with their tools [Nar96]. So, artefacts can be used to (computationally) represent any kind of environmental resource within a MAS in a uniform way: from sensors to actuators, from databases to legacy OO applications, from real-world objects to virtual blackboards.

[4] `http://cartago.apice.unibo.it`

Given its focus on artefact design, implementation, and run-time support, rather than on agent development, CArtAgO is designed so as to be as much orthogonal as possible w.r.t. the agent-platform. Nevertheless, Java bridges exist, e.g., towards both JADE and Jason. Correspondingly, CArtAgO main architectural components are:

**artefacts** artefacts are the basic bricks in the A&A meta-model, then the basic bricks in the CArtAgO framework, too. Technically, CArtAgO artefacts are Java objects equipped with Java annotations, exploited by CArtAgO run-time infrastructure to recognise available operations (aka *effectors*) and generate observable events (aka *perceptions*). Thus, artefacts are the tools for MAS designers to properly model and implement the portion of the environment agents can control / should deal with

**workspaces** workspaces play the role of the topological containers for agents and artefacts, representing the agent working environments. In particular, since every agent and every artefact are always associated to a workspace in CArtAgO, workspaces can be used to define the scope of event generation/perception for agents and artefacts

**agent bodies** by exposing *effectors* API and enabling *perception* of environment (artefact-generated) events (through sensors), CArtAgO agent bodies are the architectural components enabling agent interaction with artefacts—thus, in the very end, situatedness, at least from the individual agent viewpoint. Technically, agent bodies work as Java bridges towards existing agent platforms, such as JADE and Jason

**observable events** while implementing artefacts, CArtAgO programmers can define events to be generated in response to specific operation invocations as well as observable states to monitor for changes—generating events as soon as the state changes. Events can then be captured by sensors linked to agent bodies, either proactively got by agent minds, or automatically dispatched to the agents explicitly focussing on the artefact source of the event

Figure 4.7 below depicts the aforementioned architectural components along with their mutual relationships. In summary, CArtAgO handles situated interaction by providing artefacts as a means to mediate agents interaction with their environment—via agent bodies. As far as coordination is concerned, neither built-in services nor abstractions are given specifically for that purpose: coordination could be then achieved only by means of ad-hoc-designed social artefacts—e.g., by implementing a channel, a mailbox, a shared blackboard as a coordination artefact.

### 4.1.3 A Reference Architecture

A well-founded software architecture relies on a well-defined meta-model, that is, the set of concepts and abstractions the architecture is grounded on. Besides setting the architecture within a sound conceptual framework, this makes it possible to place the proposed

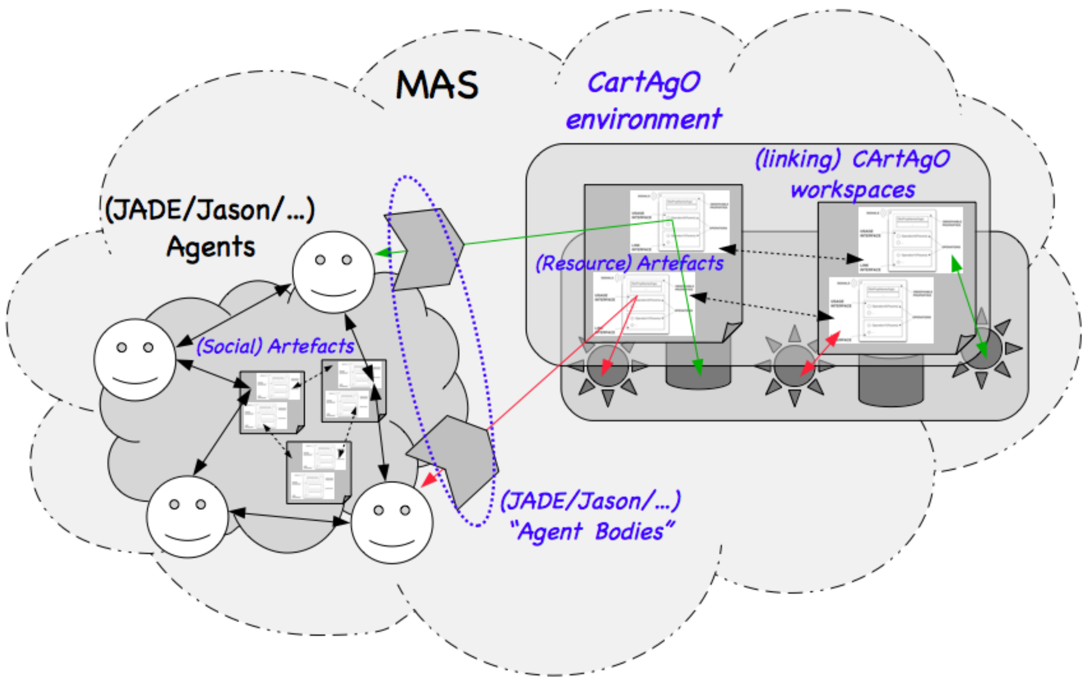


Figure 4.7: In CArtAgO, MAS environment is modelled through artefacts. Thus, situatedness is the consequence of artefacts use/observation by agent bodies. As regards interactions, artefacts can be used as well, even though they are not explicitly devoted to coordination [MO15b].

architecture within the historical perspective taken in Subsection 4.1.1, in particular, by mapping the architecture onto the meta-model depicted in Figure 4.4.

**Meta-model** In the widespread acceptation of MAS nowadays, *agents*, *environment* and *societies* are the three fundamental abstractions around which MAS should be modelled and engineered [ORV$^+$04a]. While the meta-model still adopts those three abstractions as its reference conceptual framework, there are three core concepts that motivate the architecture, which the meta-model should account for:

**activities** *goal-directed/oriented* proceedings resulting into *actions* of any sort, which make things happen within a MAS; through actions, activities in a MAS are *social* [Cas98] and *situated* [Suc87]

**environment change** the (possibly unpredictable) variations in the properties or structure of the world surrounding a MAS that affect it in any way; variations do not express any specific goal, either because this does not exist, or because it has / can not be modelled in the MAS

**dependencies** in any non-trivial MAS, activities *depend* on other activities (*social dependencies*), and on environment change (*situated dependencies*); thus, dependencies both motivate and cause *interaction*—both social and situated interaction

The core notion that links the architecture to the meta-model is the one of *event*:

**events** despite their intrinsic diversity, actions and environment change constitute altogether the only sources of dynamics in a MAS—what makes everything happen; in order to provide a uniform view of MAS dynamics, and a simpler modelling of social and situated dependencies, both actions and environment changes are represented here as *events*

**Architecture** The proposed event-driven architecture for MAS is depicted in Figure 4.8, and is made of the following components:

**agents** agents are the autonomous entities in charge of the (goal-directed/oriented, social, and situated) activities, that is, undertaking the course of actions aimed at achieving their own goal. Agents are characterised by their own goals, and articulate their activities in term of actions. Actions affects either other agents or the

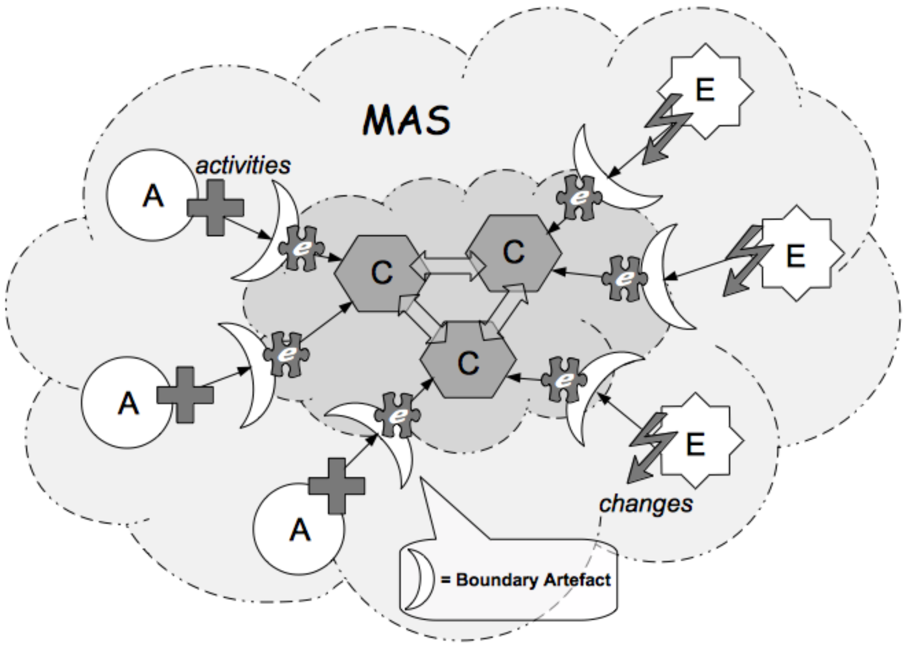


Figure 4.8: *Boundary artefacts* take care of *events* in the proposed event-driven architecture [MO15b]: they are in charge of mapping activities as well as changes onto events, to be uniformly handled by the *coordination artefacts* within the MAS —responsible for handling dependencies of any sort.

environment, leading to social (agent-agent) and situated (agent-environment) interactions. Agents in a MAS are in principle *heterogeneous* and *unpredictable*: either by design or by necessity—as in the case of open MAS

**environment resources** while the notion of environment is quite a hazy one, dealing with MAS makes it possible here to model it as populated by many items, here called *resources*, capable of interacting in some way with a computational system. Resources encapsulate the properties of the environment that are of interest for the agents in a MAS; as such, they are the instruments by which agent actions affect environment properties, and the sources of (possibly unpredictable) changes of the environment surrounding the MAS. Also environment resources in a MAS are heterogeneous and unpredictable: the variety and dynamics of resources accounts for the diversity and dynamics of MAS environment

**boundary artefacts** heterogeneity of agents and of resources, along with their dissimilar nature, might lead to theoretical and practical problems when dealing with social and situated dependencies. In order to represent all the sources of change in a MAS in a uniform way, without neglecting specificity, a mediating architectural abstraction is first of all required. This is in fact the role of *boundary artefacts* as the architectural components representing agents as well as environmental resources within the MAS. As such, they work as an interface between the agent (or, the environmental resource) and the MAS interaction space, mapping agents' activities and environment changes into events, dispatching them to the *coordination artefacts*, collecting the outcomes of dependency resolution (coordination), then dispatching outcomes back to agents and environmental resources

**coordination artefacts** coordination artefacts are the components in charge of dealing with *dependencies* in MAS—both social and situated dependencies. As both agents and resources are represented by boundary artefacts within a MAS, coordination artefacts actually work by handling dependencies between events representing agent activities and environment changes in a *uniform* way. Thus, dependencies are handled by managing interaction among agents (*social coordination*) and between agents and environmental resources (*situated coordination*)

**events** first of all, boundary artefacts generate *external* events, as the data structures reifying agent activities and environment changes. Then, coordination artefacts generate *internal* events, which represent MAS internal activity, related to the management of both social and situated interaction. As coordination artefacts handle events in order to manage dependencies, boundary artefacts translate events coming from coordination artefacts that target either agents or environment resources, thus taking care of heterogeneity of agents and resources. Correspondingly, every event records all the information potentially relevant to coordination, such as its cause (either the agent action or the environment change), spatio-temporal data (when it happened and where), the source and the target entities involved (agents, environment, coordination artefacts), and so on

Summing up, in the abstract architecture depicted by Figure 4.8 *(i) agents* and *environmental resources* interact through *boundary artefacts*, *(ii)* mapping agent activities and environment changes into *events*, *(iii)* which are then handled by *coordination artefacts*.

## 4.2 Environmental Situatedness in TuCSoN

In this section a discussion about how the TuCSoN coordination infrastructure deals with situatedness related issues is provided, from the architectural standpoint, focussing on situatedness w.r.t. the MAS environment, by implementing the reference architecture outlined. In particular, Subsection 4.2.1 describes the architectural components supporting situatedness in TuCSoN, then Subsection 4.2.2 thoroughly describes the interaction flow among the aforementioned components while enforcing situated coordination, finally Subsection 4.2.3 details how to implement situated coordination in TuCSoN and ReSpecT on a demonstrative use case.

### 4.2.1 Architectural Overview

TuCSoN represents quite a consistent example of implementation of the proposed event-driven architecture. In fact, TuCSoN main architectural abstractions (as well as run-time components) are—as depicted in Figure 4.9 below:

**agents** any computational entity choosing the TuCSoN coordination services [VO06] to interact with a TuCSoN-coordinated MAS is a TuCSoN *agent*. This choice, when admissible, is modelled by assigning an ACC (see below) to the agent, which mediates its interaction with the MAS. Agents actions result into *coordination operations*, in principle targeting the coordination media (tuple centres), actually handled by the associated ACC

**probes** environmental resources in TuCSoN are called *probes*. They are uniformly dealt with either as sources of perceptions (like *sensors*) or targets of actions (like *actuators*)—or even both. Actions over probes are called *situated operations*, and are operated by *transducers* (see below): in fact, as for agents, probes do not directly interact with the MAS, but through transducer mediation

**ACC** *Agent Coordination Contexts* [Omi02] are TuCSoN boundary artefacts devoted to agents. ACCs both enable and constraint agents interaction capabilities by exposing an API including only the *admissible coordination operations*—according, e.g., to the agent role in the MAS [VOR07]. In particular, ACCs map coordination

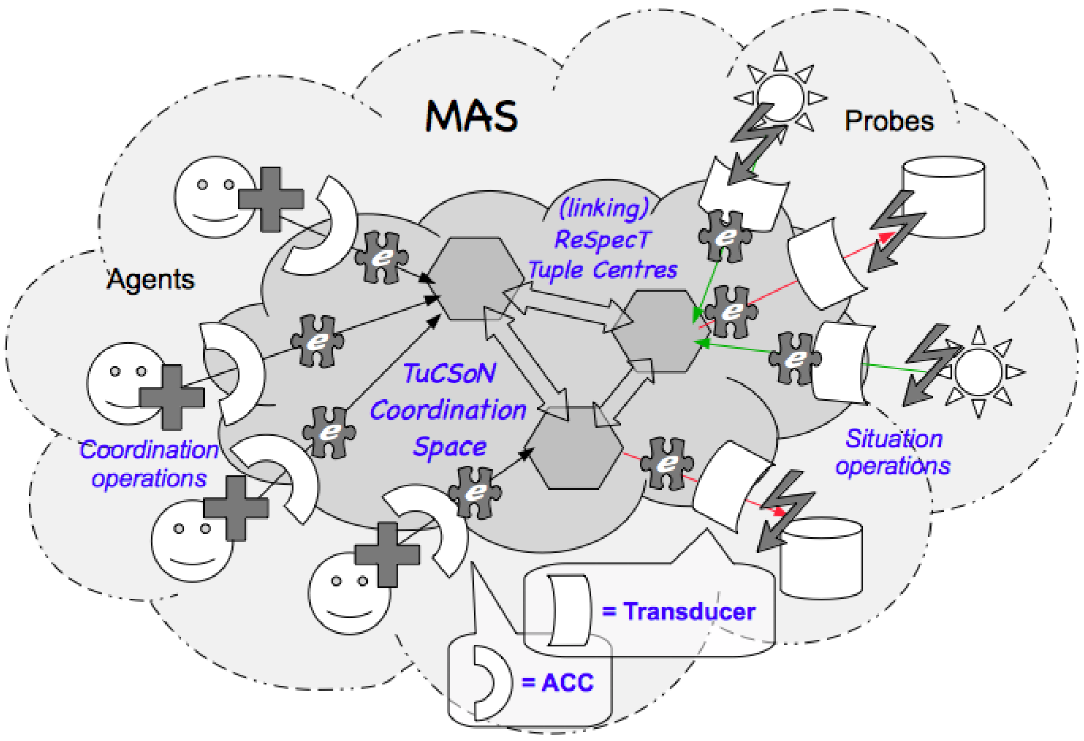


Figure 4.9: In TuCSoN, both social (agent-agent) and situated (agent-environment) interaction is *mediated* by ReSpecT tuple centres [MO15b]. The unifying abstractions in TuCSoN are ACC and transducers (as *boundary artefacts*), the TuCSoN *event model*, and the ReSpecT tuple centres (as *coordination artefacts*).

operations into events, dispatch them to the coordination medium, wait for the outcome of dependency resolution (that is, coordination), then send back to the agent the operation results. ACCs are also fundamental to guarantee and preserve agent autonomy [Omi02]: while the agent is free to choose its course of actions, its associated ACC translates the corresponding events into the MAS interaction space only in case they comply with the agent role, and the state of its interaction [VOR07]

**transducers** analogously to ACC for agents, TuCSoN *transducers* [CO09] are the boundary artefacts geared toward probes. Each probe is assigned a transducer, which is specialised to handle events from that probe, and to act on probes through situated operations. So, in particular, transducers translate probes property changes into events, which are modelled through the same general event model used for agents' operations—thus leading to a uniform MAS interaction / coordination space

**events** TuCSoN adopts and generalises the ReSpecT event model, depicted in Table 4.1, representing in a uniform way both events from agents actions and events from

| | | |
|---|---|---|
| $\langle \mathit{Event} \rangle$ | ::= | $\langle \mathit{StartCause} \rangle$ , $\langle \mathit{Cause} \rangle$ , $\langle \mathit{Evaluation} \rangle$ |
| $\langle \mathit{StartCause} \rangle$ , $\langle \mathit{Cause} \rangle$ | ::= | $(\langle \mathit{Activity} \rangle \mid \langle \mathit{Change} \rangle)$ , $\langle \mathit{Source} \rangle$ , $\langle \mathit{Target} \rangle$ , $\langle \mathit{Time} \rangle$ , $\langle \mathit{Space{:}Place} \rangle$ |
| $\langle \mathit{Source} \rangle$ , $\langle \mathit{Target} \rangle$ | ::= | $\langle \mathit{AgentId} \rangle \mid \langle \mathit{CoordArtefactId} \rangle \mid \langle \mathit{EnvResId} \rangle \mid \perp$ |
| $\langle \mathit{Evaluation} \rangle$ | ::= | $\perp \mid \{\langle \mathit{Result} \rangle\}$ |

Table 4.1: ReSpecT situated *event model* [MO15b].

| | | |
|---|---|---|
| $\langle \mathit{Activity} \rangle$ | ::= | $\langle \mathit{Operation} \rangle \mid \langle \mathit{Situation} \rangle$ |
| $\langle \mathit{Operation} \rangle$ | ::= | `out(` $\langle \mathit{Tuple} \rangle$ `)` $\mid$ `(in` $\mid$ `rd` $\mid$ `no` $\mid$ `inp` $\mid$ `rdp` $\mid$ `nop` $\mid \ldots$ `)` `(` $\langle \mathit{Template} \rangle$ [`,` $\langle \mathit{Term} \rangle$] `)` |
| $\langle \mathit{Situation} \rangle$ | ::= | `env(` $\langle \mathit{Key} \rangle$ `,` $\langle \mathit{Value} \rangle$ `)` |
| $\langle \mathit{Change} \rangle$ | ::= | `env(` $\langle \mathit{Key} \rangle$ `,` $\langle \mathit{Value} \rangle$ `)` $\mid$ `time(` $\langle \mathit{Time} \rangle$ `)` $\mid$ `from(` $\langle \mathit{Space} \rangle$ `,` $\langle \mathit{Place} \rangle$ `)` $\mid$ `to(` $\langle \mathit{Space} \rangle$ `,` $\langle \mathit{Place} \rangle$ `)` |

Table 4.2: ReSpecT *triggering events* [MO15b].

changes in the environment—as further explained by Table 4.2. Events are the connectors of the architecture, the run-time data structure reifying any relevant information about the activity or change that generated the events themselves. In particular, TuCSoN events record: the *immediate* and *primary* cause of the event [RVO08], its outcome, who is the source of the event, who is its target, when and where the event was generated. Thus, the spatio-temporal fabric is always accounted for by any TuCSoN event. Being based on the ReSpecT event model, TuCSoN events are fully inspectable by ReSpecT reactions, hence seamlessly integrated with the tuple centre programmable machinery: this means that the coordination medium – in the case of TuCSoN, ReSpecT tuple centres – is able to perform computations over MAS events

**tuple centres** ReSpecT tuple centres [OD01b] are the TuCSoN architectural component working as the coordination artefacts. They are run by the TuCSoN middleware to rule and decouple (in *control, reference, space* and *time*) dependencies between agents' activities as well as environment changes—in other words, both social as well as situated interactions [OM13]. By adopting ReSpecT tuple centres, TuCSoN relies on *(i)* the ReSpecT language to program coordination laws, and *(ii)* the ReSpecT situated event model to implement events

As depicted in Figure 4.9, TuCSoN tackles the issues of coordination and situatedness in open MAS with a *uniform and coherent* set of abstractions and architectural components:

ACC and transducers represent coordinated entities (agents as well as the environment) in the MAS, and translate activities and changes coming from them in a common event model (ReSpecT situated event model), while tuple centres coordinate both social dependencies as well as situated dependencies by allowing the management of events to be programmed using a situatedness-aware coordination language.

As a result, by implementing the event-driven architecture, TuCSoN promotes both *objective coordination* and *situatedness at the MAS level.*

### 4.2.2 Flow of Interactions

**Agent side** The *agent side* of a TuCSoN-coordinated MAS is basically represented by the run-time relationships between agents, ACC, and tuple centres. First of all, as depicted in Figure 4.10, TuCSoN agents have to acquire an ACC before issuing any sort of coordination operation towards the TuCSoN infrastructure. They do so by asking the TuCSoN middleware to release an ACC. Whether an ACC is actually released, and which one among those available[5] is dynamically determined by the TuCSoN middleware itself, based upon the agent request and its *role* inside the MAS [VOR07].

Once a TuCSoN agent obtains an ACC, all its interactions are mediated by the ACC itself. In particular, as depicted in Figure 4.11, in the case a coordination operation is requested through a *synchronous invocation*:

1. first of all (messages $2 - 2.1.2$), the target tuple centre associated to the ACC is dynamically instantiated by the TuCSoN run-time infrastructure, and its network address given to the ACC for further reference

[5]See the TuCSoN official guide at `http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide`.

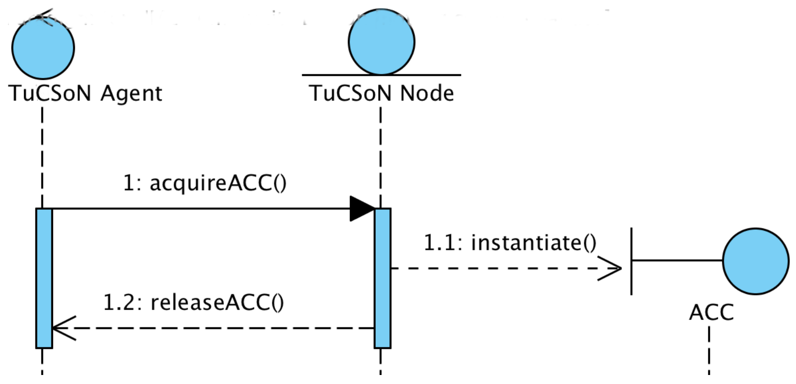


Figure 4.10: ACC acquisition by TuCSoN agents [MO14c]. Nothing can be done by an agent with the TuCSoN middleware prior to ACC acquisition.

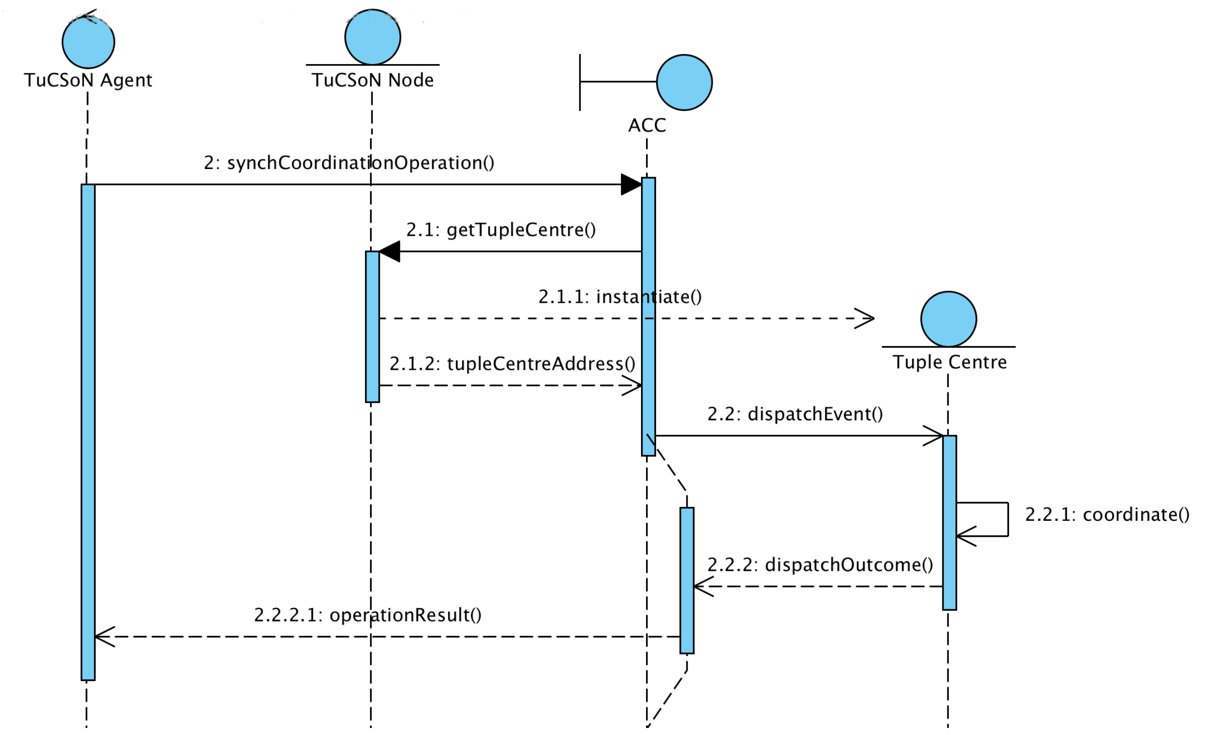


Figure 4.11: Synchronous operation invocation [MO14c]. The control flow is released back to the agent only when the operation result is available—thus, only when the coordination process ends.

2. then (message 2.2), the ACC takes charge of building the corresponding event and of dispatching it to the tuple centre target of the interaction
3. finally (messages 2.2.1 – 2.2.2.1), the ACC is notified when the outcome of the coordination operation requested is available – after a proper coordination stage, possibly involving other events from other entities – so that it can send the operation result back to the agent

Only the coordination operation request from the agent to its ACC is a *synchronous method call*: any other interaction is *asynchronous* as well as *event-driven*. This is necessary in every open and distributed scenario, and enables uncoupling in control, reference, space, and time. Nevertheless, in this scenario – synchronous operation invocation – the control flow of the caller agent is retained by the ACC as long as the operation result is not available (message 2.2.2.1).

Conversely, Figure 4.12 depicts the *asynchronous invocation* scenario: the only difference w.r.t. the synchronous one lays in the fact that the control flow is given back to the caller agent as soon as the corresponding event is dispatched to the target tuple centre (messages 3.3 – 3.4). The actual result of the requested coordination operation is

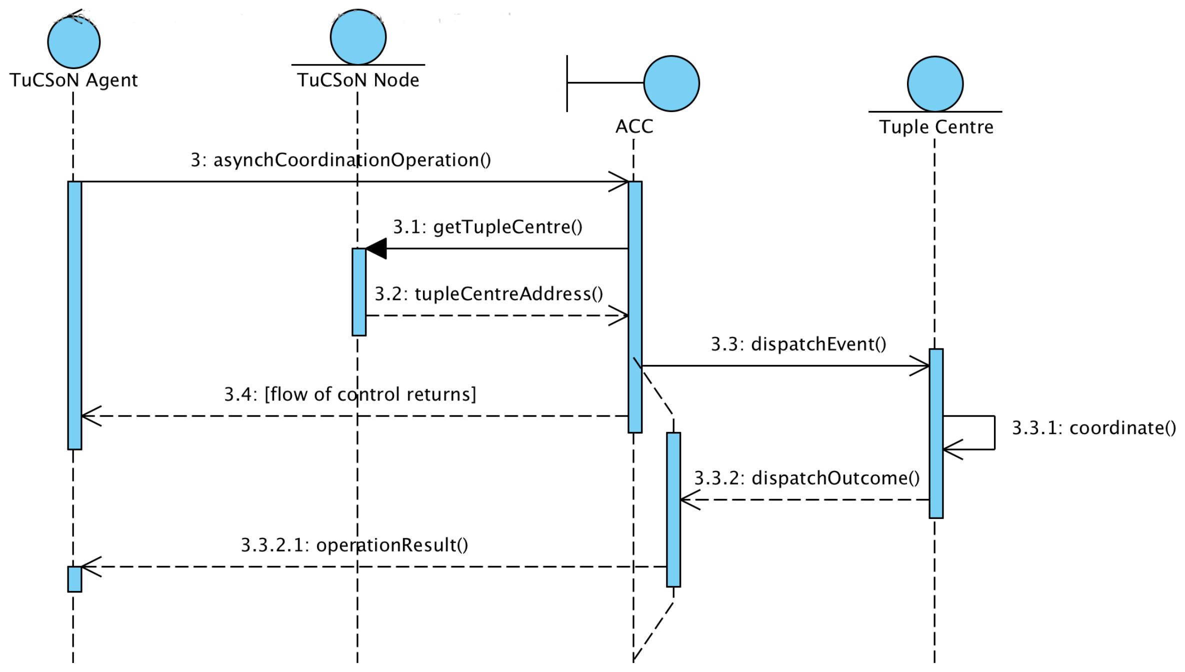


Figure 4.12: Asynchronous operation invocation [MO14c]. The control flow is released back to the agent as soon as the event related to the request is generated and dispatched by the ACC.

dispatched to the agent as soon as it becomes available, asynchronously (message 3.3.2.1). TuCSoN lets client agents choose which semantics to use for their coordination operations invocation, either synchronous or asynchronous, so as to preserve their autonomy.

As a side note, the scenario depicted in Figure 4.12 assumes that the target tuple centre is already up and running – e.g., as a consequence of a previous operation invocation – thus, the TuCSoN node simply retrieves its reference, and passes it to the ACC.

Whenever an agent no longer needs TuCSoN coordination services, it should release its ACC back to TuCSoN middleware, which promptly destroys it in order to prevent resources leakage—as depicted in Figure 4.13.

It should be noted that there is no way to re-acquire an already-released ACC – e.g., to restore interactions history –, since whenever an ACC is requested a new one is created and assigned. Since ACC are used to represent and identify agents within a TuCSoN-coordinated MAS, an agent obtaining an ACC multiple times is recognised every time as a new agent by the TuCSoN middleware.

Summing up, designers of agents exploiting TuCSoN should make their agents: *(i)* acquire an ACC; *(ii)* choose the invocation semantics for each coordination operation they perform, and *(iii)* expect operations result to be available accordingly; *(iv)* release their ACC when TuCSoN services are no longer needed—at agents shutdown TuCSoN automatically releases orphan ACCs.

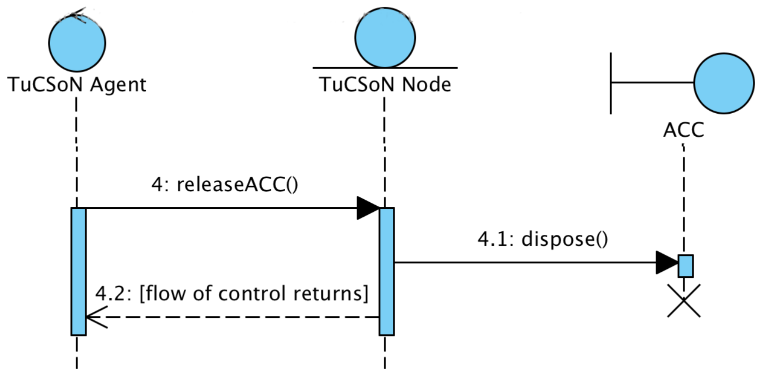


Figure 4.13: ACC release by TuCSoN agents [MO14c]. Nothing can be done by an agent with the TuCSoN middleware after ACC release.

**Environment side** On the *environment side* of the TuCSoN architecture, agents and ACCs are replaced by probes and transducers, respectively. Thus, first of all, probes should register to the TuCSoN middleware in order to get their transducer and interact—as depicted by Figure 4.14 below. After probe registration, any interaction resulting from environmental property change affecting the MAS is mediated by the transducer.

Figure 4.15 depicts the interaction among TuCSoN run-time entities in the case of a sensor probe, thus a sensor transducer, whereas Figure 4.16 shows the case of an actuator probe. By comparing the two pictures, the flow of interactions is almost the same, except for the first invocation, which depends on the nature of the probe—either sensor (Figure 4.15) or actuator (Figure 4.16).

In particular, a perception by a sensor probe works as follows:

1. first of all (messages $2 - 2.1.2$), the target tuple centre associated to the transducer is dynamically instantiated by the TuCSoN run-time infrastructure, and its network address passed to the transducer for further reference

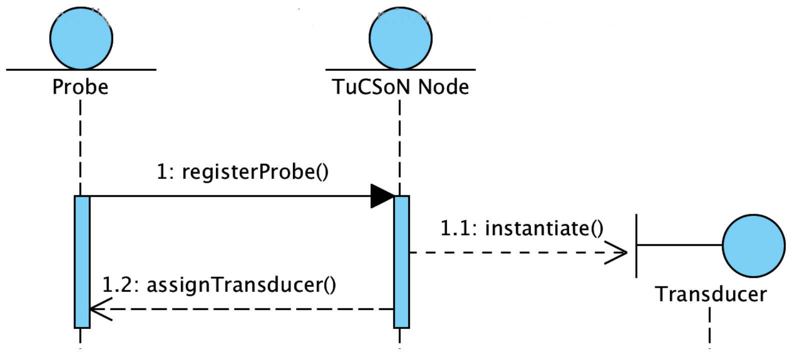


Figure 4.14: Probes registration and transducers association. No events can be perceived nor actions undertaken on a probe prior to transducer association [MO14c].

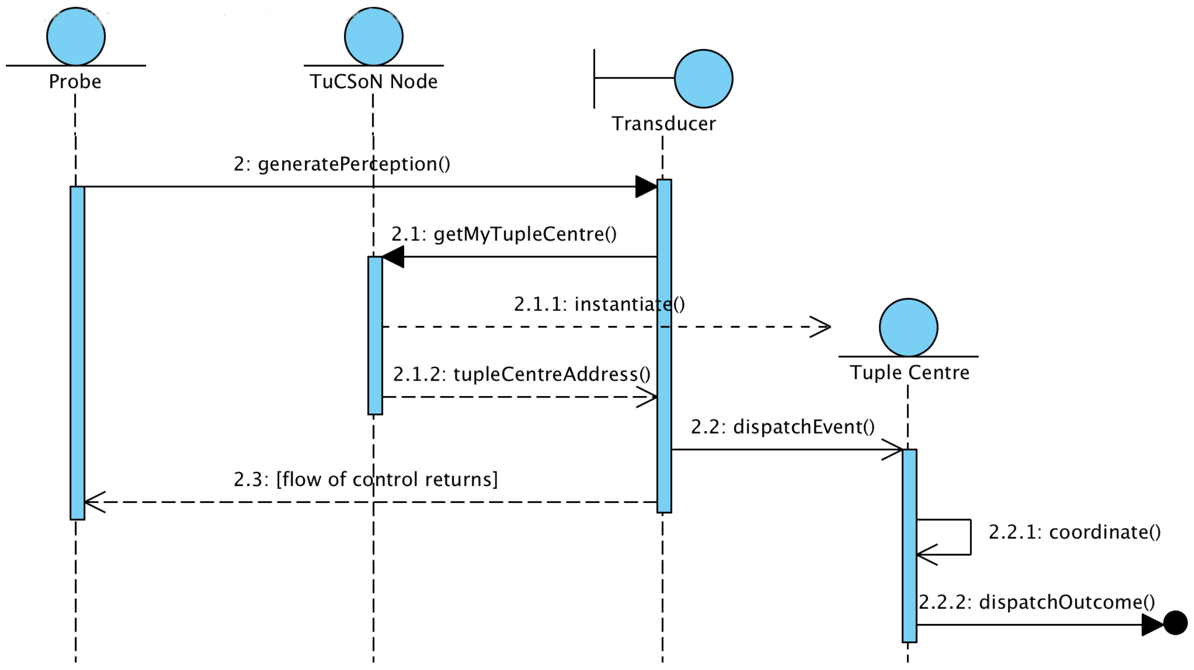


Figure 4.15: Sensor probe interaction [MO14c]. The control flow returns to the probe as soon as the environmental event is generated and dispatched by the transducer, thus, everything happens asynchronously.

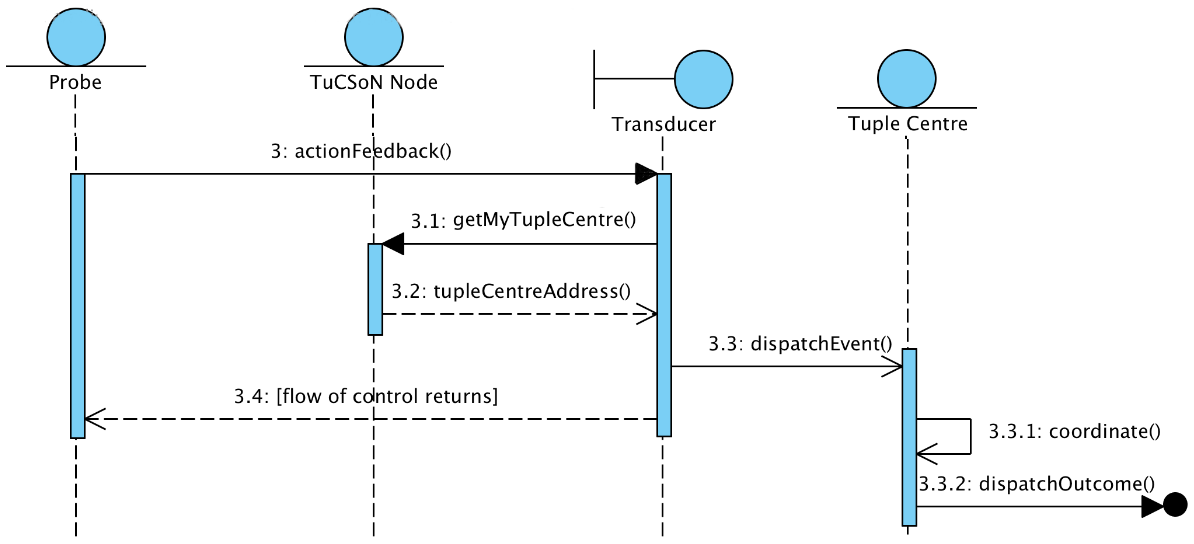


Figure 4.16: Actuator probe interaction [MO14c]. Again, everything happens asynchronously.

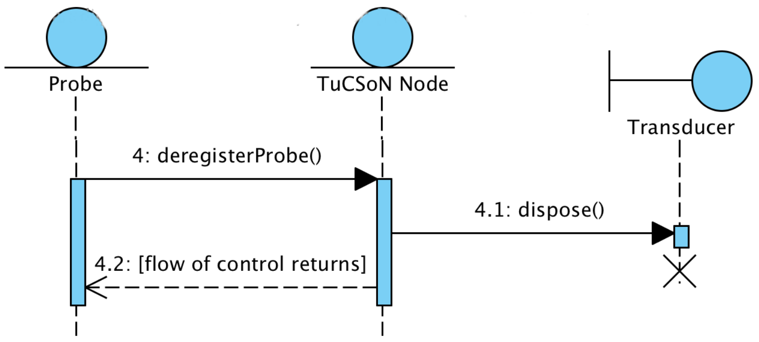


Figure 4.17: Probe deregistration [MO14c]. Nothing can be either sensed or effected by the MAS upon the deregistered probe, since the mediating transducer is no longer running.

2. then (message 2.2), the transducer builds the event corresponding to the perception operation, and dispatches it to the tuple centre target of the interaction

3. finally (messages 2.2.1 – 2.2.2), the tuple centre enacts the coordination process triggered by the event (if any), properly dispatching its outcome

As far as probe interaction is concerned, there is no distinction between synchronous or asynchronous semantics. In fact, being representations of environmental resources, probes are not supposed to expect any feedback from the MAS: they simply cause / undergo changes that are relevant to the MAS. For this reason, the semantics of situation operations invocation on probes is always asynchronous. As depicted in Figure 4.15 and Figure 4.16 in fact, the control flow is always returned to the probe as soon as the corresponding event is generated.

When a probe is no longer needed, it should be deregistered from TuCSoN, which subsequently destroys the associated transducer—as depicted in Figure 4.17 above. Wrapping up, TuCSoN situatedness services require MAS designers to: *(i)* always register probes causing their transducer instantiation; *(ii)* be aware that environmental events are always generated asynchronously; *(iv)* deregister probes when they are no longer needed—no automatic deregistration is performed by the TuCSoN middleware.

**Between agents and environment: situated coordination** Putting together the agent and the environment side of the TuCSoN event-driven architecture, Figure 4.18 and Figure 4.19 depict the synchronous interaction of an agent with a sensor, and the asynchronous interaction of an agent with an actuator, respectively.
By inspecting the whole interaction sequence, one could see how *(i)* TuCSoN ACC and transducers play a central role in supporting distribution and uncoupling of agents and probes within the MAS, and *(ii)* how TuCSoN tuple centres and the ReSpecT language are fundamental to support *situated objective* coordination [Sch01, OO03].

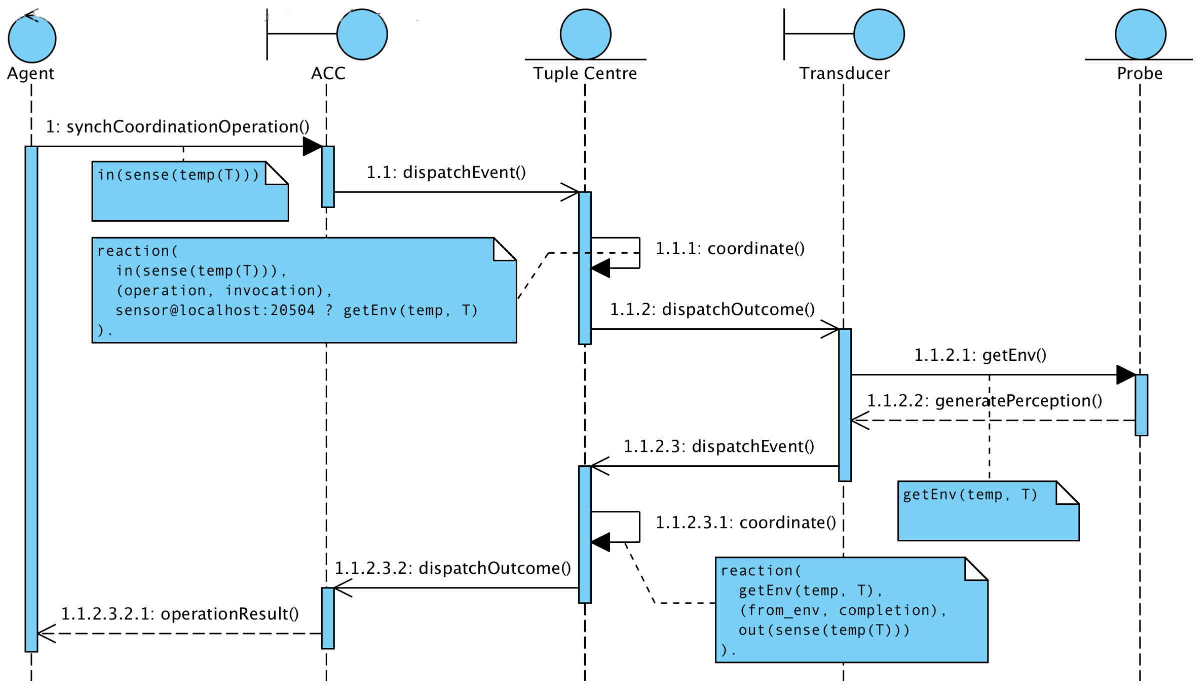


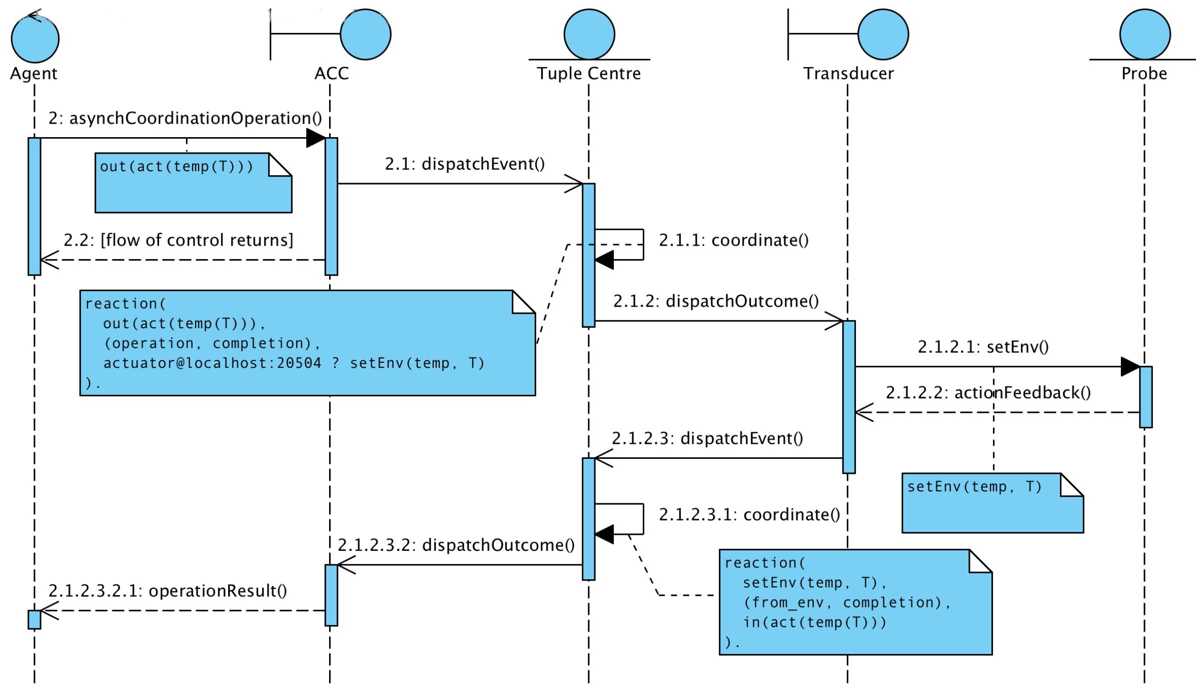
Figure 4.18: Synchronous situation operation querying a sensor [MO14c]. ReSpecT plays a fundamental role in binding both the agent coordination operation to its corresponding situation operation (annotation in step 1.1.1) and the probe response back to the agent original request (annotation in step 1.1.2.3.1).

In particular, in Figure 4.18 the agent is issuing a synchronous coordination operation request involving a given tuple (`sense(temp(T))`)—message 1.

After event dispatching (all the dynamic instantiation interactions have been left out for the sake of clarity), the tuple centre target of the operation reacts to invocation by triggering the ReSpecT reaction in annotation 1.1.1, which generates a situated event (step 1.1.2) aimed at executing a situation operation (`getEnv(temp, T)`) on the probe (`sensor`)[6]. The transducer associated to the tuple centre and responsible for the target probe intercepts the event and takes care of actually executing the operation on the probe (message 1.1.2.1). The sensor probe reply (message 1.1.2.2) generates a sequence of events propagations terminating in the response to the original coordination operation issued by the agent (message 1.1.2.3.2.1).

It is worth noticing the role of the tuple centre in supporting situatedness: in fact, step 1.1.2.3.1 properly reacts to the completion of situation operation `getEnv(temp, T)` by the sensor probe, emitting exactly the tuple originally requested by the agent (`sense(temp(T))`).

[6]Primitive `getEnv(temp, T)` is an alias for `env(temp, T)`, making explicit that the operation is meant to perceive something from the environment.



```
out(act(temp(T)))
```

```
reaction(
  out(act(temp(T))),
  (operation, completion),
  actuator@localhost:20504 ? setEnv(temp, T)
).
```

```
setEnv(temp, T)
```

```
reaction(
  setEnv(temp, T),
  (from_env, completion),
  in(act(temp(T)))
).
```

Figure 4.19: Asynchronous situation operation commanding an actuator [MO14c]. As in Figure 4.18, ReSpecT role in enabling situatedness is visible in annotations 2.1.1 and 2.1.2.3.1.

In Figure 4.19 the sequence of interactions as well as the annotations are very similar to those in Figure 4.18. In particular, annotation 2.1.1 shows how the ReSpecT reaction triggering event matches the event raised as a consequence of agent coordination operation request (`act(temp(T))`), while annotation 2.1.2.3.1 highlights how the tuple centre maps the situation operation outcome (`setEnv(temp, T)`)[7] in the original tuple (`act(temp(T))`) through a proper ReSpecT reaction. The only differences w.r.t. Figure 4.18 are the asynchronous invocation semantics used by the agent and the actuator nature of the interacting probe—thus, messages 2.1.2.1 and 2.1.2.2.

In summary, ReSpecT is fundamental to program TuCSoN tuple centres so as to correctly bind coordination operations with situation operations – while preserving interacting entities' autonomy –, ultimately supporting agent-environment interactions, thus, situatedness, in distributed, open MAS.

**Implicit methodology: explanatory example** In this section TuCSoN is deployed within a smart home scenario, with the aim of sketching the methodology implicitly suggested by TuCSoN when engineering situated coordination. In particular, a discussion

[7]Primitive `setEnv(temp, T)` is an alias for `env(temp, T)`, making explicit that the operation is meant to affect something in the environment.

about how requirement analysis, solution modelling, & solution design can be undertaken with the TuCSoN approach and its architecture is provided—implementation of a restricted scenario is left for next section (Subsection 4.2.3).

Imagine many different smart appliances (e.g., smart fridge, smart thermostat, smart lights, smart A/C, etc.) scattered in an indoor environment (e.g., a flat). Either inhabitants have an Android smartphone or a desktop PC is available in the environment (or even both): this ensures the TuCSoN middleware can be running, being the JVM its only requirement. Some kind of connection is available at least between each appliance and the smartphone/desktop—appliances may also be connected together to improve distribution thus resilience, although not strictly necessary.

Inhabitants want the smart home system to self-manage toward a given goal (e.g., always optimise power consumption) according to their preferences (e.g., prefer turning on fans rather than switching A/C on), while keeping the capability to control it despite self-management, when desired (e.g., "I want frozen beers now, forget about power consumption!").

Starting from requirement analysis, the core desiderata of the proposed scenario is, essentially, that environmental resources (e.g., the A/C, the fridge, etc.) should be able to adapt to environment change (e.g., temperature drops, food depletion, etc.) as well as users' actions (e.g., "I'm coming home late, order pizza"), striving to achieve a system goal (e.g., optimise inhabitants comfort) while accounting for each user's desires.

According to the TuCSoN meta-model as described in Subsection 4.1.3, this can be interpreted as follows:

- users continuously and unpredictably perform their daily *activities*...
- ...which may *depend* on the environment being in a certain enabler state (e.g., food must be available to enable cooking dinner), as well as may both impact and be affected by the environment...
- ...causing some *change* to happen (e.g., "since I'll be late, delay lights turning on", "there is no food, thus I must go to the grocery shop")

Once recognised that activities and environment change both make *events* happen, thus managing dependencies between activities and change ultimately resort to managing events, a perfect and complete match with TuCSoN reference meta-model is achieved.

As far as the design phase of the software engineering methodology is concerned, once the problem at hand (smart home appliances coordination) has being re-interpreted in light of the TuCSoN meta-model, TuCSoN architecture provides all the necessary components to design a solution. Thus:

- users activities are generated by *agents*, making it possible to also ascribe goals to actions, and mediated by *ACCs*, enabling and constraining interactions according to the system goals

- changes in the environment are generated by *probes* and mediated by *transducers*, enabling uniform representation of properties despite appliances heterogeneity
- activities and change (thus ACCs and transducers) generate *events* as their own representation within the situated MAS coordinated by TuCSoN, which are then managed by *tuple centres* suitably programmed with adaptable coordination laws

Consequently: agents are deployed to users' personal devices (smartphone/desktop pc), probes are deployed to home appliances, ACCs and transducers are deployed either on-board along with agents and probes, respectively (e.g., on the smartphone), or remotely (e.g., on the desktop), tuple centres are deployed again either on board or remotely, all performing activities and enacting/undergoing changes generating events, automatically handled by TuCSoN according to the designed coordination laws.

It should be noted that a similar scenario has been depicted in [Den14], although much more thoroughly. There, TuCSoN is taken as the underlying infrastructure on top of which the Butlers Architecture for smart home management is deployed. In particular, agents are used therein to model environmental resources as well (e.g., home appliances), whereas the proposed approach would have modelled them as probes, thus handled (coordinated) within the MAS by transducers.

Benefits of doing so are not limited to a cleaner architecture and separation of concerns, but also include smaller computational load (transducers are simpler than full-fledged agents), better run-time adaptiveness (replacing a transducer is much simpler than replacing an agent), improved management of heterogeneity (despite probes API differences, transducers map any event to a common event model).

### 4.2.3 Implementation Methodology

In this section it is discussed how to implement probes and transducers, how to make the TuCSoN middleware aware of them, and how to program TuCSoN tuple centres to inspect and manipulate TuCSoN situatedness-related events, ultimately engineering situated coordination.

Generally speaking, implementing situated coordination within a MAS using TuCSoN, amounts to deal with the following tasks:

1. implementing the probes—sensor probes and actuator probes. Typically, this does not require implementing, e.g., the software drivers for the resources: designers can simply wrap existing drivers in a Java class interacting with TuCSoN transducers, implementing the `ISimpleProbe` Java interface (Figure 4.20)
2. implementing the transducers associated to the sensor and actuator probes, by extending the TuCSoN `AbstractTransducer` Java class (Figure 4.22)

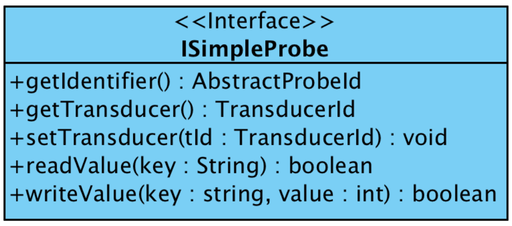

<<Interface>>
ISimpleProbe
+getIdentifier() : AbstractProbeId
+getTransducer() : TransducerId
+setTransducer(tId : TransducerId) : void
+readValue(key : String) : boolean
+writeValue(key : string, value : int) : boolean

Figure 4.20: Interface to be implemented by probes [MO14a].

3. interacting with the *transducer manager* singleton entity (Figure 4.25) to request its services, which is responsible for probes and transducers association in TuCSoN. The transducer manager listens to incoming requests for probes (de)registration and transducers (de)association, booting and setting up the two sibling sides of the transducer—the node and probe sides

4. programming TuCSoN tuple centres with ReSpecT in order to implement the coordination policies that ensure situated coordination of the MAS

As a running example, the scenario implemented within TuCSoN latest distribution[8], in package `alice.tucson.examples.situatedness`, is taken as a reference: its simplicity allows to clearly describe design and implementation issues without losing generality.

There, a situated, intelligent thermostat (`Thermostat.java`) is in charge of keeping a room temperature between 18 and 22 degrees. To this end, it interacts with a sensor (`ActualSensor.java`) and an actuator (`ActualActuator.java`): the former is queried by the thermostat to perceive the temperature, whereas the latter is prompted to change the temperature upon need. Both the sensor and the actuator, as probes, interface with the MAS (which, in this simple case, is represented by the thermostat TuCSoN agent alone) through one transducer each (respectively, `SensorTransducer.java` and `ActuatorTransducer.java`).

According to the TuCSoN approach, in order to promote distribution of the application logic, the transducers and the thermostat are associated each with their own tuple centre (`tempTc` for the thermostat agent, `sensorTc` for the sensor transducer and `actuatorTc` for the actuator transducer), suitably programmed through ReSpecT reactions (`sensorSpec.rsp` for the sensor transducer and `actuatorSpec.rsp` for the actuator transducer) that handle the specific interaction with the MAS.

Finally, the core logic of the application is implemented in `Thermostat` Java class.

Task 1 just requires MAS designers to implement the five methods of the `ISimpleProbe` interface (Figure 4.20 below) as a non-abstract Java class (Figure 4.21)—classes `ActualSensor.java` and `ActualActuator.java`:

`getIdentifier` retrieving this probe ID

[8]`TuCSoN-1.12.0.0301`, available at `http://tucson.unibo.it`.

`getTransducer` retrieving this probe associated transducer—if any

`setTransducer` to associate an existing transducer to this probe

`readValue` to *perceive* the resource—mandatory for sensors

`writeValue` to *act* on the resource—mandatory for actuators

Whereas the probe ID is assigned by the programmer at construction time, its association with the transducer occurs dynamically at run-time—hence the `setTransducer` method is usually called by the TuCSoN middleware. To operate on the probe, the methods `readValue` and/or `writeValue` (depending on whether the probe can behave as a sensor, an actuator, or both) should implement the logic required to interact with the actual probe—either a simulated environmental resource, or a real-world object. By completing task 1, the probe side of the transducer is partially implemented.

Since the transducer logic is fixed – in particular, capturing events from both probes and tuple centres – an abstract Java class is provided for extension by the TuCSoN middleware for task 2: `AbstractTransducer` implementing `TransducerStandardInterface`—as depicted in Figure 4.22. Therefore, only two methods have to be implemented:

**getEnv** to *sense* an environmental property change—usually, implemented by transducers assigned to sensors

```
@Override
public boolean readValue(final String key) {
  // field 'tid' stores transducer's id
  if (this.tid == null) {
    // no transducer associated yet!
    return false;
  }
  // field 'transducer' stores transducer's reference
  if (this.transducer == null) {
    this.transducer = TransducersManager.INSTANCE.getTransducer(
                this.tid.getAgentName()
              );
  }
  try {
    // probe's interaction logic
    ...
    this.transducer.notifyEnvEvent(
      key, value, AbstractTransducer.GET_MODE // sensor
    );
    ...
    return true;
  } catch (...) {
    return false;
  }
}
```

Figure 4.21: Stripped-down version of the code from `ActualSensor.java` [MO14a]. Method `writeValue` in `ActualActuator` class is similar, thus not reported here.

**setEnv** to *effect* an environmental property change—usually, implemented by transducers assigned to actuators

Both methods are automatically called by the TuCSoN middleware whenever an event generated by an environmental property change is raised either by the associated probe (the `notifyEnvEvent` method in `TransducerStandardInterface`—see Figure 4.22) or by the associated tuple centre (the `notifyOutput` method in `TransducerStandardInterface` automatically called by the TuCSoN middleware in response to ReSpecT primitives such as `getEnv`—see Figure 4.27).

The aforementioned methods have to be implemented so as to actually dispatch to the probes the command to either sense an environmental property (method `getEnv`) or change it (method `setEnv`)—the simplest possible implementation is shown in Figure 4.23 below in the actuator case.

Figure 4.24 sums up the dependencies existing between transducers and probes.

Tasks $1, 2$ complete the implementation of the transducer probe side, also automatically achieving part of the transducer node side. Once both probes and transducers are implemented, MAS designers should exploit TuCSoN services in order to register them and to associate them through the transducer manager, which exposes the following API (Figure 4.25 below):

`createTransducer` to create a new transducer associated to the given probe and bound to the given tuple centre

`addProbe` to attach a probe to a given transducer

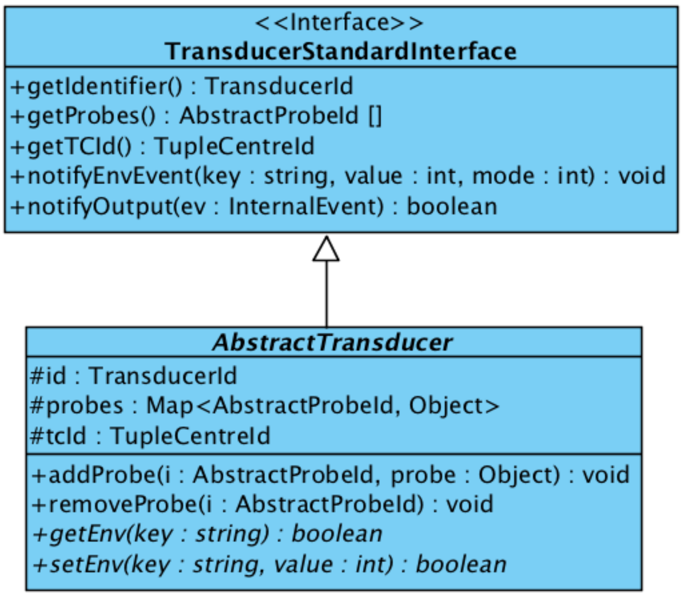

<<Interface>>
TransducerStandardInterface
+getIdentifier() : TransducerId
+getProbes() : AbstractProbeId []
+getTCId() : TupleCentreId
+notifyEnvEvent(key : string, value : int, mode : int) : void
+notifyOutput(ev : InternalEvent) : boolean
AbstractTransducer
#id : TransducerId
#probes : Map<AbstractProbeId, Object>
#tcId : TupleCentreId
+addProbe(i : AbstractProbeId, probe : Object) : void
+removeProbe(i : AbstractProbeId) : void
+getEnv(key : string) : boolean
+setEnv(key : string, value : int) : boolean

Figure 4.22: Class to be extended by custom transducers and its interface [MO14a].

```java
@Override
public boolean setEnv(final String key, final int value) {
  boolean success = true;
  // field 'probes' stores this transducer's probes
  final Object[] keySet = this.probes.keySet().toArray();
  ISimpleProbe p;
  for (final Object element : keySet) {
    p = (ISimpleProbe) this.probes.get(element);
    // try to effect the property change
    if (!p.writeValue(key, value)) {
      success = false;
      break;
    }
  }
  return success;
}
```

Figure 4.23: Stripped-down version of the code from `ActuatorTransducer.java` [MO14a]. Method `getEnv` in `SensorTransducer` class is similar, thus not reported here.

`removeProbe` to detach a probe from its transducer

`getTransducer` to retrieve a transducer's reference given its id

`stopTransducer` to destroy a given transducer

These methods are usually exploited by the agent in charge of configuring the MAS—e.g., the `Thermostat` class. It is worth noting that in order to enable dynamic and

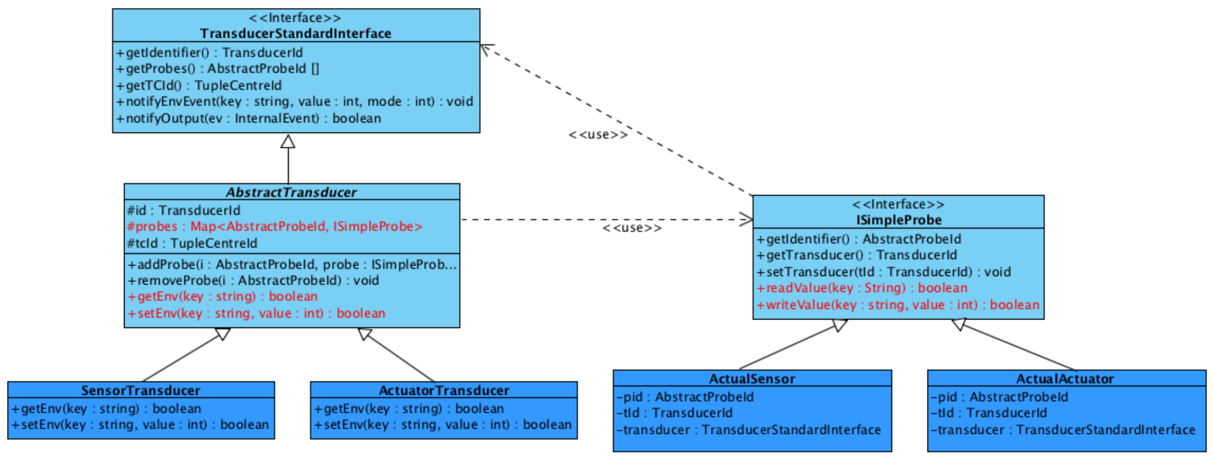
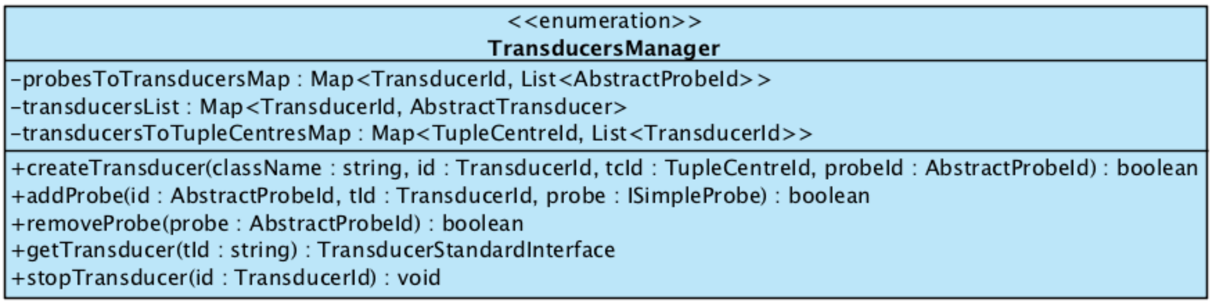

<<Interface>>
TransducerStandardInterface
+getIdentifier() : TransducerId
+getProbes() : AbstractProbeId []
+getTCId() : TupleCentreId
+notifyEnvEvent(key : string, value : int, mode : int) : void
+notifyOutput(ev : InternalEvent) : boolean
<<use>>
AbstractTransducer
#id : TransducerId
#probes : Map<AbstractProbeId, ISimpleProbe>
#tcId : TupleCentreId
+addProbe(i : AbstractProbeId, probe : ISimpleProb...
+removeProbe(i : AbstractProbeId) : void
+getEnv(key : string) : boolean
+setEnv(key : string, value : int) : boolean
<<use>>
<<Interface>>
ISimpleProbe
+getIdentifier() : AbstractProbeId
+getTransducer() : TransducerId
+setTransducer(tId : TransducerId) : void
+readValue(key : String) : boolean
+writeValue(key : string, value : int) : boolean
SensorTransducer
+getEnv(key : string) : boolean
+setEnv(key : string, value : int) : boolean
ActuatorTransducer
+getEnv(key : string) : boolean
+setEnv(key : string, value : int) : boolean
ActualSensor
-pid : AbstractProbeId
-tId : TransducerId
-transducer : TransducerStandardInterface
ActualActuator
-pid : AbstractProbeId
-tId : TransducerId
-transducer : TransducerStandardInterface

Figure 4.24: Dependencies among transducers and probes [MO14a]. Transducers dispatch to probes the requests to undertake the situation actions issued by agents (`AbstractTransducer` methods in red font), whereas probes rely on transducers to notify the outcome of a situation operation back to agents (`ISimpleProbe` methods in red font).



```
<<enumeration>>
TransducersManager
-probesToTransducersMap : Map<TransducerId, List<AbstractProbeId>>
-transducersList : Map<TransducerId, AbstractTransducer>
-transducersToTupleCentresMap : Map<TupleCentreId, List<TransducerId>>
+createTransducer(className : string, id : TransducerId, tcId : TupleCentreId, probeId : AbstractProbeId) : boolean
+addProbe(id : AbstractProbeId, tId : TransducerId, probe : ISimpleProbe) : boolean
+removeProbe(probe : AbstractProbeId) : boolean
+getTransducer(tId : string) : TransducerStandardInterface
+stopTransducer(id : TransducerId) : void
```

Figure 4.25: The transducer manager [MO14a].

distributed addition/removal of transducers and probes, as well as dynamic change of their associations, all the services are also available via TuCSoN coordination operations.

In particular, TuCSoN agents may benefit of transducer manager services also by emitting special tuples in the built-in `'$ENV'` tuple centre, available in any TuCSoN node—the syntax of tuples can be found in TuCSoN official guide[9]. This is, e.g., the choice of the `Thermostat` class as shown in Figure 4.26 below, which establishes the communication channel between the transducer probe side and its node side sibling.

A tuple centre is the coordination medium programmed to effectively enable situated interactions between agents and transducers. Thus, the last development task MAS de-

[9] `http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide.`

```
 1 public static void main(final String[] args) {
 2   ...
 3   final TucsonTupleCentreId configTc =
 4     new TucsonTupleCentreId("'$ENV'", Thermostat.DEFAULT_HOST,
 5       Thermostat.DEFAULT_PORT);
 6   ...
 7   // tuple reifying createTransducer method call
 8   final LogicTuple sensorTuple = new LogicTuple(
 9     "createTransducerSensor",
10     new TupleArgument(sensorTc.toTerm()), // the transducer's tuple centre
11     new Value( // the class implementing it
12       "alice.tucson.examples.situatedness.SensorTransducer"),
13     new Value("sensorTransducer"),
14     new Value( // the class implementing its probe
15       "alice.tucson.examples.situatedness.ActualSensor"),
16     new Value("sensor"));
17   acc.out(configTc, sensorTuple, null);
18         ...
```

Figure 4.26: Stripped-down version of code from `Thermostat.java` [MO14a]. Insertion in `'$ENV'` tuple centre of the tuple built in lines $9 - 17$ is equivalent to calling method `crateTransducer`. Nevertheless, this allows the exploitation of the services of a given TuCSoN node from a remote location—in fact, `configTc` may store the id of a remote tuple centre, deployed on another node of the network w.r.t. the caller.

signers have to undertake so as to correctly exploit TuCSoN situated coordination services is to connect the agents and the environment – technically, the probes – by means of the TuCSoN tuple centres, programmed via the ReSpecT language. In fact, as described in Subsection 4.1.3, agents and probes – or better, ACC and transducers – do not directly interact: all the interactions happen through coordination operations provided by the TuCSoN middleware—in particular, by TuCSoN tuple centres.

Therefore, focussing on situation operations, whenever agents need to interact with a probe, they perform a coordination operation on the tuple centre bound to the transducer responsible for that probe. This is what makes it possible to reify situation operations into ReSpecT events, which are to be managed by ReSpecT reactions—and thus govern the overall event-driven MAS [OM13].

In the case owhermostat scenario, taking into account the situated interaction with the sensor (`ActualSensor.java`), the ReSpecT specification tuples in Figure 4.27 below have to be put in the tuple centre associated to the sensor transducer (`'sensorTc'`, bound to `TransducerSensor.java` which is responsible for `'sensor'` probe). Although the code shown in Figure 4.27 is taken from the specific example, the ReSpecT program is general, since it implements a pattern that is valid for any situated interaction:

- reaction $1 - 5$ maps agents coordination operations requests (external events) into situation operations commands (internal events)
- reaction $6 - 10$ maps situation operation replies (from probes, external events) into coordination operations outcomes (internal events)

By completing task 4 through ReSpecT reactions, MAS designers explicitly exploit the ReSpecT event model – in particular triggering events listed in Table 4.2 on page 89 – to support situatedness, binding together events coming from the agent through its ACC with events going toward the environment through its transducer (Figure 4.27, reaction $1 - 5$)—and, dually, from the environment to the agents (Figure 4.27, reaction $6 - 10$).

```
 1 reaction(
 2      in(sense(temp(T))), // agent request
 3      (operation, invocation),
 4      sensor@localhost:20504 ? getEnv(temp, T) // perception request
 5 ).
 6 reaction(
 7      getEnv(temp, T), // perception reply
 8      (from_env, completion), // environment filter
 9      out(sense(temp(T)))
10 ).
```

Figure 4.27: Stripped-down version of the code from `sensorSpec.rsp` [MO14a]. `'sensor'` is the probe ID of the probe target of the situation operation request: the id of its transducers is automatically retrieved by TuCSoN middleware at run-time, hence transducer mediation is transparent to the ReSpecT programmer.

Technically, this last step in MAS design using TuCSoN links the node side of ACC with the node side of transducers, enacting the very notion of situatedness. The last code snippet in Figure 4.28 is meant to show how the application logic – the thermostat aimed at keeping temperature between LOW and HIGH thresholds – is linked to the situatedness machinery—sensor and actuator probes as well as their transducers.

In particular, line 10 shows TuCSoN coordination operation invocation causing ReSpecT reactions in Figure 4.27 to trigger, leading to stimulate ActualSensor through its transducer SensorTransducer—transparently to the designer of the application logic. Conversely, line 27 shows how Thermostat interacts with ActualActuator (through its transducer ActuatorTransducer) to properly command the needed temperature adjustments (again, transparently).

The same sort of transparency is provided to ReSpecT programmers, as they have no need to know the internal machinery of probes but just transducer API, and to probes programmers, since they deal with ISimpleProbe and TransducerStandardInterface API solely. This supports and promotes a clear *separation of concerns*: application logic (agent) programmers, coordination (ReSpecT) programmers, and environment (probes and transducers) programmers each may focus on their task, just relying on others adhering to TuCSoN API.

```
 1  /* Start perception-reason-action loop */
 2  LogicTuple template;
 3  ITucsonOperation op;
 4  int temp;
 5  LogicTuple action = null;
 6  for (int i = 0; i < Thermostat.ITERS; i++) {
 7    /* Perception */
 8    template = LogicTuple.parse("sense(temp(_))");
 9    op = acc.in(sensorTc, template, null); // see line 2 in Figure 4.27
10    if (op.isResultSuccess()) {
11      temp = op.getLogicTupleResult().getArg(0).getArg(0).intValue();
12      /* Reason */
13      if ((temp >= Thermostat.LOW) && (temp <= Thermostat.HIGH)) {
14        continue;
15      } else if (temp < Thermostat.LOW) {
16        action = LogicTuple.parse("act(temp(" + ++temp + "))");
17      } else if (temp > Thermostat.HIGH) {
18        action = LogicTuple.parse("act(temp(" + --temp + "))");
19      }
20      /* Action */
21        // 'act' ReSpecT reactions are similar to those in Figure 4.27
22      acc.out(actuatorTc, action, null);
23    }
24  }
```

Figure 4.28: Stripped-down version of the code from Thermostat.java [MO14a]. It should be noted that the thermostat interacts solely with TuCSoN tuple centres, being transducers (thus probes) interactions transparently delegated to the TuCSoN middleware—through the ReSpecT reactions in Figure 4.27.

### 4.2.4 Discussion of Results

The need for *situatedness* in multi agent systems (MAS) deployed within pervasive computing scenarios is often translated into the requirement of being sensitive to *environment change* [FM96], possibly affecting the environment in turn. This requirement lays at the core of the notion of *situated action* – complementing that of *social action* [Cas98] –, as those actions arising from strict interaction with the environment [Suc87].

This leads to recognise *dependencies* among agents and the environment as a fundamental source of complexity within a MAS—the other being dependencies between agents' activities [MC94]. Therefore, *coordination* – as the discipline of managing dependencies [MC94] – could be used to deal with both *social* and *situated interaction*, by exploiting *coordination artefacts* for handling both social and situated dependencies [OM13].

Accordingly, in this section a novel *situated coordination* approach has been described, promoted by the TuCSoN model and technology for agent coordination [OZ99], to handle situatedness in MAS as a coordination issue. In particular, description covered which support TuCSoN provides to MAS programmers in each macro-stage of a typical software engineering process applied to a MAS: the abstractions available for the requirement analysis (Subsection 4.1.3), the run-time architecture to refer to during the design phase (Subsection 4.2.1 and Subsection 4.2.2), the API provided to support implementation of the concept of situated coordination (Subsection 4.2.3).

The solutions adopted by the TuCSoN technology to deal with the issues of engineering situated MAS within pervasive computing context have also been highlighted: in particular, the need to rely on mediating abstractions such as ACC, transducers, and tuple centres as the means to uncouple individual components (agents and probes) interactions, along with the need for an asynchronous event-driven communication model to correctly deal with the most common issues of system distribution.

### 4.2.5 Related Work

By considering the technologies described in Subsection 4.1.2, one can easily see how they could be read as progressively improving the way in which MAS deal with environment modelling and situatedness.

The first one, Jade, provides more or less nothing on the environment side, being focussed on supporting agents with a set of services, mostly dedicated to communication, mobility and interoperability. On the coordination side, Jade adopts ACL messages as the data structure to exchange and FIPA protocols as the only means to manage (social) dependencies—thus, in a subjective way. Thus, situated dependencies should be managed without a specific abstraction, by falling back to agents and message passing re-use (or abuse). This can be seen also by considering the (implicit) meta-model behind Jade, that of Figure 4.3, which somehow recognises the existence of an interaction space between agents, but fails short in taking also MAS environment into account.

It should be noted that an integration between Jade and TuCSoN has been proposed in [ORV$^+$04b], allowing Jade to benefit of TuCSoN coordination services. In principle, this allows Jade to also benefit of all the architectural components provided by TuCSoN—thus, ultimately, to benefit of the proposed architecture. Nevertheless, the aforementioned integration is not worth to be considered here for a few reasons.

First of all, back in 2004, when [ORV$^+$04b] was published, transducers were not yet existing, being them defined first in 2009, when [CO09] was published: thus, no dedicated abstraction (boundary artefact) nor architectural component (the transducer itself) was provided specifically to deal with environment resources of any sort.

Second, [ORV$^+$04b] had the specific goal of enabling seamless integration of subjective and objective coordination approaches, thus the focus of the paper was on interpreting agents actions over coordination artefacts (only) as physical acts, according to the FIPA standard implemented by Jade: thus, no other kinds of actions (e.g., situated actions) were considered.

Lastly, the fundamental role played by the event abstraction as the glue supporting a uniform interpretation of both social and situated actions (activities and environment change) was not recognised.

When considering Jason, environment modelling improves thanks to an explicit Java environment layer, which enables agents to sense perceptions (mapped into beliefs) and to act on effectors (mapped into Jason internal actions). Furthermore, the notion of event as the central abstraction around which the whole agent inner reasoning cycle as well as its perception-action loop should revolve is recognised in Jason.

Nevertheless, something is still missing on the coordination side: Jason features nothing more than message passing and plans/goals sharing, thus leaving coordination issues to agents themselves. For these reasons, Jason (implicit) meta-model corresponds to the one depicted in Figure 4.2, although Jason support to situatedness is better w.r.t. Jade.

Finally, CArtAgO is the technology whose architecture is clearly the closest to ours, as proposed in Subsection 4.1.3—hence, to the TuCSoN architecture, too. To some extent, in fact, the following mapping could be attempted: CArtAgO artefacts are explicitly conceived to model and implement environmental resources (in some sense, the union of probes and transducers, in TuCSoN); agent bodies could be classified as (sorts of) boundary artefacts (they may also resemble TuCSoN ACCs and transducers) although the latter concept is much more generally applicable; CArtAgO observable events are one of the possible implementations of the event abstraction (as TuCSoN events are based on ReSpecT event model). Also, CArtAgO artefacts could as well be used to implement coordination artefacts, e.g., TuCSoN tuple centres.

Nevertheless, the mapping is actually quite imprecise as well as somewhat unnatural. Focussing, for instance, on CArtAgO implementation of the event abstraction, the difference is quite evident: CArtAgO observable events are bound to the artefact they come from, as a means to check the outcome of agents situated actions and to reactively respond to environment stimuli—hence they reify only situated dependencies; whereas events in TuCSoN are the reification of whatever happens within the MAS—hence social dependencies, too.

A much more general difference lies in the artefact abstraction itself: whereas CArtAgO allows MAS designers to program any kind of artefact they need according to any sort of interaction model they decide to ascribe to the artefact (e.g., depending on wether it is a social or situated artefact), TuCSoN provides a fixed set of artefacts – that is, ACCs, transducers and tuple centres – which are both *(i)* each responsible for a general aspect of MAS situated coordination – ACCs for activities, transducers for environment changes, tuple centres for social and situated coordination – and *(ii)* actually specialisable depending on the MAS at hand—e.g., although exposing the same API, transducers behaviour can be specialised according to the nature of the resource they model.

This leads to a fundamental difference w.r.t. *uniformity*: in a CArtAgO-coordinated MAS, artefacts can be *heterogeneous* both in exposed APIs and in semantics, whereas in a TuCSoN-coordinated MAS, ACCs, transducers, and tuple centres have a well-defined semantics which remains coherent regardless of the actual specialisation required.

Furthermore, CArtAgO observable events are created by the artefact they belong to in a custom way, that is, by storing the information MAS designers believe to be useful at design time. In TuCSoN instead, the event model used by, e.g., transducers is always the same – actually, ReSpecT event model – regardless of the environmental resource nature or the MAS deployment scenario.

Among the works not considered in Subsection 4.1.2, at least the following three are worth to be mentioned here: the SADE [DMY$^+$09] and ELDA [FGMR10] development frameworks, as well as the iCore european project [VGS$^+$13].

SADE is a development environment for the engineering of self-adaptive MAS. Acknowledging the role played by the environment in MAS, authors adopt the *event* abstraction to design a self-adaptation mechanism based on the *organisation* metaphor—similar to the concept of society. In particular, environmental events are notified to agents according to a publish/subscribe architecture, possibly triggering a change of role—leading to a change of behaviour.

Nevertheless, the environment abstraction is not realised to its full extent. In fact, environmental events are generated by other agents, playing the role of wrappers of that part of the environment which should be observable to the MAS.

Furthermore, situatedness support is limited, since the only fields describing an event in [DMY$^+$09] are: *(i)* its *type*, which can refer to a change in agents' state, behaviour, role, or offered services; *(ii)* its *source* (the agent who generated it); and *(iii)* a set of *constraints* whose function is not clearly explained—at least in [DMY$^+$09].

In ELDA [FGMR10], the event abstraction is adopted to design a lightweight agent model—indeed, ELDA. In particular, any ELDA agent is a single-threaded autonomous entity interacting through asynchronous events, whose behaviour is expressed reactively in response to incoming events. Although ELDA does not natively support any environment abstraction, it has been extended to support the PACO model abstractions [HJD07], splitting MAS into four parts: agents, environment, interactions, and organisation.

Nevertheless, the extension seemingly accounts for agents' position only, and the only agent-environment interaction is due to a *monitor agent* – again, a wrapper – which continuously monitor the environment and the agents' state, triggering events when some pre-conditions are met.

The Cognitive Management Framework for the Internet of Things (IoT) [VGS$^+$13], proposed in the iCore european project, aims at developing a framework able to abstract away from the technological heterogeneity of nowadays devices while improving context-awareness. Among its proposals, a three-layered architecture is conceived to face the above challenges: the lowest layer is meant to collect Virtual Representations (VOs) of real-world objects; the mid-layer to aggregate representations in Cognitive Mashups of VOs according to the way in which they collaborate to offer higher-level functionalities; the top layer is the service-oriented layer providing users and stakeholders with the APIs bridging the gap between application and IoT resources.

This three layered-architecture – in particular the mid-layer promoting and supporting IoT resources service-oriented aggregation – somehow stresses the need to tackle situatedness-related issues as coordination ones.

## 4.3 Spatial Situatedness in TuCSoN

MAS deployed in pervasive computing scenarios are stressing more and more the requirements for coordination middleware [ZCF$^+$11]. In particular, the availability of a plethora of mobile devices, with motion sensors and motion coprocessors, is pushing forward the need for *space-awareness* of computations and systems: awareness of the spatial context is often essential to establish which tasks to perform, which goals to achieve, and how.

More generally, spatial issues are fundamental in many sorts of complex software systems, including intelligent, multi-agent, adaptive, and self-organising ones [BMS11]. In most of the application scenarios where *situatedness* plays an essential role, coordination is required to be *space aware*.

This is implicitly recognised by a number of proposals in the coordination field trying to embody spatial mechanisms and constructs into the coordination languages – such as TOTA [MZ09], $\sigma\tau$-LINDA [VPB12], GEOLINDA [PCBB07], and SAPERE [ZCF$^+$11] – which, however, are mostly tailored around specific application scenarios.

For the sake of generality then, in this section the aim is to devise out the *basic* mechanisms and constructs required to *generally* enable and promote *space-aware coordination*.

Along this line, the general notion of *space-aware coordination medium* is introduced (Subsection 4.3.1), then it is shown how the ReSpecT coordination media and language can be extended so as to support space-aware coordination (Subsection 4.3.2). After sketching the semantics of the spatial extension, Subsection 4.3.3 showcases space-aware ReSpecT potentialities by dealing with a benchmark problem in the filed of *spatial computing*:

implementing the "T program" [BDU+12].

### 4.3.1 Space-aware Coordination Media

Spatial coordination requires spatial *situatedness* and *awareness* of the coordination media, which translates in a number of technical requirements.

**Situatedness** First of all, *situatedness* requires that a *space-aware coordination abstraction* should at any time be associated to an absolute positioning, both physical (i.e., the position in space of the computational device where the medium is being executed on) and virtual (i.e. the network node on which the abstraction is executed). If not a must-have, geographical positioning is also desirable, and quite a cheap requirement, too, given the widespread availability of mapping services nowadays.

More generally, this concerns both *position* and *motion* – every sort of motion –, which in principle include speed, acceleration, and all variations in the space-time fabric, also depending on the nature of space. In fact, software abstractions may move along a *virtual* space – typically, the network – which is usually *discrete*, whereas physical devices (robots, mobile devices) move through a *physical* space, which is mostly *continuous*; software abstractions, however, may also be hosted by mobile physical devices, and share their motion. As a result, a coordination abstraction may move through either a physical, continuous space, (e.g., "I am in a given position of a tridimensional physical space") or a virtual, discrete space (e.g., "I am on a given network node").

Physical positioning could be either *absolute* (e.g., "I am currently at latitude X, longitude Y, altitude Z"), *geographical* ("I am in via Sacchi 3, Cesena, Italy"), or *organisational* ("I am in Room 5 of the DISI, site of Cesena"). Absolute positioning is often available in the days of mobile devices, usually through GPS services—which, coupled with mapping services, typically provide geographical positioning, too.

Virtual positioning is available as a network service, and might be also labelled as either absolute (in terms of IP address, for instance) or relative (as a domain/subdomain localisation via DNS). Organisational location should be instead defined application- or middleware-level, and related to either absolute or virtual positioning.

Furthermore, a notion of *locality* may be available, so as to enable the *local vs. global* dynamics typically featured by complex distributed systems such as pervasive ones. Locality could be strictly bound to positioning, but not necessarily so: being in the same location is not always the same as being in the same position.

**Awareness** The main requirement of *spatial awareness* is that the ontology of a space-aware coordination medium should contain some notion of space. This means, first of all, that the position of the coordination medium should be available to the coordination laws it contains in order to make them capable of *reasoning about space*, that is, to implement *space-aware coordination laws.* So, generally speaking, a range of predicates / functions should be provided to access spatial information associated to any event occurring in the coordination medium (e.g., where the action causing the event took place, where the coordination medium is currently executing), and to perform simple computations over spatial information.

Also, space has to be embedded into the working cycle of the coordination medium: the event model should include *spatial events*, which affect coordination by triggering some space-related computation within the coordination abstraction. In fact, associating spatial information to events is not enough: space-related laws like "when at home, switch on the lights" cannot be expressed only by referring to actions performed, but require instead a specialised notion of spatial event (such as "I am at home") to be triggered.

So, a spatial event should be generated within a coordination medium, conceptually corresponding to changes in space—so, related to *motion*, such as starting from / arriving to a place. Spatial events should then be captured by the coordination medium, and used to activate space-aware coordination laws, within the normal working cycle of the coordination abstraction.

**Spatial tuple centres** Tuple centres are introduced in TuCSoN[10] [OZ99] as coordination media meant at encapsulating any computable coordination policy within the coordination abstraction. Technically, a tuple centre is a *programmable* tuple space, i.e., a tuple space whose behaviour in response to events can be programmed so as to specify and enact any coordination policy [OD01b, Omi07]. Tuple centres can then be thought as general-purpose coordination abstractions, which can be suitably forged to provide specific coordination services.

In the same way as timed tuple centres empower tuple centres with the ability of embodying timed coordination laws [ORV05], *spatial tuple centres* extend tuple centres so as to address the spatial issues outlined in previous subsection.

First of all, the location of a tuple centre is obtained through the notion of *current place*, which could be, for instance, the absolute position in space of the computational device where the coordination medium is being executed on, or the domain name of the TuCSoN node hosting the tuple centre, or the location on the map. Then, motion is conceptually represented by two sorts of spatial events: moving from a starting place, and stopping at an arrival place—in any sort of space / place.

With respect to the formal model defined in [CO09], this is achieved by extending the input queue of the environment events to become the multiset *SitE* of time, environment, and spatial events, handled as input events by the *situation* transition ($\longrightarrow_s$)—as shortly discussed at the end of Subsection 4.3.2.

Whenever some motion of any sorts occurs (such as the physical device starting / stopping, or the node identifier changes), a spatial event is generated, and put in the multiset *SitE* of the tuple centre, to be handled by the *situation* transition. Then, analogously

[10]`http://tucson.unibo.it`

to operation, situation, and time events, it is possible to specify reactions triggered by spatial events—the so-called *spatial reactions*.

Spatial reactions follow the same semantics of other reactions: once triggered, they are placed in the triggered-reaction set and then executed, atomically, in a non-deterministic order. As a result, a spatial tuple centre can be programmed to react to the motion either in physical or in virtual space, so as to enforce space-aware coordination policies.

Finally, a simple notion of locality is provided by the TuCSoN *node* abstraction: when coordination primitives are invoked without any node specification, they are handled as implicitly referring to the *local interaction space* hosted by the node; when a node identifier is instead associated to the invocation, then the primitive explicitly refers to the *global interaction space* [OZ99].

### 4.3.2 Space-aware Extension to ReSpecT

ReSpecT tuple centres are based on first-order logic (FOL). FOL is adopted both for the communication language (logic tuples), and for the behaviour specification language (ReSpecT) [OD01a]. Basically, reactions in ReSpecT are defined as Prolog-like facts (reaction *specification* tuples) of the form

```
reaction(Activity, Guards, Goals)
```

A reaction specification tuple specifies the list of the operations (*Goals*) to be executed when a given event occurs (called *triggering event*, caused by an *Activity*) and some conditions on the event hold (*Guards* evaluate to true). These operations make it possible to insert / read / remove tuples from the tuple space and the specification space of the tuple centre, but also to observe the properties of the triggering event, as well as to invoke operations over other coordination media. The core syntax of ReSpecT is shown in Table 4.3 below.

According to the abstract model described in Subsection 4.3.1, the ReSpecT language is extended to address spatial issues *(i)* by introducing some spatial predicates to get information about the spatial properties of both the tuple centre and the triggering event, and *(ii)* by making it possible to specify reactions to the occurrence of spatial events. The extension to the ReSpecT language is shown in Table 4.4.

**Spatial observation predicates** In particular, the following *observation predicates* are introduced for getting spatial properties of triggering events within ReSpecT reactions:[11]

- `current_place(@S,?P)` succeeds if *P* unifies with the position of the node which the tuple centre belongs to

[11]A Prolog-like notation is adopted for describing the modality of arguments: `+` is used for specifying input argument, `-` output argument, `?` input/output argument, `@` input argument which must be fully instantiated.

| | | |
|---|---|---|
| ⟨*Program*⟩ | ::= | {⟨*Specification*⟩ `.`} |
| ⟨*Specification*⟩ | ::= | `reaction(` ⟨*Activity*⟩ [`,` ⟨*Guards*⟩] `,` ⟨*Reactions*⟩ `)` |
| ⟨*Activity*⟩ | ::= | ⟨*Operation*⟩ \| ⟨*Situation*⟩ |
| ⟨*Operation*⟩ | ::= | `out(` ⟨*Tuple*⟩ `)` \| (`in` \| `rd` \| `no` \| `inp` \| `rdp` \| `nop`) |
| | | `(` ⟨*Template*⟩ [`,` ⟨*Term*⟩] `)` |
| ⟨*Situation*⟩ | ::= | `time(` ⟨*Time*⟩ `)` \| `env(` ⟨*Key*⟩ `,` ⟨*Value*⟩ `)` |
| ⟨*Guards*⟩ | ::= | ⟨*Guard*⟩ \| `(` ⟨*Guard*⟩ {`,` ⟨*Guard*⟩} `)` |
| ⟨*Guard*⟩ | ::= | `request` \| `response` \| `success` \| `failure` \| `endo` \| `exo` \| |
| | | `intra` \| `inter` \| `from_agent` \| `to_agent` \| `from_tc` \| `to_tc` \| |
| | | `before(` ⟨*Time*⟩ `)` \| `after(` ⟨*Time*⟩ `)` \| `from_env` \| `to_env` |
| ⟨*Reactions*⟩ | ::= | ⟨*Reaction*⟩ \| `(` ⟨*Reaction*⟩ {`,` ⟨*Reaction*⟩} `)` |
| ⟨*Reaction*⟩ | ::= | [⟨*TupleCentre*⟩ \| ⟨*EnvRes*⟩ `?`] ⟨*Operation*⟩ \| |
| | | `env(` ⟨*Key*⟩ `,` ⟨*Value*⟩ `)` \| |
| | | ⟨*Observation*⟩ \| ⟨*Computation*⟩ \| `(` ⟨*Reaction*⟩ `;` ⟨*Reaction*⟩ `)` |
| ⟨*Observation*⟩ | ::= | ⟨*Selector*⟩_⟨*Focus*⟩ |
| ⟨*Selector*⟩ | ::= | `current` \| `event` \| `start` |
| ⟨*Focus*⟩ | ::= | (`activity` \| `source` \| `target`) `(` ⟨*Term*⟩ `)` \| `time(` ⟨*Term*⟩ `)` |

Table 4.3: ReSpecT Syntax: Core [MO13g]—no forgeability, bulk, uniform predicates.

- `event_place(@S,?P)` succeeds if *P* unifies with the position of the node where the triggering event was originated
- `start_place(@S,?P)` succeeds if *P* unifies with the position of the node where the event chain that led to the triggering event was originated

where the node position can be specified either as its absolute physical position (*S*=`ph`), its IP number (*S*=`ip`), its domain name (*S*=`dns`), its geographical location (*S*=`map`) – as typically defined by mapping services like Google Maps –, or its organisational position (*S*=`org`)—that is, a location within an organisation-defined virtual topology.

As an example, the execution of the reaction specification tuple

```
reaction(
  in(q(X)),
  ( operation, completion ),
  (
     current_place(ph,DevPos),
```

| | | |
|---|---|---|
| ⟨*Specification*⟩ | ::= | `reaction(` ⟨*Activity*⟩ \| ⟨*Change*⟩ [`,` ⟨*Guards*⟩] `,` ⟨*Reactions*⟩ `)` |
| ⟨*Situation*⟩ | ::= | `env(` ⟨*Key*⟩ `,` ⟨*Value*⟩ `)` \| |
| ⟨*Change*⟩ | ::= | `env(` ⟨*Key*⟩ `,` ⟨*Value*⟩ `)` \| `time(` ⟨*Time*⟩ `)` `from(` ⟨*Space*⟩ `,` ⟨*Place*⟩ `)` \| `to(` ⟨*Space*⟩ `,` ⟨*Place*⟩ `)` |
| ⟨*Guard*⟩ | ::= | `request` \| `response` \| `success` \| `failure` \| `endo` \| `exo` \| `intra` \| `inter` \| `from_agent` \| `to_agent` \| `from_tc` \| `to_tc` \| `before(` ⟨*Time*⟩ `)` \| `after(` ⟨*Time*⟩ `)` \| `from_env` \| `to_env` \| `at(` ⟨*Space*⟩ `,` ⟨*Place*⟩ `)` \| `near(` ⟨*Space*⟩ `,` ⟨*Place*⟩ `,` ⟨*Radius*⟩ `)` |
| ⟨*Focus*⟩ | ::= | `(activity` \| `source` \| `target)` `(` ⟨*Term*⟩ `)` \| `time(` ⟨*Term*⟩ `)` \| `place(` ⟨*Space*⟩ `,` ⟨*Term*⟩ `)` |
| ⟨*Space*⟩ | ::= | `ph` \| `ip` \| `dns` \| `map` \| `org` |

Table 4.4: Spatial extensions to ReSpecT [MO13g]—only the definitions introduced / affected by the spatial extension are shown.

```
      event_place(ph,AgentPos),
      out(in_log(AgentPos,DevPos,q(X)))
   )
 ).
```

inserts a tuple (`in_log/3`) with spatial information each time a TuCSoN agent retrieves a tuple of the form `q(_)` from the tuple centre, actually implementing a *spatial log*, to track absolute positions of both the querying agent and the device hosting the tuple centre.

**Spatial guard predicates** Also, the following *guard predicates* are introduced to select reactions to be triggered based on spatial event properties:

- `at(@S,@P)` succeeds when the tuple centre is currently executing at the position `P`, specified according to `S`
- `near(@S,@P,@R)` succeeds when the tuple centre is executing at the position included in the spatial region with centre `P` and radius `R`, specified according to `S`

So, for instance, `near(dns,'apice.unibo.it',2)` succeeds when the tuple centre is currently executing on a device whose second-level domain is `.unibo.it`.

**Spatial event descriptors** Reactions to spatial events are specified similarly to ordinary reactions, by introducing the following new event descriptors:

- `from(?S,?P)` matches a spatial event raised when the device hosting the tuple centre starts moving from position `P`, specified according to `S`
- `to(?S,?P)` matches a spatial event raised when the device hosting the tuple centre stops moving and reaches position `P`, specified according to `S`

As a result, the following are admissible reaction specification tuples dealing with spatial events:

```
reaction(from(?Space,?Place), Guards, Goals).
reaction(to(?Space,?Place), Guards, Goals).
```

As a simple example, consider the following specification tuples (wherever `Guards` is omitted, it is by default `true`):

```
reaction( from(ph,StartP),
  ( current_time(StartT)
    out(start_log(StartP,StartT)) )).
reaction( to(ph,ArrP),
  ( current_time(ArrT)
    out(stop_log(ArrP,ArrT)) )).
reaction( out(stop_log(ArrP,ArrT)),
  ( internal, completion ),
  ( in(start_log(StartP,StartT))
    in(stop_log(ArrP,ArrT))
    out(m_log(StartP,ArrP,StartT,ArrT)) )).
```

which altogether record a simple *physical motion log*, including start / arrival time and position. In fact, the first reaction stores information about the beginning of a physical motion in a `start_log/2` tuple, the second the end of the motion in a `stop_log/2` tuple, whereas the last removes both sort of tuples and records their data altogether in a `m_log/4` tuple, representing the essential information about the whole trajectory of the mobile device hosting the tuple centre.

**Semantics** The basic ReSpecT semantics was first introduced in [OD01a], then extended towards time-aware coordination in [ORV05], re-shaped to support the notion of coordination artefact in [Omi07], finally enhanced with situatedness in [CO09]—which represents the reference semantics for ReSpecT until now.

In order to formalise the semantics for the space-aware extension of ReSpecT, two are the main changes with respect to [CO09]. First of all, a new, generalised event model should be defined to include both spatial events, and spatial information for any sort of event. Then, the *environment* transition, already handling both time and general environment events, should be extended to include spatial events—so as to handle the full spectrum of situatedness-related events. All other required extensions (such as the formalisation of each spatial construct's semantics) are technically simple, and trivially extend tables in [CO09].

The first fundamental extension to the event model is depicted in Table 4.4: a new sort of *spatial* $\langle Activity \rangle$ is introduced. In particular, the notion of $\langle Situation \rangle$ is extended with the two spatial activities `from(` $\langle Space \rangle$ $\langle Place \rangle$ `)`, `to(` $\langle Space \rangle$ $\langle Place \rangle$ `)`, reflecting the initial and final stages of a motion trajectory, respectively—in whatever sort of space.

However, spatial extension of the event model cannot be limited to introducing spatial activities: another issue is represented by *spatial qualification* of events, that is, in short, making *all* ReSpecT events featuring spatial properties—in the same way as temporal properties were introduced for all ReSpecT events in [ORV05]. This is represented by the $\langle Place \rangle$ property featured by $\langle Cause \rangle$ – and $\langle StartCause \rangle$, of course –, as shown in Table 4.5 below, where the extended ReSpecT event model is depicted.
Essentially, all ReSpecT events are in principle qualified with both time and space properties—the latter one defined as the position (in whichever sort of space) where the (initial) cause of the event takes place. Of course, properties may be actually defined or not at execution time, depending on the facility available when the event is generated.

For instance, if absolute physical positioning is made available by the hosting device, and the device is currently in location `P` when an event is generated, the coordination middleware associates `P` to the event as its physical location—which otherwise would be set to undefined.

According to [CO09], the operational semantics of a ReSpecT tuple centre is expressed by a transition system over a state represented by a labelled triple ${}^{OpE,SitE}\langle Tu, Re, Op\rangle_n^{OutE}$ (abstracting away from the specification tuples $\Sigma$, which are not of interest here). In particular, $Tu$ is the multiset of the ordinary tuples in the tuple centre; $Re$ is the multiset of the triggered reactions waiting to be executed; $Op$ is the multiset of the requests waiting for a response; $OpE$ is the multiset of incoming $\langle Operation \rangle$ events; $SitE$ is the multiset of incoming $\langle Situation \rangle$ events, including time, spatial, and general environment events; $OutE$ is the outgoing event multiset; $n$ is the local tuple centre time.

$OutE$ is automatically emptied by emitting the outgoing events, with no need for

| | | |
|---|---|---|
| $\langle Event \rangle$ | ::= | $\langle StartCause \rangle$ , $\langle Cause \rangle$ , $\langle Evaluation \rangle$ |
| $\langle StartCause \rangle , \langle Cause \rangle$ | ::= | $\langle Activity \rangle$ , $\langle Source \rangle$ , $\langle Target \rangle$ , $\langle Time \rangle$ , $\langle Space{:}Place \rangle$ |
| $\langle Source \rangle , \langle Target \rangle$ | ::= | $\langle AgentId \rangle \mid \langle TCId \rangle \mid \langle EnvResId \rangle \mid \bot$ |
| $\langle Evaluation \rangle$ | ::= | $\bot \mid \{\langle Result \rangle\}$ |
| $\langle Place \rangle$ | ::= | $\langle GPSCoordinates \rangle$ , $\langle IPAddress \rangle$ , $\langle DomainName \rangle$ , $\langle MapLocation \rangle$ , $\langle VirtualPosition \rangle$ |

Table 4.5: Extending ReSpecT events with space [MO13g].

special transitions. In the same way, *OpE* and *SitE* are automatically extended whenever a new incoming (either operation or situation) event enters a tuple centre—again, no special transitions are needed for incoming events. In particular, *SitE* is added new environment events by the associated transducers [CO09], new time events by the passing of time [ORV05], and – in the spatial extension of ReSpecT presented here – also new spatial events whenever some sort of motion takes place.

So, as described in [CO09], the behaviour of a ReSpecT tuple centre is modelled by a transition system composed of four different transitions: *reaction* ($\longrightarrow_r$), *situation* ($\longrightarrow_s$), *operation* ($\longrightarrow_o$), *log* ($\longrightarrow_l$). Quite intuitively, spatial events are handled – in the same way as time and environment events – by the *situation* transition, which triggers ReSpecT reactions in response to spatial events. As a result, the *situation* transition is the fundamental (and now finally complete) ReSpecT machinery supporting situatedness in the full acceptation of the term—that is, suitably handling reactiveness of the coordination abstraction to time, space, and general environment events.

### 4.3.3 Expressiveness Showcase

In [BDU+12], a layered architecture for devices running *spatial computing* programs is described, which helps bridging the gap between the hardware and *Spatial Computing Languages* (SCL):

**physical platform** the lowest level in the hierarchy, identifying the medium upon which the computation actually executes—e.g., a smartphone, a drone with a whole set of sensors and actuators, even a virtual device in the case of a simulation

**system management** typically the OS layer, abstracting away from physical details, (hopefully) providing all the low-level drivers needed by spatial applications—e.g., for a GPS module, or a motion engine

**abstract device** the top abstraction level exposing the basic API for SCL—e.g., a clock service, GPS coordinates tracking, and the like

Independently of the layer of abstraction at which a given SCL can be placed, as well as of the kind of ADM it implements, three classes of operators are required to reach maximal expressiveness and computational power—the sort of spatial Turing equivalence discussed in [Bea10]:

**measure space** transforming spatial properties into computable information—e.g., distances, angles, areas

**manipulate space** translating information into some modification of the spatial properties of the device—e.g., turning wheels to face a given direction, slowing down the motion engine

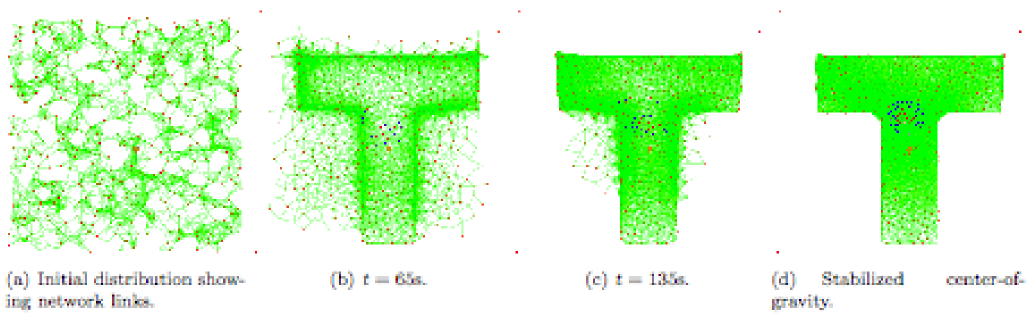


Figure 4.29: T-Program run from [BDU+12], implemented using the Proto language [VBC11].

**compute** besides usual computation, any kind of spatial-pointwise operation, e.g., an interaction, or a non-spatial sensor or actuator operation—e.g., a light sensor

A fourth class (*physical evolution*) looks more like a sort of assumption over the (possibly autonomous) dynamics a given program/device can rely upon—e.g., the existence of actuators responding to the program/device commands, or the independent motion of a colony of cells.

As a reference benchmark to test the expressive power and the computational completeness of SCL, the "T-Program" is proposed in [BDU+12], consisting of the following three stages, depicted in Figure 4.29 below in the case of the Proto language [VBC11]: *(i)* cooperatively creating a local coordinate system; *(ii)* moving devices to create a "T-shaped" structure; *(iii)* computing the centre of gravity of the structure and draw a ring around it. Stage *(i)* requires the capability to measure the spatial context where the program/device lives; stage *(ii)* requires the ability to manipulate the spatial properties of each device (thus relying also on the fourth category); stage *(iii)* requires both computational capabilities and, again, measuring capabilities.

The spatial extension to ReSpecT just proposed meets all the requirements to successfully implement the above benchmark at the level of the ADM. In fact:

- a combination of three *Observation Predicates* is given to measure spatial properties:
  - `current_place` measuring where the tuple centre executing the current ReSpecT reaction is
  - `event_place`, `start_place` measuring respectively where the *direct cause* and *start cause* [Omi07] of the event triggering the current ReSpecT computation took place
- given that the modification of spatial properties is necessarily bound to the facilities provided by the host device, the manipulation requirement can be addressed by situatedness-related constructs of ReSpecT:

  - ⟨*EnvResId*⟩ `?` `env`(⟨*Key*⟩, ⟨*Value*⟩) precisely meant to be used as an interface to device actuators, allowing an agent to dispatch commands to a device ⟨*EnvResId*⟩ at the most appropriate level of abstraction—that is, as a part of the environment managed through the coordination medium

- while the computing requirement is orthogonal w.r.t. the spatial dimension, some predicates may be considered here as they ease the process of computing over spatial information—in particular, guards:
  - `at`(⟨*Place*⟩) triggers a reaction when the reacting tuple centre is at a given location
  - `near`(⟨*Place*⟩, ⟨*Radius*⟩) triggers a reaction when the reacting tuple centre is near a given location

Finally, it should be noted that some constructs are left out, in particular those in ⟨*TCEvent*⟩. They belong to the aforementioned fourth class (physical evolution), since they allow the Abstract Device – that is, the ReSpecT VM – to *perceive motion events* generated either autonomously or on demand by the physical device hosting the VM.

Respectively, `from`(⟨*Place*⟩) is the spatial event generated by the ReSpecT VM when the host tuple centre *starts moving* (leaving a location), whereas `to`(⟨*Place*⟩) is the spatial event generated when it *stops moving* (approaching a location). These events are meant to reify a *change of state* in the spatial dimension of computation: therefore, no events have to be generated while the VM is staying still, since there is no state change to reify.

Now follows description of how space-aware ReSpecT can successfully implement the T program benchmark.

**Coordinate system** Setting a local coordinate system basically amounts at *(i)* choosing an origin node, *(ii)* making it spread a vector tuple to neighbours, then (recursively) *(iii)* making them increment the vector, and *(iv)* forwarding it to neighbours. Thus, the basic mechanism needed by the VM at the application level is *neighbourhood spreading*. Assuming a *physical* neighbourhood relation is used, the following reactions – installed on every node – achieve the goal [MO13f]:

```
// Check range then forward.
reaction( out(nbr_spread(msg(Msg),nbr(Dist),req(ID))),
  ( completion, success ),
  ( no(req(ID)), out(req(ID)), // Avoid flooding
    current_place(Me), event_place(Sender),
    within(Me,Sender,Dist), // Prolog computation
    out(msg(Msg)),
    rd(nbrs(Nbrs)), // Neighbours list
    out(forward(Msg,Dist,req(ID),Nbrs))
) ).
// Delete multicast request.
reaction( out(nbr_spread(msg(Msg),nbr(Dist),req(ID))),
  ( completion, success ),
    in(spread(msg(Msg),nbr(Dist),req(ID)))
).
```

```
16 // Forward to every neighbour.
17 reaction( out(forward(Msg,Dist,req(ID),[H|T])), // Some Nbrs
18   ( intra, completion ),
19   ( H ? out(spread(msg(Msg),nbr(Dist),req(ID))), // Forward
20     out(forward(Msg,Dist,req(ID),T)) // Iterate
21 ) ).
22 // Delete iteration tuple.
23 reaction( out(forward(Msg,Dist,req(ID),Nbrs)),
24   ( intra, completion ),
25   in(forward(Msg,Dist,req(ID),Nbrs)) // Delete it anyway
26 ).
```

Reactions 2-10 and 12-15 manage spreading requests: the former checks if the incoming request has been already served, gets reacting node position and sender node's one, checks if it's in the desired range, and if so stores the tuple (`Msg`) then starts forwarding it; the latter simply removes the request. Reactions 17-21 and 23-26 manage the forwarding process, that is, iterate neighbours forwarding the spreading command. Neighbourhood is set here at the VM level through a tuple `nbrs([nbr_1,...,nbr_N])`, but could also be set at the middleware level by using a coordination middleware such as TuCSoN [OZ99].

**"T-shape"** To arrange nodes (tuple centres) so as to form a T-shaped structure, it is needed to *(i)* define spatial constraints representing the T (how much tall, fat, etc.), then *(ii)* make every node move so as to satisfy them. Thus, the basic mechanism needed at the VM level is *motion monitoring and control* [MO13f]:

```
1  // Compute motion vector then start moving.
2  reaction( out(move(Constraints)),
3    ( completion, success ),
4    ( current_place(Here),current_time(Now),Check is Now+1000,
5      direction(Constraints,Here,Vec), // Prolog computation
6      out_s(
7        // Reaction 12-22
8      ),
9      engine ? env(mode,'on'), engine ? env(dir,Vec) // Start actuators
10 ) ).
11 // Motion constraints monitoring.
12 reaction( time(Check),
13   internal,
14   ( current_place(Here),
15     rd(move(Constraints)),
16     ( check(Here,Constraints), // Prolog computation
17       engine ? env(mode,'off')
18     ;
19       current_time(Now), Check is Now+1000,
20       out_s(
21       // Reaction 12-22
22 ) ) ) ).
23 // Arrival clean-up.
24 reaction( to(Dest),
25   internal,
26   in(move(Constraints))
27 ).
```

An interesting feature of ReSpecT is exploited in the code above. Besides reacting to a motion request by properly controlling actuators, Reaction 2-10 performs a *meta-coordination* *operation*, by inserting a new coordination law in the tuple centre (Reaction 12-22), which is responsible for arrival check.

**Focal point** To first compute the focal point (FC) of the T-shape, then draw a sphere around it, two basic mechanisms are needed, both very similar to the *neighbourhood spreading* previously shown—thus whose code is not reported to avoid redundancy: *(i)* a *bidirectional* neighbourhood spreading to collect replies to sent messages – enabling to aggregate all the node's coordinates and counting them – and *(ii)* a *spherical multicast* to draw the ring pattern.

The code for spherical multicast is almost identical to neighbourhood spreading, but with a fundamental difference (besides the `req(ID)` test to avoid flooding), that is, the use of observation predicate `start_place` instead of `event_place`. This replacement (almost) alone stops the spreading process and completely change the *spatial properties* of communication.

This is a notable example of the expressiveness of the language extension just described.

### 4.3.4 Discussion of Results

In this section, how *spatial situatedness* can be provided to the application level by enhancing the ReSpecT coordination language with spatial-related constructs enabling *space-awareness* has been discussed. To demonstrate space-aware ReSpecT expressiveness, an implementation of the T program benchmark was provided, as defined in the context of spatial computing languages, and specifically conceived to guarantee spatial Turing-completeness of a language.

This focus on the language level of a coordination middleware, completes the proposed approach to deal with situatedness-related issues in pervasive MAS, complementing the architectural aspects discussed in Section 4.2.

As an aside, the proposed language extension is supported by the porting of the TuCSoN infrastructure over the Android platform[12], where it can benefit from therein provided location services. Also, the extension makes it possible to rethink the actual architecture of the Home Manager[13] middleware for smart home appliances [DC15], delegating geolocation-related issues to the underlying TuCSoN infrastructure, instead of relying on ad-hoc software agents.

[12]At the time of writing, the official release is not yet available, but the codebase under refinement is available at `http://bitbucket.org/smariani/tucsonandroid`.

[13]`http://apice.unibo.it/xwiki/bin/view/Products/HomeManager`

## 4.4 Remarks & Outlook

The research work undertaken in this chapter is necessary to provide $\mathcal{M}o\mathcal{K}$ with a well-suited and well-grounded concrete platform to implement a first prototype, and to deploy the prototype within early evaluation scenarios.

The situated architecture thoroughly described in Subsection 4.2.1 is necessary to provide $\mathcal{M}o\mathcal{K}$ with the context-awareness needed by socio-technical systems, as discussed in Section 5.2 of Chapter 5. Once social and situated interactions generate events according to a situated event model, coordination services, as well as the application at hand, have access to all the relevant information, either for the purpose of coordination, or for users to improve their collaboration toward reaching their business goals.

Within $\mathcal{M}o\mathcal{K}$, e.g., through situated events reifying social and situated interaction, users may become aware of each other activities, and take advantage of this awareness. Furthermore, being all the interactions mediated by the environment, stigmergic and observation-based coordination are enabled and promoted by default, providing thus all the necessary ingredients for supporting the kind of user behaviour driven coordination envisioned in $\mathcal{M}o\mathcal{K}$.

The language extension extensively described in Subsection 4.3.2 is necessary, too, for two reasons at least. The first one, most obvious, is to enable the coordination laws to get advantage of the situatedness of coordination-related events generated by the underlying infrastructure. The second, possibly less apparent, is that of allowing adaptiveness of the coordination laws to the ever-changing computational context within which social and situated activities happen.

Thanks to ReSpecT programs capability to change themselves at run-time, any aspect of the coordination laws installed within TuCSoN tuple centres may be re-programmed anytime. In the case of the $\mathcal{M}o\mathcal{K}$ prototype described in Section 7.1 of Chapter 7, this means any facet of artificial chemical reactions, such as their rate, as well as of the chemical engine resembling chemical compartments, may be changed by the middleware itself, in face of, e.g., users' interactions.

# Chapter 5

# Coordination Issues in Knowledge-Intensive Socio-Technical Systems

In this chapter a novel approach to coordination in knowledge-intensive Socio-Technical Systems (STS) is skecthed, grounded on the cognitive theory of Behavioural Implicit Communication (BIC).

Accordingly, firstly those challenges peculiar to either Knowledge-Intensive Environments (KIE) or STS, which mostly impact the issue of coordination, are discussed (Section 5.1); then, some research works applying to computer-based Multi-Agent Systems (MAS) principles borrowed from Activity Theory, (cognitive) stigmergy, and BIC are briefly reviewed (Section 5.2); finally, the notion of perturbation action is introduced, connected to BIC notion of tacit message, while sketching the basic idea behind the proposed approach to self-organising coordination in STS (Section 5.3)—thoroughly detailed in Part II of this thesis.

## 5.1 Socio-Technical Systems & Knowledge-Intensive Environments

### 5.1.1 Challenges of Socio-Technical Systems

*Socio-technical systems* (STS) arise when *cognitive* and *social interaction* is mediated by information technology rather than by the natural world [Whi06]. As such, STS include non-technical elements such as people, processes, regulations, etc., which are inherent parts of the system. Since social activity is fluid and nuanced, STS are technically difficult to design properly – especially from a coordination perspective – and often awkward to use [Ack00]. Also, a number of peculiarities, with related engineering challenges, have been highlighted by various research works [Ack00]. Among the many:

- STS have *emergent* properties, which cannot be attributed to individual parts of the system, depending on the relationships and *dependencies* between system components. Given this complexity, the aforementioned properties can only be evaluated once the system has been assembled and deployed, not at design time
- STS are often *non-deterministic*, that is, when presented with a specific input, they may not always produce the same output. This happens because system behaviour depends on human operators, and people do not always react in the same way to the same situation
- people may use the STS in ways completely *unpredictable* from the designers standpoint. In large-scale open networks like the Internet, users behaviour is uncontrolled, so that very few assumptions can be made about it. In particular, it is almost impossible to globally foresee and influence the space of potential interactions. Most probably, users, as well as software components, will behave in a self-interested fashion, which may help to *anticipate* some of their actions, and may provide some clues on how to design coordination strategies at the micro level [OO02]
- *awareness*, that is, knowing who is present, and *peripheral awareness*, namely monitoring of others' activity, are fundamental in STS [HS96], because visibility of information flow – thus *observability* of dependencies – enables learning and greater efficiency [Hut95]—as well as *observation-based coordination* [PCF07]
- people not only *adapt* to their systems, they adapt their systems to their needs (*co-evolution*) [Orl92, OBS96]

Failing to recognise one of the above facets of STS, thus neglecting to address the corresponding issue, inevitably leads to a *socio-technical gap* in the STS, that is, a gap between what the computational platform strive to provide, and what the users participating the STS are expecting to have [Ack00].

It should be noted that all the mentioned peculiarities imply challenges which is possible to approach from a *coordination perspective*, in particular, exploiting coordination techniques supporting *programmable* (to deal with unpredictability and adaptation, mostly), *self-organising* (to account for emergence and non-determinism), *situated* (supporting awareness) coordination.

Not by chance, this is exactly the approach to coordination discussed in chapters 2-3 (programmability & self-organisation) and 4 (situatedness).

### 5.1.2 Challenges of Knowledge-Intensive Environments

In general, data are considered as raw facts, information is regarded as an *organised* set of data, and knowledge is perceived as *meaningful* information [Bha01]. In a dynamic business environment, where an organisation faces unexpected and novel problems, a computational platform can be used, at best, as an *enabler* to turn data into information, but it is only through people – hence, considering the STS *as a whole* – that information is *interpreted* and turned into knowledge.

Following [Bha01], *Knowledge Management* (KM) is a complex socio-technical process encompassing knowledge creation, validation, distribution, presentation, and application, each facet bringing along its own issues and computational challenges:

**creation** refers to the ability of an organisation to develop *novel* and *useful* information, processes, best practices, solutions, etc. Knowledge creation is an *emergent* process in which experimentation and pure *chance* play an important role. Providing the means to undertake trial-and-error experiments, and to spot knowledge creation opportunities, is a great computational challenge for a KM platform

**validation** refers to the extent to which a firm can reflect on knowledge and evaluate its *relevance*, pertinency, and effectiveness for the existing organisational environment and business goals. Knowledge validation is a painstaking process of continually monitoring, evaluating, and *refining* the knowledge base to suit the existing, potential, or foresee realities. As the realities change, so may arise the need to convert parts of knowledge back into information, then data, which may finally be *discarded*. Designing computational techniques seamlessly integrating with users workflows, while transparently assisting them in all the stages involved in knowledge validation, is far from trivial

**distribution** refers to the distribution of organisational knowledge in different locations, embedded into different artefacts and procedures, and stored into different storage media—which is challenging by itself. Furthermore, *interactions* between organisational technologies, techniques, and people can have *direct bearing* on knowledge distribution, introducing the need to deal with some degree of *uncertainty* – about, e.g., where to find a relevant piece of information – and demanding for adaptation techniques supporting continuous and *autonomous* knowledge re-distribution

**presentation** refers to each storage medium requiring its different means of knowledge *(re)presentation*, thus, organisational members often find it difficult to re-configure, re-combine, and *integrate* knowledge from these distinct and disparate sources. Though organisational members may find the relevant pieces of information by organising data into separate databases, they will still find it difficult to integrate and interpret information without a common representation

**application** refers to making knowledge more *active* for the firm in creating values. The criteria for evaluating relevance of knowledge are not often readily apparent. However, the technological platform could provide means of experimentation to assess the potential of knowledge and readily exploit it

Two main strategies for KM are employed by early adopters of the principle [MAYA01], striving to address the above challenges:

- the *process-centred* approach, which understands KM as a *social*, communication based process. Here knowledge is closely tied to the person who developed it and is shared mainly through person-to-person contacts. The main purpose of information technology is then to *enable* people communicate knowledge, not to store it
- the *product-centred* approach, focussing on knowledge *reification* into documents, then on their creation, storage, and reuse in computer-based corporate memories

Whatever the approach, the connections that KM software must facilitate are between people as much as they are between people and information systems. In particular, the software must support fruitful exchange of knowledge, and transformation from *tacit* to *explicit* knowledge [MAYA01].

Yet again, the whole process of KM, in most if not all of its facets, can be approached from a *coordination perspective*, where coordination policies and mechanisms may be applied not solely to the agents (either software or human) participating the management process, but also to the raw data subject of the process. This way, data may become organised information *autonomously*, guided by coordination mechanisms seamlessly integrating knowledge workers' own workflows with the organisation long-term goals—e.g., creation, validation, and distribution, may be (partially) delegated to the coordination infrastructure underlying the KM platform. In turn, knowledge workers may find knowledge patterns more easily by having the KM platform (thus, the underlying coordination mechanisms) assist them, by, e.g., presenting the right information at the right time in the right place.

One possible way to do so, in the broad sense of taking coordination as the ground upon which to build KM processes in STS, is outlined in the following subsection, as well as the subject of the other sections of this chapter.

### 5.1.3 Research Roadmap

Among the issues just described, many can be tackled by exploiting existing coordination abstractions and mechanisms. E.g., the need for *awareness* manifested by STS can be directly supported through the notion of *coordination artefacts*, as provided by the A&A meta-model for MAS [ROD03]—see Subsection 5.2.1. There, artefacts are computational resources at agents' disposal, featuring, among the many properties, *observability* of interactions, which directly enables peripheral awareness. Also, *malleability* of artefacts, that is, their ability to change behaviour dynamically, may be fruitfully exploited to support *adaptation* of the system's provided functionalities, as well as of other non-functional properties, according to users' interactions and ever-changing needs.

It should be noted that KM processes may be engineered on top of coordination abstractions and mechanisms. E.g., consider that both traditional approaches to KM, that is, process-centred and product-centred, may be conveniently integrated through the notion of artefact, too: coordination artefacts as enablers of people communication – for the former approach – and resource artefacts as reification of (possibly implicit) knowledge—for the latter. Issues of KM may be then readily delegated to artefacts, exploiting their distinguishing features. In the case of knowledge distribution, e.g., the *linkability* feature comes in hand [ORZ06], by allowing delegation of distribution of knowledge, as well as of the computational load needed to make it accessible, to the KM platform.

Besides the A&A meta-model, other coordination-related mechanisms may deal with the challenges described in previous section. E.g., *stigmergic interaction* [Par06] leads to an *emergent* phenomenon of global coordination arising from *local* interactions which tolerates quite well *uncertainty* of information and *unpredictability* of behaviour—besides directly supporting peripheral awareness, if not on actions on their post-hoc traces.

For all these reasons, and many more that are detailed in dedicated paragraphs in the following, next section reviews the aforementioned literature – that is, the A&A meta-model and stigmergic interaction – and more, shaping the conceptual framework on top of which the approach to *user-driven coordination* is built, as sketched in Section 5.3. Besides this, the conceptual framework is exploited in the *M*olecules *o*f *K*nowledge model described in Part II of this thesis.

## 5.2 From Activity Theory to Behavioural Implicit Communication

### 5.2.1 Activity Theory for Multi-Agent Systems

*Activity Theory* (AT) is a social psychological theory born in the context of Soviet Psychology [V$^+$78]. Nowadays, it is widely applied in computer science, especially in Computer Supported Cooperative Work (CSCW) and Human Computer Interaction (HCI) [Nar96]. AT is a general framework for conceptualising human activities: according to AT, any activity carried on by one or more participants (agents) of an organisation (MAS) cannot be understood without considering the tools, or *artefacts*, enabling actions and mediating interaction [ORV08].

Thus, on the one side, artefacts mediate interaction among individuals, as well as between individuals and their environment; on the other side, artefacts represent that part of the (computational) *environment* the individuals want/have to deal with. Artefacts are not only *physical*, such as shelves, doors, phones, and whiteboards, but may also be *cognitive*, such as operating procedures, heuristics, scripts, individual and collective experience, or even both, such as operating manuals and computers.

Adopting AT as a conceptual framework for MAS leads to the fundamental recognition that agents are not the sole abstraction to design MAS: artefacts, too, are necessary [ORV08]. As developed by Ricci et al. while defining the A&A meta-model for MAS

[ROD03], then, activity theory promotes the notion of *coordination artefact* to identify those artefacts used specifically for coordination purpose. Along this line, coordination artefacts represent a straightforward generalisation of the notion of *coordination medium* [Cia96], coming from the fields of coordination models and languages—including abstractions like tuple spaces, channels, pheromone infrastructures [PBS02], among the many.

The A&A meta-model is characterised in terms of three basic abstractions [ORV08]:

**agents** representing pro-active components of the MAS, encapsulating the *autonomous* execution of some kind of activities inside some sort of environment

**artefacts** representing reactive components of the MAS, intentionally constructed, shared, manipulated and used by agents (or by the MAS designer), to support agents activities – either cooperatively or competitively – and/or MAS non-functional properties

**workspaces** as the computational containers of agents and artefacts, useful for defining the *topology* of the environment, and providing a way to define a notion of *locality*

**Agents** From a computational viewpoint, autonomy means that agents encapsulate (their thread of) control. So, agents never give up control, nor are controlled by anything, unless they deliberate to do so, of course—"agents can say no", according to Odell [Ode02]. Only data (information, knowledge) crosses agent boundaries. As a result, the interpretation of a MAS is that of a multiplicity of distinct loci of control, interacting with each other by exchanging information [ORV08].

Literally, the etymology of the word "agent" – from Latin "agens" – means "the one who acts", thus, the agent notion should come equipped, by definition, with a notion of *action*. Whatever the model, the notion of action is intrinsically connected to the notion of *change*: an agent acts in order to change something, which, in the context of a MAS, can be either another agent, or the environment [VHR$^+$07, WOO07]. The only way to directly affect another agent state is usually thought to be through direct information exchange, that is, through a speech act—or communication action. Instead, change to the MAS environment is more easily thought of as the result of physical actions—which may be undertaken on a computational, or simulated environment, too, of course.

Any ground model of action is strictly coupled with the context where the action takes place. In this sense, autonomous agents are essentially *situated* entities, since any agent is strictly coupled with the environment where it lives and (inter)acts [ORV08].

**Artefacts** Artefacts, on the contrary, are not autonomous: since they are designed to serve some purpose for agents, artefacts do not follow their own course of action. As such, artefacts are (computationally) reactive, that is, they behave in response to agent use, and their function just needs to emerge when they are used by an agent [ORV08].

Also, their function is expressed in terms of change to the environment, that is, what the artefact actually does when used by an agent, which makes artefacts intrinsically *situated*, too. Finally, since they are situated, artefacts are easy to be thought of as reactive to changes in the environment.

Among the main properties of artefacts, one could list [ROV$^+$07]: (i) *inspectability* and controllability, that is, the capability of observing and controlling artefacts structure, state and behaviour at run-time; (ii) *malleability* (or, forgeability), that is, the capability of changing / adapting artefacts function at run-time, according to new requirements or unpredictable events occurring in the open environment; and (iii) *linkability*, that is, the capability of composing distinct artefacts at run-time as a form of service composition, so as to scale up with complexity of the function to be provided.

**Workspaces** They conceptually contain agents and artefacts, and computationally provide MAS with a notion of *locality* – that is, agents and artefacts locally available in a given computational node – useful to shape the MAS *topology*—in terms of neighbourhood relations between workspaces. Besides this, definition of workspaces has been never further detailed, basically because according to the A&A meta-model, anything interesting inside a workspace has to be represented by a dedicated artefact.

In Subsection 5.2.3, the notion of *smart environment* introduced in [TCR$^+$05] is discussed, and taken as a possible enhancement to the notion of A&A workspace, integrating ingredients borrowed from AT, stigmergy, and BIC (all described in following sections).

**Why AT?** It is apparent how AT, and then the A&A meta-model, may prove extremely useful as a reference framework for conceiving and designing the computational part of a STS devoted to KM: agents are the goal-directed/-oriented, active components of the system, such as human users and software agents, undertaking (epistemic) actions aimed at reaching their own business goals; artefacts are the (computational) tools in their hands, including those services devoted to KM and collaboration, as well as the documents subject of the management process; workspaces are the working environments providing the coordination services necessary to deal with the issues outlined in Section 5.1—e.g., programmability for adaptiveness & unpredictability, self-organisation for emergence & non-determinism, observability for awareness, and so on.

### 5.2.2 Stigmergy and Cognitive Stigmergy

**Stigmergy** The notion of *stigmergy* generally refers to a set of coordination mechanisms mediated by the *environment* [Omi12]. For instance, in ant colonies chemical substances (pheromone) act as environment *markers* for specific social activities, and drive both the individual and the social behaviour of ants. While the notion of stigmergy has undergone a large number of generalisations / specialisations / extensions [Par06, ROV$^+$07] w.r.t. the original definition [Gra59], and even a larger number of implementations in systems of many sorts, its main features are always the same. In short, *stigmergic coordination* requires that:

- some interacting agents perform some action on the environment that leaves some *traces*, or markers, which can then be *perceived* by other agents and *affect* their subsequent behaviour
- all interactions among agents are mediated by the environment, through traces—like ant's pheromones
- emission of traces is *generative* [Gel85], thus once they are produced, traces' life is independent of the producer
- traces evolution over time depends on their relation with the environment—as in the case of pheromone diffusion, aggregation, and evaporation in ant colonies

Such a sort of interaction among agents is what produces *self-organisation*: whereas it occurs on a *local* basis, its effect is *global* in terms of the system's global structures and behaviours it originates, by *emergence* [SFH+03].

**Cognitive stigmergy** A number of relevant works in the field of cognitive sciences point out the role of stigmergy as a fundamental coordination mechanism also in the context of human societies and organisations [SZ01, SW04]. As noted in [ROV+07]:

- modifications to the environment (e.g., traces) are often amenable of an *interpretation* in the context of a shared, conventional system of signs
- the interacting agents feature *cognitive* abilities that can be used in stigmergy-based interactions

Based on this, the notion of *cognitive stigmergy* is introduced in [ROV+07] as a first generalisation of stigmergic coordination exploiting agent's cognitive capabilities to enable and promote *self-organisation* of social activities: when traces becomes signs, stigmergy becomes cognitive stigmergy. There, self-organisation is based on signs amenable of a symbolic interpretation, and involves intelligent agents, able to correctly interpret signs in the environment, so as to react properly.

Under a cognitive perspective, the (working) environment in stigmergy can be interpreted as a set of shared, observable tools (artefacts), providing specific functionalities, that are useful for agents while performing their individual work both for empowering their capabilities and for sharing information—through the environment itself.

Thus, artefacts in cognitive stigmergy are aimed, first of all, at promoting *awareness*, that is, making agents seamlessly aware of the work and practises of other agents, which could in turn *affect*, either driving or improving, their own activities. Awareness is a key aspect to support *emergent* forms of *coordination*, where there is no pre-established plan defining exactly which are the dependencies and interactions among ongoing activities [MC94] and how to manage them—on the contrary, the plan emerges along with the activities themselves.

**Why stigmergy?** Stigmergy and cognitive stigmergy directly and efficiently support both *awareness* and *peripheral awareness* in STS, be it by simply associating reactive behaviours to changes in the shared working environment (e.g., in the form of traces deposit) if non-intelligent agents are considered (e.g., simple software components), or by true signification of signs (e.g., traces amenable of symbolic interpretation) undertaken by intelligent agents (e.g., humans or BDI agents [Cas98]).

Besides awareness, it is apparent that stigmergic coordination is well suited for many facets of KM in STS, e.g., (i) as far as *knowledge creation* is concerned, attaching traces of past activities to managed information items may undoubtedly help further interpretation by collaborative agents; (ii) for *validation*, traces of activities may be exploited by the KM platform to autonomously spot stale, obsolete, wrong, or simply no longer relevant information, then discard it; (iii) as far as *distribution* is concerned, the literature about stigmergic coordination is full of examples of emergent spatial organisation of information items or emergence of spatial patterns among ensembles of cooperating agents—see [Par06] for some.

Finally, the connection between (cognitive) stigmergy and the A&A meta-model, thus activity theory, too, is obviously represented by the definition of artefact as the means to model and design computational environments within a MAS. Accordingly, (cognitive) artefacts may be the targets of agents practical/epistemic actions, and may react to actions by changing their own state – or that of *linked* artefacts – so as to reflect the traces (interpreted as, e.g., side effects) of the undertaken actions.

### 5.2.3 Behavioural Implicit Communication

*Behavioural Implicit Communication* (BIC) is a cognitive theory of communication stemming from the essential idea that usual, *practical*, even non-social behaviour can *contextually* be used as a message for communicating, and that behaviour can be communication by itself, without any modification or any additional signal or mark [Cas06]. The adjective *behavioural* is because it is just simple, non-codified behaviour. *Implicit*, because, not being specialised nor codified, its communicative character is unmarked, undisclosed, *not manifest*. In other words, in BIC communication is just a use, and at most a destination, not the shaping function [CC95].

On the contrary, communication actions are normally carried on by specialised behaviours (e.g., speech acts, and gestures). Therefore, a BIC action is a *practical action* primarily aimed at reaching a practical goal, which may additionally be interpreted as aimed at achieving a communicative goal, without any predetermined (conventional or innate) specialised meaning.

BIC can be taken as a reference for different modalities of *observation-based coordination* [PCF07]. In [TCR$^+$05] five are identified:

**unilateral** X intends to coordinate with Y by observing Y's actions

**bilateral** the unilateral form of coordination but for both agents, so: X intends to coordinate with Y by observing Y's actions, and, viceversa, Y intends to coordinate with X by observing X's actions

**unilateral-AW** a unilateral form of coordination, improved with a first degree of *awareness*: X intends to coordinate with Y by observing Y's actions, and Y is aware of it—that is, it knows to be observed

**reciprocal** a bilateral form of observation-based coordination with awareness of both the involved agents: X intends to coordinate with Y by observing Y's actions, being Y aware of it, and Y intends to coordinate with X by observing X's actions, being X aware of it

**mutual** extends the reciprocal form by introducing the explicit awareness of each other intention to coordinate (*awareness*$^2$): X intends to coordinate with Y by observing Y's actions, being Y aware of it, Y intends to coordinate with X by observing X's actions, being X aware of it, and also X is aware of Y's intention to coordinate, while Y is aware of X's intention to coordinate

BIC is necessary for mutual coordination, while it is possible and undoubtely useful in the other forms of observation-based coordination.

It should be noted that *stigmergy* seems very similar to BIC. However, definition of the former is unable to distinguish between the communication and the *signification* processes. Put in other words, according to [TCR$^+$05], one does not want to consider an escaping prey as doing communicative actions to its predator, notwithstanding that the effects of the first actions elicit and influence the actions of the latter.

Nevertheless, as in BIC, stigmergic communication does not exploit any specialised communicative action, but just usual practical actions. Therefore, [TCR$^+$05] considers stigmergy as a subcategory of BIC, being communication via *long term traces*, physical practical outcomes, useful environment modifications, which preserve their practical end, but acquire a communicative function.

Under this perspective, stigmergy amounts to a special form of BIC where the addressee does not perceive the behaviour during its performance, but post-hoc traces and outcomes of it.

**Tacit messages** Tacit messages are proposed, along with the notion of BIC, to describe the messages a practical action (and its traces) may implicitly send to observers [CPT10]:

1. *informing about the presence.* "Agent $A$ is here". Since an action, an activity, a practical behaviour (and its traces), is observable in the computational environment, any agent therein – as well as the environment itself – becomes aware of $A$ existence—and, possibly, contextual information such as $A$ location. Think about turning the lights on, when going out, in a room clearly visible from the outside. It

is a signal left for a potential intruder to persuade him that somebody is at home. The light in itself has not a conventional meaning, but a possible inference that can be drawn by observing it is exploited to send a deceiving message. In this case, although the actual goal of the practical action is informative, the acting agent does not really want to be understood as being communicating

2. *informing about the intention.* "Agent $A$ plans to do action $\beta$". If the agents' workflow determines that action $\beta$ follows action $\alpha$, peers (as well as the environment) observing $A$ doing $\alpha$ may assume $A$ next intention to be "do $\beta$". Consider the phenomenon of trust dynamics, where the fact that an agent trusts another one increases the latter trustworthiness. The trusting agent may intentionally exploit this process by, e.g., delegating to a subordinate agent a critical task: the trace is an implicit signal of the intention to trust

3. *informing about the ability.* "$A$ is able to do $\phi_i,\ i \in \mathbb{N}$". Assuming actions $\phi_i,\ i \in \mathbb{N}$ have similar pre-conditions, agents (and the environment) observing $A$ doing $\phi_i$ may infer that $A$ is also able to do $\phi_{j \neq i},\ j \in \mathbb{N}$. In the context of supervised learning (e.g., teacher-scholar relationship), each action (of the scholar) is also a message to the supervisor (teacher), implicitly communicating improvements and acquired abilities. For teamwork activities, the ability of agents to signal their own abilities is a crucial message that can be used to speed up team formation

4. *informing about the opportunity for action.* "$p_i,\ i \in \mathbb{N}$ is the set of pre-conditions for doing $\alpha$". Agents observing $A$ doing $\alpha$ may infer that $p_i,\ i \in \mathbb{N}$ hold, thus, they may take the opportunity to do $\alpha$ as soon as possible. Think about lines in, e.g., a post office: they are evident signs informing about which are the active counters—which are the condition for action. The queue line also informs about the fact that the others are waiting to act, thus: the condition for acting is there, but not available yet. Finally, the physical shape of the line also informs on who is the last person waiting, back to whom the newcomer has to wait

5. *informing about the accomplishment of an action.* "$A$ achieved $S$". If $S$ is the state of affairs reachable as a consequence of doing action $\alpha$, agents observing $A$ doing $\alpha$ may infer that $A$ is now in state $S$. Consider a child showing her mother she is eating a given food, or a psychiatric patient showing the nurse he is drinking his drug. It is not the ability of eating or drinking that is relevant here, but that the eating or drinking action is being accomplished. This kind of message is particularly important in satisfaction of social commitments, expectations, obligations, and the like

6. *informing about the goal.* "$A$ has goal $g$". By observing $A$ doing action $\alpha$, peers of $A$ may infer $A$'s goal to be $g$, e.g., because action $\alpha$ is part of a workflow aimed at achieving $g$—likewise for the environment. In the context of a team sport, e.g.,

football, by kicking the ball in one direction (action), a player is communicating with a team member to sprint in that precise direction (goal)

7. *informing about the result.* "Result $R$ is available". If peer agents know that action $\alpha$ leads to result $R$, whenever agent $A$ does $\alpha$ they can expect result $R$ to be soon available—in case action $\alpha$ completes successfully. Suppose that while doing the dishes, a glass is dropped and breaks into pieces. You decide not to remove the fragments in order to let your husband understand that, although he was convinced that they were unbreakable glasses, this glass being struck, it breaks. A trace thus, can be an implicit signal not only when it is the result of an action, but also a consequence of abstaining from acting

The above categorisation is general enough to suit several different business domains and practical actions. In fact, it is used in Subsection 5.3.2 to categorise tacit messages devised out by analysing real-world STS.

**Why BIC?** BIC is taken as the reference framework for approaching the issue of coordination in knowledge-intensive STS. The reason is that BIC provides a sound cognitive and social model of action and interaction both w.r.t. to human agents and w.r.t. computational agents. Furthermore, as clarified in next section, BIC can be applied to computational environments as well, fully supporting environment-mediated, observation-based coordination. Also, the interpretation of BIC referenced so far, that is, the one described in [TCR+05], encompasses stigmergic coordination, too, as well as is deeply intertwined with the notion of (cognitive) artefact [TCR+05].

As regards KM in STS, BIC is obviously well suited to improve existing approaches to KM, be them process-centred or product-centred, partly due to its roots in activity theory and stigmergy, partly thanks to its notion of non-manifest, non-codified communication, reified by tacit messages. Furthermore, the notion of computational smart environment discussed in next section, is directly enabled by BIC, and is a solid reference for the kind of self-organising KM system envisioned in this thesis, and targeted by the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model described in Part II.

### 5.2.4 Toward Computational Smart Environments

The nature of a *working environment* (computational workspace) – which depends on the tools (artefacts) shaping it – determines the effectiveness of the activities of the actors (agents, users) that are immersed in it [TCR+05]. Then, the purpose of an activity is not merely to change the environment in a way that (presumably) leads to goal satisfaction, as typically assumed in the literature.

In general, people (software agents) undertake actions to save attention, memory and computation; people recruit external elements to reduce their own cognitive (computational) effort by distributing load; and so on. This makes sense only if agents (either software or humans) and their activities are conceived as *situated* [TCR+05], that is, strongly coupled with their environment.

As a result, environment design should not merely be aimed at helping agents to achieve their goals, but also to make other actions as easy as possible—such as *epistemic*, and *coordinative* actions.

Also, [TCR+05] notices that cognitive processes exist in MAS that do not belong to individual agents: the MAS environment may participate, too, by enabling and mediating individual agent actions as well as social agent interaction, through the knowledge it embeds either implicitly or explicitly. Thus, knowledge is distributed in the environment and is encapsulated within cognitive artefacts, and the structure of the environment, as well as of the knowledge it contains, affect the activities of agents within the MAS.

According to [SS00], "computer supported collaborative work (CSCW) seems to be pursuing two diverging strategies", leading to distinct trends in CSCW research: the first, stemming from workflow management systems, tends to privilege *automatisation* of coordination; the second, takes *flexibility* of interaction as its main goal. That is, the former approach stresses the role of computational entities prescribing the rules of collaboration (like workflow engines), the latter mostly leverage on the intelligent coordination capabilities of collaborative entities (like humans, or intelligent agents). So, there is a gap between two strategies, that [SS00] proposes to close by dealing with two key issues: *mutual awareness* and *coordinative artefacts*.

Mutual awareness means that actors of a collaboration activity *affect* and mutually *perceive* others' activities in the common field of work – the shared workspace – which can (partially) reveal/conceal them. Mutual awareness is then the basis for opportunistic, ad hoc alignment and improvisation, which ensures flexibility to collaborative activities. Coordinative (or, coordination) artefacts instead, encapsulate those portions of the coordination responsibilities that is better to automatise, e.g., for the sake of efficiency. So, on the one hand, coordinative artefacts define and govern the space of the admissible interaction, while, on the other hand, they do not impose a pre-defined course of actions unnecessarily reducing flexibility (they do not work as commanders) [TCR+05].

BIC and (cognitive) stigmergy seem to provide the necessary mutual awareness envisioned by [SS00], while (cognitive) coordination artefacts the required coordinative capabilities. But in order for a computational work environment to fully support BIC-based coordination, at least three different conditions have to hold [TCR+05]:

1. *observability* of practical actions and their traces must be a property of the environment where agents live. The environment can enable visibility of others, or constrain it—in the same way that sunny or foggy days affect perception. It could also enable agents to make themselves observable or hide their presence on purpose

2. agents should be able to *understand* and *interpret* (or to learn to react to) a practical action. A usual practical action becomes then a (implicit) message when the acting agent knows others are observing and will understand his behaviour

3. agents should be able to understand the *effect* that their actions have on others, so as to begin acting in the usual way also/only to obtain a desired/expected reaction

Based on these requirements, two types of *computational smart environment* are defined in [TCR+05]: common environment and shared environment.

Agents that live in a *common environment* (*c-env*) are agents whose actions and goals interfere (positively or negatively), thus need coordination to manage this interference. Agents can observe just the state of the environment and act on that basis, without having access to the actions of their peers. Even a trace is seen as part of the environment and not as a product of other agents. A general property of a c-env is that it enables agents to modify the environment state while keeping track of it. A *shared environment* (*s-env*) instead, is a particular case of a c-env that enables *(i)* different forms of observability of each other action executions, as well as *(ii)* awareness of this observability.

It is exactly this kind of smart environment, leveraging BIC and (cognitive) stigmergy through coordination artefacts, that is envisioned in Part II of this thesis with the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model and technology for self-organisation of knowledge in knowledge-intensive STS. There, shared workspaces act as active repositories of information, autonomously and continuously monitoring users' activity with the aim of exploiting their tacit messages and traces of actions so as to enable anticipatory coordination [PCF07], while assisting usual workflows related to KM by autonomously dealing with some of the aforementioned facets of KM.

## 5.3 Behavioural Implicit Communication in Real-world STS

In this section a survey of a few real-world STS, heterogeneous in goals, architecture, and functionalities, is discussed, seeking for the practical (virtual) actions they provide (Subsection 5.3.1). Then, in Subsection 5.3.2, the pool of actions the STS have in common, that is, those actions having different names but the same (epistemic) goal, is devised out, as well as the tacit messages the common actions may convey and the consequent perturbation actions they may bring along.

The purpose of the section is to bridge the gap between theory and practice, showing that BIC is not only a fancy cognitive theory, but a pervasive facet of everyday technological activities.

### 5.3.1 Survey of Actions

It should be noted that the following list is not comprehensive: for each STS covered by the survey, the focus is solely on those actions which could more easily be interpreted in *epistemic terms*, and which could be defined *positive*, in the sense of adding information and connections, not removing them—e.g., the "post" action vs. the opposite, "delete post" action. Also, features not available to the average user are ignored, e.g., companies, groups, public pages and profiles, premium services.

As far as Facebook is concerned, actions considered are:

**post** publish something on Facebook wall, e.g., a text, an image, a video, or an hyperlink. The published piece of information may be publicly visible, or its visibility can be constrained with a fine granularity

**like** manifest interest in a piece of information, e.g., a post, a re-post, or a comment. It should be noted that interest is usually positive, meaning that the user liking the post (or a comment to a post) enjoys or agrees with the information, but could also be negative, if the opposite is true, depending on the acting user habits and shared conventions among his/her friends

**comment** comment a post with information, e.g., text, an image, a video, or hyperlinks

**reply** reply to a comment with another comment. The difference w.r.t. a comment, is that the subject of a comment is a post, while the subject of a reply is a comment

**share** share someone else's post, either with an additional comment, or as it is—the source is automatically cited. The target audience is limited by the original post visibility constraints

**save** store a post for later retrieval. The saved piece of information is placed in a special section of a user workspace, and is periodically re-brought to user attention

**tag friends** explicitly bring to a given friend attention the piece of information he/she is tagged in. As a consequence, an explicit notification is sent to the tagged friend

**add friend** include a given person in the list of friends, that is, the list of observable people who can in turn observe the acting user. Observability potentially includes anything visible on the activity registry each Facebook user has—fine grained visibility details may be tuned at will

**search** search for any information, including posts, public pages, interest groups, people

As far as Twitter is concerned, actions considered are:

**tweet** post something to Twitter wall. Besides text, pictures, hyperlinks, and Facebook-like items, also a poll can be published

**re-tweet** re-publish someone else's tweet, either with an additional comment, or as it is—the source is automatically cited

**reply** reply to a tweet with another tweet specifically directed to the author of the replied tweet, and referring to the original tweet—the author is automatically tagged and the source tweet included

**favourite** manifest interest in a tweet. Similarly to Facebook "like", interest is usually positive

**share privately** send the link to a tweet to a friend with a private message

**follow** include a given profile in the list of followed profiles, that is, the list of observable profiles you get notifications from. Differently from Facebook friendship, Twitter "following" relationship is not symmetric

**search** search for free terms, profiles, hashtags

As far as Google$^+$ is concerned, actions considered are:

**post** post something on Google$^+$ wall

**share** share someone else's post, either with an additional comment, or as it is—the source is automatically embedded in the new post

**+1** manifest interest in a piece of information, e.g., a post or a comment. As for Facebook and Twitter, usually a positive interest

**comment** comment a post with text, or hyperlinks

**reply** reply to a comment with another comment

**add people** add people to the network of friends, that is, the list of observable people who can in turn observe the acting user

**search** search for people, profiles, communities, collections of information

As far as LinkedIN is concerned, actions considered are:

**post** post something on your LinkedIN profile, e.g., text, an hyperlink, a picture

**suggest** manifest interest in a piece of information, e.g., a post or a comment

**comment** comment a post with other information, e.g., text or hyperlinks

**share** share someone else's post, either with and additional comment, or as it is—the source is automatically cited

**connect** include a given person in the list of connections, that is, the list of observable people who can in turn observe the acting user

**search** search for people, job offers, firms, groups, universities, posts, even personal mail

As far as Mendeley is concerned, actions considered are:

**publish** publish an already published paper. The paper becomes available in the acting user's profile, and searchable through Mendeley search facilities

**post** publish something on Mendeley news feed, e.g., text or hyperlinks

**like** manifest interest in a piece of information, e.g., a publication, or a post

**comment** comment a post with other information, e.g., text or hyperlinks

**cite** share someone else's publication, either with an additional comment, or as it is, while attributing authorship

**download** download the publication, if made available by the author

**follow** include a given author in the list of followed profiles, that is, the list of observable authors you get notifications from. Similarly to Twitter following relationship, this is not symmetric

**search** search for people, papers, or interest groups

As far as Academia.edu is concerned, actions considered are:

**publish** publish a paper, either already published, or new. The paper becomes available in the acting user's profile, and searchable through Academia.edu search facilities

**bookmark** store a publication in a special, public bookmarks section for later retrieval

**download** download the publication, if made available by the author, otherwise ask for a private copy

**follow** include a given author in the list of followed profiles, that is, the list of observable authors you get notifications from. As for Mendeley following relationship, this is not symmetric

**search** search for papers, people, universities

As far as ResearchGate is concerned, actions considered are:

**publish** publish a piece of information, whose type can be dynamically designed. Supported publication types are, among the many, article, book, code, dataset, patent, presentation, thesis, and so on. Published information becomes available from the publisher profile, and searchable through ResearchGate search facilities

**comment** comment a published item with other information, e.g., text or hyperlinks

**download** download a given piece of information, if made available by the author, otherwise ask for a private copy

**follow** follow an author, a publication, or a question. Regardless of the subject of the follow action, the follower now gets updates regarding modifications of the followed source of information—e.g., new publications from authors, new comments on publications, new answers to questions. The following relationship is not symmetric, whatever are the involved entities

**ask** publish a question on your profile, which anybody can publicly answer to

**answer** publicly answer a published question

**vote** publicly up-vote or down-vote either a question or an answer

**endorse** publicly endorse a given skill of a given person

**search** search for people, publications, questions, job offers

As far as Storify is concerned, actions considered are:

**add source** add a source of information to your story. The admissible sources are, among the many, other Storify elements, Twitter tweets and user profiles, Instagram[1] posts and users, YouTube videos, any Google search result, and many more

**add content** add content to your story, taken from one of the already added sources

**comment story** add comments to a specific content of a given story

**publish story** publish a story. From now on, the story can be publicly used a source of information into other stories

**like story** manifest interest in a whole story

**share story** share someone else's story

**search stories** search for stories based on free terms

**follow** include a given storyteller in the list of followed profiles, that is, the list of observable profiles you get notifications from. Like in the case of Twitter, the relation is not symmetric

[1]`http://www.instagram.com`

### 5.3.2 Factorisation of Common Actions

Based on the extensive survey just described, a core set of common actions shared by almost all the heterogeneous STS just described are indentified, whose goals, despite their diversity, are almost identical—at the level of abstraction suggested by BIC theory:

**share** the share action encompasses posting novel information, sharing or citing someone else's posts, publishing papers or stories, asking questions, and so on. Namely, any action whose effect is that of adding information to the system

**mark** the mark action encompasses liking a post, voting a question/answer, bookmarking a publication, giving +1 to a post, and so on. Namely, any action whose effect is that of marking information as relevant or not, qualitatively or quantitatively

**annotate** the annotate action encompasses, besides the obvious annotation to stories, comments to existing posts, replies to comments, answers to questions. Namely, any action whose effect is that of attaching a piece of information to an existing information

**connect** the connect action encompasses adding friends, following people or sources of information, and adding content and sources to stories. Namely, any action expanding the network of relationships between a user and other users or sources of information

**harvest** the harvest action encompasses all kinds of search actions, whatever their target is. Namely, any action aimed at acquiring knowledge about either potential sources of information or potentially relevant pieces of information

**Tacit messages from actions** Each of the common actions just devised out may convey different tacit messages, depending on a number of factors, such as the context within which the action occurred, the source and target of the action, and the degree of mutual awareness the specific platform supports.

In the following, a few tacit messages are sketched for each action, so as to give the reader a clue about how BIC may be used in real-world scenarios—tacit messages are referred to using their numbers, as in Subsection 5.2.3.

**share** re-publishing or mentioning someone else's information can convey, e.g., tacit messages $1, 3, 5$. If $X$ shares $Y$'s information through action $a$, every other agent observing $a$ becomes aware of existence and location of both $X$ and $Y$ (1). The fact that $X$ is sharing information $I$ from source $S$ lets $X$'s peers infer $X$ can manipulate $S$ (3). If $X$ shared $I$ with $Z$, $Z$ may infer, e.g., that $X$ expects $Z$ to somehow use it (5)

**mark** marking as relevant a piece of information can convey, e.g., tacit messages $1, 4$. If the socio-technical platform lets $X$ be aware of $Y$ marking information $I$ as relevant, $X$ may infer that $Y$ exists (1). If $Y$ marks as relevant $I$ belonging to $X$, $X$ may infer that $Y$ is interested in her work, perhaps seeking for collaborations (4)

**annotate** annotating a piece of information can convey, e.g., tacit messages $5, 6$. Since $X$ annotated, e.g., $Y$'s post, any agent observing $X$ may infer she just finished reading that post (5). Furthermore, by interpreting the content of the annotation, agents peers of $X$ may infer the motivation behind $X$'s annotation (6)

**connect** subscribing for updates regarding some piece of information or some user can convey tacit messages $2, 4$. Since $X$ manifested interest in $Y$'s work through subscription, $Y$ may infer $X$ intention to use it somehow (2). Accordingly, $Y$ may infer the opportunity for, e.g., collaboration (4)

**harvest** performing a search query to retrieve information can convey, e.g., tacit messages $1, 2, 4$—it should be noted, however, that which assumptions to make about a search action heavily depends on which search criteria are supported. If $X$ search query is observable by peer agents, they can infer $X$ existence and location (1). Also, they can infer $X$ goal to acquire knowledge related to its search query (2). Finally, along the same line, they can take the chance to provide matching information (4)

**Perturbation actions from tacit messages** The last step still missing to bridge the gap between theory and practice regarding BIC in real-world STS, is to answer the question: how can a coordination middleware, underlying the socio-technical platform for KM, take advantage of all the possible tacit messages, just ascribed to (inter-)actions, for the benefit of the coordination process? The answer proposed in the following is: through *perturbation actions*.

Perturbation actions are computational functions/processes changing the state of a STS in response to users' interactions, but *transparently* to them [MO15a]. Perturbation actions exploit the implicit information conveyed by tacit messages, leveraging the *mind-reading* and *signification* abilities ascribed to agents (either software or human), as well as to the (smart) computational environment, with the aim of *tuning* the coordination process so as to better support the ever-changing KM related needs of the socio-technical platform, and its users.

Accordingly, perturbation actions may, e.g., *(i)* spread discovery messages informing agents about the presence and location of others (tacit message *presence*), *(ii)* establish privileged communication channels between frequently interacting agents (*opportunity*), *(iii)* encourage/obstruct some desirable/dangerous interaction protocols or situated actions (*intention*, *ability*, *goal*), *(iv)* recommend to users novel, potentially interesting information as soon as it is available (*accomplishment*, *result*).

Not by chance, perturbation actions are one essential abstraction of the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model, as described in Chapter 6 in Part II of this thesis, where their role and how they work in a concrete middleware is thoroughly described. Here, the survey is concluded by providing some clues about which possible perturbation actions may be put to work by the aforementioned real-world socio-technical platforms, based on the devised pool of common actions. However, what follows is a speculation on what could be possibly done behind the scenes of these platforms, since for many of them no public disclosure of, e.g., ranking algorithms is available.

**share** provided by Facebook, Twitter (retweet), G+, LinkedIN, Mendeley (post), Storify, Academia.edu (publish), ResearchGate (publish), etc. It is likely to help the STS platform, underlying the social network application, in

- suggesting novel connections, based on common interests
- ranking feeds in the newsfeed timeline, based on similarity
- tuning personalised advertising policies, based on posts topic
- recommending content (e.g., job offers, publications, news stories), based on similarity

**mark** provided by Facebook (like), Twitter (like), G+ (+1), LinkedIN (suggest), Mendeley (like), Academia.edu (bookmark), ResearchGate (follow/download), Storify, etc. It is likely to influence the STS similarly to what described above, with the addition of enabling/strengthening notifications for the marked items

**annotate** provided by almost any STS, it is likely to help the STS platform by

- suggesting novel connections, based on shared annotations
- recommending content (e.g., job offers, publications, news stories), based on annotated items
- enabling/strengthening notifications for the annotated items

**connect** provided by Facebook (add friend), Twitter (follow), G+ (add), LinkedIN, Mendeley (follow), Academia.edu (follow), ResearchGate (follow), etc. It is likely to help the STS platform by

- suggesting further connections
- activating/ranking feeds in the newsfeed timeline
- enabling/strengthening notifications for new/old connections

**harvest** provided by almost every social network, it is the epistemic action by its very definition, thus may be exploited by the STS platform in a wide number of ways, heavily depending on which are the searchable information items, and which are the searching parameters. Among the many possible reactions:

- re-organise the knowledge graph internally used by the STS
- tune the algorithm providing suggestions
- improve personalised advertising policies
- suggest novel connections, based on search terms
- ranking feeds in the newsfeed timeline, based on search results

## 5.4 Remarks & Outlook

The material presented in this chapter is fundamental to the conception of the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model, and affects the whole thesis, too.

First of all, the A&A meta-model is the reference framework for thinking about MAS modelling and engineering. The notion of *coordination artefact* as mediator of both social and situated interaction within any distributed computational system is central not only to $\mathcal{M}o\mathcal{K}$, but to the whole thesis—as witnessed by Chapter 4, in particular. The same holds true for stigmergic coordination, either cognitive or not.

The whole approach to coordination in the data-driven way based on tuple spaces encourages and directly supports adoption of *environment mediated interaction*, where the environment is the network of tuple spaces itself.

Furthermore, the idea of interpreting the (computational) environment as an *active* component of the system, able to *autonomously* perform actions (think about pheromone diffusion, aggregation, and evaporation), is used not only in $\mathcal{M}o\mathcal{K}$, but also in the chemical metaphor for self-organising coordination described in Chapter 2, where the environment is a tuple space enriched with a pro-active behaviour—the simulation of a chemical solution dynamics.

Focussing the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model in particular, besides the notions of artefact, workspace, and stigmergic coordination being enabler for $\mathcal{M}o\mathcal{K}$ technology, the socio-cognitive theory of BIC is the "big deal". BIC provides the conceptual framework to knowledgeably fill a gap in current coordination models and languages theory and practice, that is, the fact the coordination model obviously affects users' interactions, but there is no way to have users affect the coordination processes while performing their usual activities. Furthermore, BIC provides the means to do so *transparently* to users, as an *implicit* facet of their everyday (interaction) activities.

Being able to do a similar thing is of paramount importance, especially within the context of knowledge-intensive STS. As already pointed out, *mutual awareness* (including peripheral awareness, too), as well as the ability to delegate activities to *coordinative artefacts* – according to [SS00] terminology – is the only path toward bridging the gap between current approaches to KM.

For all these reasons, the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model discussed in the upcoming Part II is heavily centred around the notions of tacit message (*traces* in $\mathcal{M}o\mathcal{K}$) and perturbation action—besides being obviously based on situatedness of interactions and chemical-inspired self-organisation.

# Part II

# Self-organisation of Knowledge in $\mathcal{M}$olecules $\mathcal{o}$f $\mathcal{K}$nowledge

In this second part of this thesis, the $\mathcal{M}$olecules $\mathcal{o}$f $\mathcal{K}$nowledge ($\mathcal{MoK}$) model and technology for self-organisation of knowledge in knowledge-intensive socio-technical systems is presented. $\mathcal{MoK}$ is the result of the integration of all the different, but complementary, contributions presented so far, in Part I of this thesis.

In fact: *(i)* it is a self-organising biochemical coordination model, where coordination laws are biochemical reactions, information items are molecules, and the agents to coordinate are the chemists working with a "knowledge-based chemical solution"—Chapter 3; *(ii)* it has been prototyped as a middleware layer for knowledge management applications, by implementing it upon the TuCSoN coordination infrastructure, fully exploiting its situated architecture, for distribution, and the ReSpecT language, for spatio-temporal awareness and adaptiveness—Chapter 4; *(iii)* it integrates a socio-cognitive theory of action and interaction since its very foundation, adopting the behavioural implicit communication perspective over anticipatory coordination via tacit messages and stigmergic coordination—Chapter 5.

# Chapter 6

# The $\mathcal{M}$olecules $\mathcal{o}$f $\mathcal{K}$nowledge Model

Nowadays, ICT systems have gone far beyond pure *algorithmic*, Turing Machine like computation, as Turing himself anticipated long ago with o-machines and c-machines [Tur39], and Wegner later highlighted [Weg97] with its equation "computation = algorithm + *interaction*". The question "how to manage interactions?" then, originated a whole research landscape, branded *coordination models and languages*, which is specifically devoted to figure out how to *manage dependencies* arising from interacting activities in systems of any sort [MC94]. However, in recent years a plethora of novel *open*, *pervasive*, highly *dynamic*, and mostly *unpredictable* systems, such as self-organising ones, have surged, presenting brand new challenges demanding for innovative coordination approaches [OV11].

*Socio-Technical Systems* (STS) and *Knowledge-Intensive Environments* (KIE) are both blatant examples of such a sort of systems, being them conceived and designed to combine business processes, technologies and people's skills [Whi06] to store, handle, and make accessible large repositories of information [Bha01].

Managing their *interaction space* is of paramount importance, for guaranteeing their functionalities, as well as for providing desirable non-functional properties [SS00]—e.g., scalability, fault-tolerance, self-* properties in general. However, engineering coordination mechanisms and policies accordingly is far from trivial, mostly due to the peculiar characteristics of the aforementioned systems highlighted in Chapter 5.

The *data-driven approach* to coordination [DPHW05], such as tuple space based models [Gel85], aims at coordinating interacting agents by properly managing access to information, rather than by directly *commanding* the parties about what to do, and when. A similar approach seems very suitable for the above described knowledge-intensive STS, where data, information, and then knowledge drive both the business goals of the organisation as a whole, as well as the everyday activities of the co-workers sharing the computational platform in charge of knowledge management.

Therefore, a possibly novel research thread departs from the question: why to stick viewing data as passive, "dead" things to run algorithms upon in the traditional I/O paradigm? Why not to foster a novel interpretation of both knowledge management and data-driven coordination in general, where *data is alive*, a living thing spontaneously and continuously evolving so as to better serve the ever-changing needs of the organisation?

The Molecules of Knowledge model is precisely devoted to answer the question, providing a *self-organising*, *user-centric* approach to coordination for knowledge-intensive STS.

**Overview** A MoK-coordinated system is a network of MoK *compartments* (information containers), in which MoK *seeds* (sources of information) continuously and autonomously inject MoK *atoms* (atomic information pieces).

MoK atoms may then aggregate into *molecules* (composite information chunks), diffuse to neighbouring compartments, lose relevance as time flows, gain relevance upon users interactions, or have their properties modified, as a consequence of user agents' interactions.

These autonomous and decentralised processes are enacted by MoK *reactions* (the coordination laws dictating how the system evolves), and are influenced by MoK *enzymes* (reification of user agents' actions) and by MoK *traces* (actions' side effects).

Traces are transparently, and possibly unintentionally, left within the working environment by MoK *catalysts* (users, either human or software agents) while performing their activities.

## 6.1 Core Abstractions

The MoK model is built around the following abstractions.

**Seeds** As MoK *sources of information*, seeds are meant to represent any facet of a source of information and to make available any information therein contained which may be useful for the application at hand. Sources of information reified by seeds may be legacy DBs, web pages, RSS channels, files on disk, sensor devices, and so on: namely anything capable of generating raw data or (un)structured information.

Seeds generate information *continuously* and *autonomously*, that is: any data chunk is produced by a seed multiple times, not once, at a *rate* depending on heterogeneous and dynamic *contextual information*—e.g., congestion of the MoK system, predicted relevance of the information, users' preferences, and so on; any data chunk is produced spontaneously by the seed, with no need for external intervention—still allowed for agents willing to influence atoms generation process.

Seeds may be formally defined as follows:

$$\texttt{seed}(\mathit{Src},\mathit{Atoms})_c, \quad c \in \Re^+$$

$$\mathit{Src} ::= \texttt{URI}$$

$$\mathit{Atoms} ::= \mathit{atom}^i_{c_i} \mid \mathit{atom}^i_{c_i},\mathit{Atoms}, \quad c_i \in \Re^+,\ i \in \mathbb{N}$$

where: `Src` is meant to provide an always up-to-date reference to the original source of information, wherever it is, and in whichever format it is persistently stored—e.g., the URL of a web page, the absolute path of a file on disk, and so on; `Atoms` is the set of atoms to be released into the compartment where the seed lives, each with its own initial concentration value.

**Atoms** As $\mathcal{M}o\mathcal{K}$ *atomic unit of information*, atoms are meant to represent any piece of information which can be regarded as being *atomic*, that is, non further decomposable in smaller chunks—the granularity of information is decidable either by agents manually creating atoms, or by $\mathcal{M}o\mathcal{K}$ itself automatically extracting atoms from seeds.

Accordingly, atoms may represent table cells of a relational DB, nodes in a graph DB, document content or metadata in a document-oriented DB; hyperlinks of a web page, feeds of an RSS channel, posts or comments in a blog; word, phrases, or paragraphs in an article, and so on.

Atoms are *continuously* and *autonomously* injected by seeds into compartments, where they become susceptible to $\mathcal{M}o\mathcal{K}$ computational process—e.g., they can be aggregated into more complex information heaps, exploited to inference novel knowledge, or shared with other membrane-connected compartments.

Atoms may be formally defined as follows:

$$\texttt{atom(}\textit{src}\texttt{,}\textit{Content}\texttt{,}\textit{Meta-info}\texttt{)}_c, \quad c \in \Re^+$$

$$\textit{Content} ::= \texttt{raw information}$$
$$\textit{Meta-info} ::= \texttt{(semantic) metadata}$$

where: `Content` is the raw piece of atomic information the atom conveys – e.g., records of a DB, comments of a blog, sentences of an article, and so on –, whereas `Meta-info` is meant to store any kind of metadata, related to the atom's content, supporting knowledge inference & discovery—e.g., the DB schema the content must comply to, a set of synonyms for a set of words, the categorisation of an article topic based on a dedicated ontology, and so on.

**Molecules** As $\mathcal{M}o\mathcal{K}$ *composite unit of information*, molecules are meant to represent any piece of information which cannot be regarded as being atomic. Accordingly, molecules may represent table rows in a relational DB, sub-graphs in a graph DB, a collection of documents in a document-oriented DB; hyperlinks chains in a web page, related feeds of an RSS channel, posts with comments in a blog; similar words, related phrases, or sections in an article, and so on.

As such, molecules are, in general, *collections of semantically related atoms*, where the semantics underlying the aggregation process depends on the business domain of the application at hand—e.g., for a citizen journalism IT platform, the driving force may be the semantic similarity of news articles based on a given ontology.

Molecules are *continuously* and *autonomously* generated by $\mathcal{M}o\mathcal{K}$ computational processes, to whom they are susceptible likewise atoms—still, agents, too, can manually create, manipulate, and share molecules. Molecules may be formally defined as follows:

$$\texttt{molecule(}\mathit{Atoms}\texttt{)}_c, \quad c \in \Re^+$$

$$\mathit{Atoms} ::= \mathit{atom}^i \mid \mathit{atom}^i, \mathit{Atoms}\,, \quad i \in \mathbb{N}$$

$$\{\mathit{Atoms} \mid \mathcal{F}_{\mathcal{M}o\mathcal{K}}(\mathit{atom}^i, \mathit{atom}^j) = \delta \geq th\}, \quad i \neq j \in \mathbb{N}, \quad \delta, th \in (0,1] \in \Re$$

where the atoms aggregated by a molecule have to be, according to $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$, sufficiently ($\delta$) semantically related – the threshold $th$ defining "sufficiently" being necessarily application specific –, and each atom within the molecule must be unique—it is meaningless to aggregate identical atoms.

**Compartments** As $\mathcal{M}o\mathcal{K}$ *knowledge containers*, compartments are the *computational* abstraction in $\mathcal{M}o\mathcal{K}$, responsible for (i) handling the whole information lifecycle – from storage to sharing, to knowledge inference – as well as its availability to agents, and for (ii) scheduling and executing the processes manipulating information and supporting knowledge inference, discovery, and sharing.

As such, compartments act as *active repositories of knowledge*, not merely storing information persistently to make it available to agents upon need, but also *autonomously* evolving information – possibly improving its quality and inferring novel knowledge –, and pro-actively sharing information with other compartments—possibly moving it closer to where it is needed more.

Compartments may be formally defined as follows:

$$[\![\mathit{Seeds}\,, \mathit{Atoms}\,, \mathit{Molecules}\,, \mathit{Enzymes}\,, \mathit{Traces}\,, \mathit{Reactions}\,, \asymp^j]\!], \quad j \in \mathbb{N}$$

where brackets $[\![\cdot]\!]$ denote the compartment, and symbol $\asymp$ denotes a membrane between the defined compartment and others (iterated over by $j$).

**Membranes** As $\mathcal{M}o\mathcal{K}$ *interaction channels*, membranes are the *communication* abstraction in $\mathcal{M}o\mathcal{K}$, as well as its *topological* abstraction.

On the one hand, membranes enable exchange of information between compartments; on the other hand, by establishing 1 : 1 communication channels, membranes implicitly define the notions of *locality* and *neighbourhood* in $\mathcal{M}o\mathcal{K}$. Locality, because they forbid interaction between compartments which are not coupled by any membrane, thus confine the computational processes therein. Neighbourhood, because they allow computational processes to involve more than one compartment, given they are coupled by a membrane—e.g., sharing of information necessarily involves a sending compartment and, at least, a receiving compartment.

Membranes may be formally defined as follows:

$$[\![\ldots]\!]_i \asymp [\![\ldots]\!]_j, \quad i \neq j \in \mathbb{N}$$

where symbol $\asymp$ denotes the membrane connecting compartments $i$ and $j$.

**Catalysts** As $\mathcal{MoK}$ *knowledge workers*, catalysts represent within $\mathcal{MoK}$ the agents – either software or humans – which need to undertake *(epistemic) actions* over the information living within the $\mathcal{MoK}$ system, in order to achieve their business goals.

As a side effect of their activity, catalysts *influence* – intentionally or not – both (i) the way in which $\mathcal{MoK}$ processes apply to information and knowledge, and (ii) information properties—such as relevance and location. Influence is based on both (i) the *epistemic nature* of the activity taking place – e.g., a "search" action vs. a "share" action –, and (ii) contextual information regarding the action itself—e.g., which information is involved, where it is, who is doing the action, when, and so on.

As long as the goal is that of improving information quality, promoting knowledge inference and discovery, and supporting effective information sharing, any system property and any information available to $\mathcal{MoK}$ may be affected—e.g., tuning processes working rate, manipulating information content and location, and so on.

Catalysts may be formally defined as follows:

$$Catalyst \simeq (\alpha \dagger [\![\ldots]\!]).Catalyst$$

$$\alpha \in \{\texttt{share(}\mathit{Reactant}\texttt{)} \mid \texttt{mark(}\mathit{Reactant}\texttt{)} \mid \texttt{annotate(}\mathit{Reactant}\texttt{)} \mid \texttt{connect(}\mathit{Reactant}\texttt{)} \mid \texttt{harvest(}\mathit{Reactant}\texttt{)}\}$$

$$\mathit{Reactant} ::= \mathit{seed} \mid \mathit{atom} \mid \mathit{molecule}$$

where $\alpha$ denotes actions available to catalysts, and symbol $\dagger$ denotes application of the action to a compartment—a catalyst behaviour is then modelled ($\simeq$) as the sequence of actions she performs within different compartments.

**Enzymes** As $\mathcal{MoK}$ *reification of actions*, enzymes reify within $\mathcal{MoK}$ (i) the *epistemic nature* of all the actions available to agents for information handling and knowledge discovery, as well as (ii) any *contextual information* related to actions.

The former is vital to $\mathcal{MoK}$ because it enables *adaptiveness*, that is, the ability of $\mathcal{MoK}$ computational processes to dynamically change their functioning according to what the agents are doing—e.g., search queries for a given keyword may increase the frequency at which related information is shared from neighbouring compartments.

The latter is fundamental because it enables *situatedness* & *awareness*, that is, the ability of $\mathcal{MoK}$ to precisely characterise actions in space, time, and w.r.t. any other property of the computational environment they are taking place within—e.g., who is taking action, where (e.g., in which compartment), when, and so on.

Enzymes are *automatically* and *transparently* (to agents) produced by the compartment where the action is taking place, which means (i) agents need to do nothing else beyond their usual actions to generate enzymes – thus there is no cognitive/computational overhead for them –, and that it is not necessary for agents to be aware of enzymes being released whenever they undertake actions—nevertheless, awareness may help them gain better assistance from $\mathcal{MoK}$, as discussed while introducing Behavioural Implicit Communication (BIC) in Chapter 5.

Enzymes may be formally defined as follows:

$$\texttt{enzyme(}\mathit{Species}\texttt{,}s_{species}\texttt{,}\mathit{Reactant}\texttt{,}\mathit{Context}\texttt{)}_c, \quad c, s \in \Re^+$$

$$\mathit{Species} ::= \texttt{share} \mid \texttt{mark} \mid \texttt{annotate} \mid \texttt{connect} \mid \texttt{harvest}$$
$$\mathit{Reactant} ::= \mathit{atom} \mid \mathit{molecule}$$
$$\mathit{Context} ::= \texttt{any contextual info}$$

where: *Species* denotes the epistemic nature of the action the enzyme reifies, and contributes to determine the perturbation action carrying out the influences of the action on the $\mathcal{MoK}$ system—through traces, see below; $s_{species}$ depends on *Species* and denotes the strength of the enzyme, that is, the magnitude of the relevance boost brought to the semantically related atom or molecule; *Context* is meant to track any contextual information regarding the action which may be useful for coordination purpose (e.g., time of execution, outcome, previous & following action, etc.).

**Traces** As $\mathcal{MoK}$ *reification of actions' effects*, traces reify within $\mathcal{MoK}$ any (side) effect that any action could cause, that is, any modification to $\mathcal{MoK}$ computational environment, which is due to the action but may not be its intentional primary effect. A category of (side) effects of actions, particularly interesting for $\mathcal{MoK}$, is that of *tacit messages* [CPT10], thoroughly discussed in Chapter 5—and briefly recalled in Section 6.3.

Traces are *automatically* and *transparently* (to agents) produced by the compartment where the action took place, similarly and consequently to enzymes. Together, enzymes and traces enable *anticipatory coordination* in $\mathcal{MoK}$—see Section 6.3.

Traces may be formally defined as follows:

$$\texttt{trace(}\mathit{Msg}\texttt{,}\mathit{Target}\texttt{)}_c, \quad c \in \Re^+$$

$$\mathit{Msg} ::= \texttt{presence} \mid \texttt{intention} \mid \texttt{ability} \mid \texttt{opportunity} \mid \texttt{accomplishment} \mid \texttt{goal} \mid \texttt{result}$$
$$\mathit{Target} ::= \mathit{seed} \mid \mathit{atom} \mid \mathit{molecule} \mid \asymp \mid \mathit{Reaction}$$

where: *Msg* is the tacit message conveyed by the trace, depending on the corresponding enzyme's species and action context, and (possibly) modulating the way the trace affects the system state—see Section 6.3 for details; *Target* is what could be affected by the trace, depending on the generating enzyme species and reactant.

Along with *Msg*, any additional information regarding the tacit message may be stored in a $\mathcal{MoK}$ trace – e.g., the acting agent location –, to be later retrieved by $\mathcal{MoK}$ reactions – applying perturbation actions, in particular – when needed.

**Perturbations** As *MoK interactions' side effects*, perturbations are the *computational functions* in charge of applying (application specific, custom defined) side effects of agents' interactions with MoK, actually enacting the *influence* on information lifecycle – as well as on knowledge inference, discovery and sharing – MoK agents may have.

Accordingly, perturbation actions may affect both (i) MoK autonomous processes functioning – e.g., their application rate, their targets, and so on – and (ii) information and system properties—e.g., content, location, relevance, and so on.

Perturbation actions are associated to MoK traces – deposited by enzymes, in turn – and *autonomously* carried out by MoK compartments as soon as possible, which means, when (i) the trace causing the perturbation is present in the compartment, and (ii) the body of knowledge or the system property target of the perturbation is available for manipulation.

Summing up, perturbation actions enact the influence of agents on the body of knowledge in MoK, coherently w.r.t. the side effects of their interactions as represented by traces, which are automatically generated from the original (epistemic) action as reified by enzymes.

Perturbation actions may be formally defined as follows:

$$\texttt{perturbation}(P_{species}, \mathit{Target})$$

$$P_{species} ::= \texttt{attract} \mid \texttt{repulse} \mid \texttt{approach} \mid \texttt{drift-apart} \mid \texttt{strengthen} \mid \texttt{weaken} \mid \texttt{boost} \mid \texttt{wane}$$

$$\mathit{Target} ::= \mathit{seed} \mid \mathit{atom} \mid \mathit{molecule} \mid \asymp \mid \mathit{Reaction}$$

where $P_{species}$ is the perturbation action caused by the trace, depending on the trace tacit message and its target, as well as (indirectly) from the original enzyme species and context—see Section 6.3 for details on correspondences.

Description of the meaning and purpose of each perturbation action is detailed in Subsection 6.3.1. Nevertheless, a brief overview is undoubtedly useful: `attract` perturbation action brings information closer to where the original action took place (`repulse` does the opposite); `approach` facilitates exchange of information between compartments (the one where the action took place and the one where the trace ends up being, `drift-apart` does the opposite); `strengthen` increases concentration of information (`weaken` does the opposite); `boost` increases rate of a MoK reaction (`wane` does the opposite).

**Reactions** As *MoK knowledge dynamics processes*, reactions are the *autonomous* and *decentralised* processes supporting (meta) information handling as well as knowledge inference, discovery and sharing in MoK.

Autonomous, because reactions are scheduled and executed by MoK compartments according to *dynamic rate expressions* – inspired by rate expressions of natural chemical reactions –, which are meant to support *awareness* of the contextual information which may affect reactions application. This in turn enables *adaptiveness* to the external influences put by interacting agents.

Decentralised, because $\mathcal{M}o\mathcal{K}$ reactions may apply to seeds, atoms, molecules, as well as enzymes and traces, depending on their nature and purpose, but anyway they only rely on local information since they can only affect *neighbouring* compartments, at most—many reactions are confined within a single compartment.

The rationale driving reactions application to seeds, atoms, etc. – that is, to which specific seed, atom, etc. a given reaction is applied to – is *similarity* between the reactant *templates* used in the left hand side of the reaction and the actual reactants available in the reacting compartment.

The semantics underlying this similarity measure is business domain specific—e.g., in the case of a IT platform for collaboration and sharing of papers between researchers, documents similarity measures such as squared Euclidean distance, cosine similarity, and relative entropy are all reasonable alternatives.

Details regarding the nature and purpose of each $\mathcal{M}o\mathcal{K}$ reaction, as well as description of requirements for the similarity measure, are given in Section 6.2 and Section 6.4, respectively. Reactions formalisation, too, follows in next section.

## 6.2 Focus on Reactions

In this section $\mathcal{M}o\mathcal{K}$ reactions are thoroughly described. First, their structure and their execution rate are formally defined (Subsection 6.2.1), then, a report on their evaluation is provided, by simulating their interplay according to custom kinetic rates, using the BioPEPA tool for chemical solutions simulation (Subsection 6.2.2). This also motivates why that specific rate expressions have been chosen.

It should be noted that, to enhance readability and avoid redundancy, from the kinetic rate expressions of each reaction has been omitted a multiplicative parameter $\hbar$, meant to represent a hook for tuning $\mathcal{M}o\mathcal{K}$ reactions' application rates—e.g., through $\mathcal{M}o\mathcal{K}$ perturbation reaction. This parameter is initially set to 1, thus does not affect the effective rate of execution. Then, as the $\mathcal{M}o\mathcal{K}$ system evolves, it may be either increased or decreased so as to speed up or slow down the reaction rate upon need—e.g., due to catalysts interactions.

### 6.2.1 Formal Description

As briefly described in Section 6.1, reactions are $\mathcal{M}o\mathcal{K}$ *autonomous* and *decentralised* processes supporting (meta) information handling as well as knowledge inference, discovery and sharing—essentially, they determine the global behaviour of a $\mathcal{M}o\mathcal{K}$ system, together with agents' interactions. On purpose, $\mathcal{M}o\mathcal{K}$ features the following reactions.

**Injection** $\mathcal{M}o\mathcal{K}$ *injection reaction* generates atoms from seeds, at a given rate, putting them into the compartment where the seed is. Injection reaction may be formally defined as follows:

$$\texttt{seed}(\mathit{src},\mathit{Atoms}) \xrightarrow{r_{inj}} \texttt{seed}(\mathit{src},\mathit{Atoms}) + \mathit{atom}^{i}_{c+c_i}, \quad \forall \mathit{atom}^{i}_{c_i} \in \mathit{Atoms}$$

$$r_{inj} = \frac{1}{1+\texttt{time}} * \texttt{diff}(\sharp \mathit{seed}, \sharp \mathit{atom})$$

where:

- superscript $i$ denotes the i-th atom in the set of atoms $\mathit{Atoms}$—and, accordingly, its concentration, denoted by subscript $c_i$
- $\sharp$ is the operator returning concentration of its operand
- $\texttt{diff}(\cdot,\cdot)$ is a function computing any one notion of difference between concentrations (e.g., subtraction)
- `time` is the operator measuring the flow of time

In particular, given a seed able to generate a set of atoms, each of them is injected in the local compartment, with its own concentration – adding up to the concentration of identical atoms already present –, every $\frac{1}{r_{inj}}$ time steps[1].

The dynamic rate expression of the injection reaction represents a trade-off between two contrasting needs: on one hand, atoms should be *perpetually* injected into a $\mathcal{MoK}$ system, since there is no reasonable way to know a-priori *when* some information will be useful the most; on the other hand, it is desirable to avoid *flooding* the system without any control on the quantity of atoms – thus the size of the knowledge base – in play.

Accordingly, $r_{inj}$ is designed to: (i) slower the injection process as time passes, thus as the compartment likely becomes more and more congested, (ii) comply to a threshold enforcing an upper bound on the concentration of injected atoms—as soon as saturation is reached, the injection reaction stops until the atoms' concentration lowers.

**Aggregation** $\mathcal{MoK}$ *aggregation reaction* ties together *semantically related* atoms into molecules, or molecules into other molecules, at a given rate. Aggregation reaction may be formally defined as follows:

$$\mathit{Reactant}' + \mathit{Reactant}'' \xrightarrow{r_{agg}} (\mathit{Reactant}' \oplus \mathit{Reactant}'') + (\mathit{Reactant}' \ominus \mathit{Reactant}'')$$

$$\left.\begin{aligned} \oplus &= \{\mathit{Reactants} \mid \mathcal{F}_{\mathcal{MoK}}(\mathit{Reactant}', \mathit{Reactant}'') = \delta \geq th\} \\ \ominus &= \{\mathit{Reactants} \mid \mathcal{F}_{\mathcal{MoK}}(\mathit{Reactant}', \mathit{Reactant}'') = \delta < th\} \end{aligned}\right\} \delta, th \in (0,1] \in \Re$$

$$\mathit{Reactant} ::= \mathit{atom} \mid \mathit{molecule}$$

$$\mathit{Reactants} ::= \mathit{Atoms} \mid \mathit{Molecules}$$

$$r_{agg} = \frac{\texttt{time}}{\sharp \mathit{Reactant}^{lhs}}$$

[1]Approximately, due to the stochastic nature of the scheduling algorithm used by $\mathcal{MoK}$ compartments to execute reactions, detailed in Section 7.1 of Chapter 7.

where:

- same apices ($'$) denote same reactants
- *th* is an arbitrary threshold, necessary to determine whether different information pieces are sufficiently semantically related to be aggregated
- $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ is the function computing *semantic similarity* in $\mathcal{M}o\mathcal{K}$, thoroughly described in Section 6.4
- $\oplus$ is the operator returning the set of reactant obtained by extracting from the two input sets only those reactants for which $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ result is above (or equal to) the threshold
- $\ominus$ is the operator returning the set of reactants obtained by extracting from the two input sets only those reactants for which $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ result is below the threshold—thus, it is the dual operator of $\oplus$
- superscript *lhs* denotes the left-hand-side of the reaction—notation $\sharp\mathtt{Reactant}^{lhs}$, therefore, counts the number of either one among reactants

and concentration values are not specified because they all equal 1. In particular, given two molecules – or two atoms, or one atom and a molecule –, every $\frac{1}{r_{agg}}$ time steps they are joined into a new one, binding together semantically related atoms while non-related ones are released back into the reacting compartment.

The dynamic rate expression of the aggregation reaction is meant to enforce a direct proportionality between the size of a molecule and the speed of aggregation: the bigger a molecule is – that is, the more knowledge a molecule is conveying –, the faster it will aggregate—that is, the more frequently it will attract other atoms/molecules.

**Diffusion** $\mathcal{M}o\mathcal{K}$ *diffusion reaction* moves atoms, molecules, and traces among *neighbouring* compartments, at a given rate. Diffusion reaction may be formally defined as follows:

$$[\![\mathtt{Reactants}' \cup \mathtt{Reactant}]\!]_i \asymp [\![\mathtt{Reactants}'']\!]_j \xrightarrow{r_{diff}} [\![\mathtt{Reactants}']\!]_i \asymp [\![\mathtt{Reactants}'' \cup \mathtt{Reactant}]\!]_j, \quad i \neq j \in \mathbb{N}$$

$$\mathtt{Reactant} ::= \mathtt{atom} \mid \mathtt{molecule} \mid \mathtt{trace}$$
$$\mathtt{Reactants} ::= \mathtt{Atoms} \mid \mathtt{Molecules} \mid \mathtt{Traces}$$

$$r_{diff} = \mathcal{d} * \mathtt{diff}([\![\sharp\mathtt{Reactant}]\!]_i, [\![\sharp\mathtt{Reactant}]\!]_j), \quad \mathcal{d} \in (0, 1] \in \Re$$

where:

- brackets $[\![\cdot]\!]$ delimit compartments

- subscript $i, j$ identify a specific compartment
- symbol $\asymp$ denotes membranes connecting compartments
- $d$ is an arbitrary weight factor tuning diffusion strength

In particular, given two compartments connected by a membrane, a molecule, an atom, or a trace, from either of the two, is removed and sent to the other – one unit of concentration at a time –, at the rate given by $r_{diff}$ dynamic rate expression.

The rate expression is meant to (i) provide an upper bound on diffusion, avoiding unbounded proliferation of foreign information pieces in compartments which are not their origin – here, diffusion asymptotically tends to balance distribution of reactants in neighbouring compartments –, (ii) provide a hook to fine-tune the extent to which reactants are diffused—here, only a fraction of the reactants are allowed to diffuse.

**Decay** *MoK decay reaction* decreases the concentration of atoms, molecules, enzymes and traces as *time flows*. Decay reaction may be formally defined as follows:

$$\mathtt{Reactant}_c \xrightarrow{r_{dec}} \mathtt{Reactant}_{c-1}, \quad c \in \Re^+$$

$$\mathtt{Reactant} ::= \mathtt{atom} \mid \mathtt{molecule} \mid \mathtt{enzyme} \mid \mathtt{trace}$$

$$r_{dec} = \begin{cases} \mathtt{fMA}(a_R) * \mathtt{log(1+time)}, & a_R \in \Re^+, \quad \mathtt{Reactant} \setminus \mathtt{trace} \\ \mathtt{diff}(\sharp[\Uparrow \mathtt{trace}], \sharp \mathtt{trace}) * \mathtt{log(1+time)} \end{cases}$$

where:

- `fMA` is the function implementing the *law of mass action* [Car08], that is, the mathematical model explaining and predicting the behaviour of solutions in dynamic equilibrium[2]
- $a_R$ is the affinity constant needed by `fMA`—which, here, can be diverse for each *Reactant* type and dynamically adapted
- symbol $\Uparrow$ denotes where a *MoK* abstraction comes from—in this case, it states that $\sharp[\Uparrow \mathtt{Trace}]$ is the concentration of the enzyme who generated the trace to decay

[2]The law states that the rate of an elementary reaction ($r_f$) – a reaction that proceeds through only one transition state, that is, one mechanistic step – is proportional ($k_f$) to the product of the concentrations of the participating molecules ($R^1, R^2$): $r_f = k_f[R^1][R^2]$. $k_f$ is called *rate constant* and, in chemistry, is a function of participating molecules affinity.

In particular, given an atom, molecule, enzyme, or trace, its concentration is decreased by 1 at the rate given by $r_{dec}$ dynamic rate expression.

This rate expression is designed by recognising that time dependency alone is not enough for a meaningful decay behaviour: it would simply end-up slowing down the saturation process provided by injection reaction.

Hence, logarithmic time dependency guarantees a smoother decay process while dependency on reactants' concentration makes the process asymptotically tend to homogenise information relevance.

**Reinforcement** $\mathcal{M}o\mathcal{K}$ *reinforcement reaction* increases the concentration of atoms and molecules according to $\mathcal{M}o\mathcal{K}$ agents' *interactions*, at a given rate. Reinforcement reaction is formally defined as follows:

$$\texttt{enzyme(}\mathit{species}\,,s_{species}\,,\mathit{Reactant}\,,\mathit{Context}\texttt{)} + \mathit{Reactant}'_c \xrightarrow{r_{reinf}}$$
$$\texttt{enzyme(}\mathit{species}\,,s_{species}\,,\mathit{Reactant}\,,\mathit{Context}\texttt{)} + \mathit{Reactant}'_{c+s}, \quad c, s \in \Re^+$$

$$\mathcal{F}_{\mathcal{M}o\mathcal{K}}(\mathit{Reactant}, \mathit{Reactant}') = \delta \geq th, \quad \delta, th \in (0,1] \in \Re$$

$$\mathit{Reactant} ::= \mathit{atom} \mid \mathit{molecule}$$

$$r_{reinf} = \texttt{diff(}\sharp[\Uparrow \mathit{Reactant}], \sharp \mathit{Reactant}\texttt{)}$$

where, in this case, symbol $\Uparrow$ states that $\sharp[\Uparrow \mathit{Reactant}]$ is either the concentration of the seed source of the atom to reinforce, or an average of the concentrations of the seeds sources of the atoms aggregated by the molecule to reinforce.

In particular, given an enzyme and a semantically related atom or molecule, the concentration the reactant is increased according to enzyme's specification, every $\frac{1}{r_{reinf}}$ time steps.

The dynamic rate expression of the reinforcement reaction is meant to enforce *situatedness* and *awareness* of the feedback $\mathcal{M}o\mathcal{K}$ agents implicitly provide to $\mathcal{M}o\mathcal{K}$. Thus, feedback should (i) be prompt, that is, rapidly and highly increase concentration despite decay, (ii) be bounded, both in time and space—enzymes do not diffuse and reinforcement does not consider neighbourhoods, (iii) depend on the nature of interactions, that is, on enzymes' epistemic nature (the species).

**Deposit** $\mathcal{M}o\mathcal{K}$ *deposit reaction* generates traces from enzymes, at a given rate, putting them into the compartment where the *(inter-)action* originally releasing the enzyme took place. Deposit reaction may be formally defined as follows:

$$\texttt{enzyme(}\mathit{species}\,,s\,,\mathit{Reactant}\,,\mathit{Context}\texttt{)} \xrightarrow{r_{dep}}$$
$$\texttt{enzyme(}\mathit{species}\,,s\,,\mathit{Reactant}\,,\mathit{Context}\texttt{)} + \texttt{trace(}\mathit{Msg}\,,\mathit{Target}\texttt{)}_{c+s}$$

$$r_{dep} = \frac{1}{1+\texttt{time}} * \texttt{diff(}\sharp \mathit{enzyme}, \sharp \mathit{trace}\texttt{)}$$

In particular, given an enzyme of a certain species, thus able to generate that species of traces, every $\frac{1}{r_{dep}}$ time steps $s$ concentration of the trace is deposited in the local compartment.

The dynamic rate expression of the deposit reaction is the same as that of the injection reaction, thus represents the same trade-off between *flooding* and *availability*. Nevertheless, remember that seeds do not decay while enzymes do, thus, the run-time dynamics of the two reactions may be considerably different.

**Perturbation** *$\mathcal{MoK}$ perturbation reaction* carries out the *(side) effects* of (interaction) activities undertaken by $\mathcal{MoK}$ agents, at a given rate, considering the enzymes' species and traces' tacit message. Perturbation reaction may be formally defined as follows:

$$\texttt{trace(}\mathit{Msg}\texttt{,}\mathit{Target}\texttt{)} + \mathit{Target}' \xrightarrow{r_{pert}} \dagger(\texttt{perturbation(}P_{species}\texttt{,}\mathit{Target}\texttt{)}, \mathit{Target}')$$

$$\mathit{Target} ::= \mathit{seed} \mid \mathit{atom} \mid \mathit{molecule} \mid \asymp \mid \mathcal{Reaction}$$

$$r_{pert} \propto (p_{species}, \mathit{Reactant}', \mathit{trace})$$

where:

- symbol $\dagger$ denotes application of perturbation action $p_{species}$ to $\mathit{Target}'$
- symbol $\propto$ is the usual symbol denoting some kind of proportionality—in this case, it states that the dynamic rate expression of perturbation reaction should be a function of the perturbation action, of any property of the matching target, and of any property of the trace itself

In particular, given a trace of a certain species and its target, the perturbation action corresponding to that species is started, at the rate given by $r_{pert}$ dynamic rate expression.

This rate expression depends on the specific perturbation action the trace applies, in relation to its species and tacit messages being conveyed. Nevertheless, it is likely to also depend on the concentration of the trace and the target involved.

The autonomous computational process started by the perturbation action may affect any property of the target involved – e.g., increasing/decreasing reactants' concentration, or changing their location in the neighbourhood –, as well as any property of the $\mathcal{MoK}$ system—e.g., tuning rate expressions.

### 6.2.2 Evaluation

Simulation is widely recognised as a fundamental development and evaluation stage in the process of designing and deploying both MAS as well as biochemical processes [MAC+07, GVO06]. This is mostly due to the high number of system parameters needed, the huge number of local interactions between components, the influence of randomness and probability on system evolution.

A number of different simulation tools capable of modeling biochemical-like processes exist, either born in the biochemistry field (see [AAS06] for a survey) or in the Multi-Agent Based Simulation research area (survey in [NM09]). Among the many, ALCHEMIST [PMV13], PRISM [KNP11], and BioPEPA [CH09] at least, are worth to be mentioned.

The choice fell on the latter for its appealing features – briefly described in following paragraph – which perfectly suit the purpose of this section.

**The BioPEPA tool** BioPEPA [CH09] is a language for modeling and analysis of biochemical processes. It is based on PEPA [GH94], a process algebra originally aimed at performance analysis of software systems, extending it to deal with some features of biochemical networks, such as stoichiometry and different kinds of kinetic laws—including the law of mass action. The most appealing features of BioPEPA are:

- custom kinetic laws represented by means of *functional rates*
- definition of stoichiometry (how many molecules of a given kind participate) and role played by the species (reactant, product, enzyme, ...) in a given reaction
- theoretical roots in CTMC semantics—behind any BioPEPA specification lies a stochastic labelled transition system modeling a CTMC

In BioPEPA, rate expressions are defined as mathematical equations involving reactants' concentrations (denoted with the reactant name and dynamically computed at run-time) and supporting mathematical operators (e.g., `exp` and `log` functions) as well as built-in kinetic laws (e.g., the law of mass action, denoted with the keyword `fMA`) and time dependency (through the variable `time`, changing value dynamically according to the current simulation time step)[3].

**Technical notes on experiments** Each of the following experiments has been performed by using Gillespie's stochastic simulation algorithm in 100 independent replications. Each of the following plots has been directly generated from BioPEPA as a result of the correspondent experiment—hence, of the 100 Gillespie runs. In each chart, the $x$-axis plots the time steps of the simulation, whereas the $y$-axis the concentration level of the reactants expressed in units of molecules.

It should be noted that $\mathcal{M}o\mathcal{K}$ deposit and perturbation reactions have been left out of the following experiments. The motivation for ignoring the former is that it is identical to injection, except it involves different reactants. For the latter, besides being necessarily application-specific, it already takes part in the experiments of Subsection 6.3.2.

Differently from the experiments described in Section 3.12 in Part I of this thesis, here one reaction at a time is incrementally added to the scenario to be simulated—whereas in Section 3.1.2 each reaction has been studied in isolation. This allows evaluation of

[3]To learn more about BioPEPA syntax, please refer to [CH09].

*(i)* the interplay between the different reactions constituting the $\mathcal{MoK}$ model, and *(ii)* the whole pool of $\mathcal{MoK}$ reactions altogether – as the last step –, to understand if the global behaviour exhibited is an enabler for the kind of self-organising behaviour desired in $\mathcal{MoK}$.

Besides, the purpose of the experiments are different: in Chapter 3 the aim was at motivating the proposed approach to self-organising coordination by showcasing the expressiveness of custom kinetic rates for artificial chemical reactions, whereas here the focus is on the $\mathcal{M}$olecules $\mathcal{o}$f $\mathcal{K}$nowledge model, with the aim of engineering custom kinetic rates enabling the desired self-organising behaviour.

As a last note, the codebase tracking the BioPEPA specifications used in the examples is publicly available under LGPL license at `http://bitbucket.org/smariani/mok-biopepa`.

**Injection rate** Injection reaction has been formalised in Subsection 6.2.1 as follows:

$$\texttt{seed}(\mathit{src}, \mathit{Atoms}) \xrightarrow{r_{inj}} \texttt{seed}(\mathit{src}, \mathit{Atoms}) + \mathit{atom}^{i}_{c+c_i}, \quad \forall \mathit{atom}^{i}_{c_i} \in \mathit{Atoms}$$

$$r_{inj} = \frac{1}{1+\texttt{time}} * \texttt{diff}(\sharp \mathit{seed}, \sharp \mathit{atom})$$

Two contrasting needs have been addressed: on one hand, atoms should be *perpetually* injected into the $\mathcal{MoK}$ system, since there is no way to know a-priori *when* some information will be useful; on the other hand, one would likely avoid flooding the system without any control on how many atoms are in play. Thus, three options seem viable:

1. have injection rate decreasing as time passes
2. enforce some kind of saturation to stop injection
3. a combination of the two

Figure 6.1 below shows option (1) in blue, option (2) in yellow and option (3) in red. The green dashed line plots the law of mass action rate, whereas horizontal lines are the sources. Figure 6.2 shows the BioPEPA specification used.

Clearly, using the rate expression based on the law of mass action is out of question: its behaviour follows none of $\mathcal{MoK}$ injection reaction desiderata. Once discarded also option (1), whose trend is clearly too slow in reaching saturation, options (2) and (3) may seem almost identical. Actually they are not:

- option (2) is saturation-driven only, thus if at some point in time `atom_sports` will suddenly decrease in concentration – e.g., due to agents consuming them – they will go back to saturation-level as fast as possible, no matter how long their sources are within the system
- option (3) instead, makes the saturation process time-dependant. In particular, the longer `source_sports` are within the system, the slower saturation will be

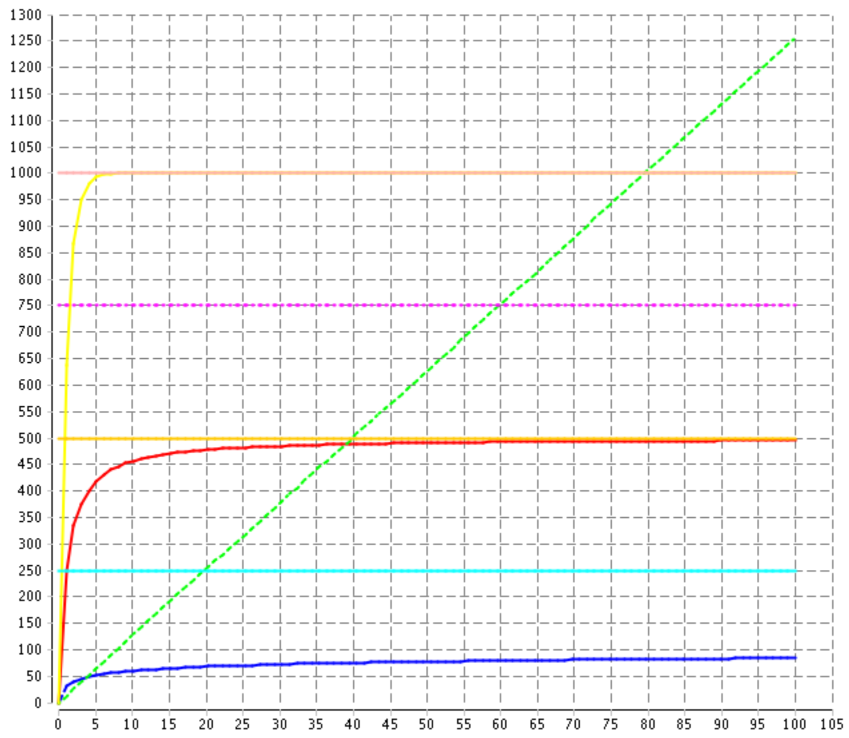


Figure 6.1: Comparison of kinetic rate expressions for atoms injection [Mar13b]. Horizontal lines represent correspondent seeds' concentration: purple dashed for option (1), pink for option (2), orange for option (3), light-blue for option (4).

```
injE = [source_economics/atom_economics * (1 / (1 + time))];  // option (1)
injS = [source_sports - atom_sports];                         // option (2)
injC = [(1 / (1 + time)) * (source_crime - atom_crime)];      // option (3)
injP = [fMA(0.05)];                                           // option (4)
```

Figure 6.2: The `fMA` keyword calls a built-in function to compute the law of mass action [Mar13b]. Its only parameter is the rate constant. The fMA implicitly consider reactants involved in the reaction exploiting its correspondent functional rate—for the full BioPEPA specification, please refer to [Mar13a].

Choosing among the two essentially depends on how fast information loses relevance compared to how frequently users are expected to interact with the system. Being knowledge-intensive STS intrinsically fast-paced ones, for $\mathcal{MoK}$ option (3) is preferred.

**Decay rate** Decay reaction has been formalised in Subsection 6.2.1 as follows:

$$\mathtt{Reactant}_c \xrightarrow{r_{dec}} \mathtt{Reactant}_{c-1}, \quad c \in \Re^+$$

$$r_{dec} = \begin{cases} \texttt{fMA}(a_R) * \texttt{log(1+time)}, & a_R \in \Re^+, \quad \mathtt{Reactant} \setminus \mathtt{trace} \\ \texttt{diff}(\sharp[\Uparrow \mathtt{trace}], \sharp \mathtt{trace}) * \texttt{log(1+time)} \end{cases}$$

Decay is an effective way to resemble the relationship between information relevance and time flow. In many application scenarios, in fact, information tend (on average) to lose (potential) relevance as time passes by: journalistic news do so, academic papers do so, posts in social network, fitness data, sensor streams, and so on. Furthermore, decay enforces a kind of *negative feedback* which, together with the *positive feedback* provided by $\mathcal{M}o\mathcal{K}$ enzymes, enables the *feedback loop* peculiar of self-organising systems.

Nevertheless, time dependency alone seem not enough for a meaningful decay behaviour: by using, e.g., a fixed rate one would end-up simply slowing down the saturation process provided by injection reaction. Hence, Figure 6.3 shows three different combinations of time dependency and concentration dependency for $\mathcal{M}o\mathcal{K}$ decay reaction—a fourth one (yellow line), based on the law of mass action, is given for comparison purpose:

1. linear time dependency + relative concentration dependency (blue dashed line)
2. logarithmic time dependency + relative concentration dependency (red line)
3. linear time dependency + built-in law of mass action (green dashed line)

Figure 6.4 shows the BioPEPA specification used[4]. The law of mass action is unsatisfactory, as well as option (1). Options (2) and (3) seem both viable instead.

The choice is mostly driven by how fast are the dynamics of the scenario in which $\mathcal{M}o\mathcal{K}$ has to be deployed, thus how fast information should lose relevance. Nevertheless, it should be noted that option (3) has an additional parameter w.r.t. option (2): the law of mass action rate constant. Even if the aforementioned parameter is made dynamic – e.g., the ratio between sources and atoms concentrations as done in options (1), (2) – the trend still would not match the desiderata for $\mathcal{M}o\mathcal{K}$ decay reaction—compare with yellow line of Figure 4 in [Mar13a]. Thus, the choice fell on option (2).

**Aggregation rate** Aggregation reaction has been formalised in Subsection 6.2.1 as follows:

$$\mathtt{Reactant}' + \mathtt{Reactant}'' \xrightarrow{r_{agg}} (\mathtt{Reactant}' \oplus \mathtt{Reactant}'') + (\mathtt{Reactant}' \ominus \mathtt{Reactant}'')$$

$$r_{agg} = \frac{\texttt{time}}{\sharp \mathtt{Reactant}^{lhs}}$$

The main responsibility of the aggregation reaction is to cluster together similar information atoms. While doing so, a fact should be considered: as time passes by, it is likely that more and more atoms will be roaming compartments, thus, it is desirable to aggregate more, so as to have different forms of more concise information[5]—for many reasons, such

[4]Actually, the Heaviside function has been also used to counter BioPEPA tolerance to negative rates, which are meaningless—see [CH09].

[5]Aggregation may assume different forms, such as filtering, merging, fusing, etc.

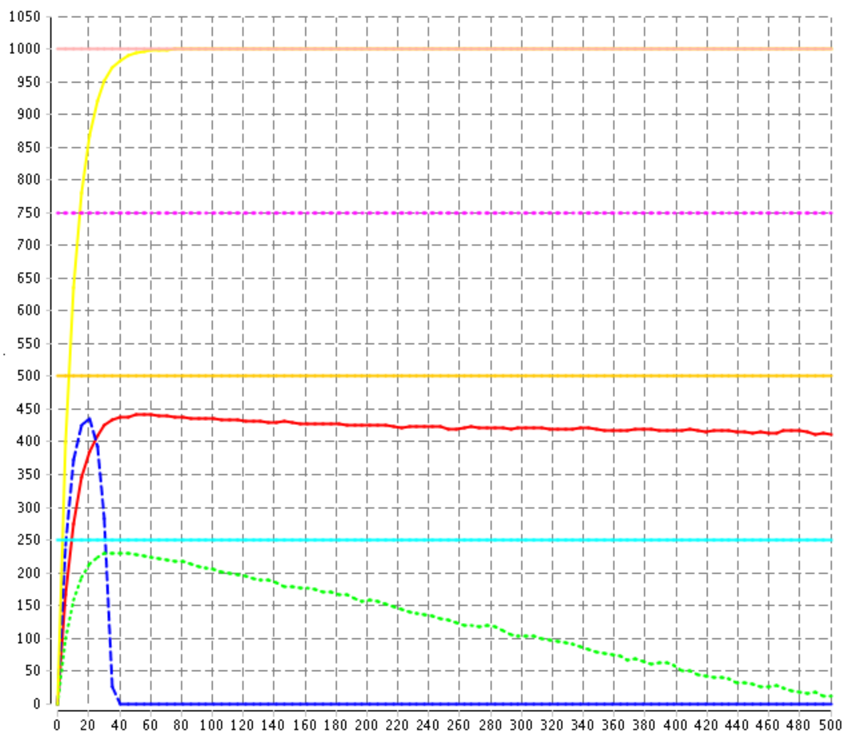


Figure 6.3: Comparison of kinetic rate expressions for atoms decay [Mar13b]. Again, horizontal lines represent correspondent seeds' concentration: purple dashed for option (1), orange for option (2), light-blue for option (3), pink for option (4).

```
decayE = [source_economics / atom_economics *     // option (1)
            time];
decayC = [source_crime / atom_crime *             // option (2)
            log(1+time)];
decayP = [fMA(0.05) * time];                      // option (3)
decayS = [fMA(0.05)];                             // option (4)
```

Figure 6.4: BioPEPA specification of rate expressions for MoK decay reaction [Mar13b].

as memory occupation.

It should be noted that molecules decay – although not implemented in the simulation to ease plots comprehension –, and that non-similar atoms are released back into the compartment during aggregation, thus it is unlikely for molecules to keep growing endlessly—unless they are really exploited by catalysts.

Based on the above discussion, three options seem viable—a fourth one, based on fMA solely, is considered for comparison purpose:

1. feature a direct proportionality w.r.t. time

2. combine direct proportionality w.r.t. time with inverse proportionality w.r.t. atoms,

so as to make the aggregation process slower when more and more atoms are present—a motivation could be that these atoms may be relevant on their own, since they are so many and not yet aggregated

3. combine direct proportionality w.r.t. time with inverse proportionality w.r.t. the generated molecule, so as to speed up the aggregation process when molecules lose relevance—a motivation could be that these molecules deserve a chance to be exploited by catalysts

Figure 6.5 shows the above described rate expressions on different time horizons (shorter on the left, longer on the right).

Option (1) aggregates atoms so fast they are almost instantly depleted, and have no chance to go nowhere near their saturation level (not depicted to ease readability of the plot, but equal to previous simulations)—orange dotted line for molecules, red line for atoms. Option (2) implements quite well the desiderata, being atoms kept slightly under their saturation level, and being molecules produced slightly faster as time passes by—pink dotted line for molecules, yellow line for atoms. Option (3) seems almost identical to option (2) on a short time period, but in the long run (picture on the right) it exhibits a more linear trend w.r.t. option (2)—light-blue line for molecules, dotted green line for atoms. Option (4) is clearly out of question, in a way very similar to option (1)—purple dotted line for molecules, blue dotted line for atoms.

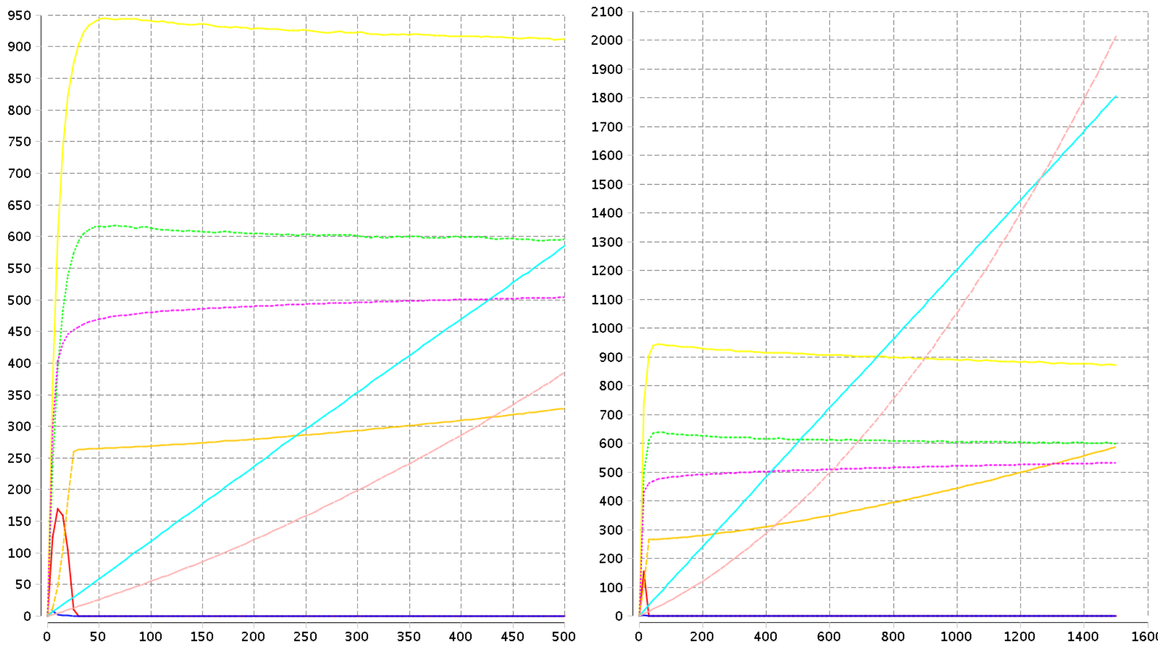


Figure 6.5: Comparison of kinetic rate expressions for molecules aggregation.

```
aggC = [fMA(0.5) * time];                 // option (1)
aggS = [fMA(0.5) * time/atom_sports];     // option (2)
aggP = [fMA(0.5) * time/molecule_politics]; // option(3)
aggE = [fMA(0.5)];                        // option (4)
```

Figure 6.6: BioPEPA specification of rate expressions for $\mathcal{MoK}$ aggregation reaction.

Figure 6.6 above shows the BioPEPA specification used[6]. Once again the choice among option (2) and (3) is mostly driven by the application-specific needs $\mathcal{MoK}$ should support. According to the deployment scenarios envisioned for $\mathcal{MoK}$, option (2) is preferred.

**Reinforcement rate** Reinforcement reaction has been formalised in Subsection 6.2.1 as follows:

$$\texttt{enzyme(}\mathit{species}\texttt{,}\mathit{s}_{species}\texttt{,}\mathit{Reactant}\texttt{,}\mathit{Context}\texttt{)} + \mathit{Reactant}'_{c} \xrightarrow{r_{reinf}}$$
$$\texttt{enzyme(}\mathit{species}\texttt{,}\mathit{s}_{species}\texttt{,}\mathit{Reactant}\texttt{,}\mathit{Context}\texttt{)} + \mathit{Reactant}'_{c+s}, \quad c, s \in \Re^{+}$$

$$r_{reinf} = \texttt{diff(}\sharp[\Uparrow \mathit{Reactant}]\texttt{,}\sharp \mathit{Reactant}\texttt{)}$$

To properly engineer $\mathcal{MoK}$ reinforcement reaction rate, it should be kept in mind what enzymes are meant for, that is, *(i)* representing a *situated interest* manifested by an agent w.r.t. a piece of knowledge – an atom or a molecule – *(ii)* be exploited to reinforce knowledge relevance within the system. The word "situated" means that reinforcement should take into account the *situatedness* of agents (inter-)actions along a number of dimensions: time, space, type—a "harvest" action, a "mark" action, etc. For these reasons, $\mathcal{MoK}$ reinforcement reaction rate should:

- be prompt, that is, rapidly increase molecules concentration—despite decay
- be limited both in time and space, to resemble relevance relationship with situatedness of (inter-)actions
- depend on the (inter-)action type—e.g., a "mark" action could inject more enzymes and/or reinforce atoms with greater stoichiometry w.r.t. a "harvest" action

Figure 6.7 clearly shows that the desiderata are fulfilled only by a reinforcement reaction having a functional dependency on the ratio between the reinforced molecule's concentration and its source own—option (1) in Figure 6.8 below. Once again, sticking with the law of mass action alone is out of question: option (2) – dashed blue line –, even if adopting a dynamic rate constant, exhibits an exceedingly high and fast peak, option (3) – red line –, using a fixed rate constant (as in the law of mass action typically is), almost completely ignores the feedback—enzymes are too slowly consumed (orange line, plotting enzymes concentration).

[6]Again, the Heaviside step function has been used to avoid negative rates.

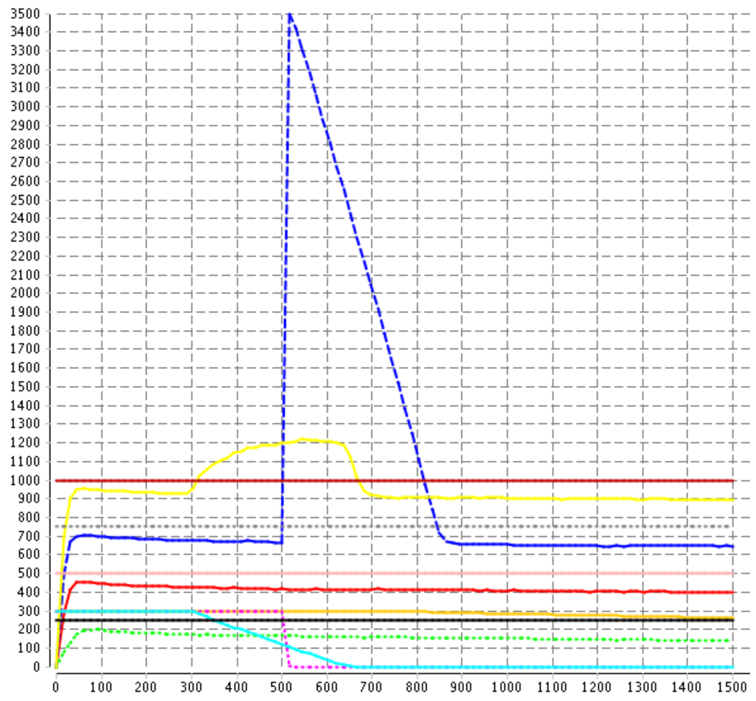


Figure 6.7: Comparison of kinetic rate expressions for atoms reinforcement [Mar13b]. Lines worth to be considered are: the yellow one, plotting option (1), the dashed blue one, plotting option (2), the red one, plotting option (3).

```
feedS = [(source_sports / atom_sports)];        // option (1)
feedE = [fMA(source_economics / atom_economics)]; // option (2)
feedC = [fMA(0.05)];                     // option (3)
```

Figure 6.8: BioPEPA specification of rate expressions for $\mathcal{M}o\mathcal{K}$ reinforcement reaction [Mar13b].

Furthermore, Figure 6.9 shows how *concentration* and *stoichiometry* can influence $\mathcal{M}o\mathcal{K}$ reinforcement reaction behaviour, effectively modeling situatedness—in particular, w.r.t. the type of (inter-)actions. In fact, in Figure 6.9 *(i)* the initial concentration of red enzymes (red line) is doubled w.r.t. yellow enzymes (yellow line) in Figure 6.7: as a result, the duration of the feedback is doubled as well; *(ii)* the stoichiometry of red atoms (red line) in reinforcement reaction is doubled w.r.t. yellow atoms (yellow line) in Figure 6.7: as a result, the intensity of the feedback is more than doubled.

Also, Figure 6.9 shows that *(i)* the opposite holds, too, that is, halving the initial concentration halves the duration of the feedback (yellow and blue lines); *(ii)* no interference happens between concentration and stoichiometry parameters.

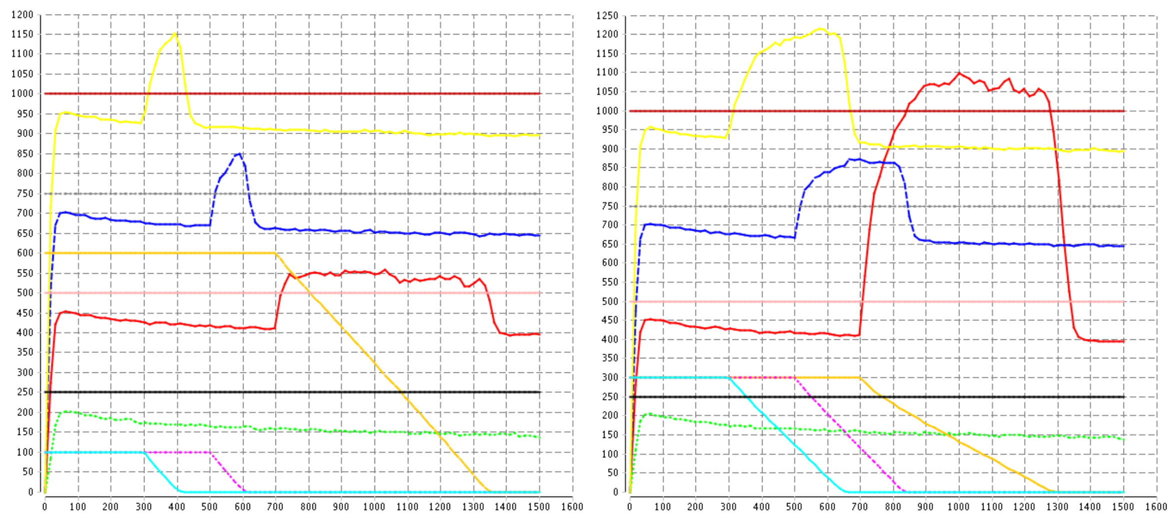


Figure 6.9: Enzymes concentration increment effect on reinforcement (on the left), and atoms stoichiometry increment effect on reinforcement (on the right) [Mar13b].

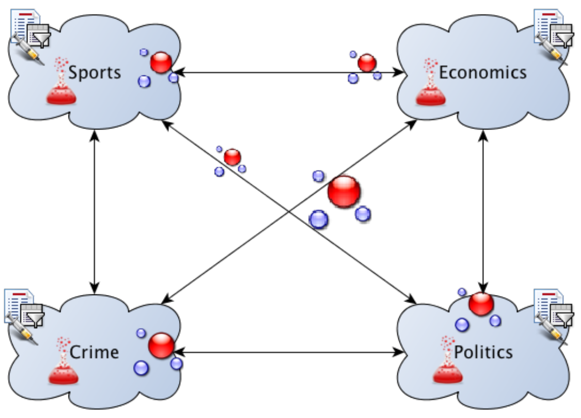


Figure 6.10: $\mathcal{M}o\mathcal{K}$ topology to experiment with diffusion reaction [Mar13b].

**Diffusion rate** Diffusion reaction has been formalised in Subsection 6.2.1 as follows:

$$[\![ \mathtt{Reactants}' \cup \mathtt{Reactant} ]\!]_i \asymp [\![ \mathtt{Reactants}'' ]\!]_j \xrightarrow{r_{diff}} [\![ \mathtt{Reactants}' ]\!]_i \asymp [\![ \mathtt{Reactants}'' \cup \mathtt{Reactant} ]\!]_j, \quad i \neq j \in \mathbb{N}$$

$$r_{diff} = \mathscr{d} * \mathtt{diff}([\![ \sharp \mathtt{Reactant} ]\!]_i, [\![ \sharp \mathtt{Reactant} ]\!]_j), \quad \mathscr{d} \in (0,1] \in \Re$$

The topology depicted in Figure 6.10 below is taken as a reference. Namely, four $\mathcal{M}o\mathcal{K}$ compartments are imagined to be connected one to each other, allowing any molecule to move anywhere.

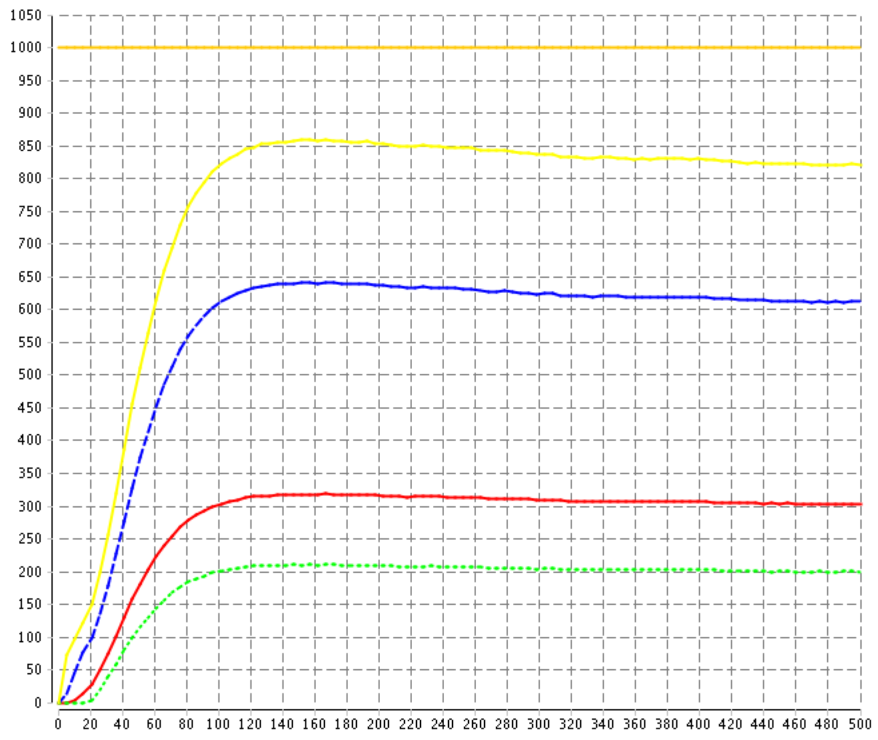


Figure 6.11: *MoK* diffusion reaction trend [Mar13b]. Yellow line plots concentration level of atoms in their origin compartment (the orange horizontal line represents their source).

```
// diffusion weight
DW = 0.75;
// diffusion functional rates (a@x => a@y)
diffSE = [DW * as@sports - as@economics]; // blue line
diffSC = [DW/2 * as@sports - as@crime]; // red line
diffSP = [DW/3 * as@sports - as@politics]; // green line
```

Figure 6.12: BioPEPA specification for diffusion rates [Mar13b]. Notation $r@c$ refers to the concentration of reactant $r$ in compartment $c$. Previous listings did not follow this notation because there was only a single compartment.

The main desiderata for *MoK* diffusion reaction are similar to those of *MoK* injection: on one hand, one would like to perpetually spread information around, because agents working in other compartments may be interested in it; on the other hand, it is also desirable to keep some degree of control about how much information is moved around.

Such a kind of degree of control can be achieved by reusing the concept of saturation, as shown by Figure 6.11: in particular, it seems reasonable to allow only a fraction of molecules to leave their origin compartment—see Figure 6.12. In practice, one can arbitrarily decrease/increase the saturation-level of the origin compartment in the destination compartment. Furthermore, they are functionally related.

As a side note, diffusion reaction featuring the law of mass action is not depicted. The motivation is that it exhibits an unexpected malfunctioning affecting also other reactions. More on this interference problem in next section.

**On the problem of interference between reactions** All the experiments shown have been conducted incrementally, that is, each *𝓜o𝒦* reaction has been added to the BioPEPA specification one at a time. As reported in [Mar13a], when adding diffusion to other *𝓜o𝒦* behaviours, BioPEPA plots highlighted some *interference* between reactions. E.g., Figure 6.13 below depicts what happened when reinforcement has been added to injection, decay and diffusion. A number of unexpected behaviours can be seen:

- first of all, desiderata for *𝓜o𝒦* reinforcement reaction are not met (dashed blue line). In particular, it seems atoms cannot go beyond their original compartment concentration level (yellow line)
- second, enzymes are not fully depleted (orange line)
- last but not least, other atoms are affected by a successful application of *𝓜o𝒦* reinforcement reaction (yellow, red and green lines): in particular, in the time interval during which enzymes are consumed, *all* other trends experiment some fluctuations

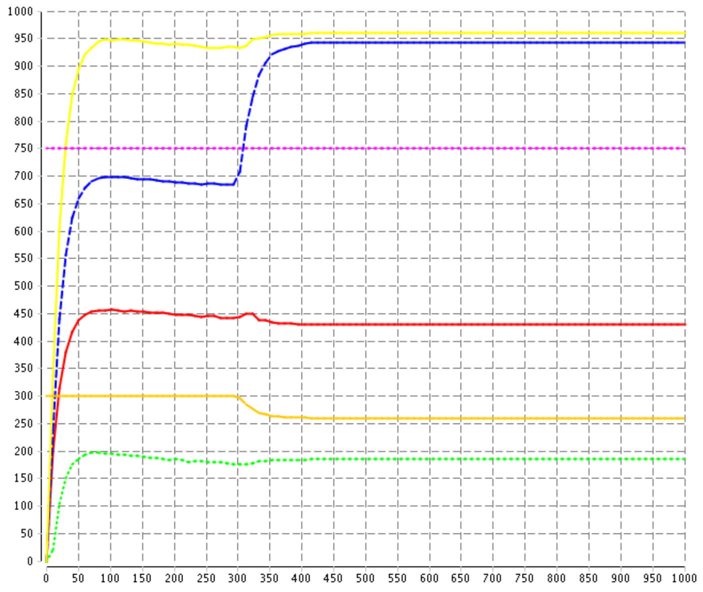


Figure 6.13: *𝓜o𝒦* reinforcement reaction addition to injection, decay and diffusion [Mar13b]. Not only enzymes are not fully depleted, but also undesirable and unexpected interferences with other reactions are clearly highlighted.

```
FF = 2;                      // feed factor > 1
feedEC = [se@economics / (ae@crime * FF)];  // option (1)-revised
```

Figure 6.14: Adjusted BioPEPA specification of rate expressions for $\mathcal{M}o\mathcal{K}$ reinforcement reaction used together with $\mathcal{M}o\mathcal{K}$ diffusion reaction [Mar13b].

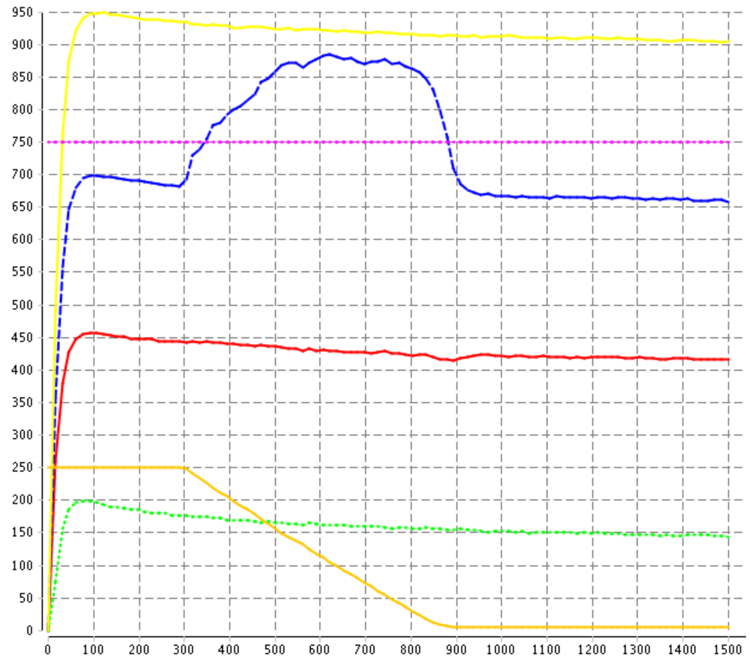


Figure 6.15: Adjusted $\mathcal{M}o\mathcal{K}$ reinforcement reaction: enzymes are now completely depleted and other reactions no longer affected [Mar13b].

The reason at the root of all these issues is still unknown: being chemical-like reactions scheduling essentially based on race conditions between the correspondent functional rates – evaluated at a given point in time –, understanding what exactly happens within the system at a given time step is not trivial at all—or even impossible, depending on the debugging services the simulation tool adopted provides.

Nevertheless, the satisfactory BioPEPA specification shown in Figure 6.14 above has been found. In particular, $\mathcal{M}o\mathcal{K}$ reinforcement reaction rate has been added a feed factor parameter, used to weight the influence of the atoms to be reinforced w.r.t. the concentration of the corresponding source in the compartment the latter belongs to.

Figure 6.15 shows that desiderata are now met successfully, including the functional dependencies on enzymes concentration and atoms stoichiometry shown in Figure 6.9.

## 6.3 Focus on Interactions

In this section how agents' interactions are handled in MoK is thoroughly described. After briefly recalling which actions taken as a reference in the modelling of STS, and what tacit messages they may convey – from Chapter 5 –, a description of the relationships between actions, enzymes' *species*, traces' *messages*, and *perturbation actions* is provided (Subsection 6.3.1).

Then follows evaluation of the proposed approach by simulating a citizen journalism scenario within which co-workers affect the *spatial displacement* of information items – actually promoting *clustering* of semantically-related information *without* the need for any semantic technology – simply by carrying out their usual activities of, e.g., information harvesting (Subsection 6.3.2).

This way, it is possible to clarify how each single action can be interpreted by the MoK model for the benefit of the coordination processes therein, and which kind of anticipative action MoK may plan to take advantage of tacit messages brought by users' actions.

### 6.3.1 From Users' Actions to MoK Perturbations

Interaction, thus *coordination* of interactions, in MoK is interpreted from the socio-cognitive perspective of BIC [CPT10], where communication does not occur through any specialised signal, but through the *practical behaviour* observed by the recipient(s). Then, actions themselves – along with their (side) effects – become the message, possibly intentionally (implicitly) sent in order to obtain a desired reaction—either by the computational environment where the action takes place, or by other agents who can observe the action and/ its effects.

This perspective leads to the argument that self-organising coordination can be based on the *observation* and *interpretation* of actions as wholes, that is, both the practical behaviour and its (side) effects—rather than solely of their effects, or traces, on the environment, as happens, e.g., for *stigmergic coordination* [Gra59].

Tacit messages have been already described in Chapter 5, along with the notion of BIC. Here follows a brief recap:

1. *informing about the presence.* "Agent $A$ is here". Since an action, an activity, a practical behaviour (and its traces), is observable in the computational environment, any agent therein – as well as the environment itself – becomes aware of $A$ existence—and, possibly, contextual information such as $A$ location

2. *informing about the intention.* "Agent $A$ plans to do action $\beta$". If the agents' workflow determines that action $\beta$ follows action $\alpha$, peers (as well as the environment) observing $A$ doing $\alpha$ may assume $A$ next intention to be "do $\beta$"

3. *informing about the ability.* "$A$ is able to do $\phi_i,\ i \in \mathbb{N}$". Assuming actions $\phi_i,\ i \in \mathbb{N}$

have similar pre-conditions, agents (and the environment) observing $A$ doing $\phi_i$ may infer that $A$ is also able to do $\phi_{j \neq i},\ j \in \mathbb{N}$

4. *informing about the opportunity for action.* "$p_i,\ i \in \mathbb{N}$ is the set of pre-conditions for doing $\alpha$". Agents observing $A$ doing $\alpha$ may infer that $p_i,\ i \in \mathbb{N}$ hold, thus, they may take the opportunity to do $\alpha$ as soon as possible

5. *informing about the accomplishment of an action.* "$A$ achieved $S$". If $S$ is the state of affairs reachable as a consequence of doing action $\alpha$, agents observing $A$ doing $\alpha$ may infer that $A$ is now in state $S$

6. *informing about the goal.* "$A$ has goal $g$". By observing $A$ doing action $\alpha$, peers of $A$ may infer $A$'s goal to be $g$, e.g., because action $\alpha$ is part of a workflow aimed at achieving $g$—likewise for the environment

7. *informing about the result.* "Result $R$ is available". If peer agents know that action $\alpha$ leads to result $R$, whenever agent $A$ does $\alpha$ they can expect result $R$ to be soon available—in case action $\alpha$ completes successfully

Within MoK, the factorisation of common actions described in Chapter 5, along with the discussion about the possible *tacit messages* and *perturbation actions* they may convey, is exploited to define precise associations between MoK enzymes' species, traces' messages, and perturbation actions. This enables to evaluate MoK behaviour in response to users interactions in a more rigorous way, e.g., through simulations and demo deployments.

Accordingly, here follows the list of MoK actions outlined in Chapter 5, now completed by a description of *(i)* the enzyme they (transparently) release within MoK, *(ii)*, the traces the enzyme may (autonomously) deposit within MoK, and *(iii)* the perturbation actions these traces may bring along.

**share** the share action is reified by enzymes with *`Species`* = `share`. Then, depending on the *`Context`* within which the action took place, `share` enzymes may deposit the following different sorts of traces:

- `presence` traces, that is, traces having *`Msg`* = `presence`, indicating to MoK presence, location, and any other available property of the acting agent
- `ability` traces, indicating to MoK the acting agent's capabilities, both in terms of what she can do, and of what she can perceive about others—e.g., what actions are available to her, on which targets, which of her traces other catalysts are perceiving, etc.
- `accomplishment` traces, indicating to MoK which state of affairs the acting agent may have reached upon successful completion of the action—e.g., which information has been manipulated and how

which, according to *(i)* their species, *(ii)* the context within which the original enzyme was released, and *(iii)* their $\mathtt{Target}$, may lead to the following perturbation actions:

- `approach` perturbations, that is, perturbation actions having $P_{species}$ = `approach`, stemming from the `presence` tacit message (and indirectly on the `share` action), aimed at bringing closer to each other the compartments of those agents interacting more often with each other information items. "Closer" here means "logically closer", that is, interactions between compartments, and the users working with them, are facilitated and promoted—e.g., diffusion rate is increased. Nevertheless, it is possible to also imagine a more physical interpretation, according to which, e.g., the involved compartments are physically migrated in geographically closer hosts, so as to, e.g., lower the communication latency
- `attract` perturbations, stemming from the `ability` tacit message, aimed at bringing to the compartment where the action took place – thus, not necessarily the one where the trace is at a given time, being traces able to diffuse whereas enzymes cannot – information similar to the one target in the `share` action—according to any notion of similarity implemented through function $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$. The opposite perturbation, `repulse`, may also be undertaken for those information items recognised as being dissimilar to the shared information.
- `boost`/`wane` perturbations, stemming from the `accomplishment` tacit message, aimed at increasing/decreasing the rate of $\mathcal{M}o\mathcal{K}$ reactions so as to facilitate the acting agent's workflow—e.g., $\mathcal{M}o\mathcal{K}$ reinforcement and decay reactions' rates may be respectively increased and decreased for information items similar to those target of the original `share` action

**mark** the mark action is reified by enzymes with $\mathtt{Species}$ = `mark`. Then, depending on the $\mathtt{Context}$ within which the action took place, `mark` enzymes may deposit the following different sorts of traces:

- `presence` traces, that is, traces having $\mathtt{Msg}$ = `presence`
- `opportunity` traces, indicating to $\mathcal{M}o\mathcal{K}$ that a new opportunity for action is now available, enabled by the action just completed successfully—e.g., a collaboration between co-workers, exploitation of additional information, etc.

which, according to *(i)* their species, *(ii)* the context within which the original enzyme was released, and *(iii)* their $\mathtt{Target}$, may lead to the following perturbation actions:

- `approach` perturbations, that is, perturbation actions having $P_{species}$ = `approach`, stemming from the `presence` tacit message (and indirectly on the `mark` action)

- `strengthen` perturbations, stemming from the `opportunity` tacit message, aimed at increasing relevance (concentration) of information semantically related to the one target of the original `mark` action. If marking information is allowed to bear a negative meaning – e.g., disliking information as opposed to liking – the opposite perturbation action, that is, `weaken`, is to be considered

Besides, it should be noted that also `attract` and `boost` (or `wane`) perturbation actions may be considered, stemming from the `opportunity` tacit message, based on the ***Context*** of the original enzyme, and the ***Target*** of the trace

**annotate** the annotate action is reified by enzymes with ***Species*** = `annotate`. Then, depending on the ***Context*** within which the action took place, `annotate` enzymes may deposit the following different sorts of traces:

- `accomplishment` traces, that is, traces having ***Msg*** = `accomplishment`
- `goal` traces, indicating to $\mathcal{M}o\mathcal{K}$ which could be the motivations behind the acting agent's behaviour, in terms of the state of affairs she aims to achieve in the medium to long term

which, according to *(i)* their species, *(ii)* the context within which the original enzyme was released, and *(iii)* their ***Target***, may lead to the following perturbation actions:

- `boost/wane` perturbations, that is, perturbation actions having $P_{species}$ = `boost` / `wane`, stemming from the `accomplishment` tacit message (and indirectly on the `annotate` action)
- `attract/repulse` perturbations, stemming from the `goal` tacit message

Besides this, also `strengthen` (or `weaken`) perturbation actions may be considered, stemming from both tacit messages, based on the ***Context*** of the original enzyme, and the ***Target*** of the trace

**connect** the connect action is reified by enzymes with ***Species*** = `connect`. Then, depending on the ***Context*** within which the action took place, `connect` enzymes may deposit the following different sorts of traces:

- `intention` traces, that is, traces having ***Msg*** = `intention`, indicating to $\mathcal{M}o\mathcal{K}$ the immediate cause of agent's action, in terms of the state of affairs she aims to achieve in the short-term—right after the action completes
- `opportunity` traces

which, according to *(i)* their species, *(ii)* the context within which the original enzyme was released, and *(iii)* their ***Target***, may lead to the following perturbation actions:

- `approach` perturbations, that is, perturbation actions having $\mathtt{P}_{species}$ = `approach`, stemming from the `intention` tacit message
- `strengthen` perturbations, stemming from the `opportunity` tacit message

Besides, also `attract` and `boost` (or `wane`) perturbation actions may be considered, respectively stemming from the `intention` and `opportunity` tacit messages, based on the ***Context*** of the original enzyme, and the ***Target*** of the trace

**harvest** the harvest action is reified by enzymes with ***Species*** = `harvest`. Then, depending on the ***Context*** within which the action took place, `harvest` enzymes may deposit the following different sorts of traces:

- `presence` traces, that is, traces having ***Msg*** = `presence`
- `intention` traces
- `opportunity` traces

which, according to *(i)* their species, *(ii)* the context within which the original enzyme was released, and *(iii)* their ***Target***, may lead to the following perturbation actions:

- `approach` perturbations, that is, perturbation actions having $\mathtt{P}_{species}$ = `approach`, stemming from the `intention` tacit message
- `attract` perturbations, stemming from the `intention` tacit message
- `strengthen` perturbations, stemming from the `opportunity` tacit message

Besides this, it should be noted that the harvest action is the utmost epistemic action, potentially conveying every sort of tacit message, thus enabling any sort of inference leading to any sort of perturbation actions. Therefore, based on the ***Context*** of the original enzyme, and the ***Target*** of the trace, any other perturbation action may be considered.

In next section, some of the above described associations, thus the whole idea of BIC driven coordination in the very end, is evaluated against a citizen journalism scenario, where simulations of user agents' interactions within a MoK-coordinated infrastructure for news management are described, to showcase how enzymes, traces, and perturbation actions may affect MoK coordination capabilities, therefore the system behaviour.

### 6.3.2 Early Evaluation: Citizen Journalism

In the following a simulation of a citizen journalism scenario is described, where users share a MoK-coordinated IT platform for retrieving, assembling, and publishing news stories.

Users have personal devices (smartphones, tablets, laptops) running the MoK middleware, which they use to *(i) search* throughout the IT platform looking for relevant information, *(ii)* modify, *annotate*, comment, etc. information so as to shape their own news story, and finally *(iii) release* their story to the public, for both reading and re-use.

For a number of reasons – among which limiting bandwidth consumption, boosting security, reducing communication latency, etc. – the extent to which a search action may extend is limited by a (possibly logical, not physical) *neighbourhood* of compartments, that is, those compartments connected by a membrane to the compartment where the search action takes place—e.g., neighbourhood can be based on 1-hop network reachability.

While performing search actions, and in general while undertaking their usual business-related activities, users release (*transparently*, possibly *unintentionally*) *enzymes* within the compartment underlying the working space they are operating on, which in turns deposit *traces* which start wandering across the network of compartments constituting the news management IT platform—recall that traces are subject to diffusion whereas enzymes are not.

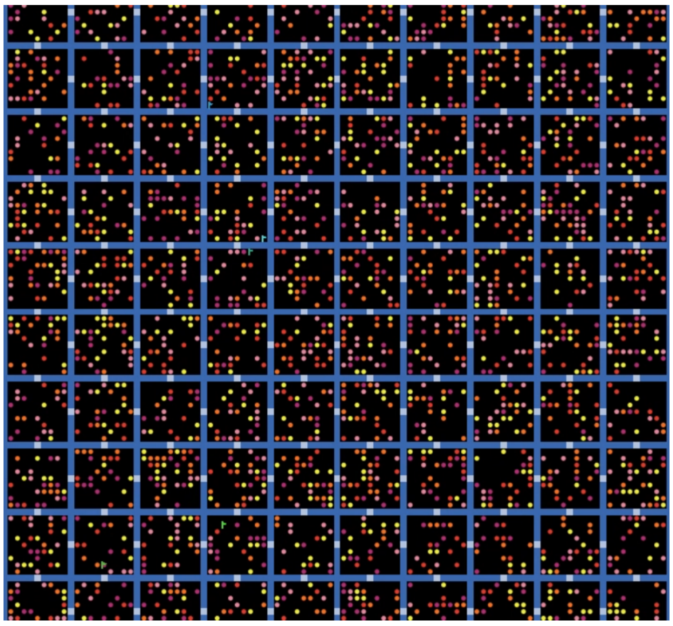
Then, the MoK middleware exploits enzymes and traces to schedule *perturbation actions* aimed at enacting some form of *anticipatory coordination* [PCF07]. In particular, within the simulated scenario users perform `harvest`, `mark`, and `share` actions, which deposit, respectively, `intention`, `opportunity`, and `presence` traces, ultimately causing `attract`, `strengthen`, and `boost` perturbation actions—according to MoK terminology.

Thus, what happens in practice is that the MoK middleware exploit users' (local) interactions to improve the (global) spatial organisation of information: whenever users implicitly manifest interest in a piece of information – through harvesting, marking, and sharing actions – the MoK middleware interpret their *intention* to exploit information, and the *opportunity* for others to exploit it as well, *attracting* similar information toward the compartment where the action took place, *increasing* concentration of the information target of the action, and *boosting* reinforcement and diffusion reactions accordingly.

Figure 6.16, Figure 6.17, and Figure 6.18 showcase how the emergent *collective intelligence* phenomena enabling anticipatory coordination is effectively supported by suitable BIC inspired abstractions and mechanisms—and, ultimately, by the Molecules of Knowledge model as a whole.

The coordination infrastructure, in fact, does not know in advance the effectiveness of its coordination activities in supporting users' workflows: it only tries to react to users' activities at its best, according to its own interpretation of users' actions in epistemic terms—e.g., through tacit messages. This is exactly what anticipatory coordination is: the infrastructure tries to foresee the user coordination needs even before users do, with the aim of satisfying them at best [MO15a].

The picture at the top, in Figure 6.16, shows the initial configuration: information molecules (coloured dots) are randomly scattered throughout a grid of networked compartments (black squares)—light-blue little squares represent membranes between compartments, allowing diffusion. The picture at the bottom, in same figure, highlights two



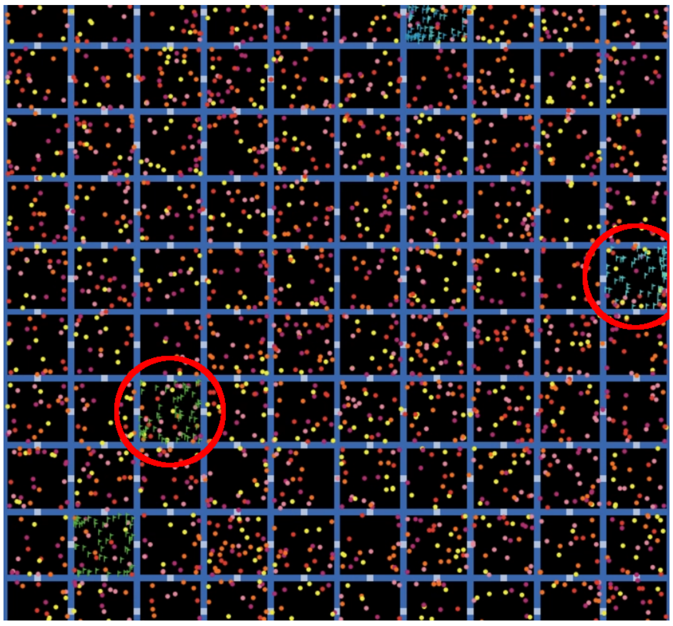
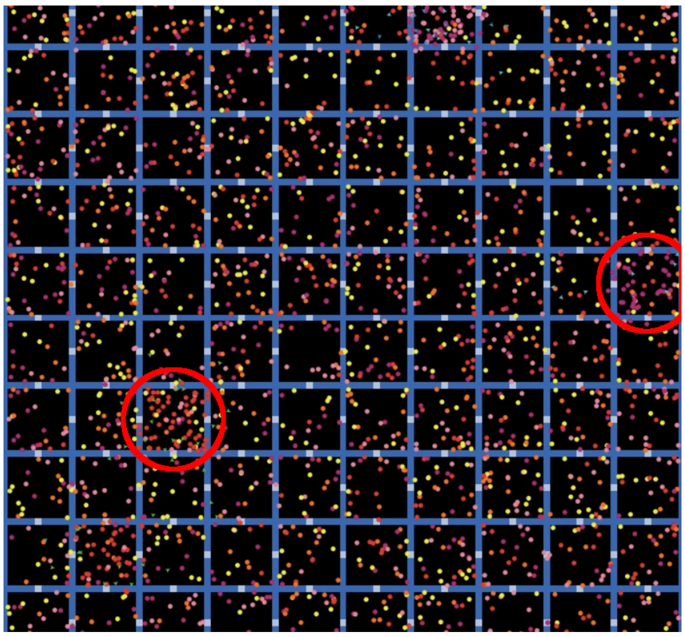


Figure 6.16: *Self-organising, adaptive anticipatory coordination* [MO15a]. The simulation starts from a situation in which atoms and molecules of information belonging to different topics (represented as differently coloured dots) is randomly scattered across catalysts' compartments (represented as black squares, delimited by blue lines, and connected by membranes represented as light-blue patches)—top figure. From time to time, users (catalysts) carry out `harvest`, `mark`, and `share` actions, releasing homonym enzymes (represented as coloured flags) within the workspace they are working on—bottom figure.



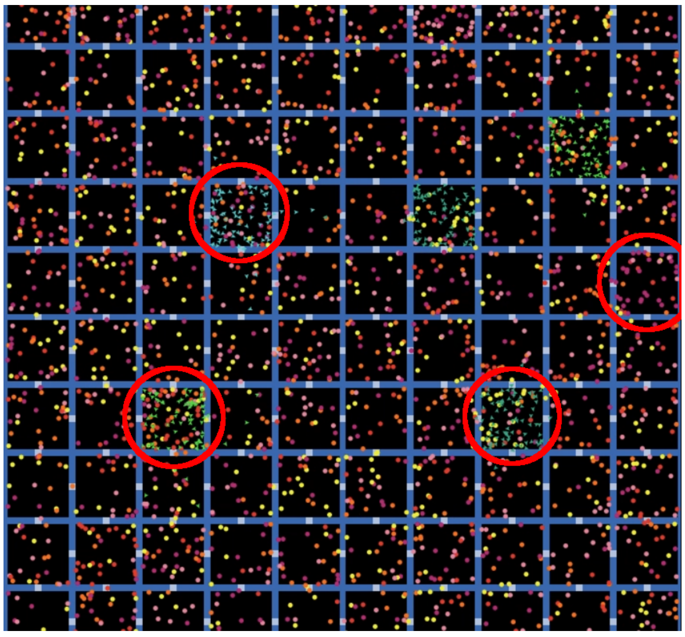


Figure 6.17: *Self-organising, adaptive anticipatory coordination* [MO15a]. As soon as enzymes are released, 𝓜o𝓚 reinforcement reaction increases concentration of the information atoms and molecules target of the original action—top figure. Then, `intention`, `opportunity`, and `presence` traces are deposited. Traces are free to wander between neighbouring compartments, crossing membranes, so as to apply their perturbation action potentially to any compartment in the network—bottom figure.

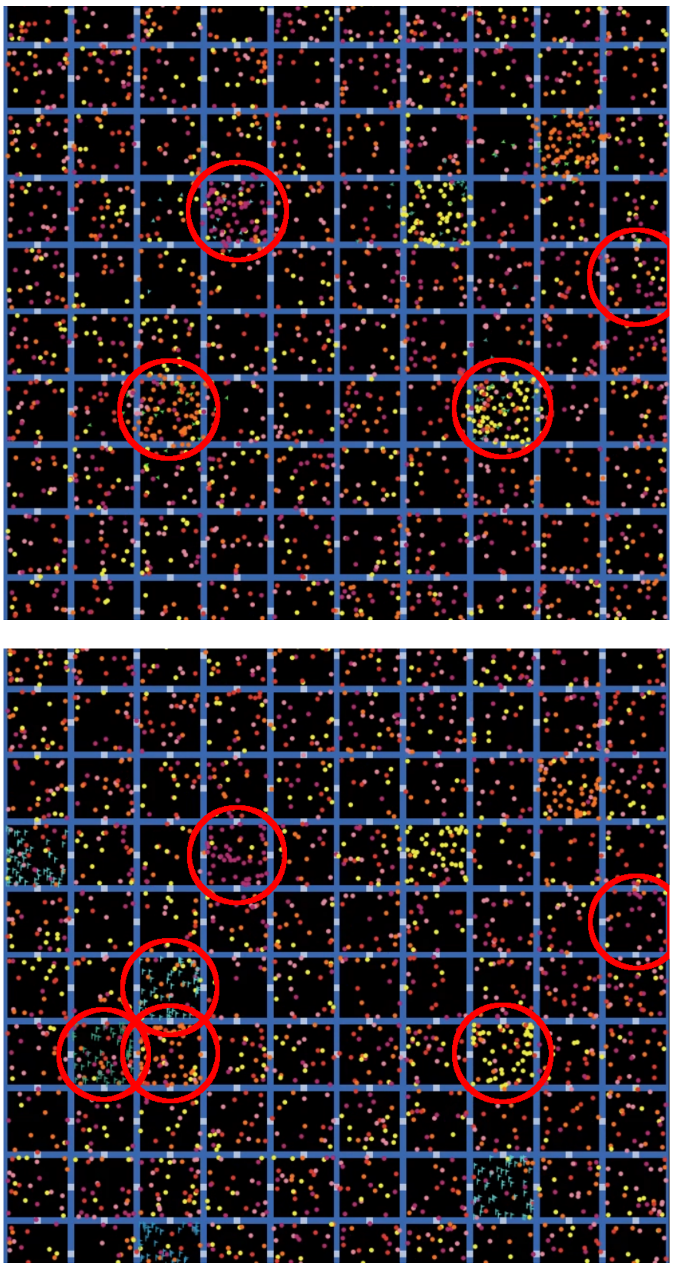


Figure 6.18: *Self-organising, adaptive anticipatory coordination* [MO15a]. In fact, clusters begin to appear – as an *emergent* phenomenon – thanks to the interplay between `attract`, `strengthen`, and `boost` perturbations, both where the original action took place as well as in distant compartments, reached by the traces, and storing compatible information—top figure. Whenever new actions are performed by catalysts, the $\mathcal{MoK}$ infrastructure *adaptively* re-organises the spatial configuration of information so as to better tackle the novel, now arising coordination needs—bottom figure.

compartments in which enzymes (coloured flags) have just been released, thus traces begin to spawn and diffuse (coloured arrows): green enzymes in the bottom-left one, cyan enzymes in the top-right one.

Colours represent semantic differences for different matches: red molecules match green enzymes/traces, orange molecules match lime enzymes/traces, yellow molecules match turquoise enzymes/traces, magenta molecules match cyan enzymes/traces, pink molecules match sky blue enzymes/traces.

The picture at the top, in Figure 6.17, showcases that the expected clusters appear: red molecules (brought by green traces' perturbation action) have the highest concentration in the bottom-left (highlighted) compartment, likewise magenta molecules (brought by cyan traces) in the top-right one. The pictures from the one at the bottom of Figure 6.17 to those in Figure 6.18 showcase that clusters are transient, hence 𝓜𝓸𝓚 coordinative behaviour *adaptive*: they last as long as users' action effects (enzymes and traces) last.

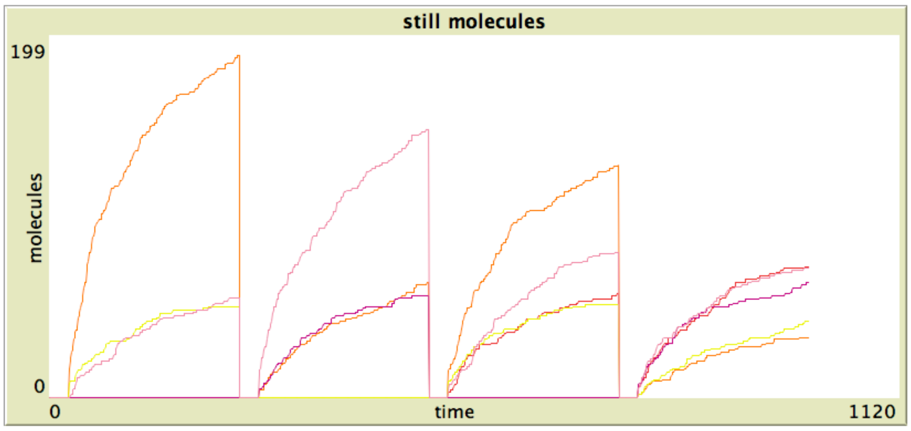
In fact, besides new clusters appearing (magenta molecules, top-left and yellow molecules, bottom-right), the previous ones either disappear (magenta cluster, top-right) or are replaced (orange cluster, bottom-left). This *adaptiveness* feature is confirmed by Figure 6.19, plotting the oscillatory trend of clustered (still) molecules and traces.

Also, the series of pictures just described highlights other desirable features of 𝓜𝓸𝓚, stemming from both its biochemical inspiration and BIC, respectively: *locality* and *situatedness* (of both computations and interactions). In fact, as neighbouring compartments can influence each other through diffusion, they can also act independently by, e.g., aggregating different molecules.

**Technical details** Technical details of the experiment are as follows[7]: 100 𝓜𝓸𝓚 compartments are networked in a grid (4 neighbours per compartment, except border)—see Figure 6.16; 2500 molecules, split in 5 non-overlapping semantic categories (representing matching with different enzymes), are uniformly sampled then randomly scattered in the grid—statistically, 500 molecules per category; 250 enzymes, split in the same categories, are generated in 5 random compartments; enzymes' categories are uniformly sampled in batches consisting of 50 enzymes each, so that generated enzymes of a given category are always multiple of 50; enzymes are generated *periodically* (every 250 time steps) and subject to decay; 2 traces per enzyme are generated, coherently with enzymes' category and according to the same time interval; traces, too, are subject to decay, although at a lower rate w.r.t. enzymes—due to their different purpose: represent *long-term effects* of actions for the former, reify *situated* actions for the latter.

The simulations proceed as follows: molecules randomly diffuse among neighbouring compartments; enzymes reify, e.g., a `harvest` action which successfully collects a set of

[7]Simulation tool used is NetLogo 5.0.5, available from `http://ccl.northwestern.edu/netlogo/`. Videos of the simulations are available on YouTube (`https://youtu.be/8ibkXdukTfk`). Source code of the simulations is to be released as a NetLogo model, available from `http://ccl.northwestern.edu/netlogo/models/community/`.



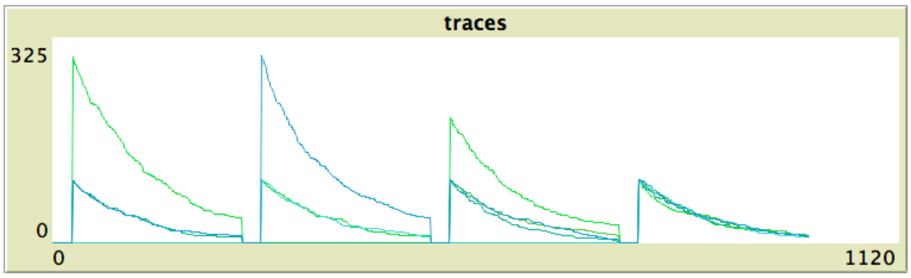


Figure 6.19: At the top, concentration of still molecules over time; at the bottom, concentration of traces over time [MO15a]. Still molecules represent molecules currently in the right compartment—the one storing matching enzymes. The oscillatory trend is due to periodic injection of enzymes (thus traces) which clears the still state of molecules. The different colours correspond to the different molecules. Traces move molecules to the right compartment through perturbations. The oscillatory trend is due to decay of traces over time. The different colours correspond to the different traces.

molecules from the local compartment; enzymes stand still in the compartment where the action took place until decay, depositing traces; traces, representing, e.g., tacit message `intention`, randomly diffuse among neighbouring compartments until either (i) decay or (ii) find a matching molecule to apply their perturbation action to; the perturbation action, e.g., `attract`, makes the involved molecule diffuse toward the compartment where the trace's father enzyme belong.

## 6.4 Focus on Similarity

In Section 7.1.4 within the upcoming chapter, it is described how the Molecules of Knowledge model is able, to some extent, to support *semantic coordination* patterns, e.g., clustering of similar information, without any sort of semantic matchmaking, similarity measures, ontology-based reasoning, or the like, but relying purely on epistemic interpretation of users' interactions.

Nevertheless, as apparent from the detailed description of $\mathcal{M}o\mathcal{K}$ reactions machinery given in Subsection 6.2.1, a $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function for *measuring similarity* is considered, and likely to deeply impact the accuracy of the self-organising behaviours achievable by the model. Although a definitive choice for $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function has not yet been made, being the issue currently still under investigation, what follows reports on some early experiments regarding application of *text mining* related techniques in a demo scenario of academic papers clustering.

### 6.4.1 Viable Approaches

Measuring similarity between $\mathcal{M}o\mathcal{K}$ atoms and molecules requires first of all to decide which kind of content these $\mathcal{M}o\mathcal{K}$ abstractions convey. In fact, measuring similarity between single words, sentences, or whole documents can be quite a different task, requiring more or less pre-processing of the raw text being the source of information.

*Text-based knowledge* has been already identified as the kind of information $\mathcal{M}o\mathcal{K}$ aims at dealing with, thus what is missing is to decide the *granularity* of this text-based knowledge, that is, namely, whether atoms will convey words and molecules sentences, or atoms sentences and molecules collections of them – possibly even whole documents –, or something else similar.

For the experiments described in Subsection 6.4.2, the choice is to put BibTeX[8] records first – suitably edited to remove fields not useful for the purpose –, then abstracts, finally full academic papers, into atoms, then to exploit $\mathcal{M}o\mathcal{K}$ *aggregation reaction* to cluster similar atoms into molecules.

It should be noted that *hierarchical clustering* may be achieved by clustering similar molecules into the same compartment, whereas dissimilar molecules into different compartments. Furthermore, depth of the hierarchy may be increased by recursively considering neighbourhoods of compartments, neighbourhoods of neighbourhoods, and so on—however, such an interesting perspective is left for future investigation. As a side note, it is interesting to recognise that this kind of *topological clustering* is similar to the clustering approach based on *self-organising maps* [LHKK96].

The vast majority of *similarity measures* (or, *correlation functions*) are based on the *vector space* representation of documents (or, more generally, text snippets), according to which each document is represented by a vector where each cell stores a word of the document and its *weight* inside the document. For the experiments in Subsection 6.4.2, weights are computed based on [Leu01], which combines Okapi's *tf* score [RWJ$^+$95] and

[8] `http://www.bibtex.org`

INQUERY's normalised *idf* score [ACC+98]:

$$v_{i,j} = \frac{tf_{i,j}}{tf_{i,j} + 0.5 + 1.5\frac{doclen_j}{avgdoclen}} \cdot \frac{log(\frac{colsize+0.5}{docf_i})}{log(colsize+1)} \tag{6.1}$$

where $v_{i,j}$ is the weight of i-*th* term within j-*th* document, $tf_{i,j}$ is the frequency of term $i$ in document $j$, $docf_i$ the number of documents in which term $i$ is, $doclen_j$ the number of terms in document $j$, *avgdoclen* the average of documents' length.

The simplest method to measure similarity between text snippets is to count the co-occurrences of words, that is, the number of words appearing in both snippets, and possibly how many times, normalising the obtained value according to the length of the text within the involved snippets.

A slightly more refined, and widely used, method is to rely on *cosine similarity* [Hua08]:

$$Sim(C_1, C_2) = \frac{\sum_{k=1}^{n} c_i(t_k) \cdot c_j(t_k)}{\sqrt{\sum_{k=1}^{n} c_i(t_k)^2 \cdot \sum_{k=1}^{n} c_j(t_k)^2}} \tag{6.2}$$

where $C_i = \{c_i(t_1), \ldots, c_j(t_n)\}$ and $C_j$ are the weight vectors of the documents to compare, and $t_k$ represents the k-*th* word. $Sim(C_1, C_2) \in [-1, 1]$, meaning that text snippets $C_1, C_2$ are identical (1), totally dissimilar $(-1)$, unrelated (0), or anything in-between.

Another method, emphasising big differences in weights, is the *mean squares difference* [HKBR99]:

$$d(C_i, C_j) = \frac{\sum_{k=1}^{n}(c_i(t_k) - c_j(t_k))^2}{n}$$

Yet another possibility, is to exploit *euclidean distance* [HNP05]:

$$dist(C_i, C_j) = \sqrt{\sum_{k=1}^{n} \mid c_i(t_k) - c_j(t_k) \mid^2}$$

Furthermore, in [JS13], a *concept-based* similarity measure is proposed, considering three different levels: sentences, whole documents, collections of documents. For each sentence (or document, or corpus), *part-of-speech* tagging [Abn97] is used to semantically interpret the content of the text, so as to define and assign concepts to that portion of text. The authors introduce three novel measures of frequency:

- for sentences, the conceptual term frequency (*ctf*), that is, the number of occurrences of a concept in that sentence
- for documents, the concept-based term frequency (*tf*), that is, the number of occurrences of a concept within the whole document
- for documents collections, the concept-based document frequency (*df*), that is, the number of documents which contain the concept

Based on these, as well as many other factors, a new similarity measure is defined [Abn97].

In the upcoming section, some of these similarity measures are put to test within a demo scenario involving academic papers clustering.

### 6.4.2 Experiments

Prior to comparing efficacy and efficiency of the different similarity measures just described, it is necessary to perform some pre-processing of the documents subject of the experiments—that is, academic papers taken from APICe online repository[9]. Pre-processing involves the most common steps of a typical text mining pipeline, such as text extraction, tokenisation, stemming, part-of-speech tagging, and the like. The aim is to obtain the aforementioned vector space representation.

In the following, only three out of the many more experiments performed are described, being the most successful ones, both w.r.t. clustering results and to efficiency. In particular, all the three experiments are based on full-text atoms, that is, atoms storing a vector space representation of the whole text of the paper they represent.

However, the experiments actually performed considered every combination of text snippet and similarity measure, that is, both BibTeX atoms, abstract-based atoms, and full-text based atoms, have been aggregated according to both the most basic similarity measure, cosine similarity, mean squares difference, euclidean distance, and concept-based similarity.

It should be noted that, being $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ similarity measure exploited in $\mathcal{M}o\mathcal{K}$ during system operation, not in a pre-processing stage, the weighting formula (6.1) has been adjusted to consider a bootstrap phase in which no other document is known, and to dynamically change according to new documents (atoms) being put into $\mathcal{M}o\mathcal{K}$ compartments.

**Most basic measure** In the first experiment, a very basic approach has been pursued, that is:

- weight assignment is simply constant, that is, each term considered for measuring similarity is assigned the constant weight $\frac{1}{n}$, where $n$ is the number of terms in the document—or, more generally, in the text snippet
- similarity measure – that is, $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ – is simply based on *weighted co-occurrences* of terms, that is, the more terms are in common between text snippets, the more snippets are considered similar
- co-occurrences of terms is checked syntactically—no semantics involved here

The role of the clustering algorithm is obviously played by $\mathcal{M}o\mathcal{K}$ *aggregation reaction*, taking two atoms, checking their similarity, then merging them in a molecule if they are similar *enough*—the cluster, either a novel one or one already existing.

[9] `http://apice.unibo.it`

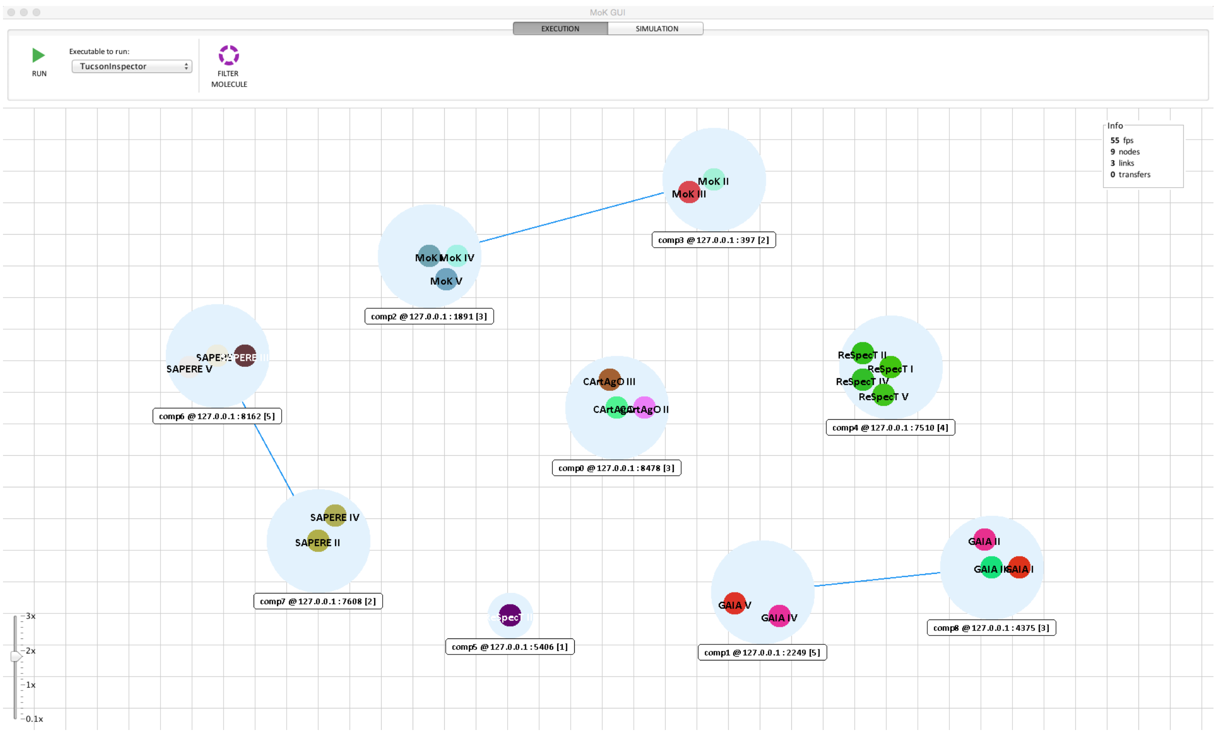


Figure 6.20: Clustering result for most basic similarity measure.

The reason to include this very basic approach into discussion of experiments stems from the on-line character of the clustering process supported by $\mathcal{M}o\mathcal{K}$: being the aggregation reaction meant to continuously execute aggregations according to its dynamic rate, it is crucial that weight assignment, co-occurrences check, and similarity measure, are as *efficient* (barely speaking, *fast*) as possible—effectiveness is not substantial in $\mathcal{M}o\mathcal{K}$, due to its very nature. In this case, e.g., time taken to compute similarity for a single comparison is the lowest experienced: just $\approx 8$ ms.

Figure 6.20 above graphically depicts the clustering results obtained. There, coloured, smaller balls are not atoms, but molecules, and light-blue, bigger balls are themselves molecules, although aggregating other molecules. Each smaller molecule has a name, roughly denoting the topic of the atoms (hence papers) it aggregates, as defined by experts in the field. The different colours roughly resemble similarity as computed by $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function. Links between bigger molecules are another graphical representation of similarity calculated by $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$: linked molecules are similar enough to be considered somehow connected (correlated), but not enough to be clustered together in a single molecule—according to the application-specific threshold considered in Subsection 6.2.1.

Figure 6.20 clearly shows that $\mathcal{M}o\mathcal{K}$ aggregation reaction has some success in finding similarity patterns, even in the case of $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ most basic implementation. E.g., although no molecule includes all the papers tagged with the same topic by experts, no molecule

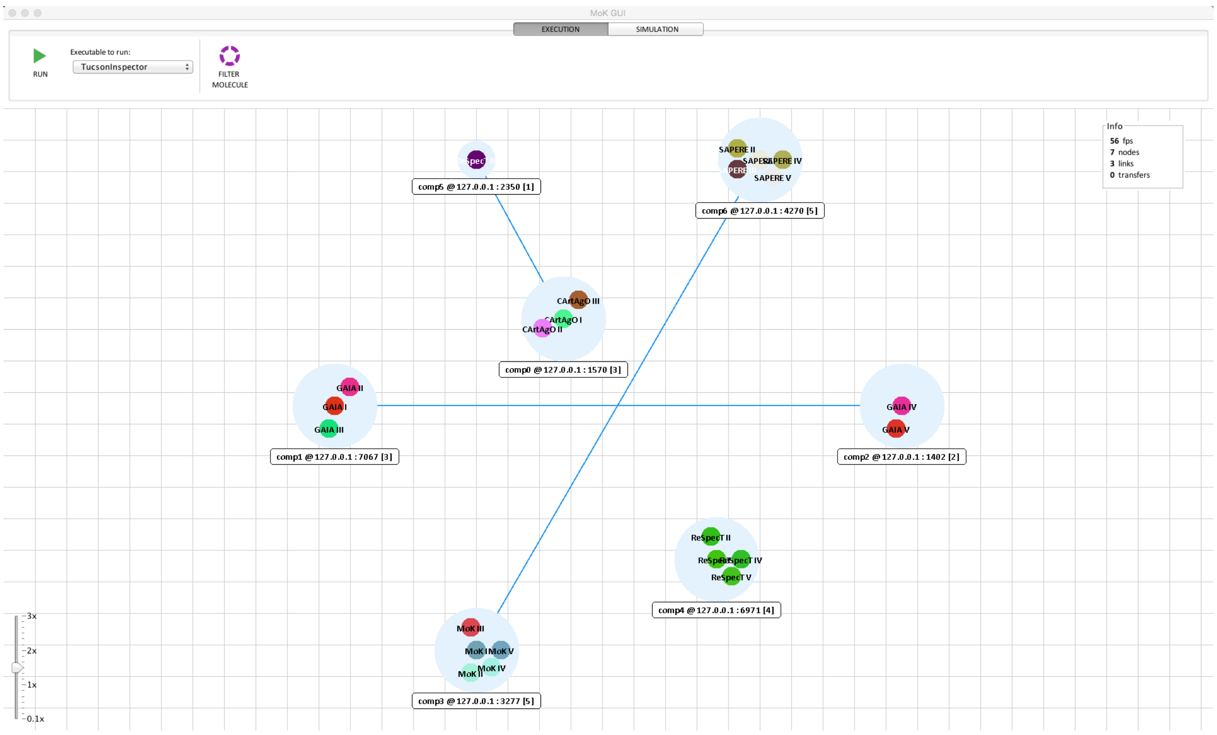


Figure 6.21: Clustering result for cosine similarity measure.

aggregates information belonging to different topics. Furthermore, similar information aggregated by different molecules is anyway recognised to be similar by $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function—a link lies between the different molecules.

**Cosine similarity** In this second experiment, the previous approach is refined as follows:

- weight assignment occurs according to (6.1)
- $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ is based on *cosine similarity*, that is, on function (6.2)
- co-occurrences of terms is still checked syntactically

Figure 6.21 shows that the above refinements lead to an improvement in the clustering results obtained by $\mathcal{M}o\mathcal{K}$ aggregation reaction feature cosine similarity measure as $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$. In fact, now all the papers belonging to the same topic, according to experts evaluation, are correctly assigned to the same cluster, that is, molecule, except for one molecule consisting of 5 papers tagged with term `ReSpecT` (see compartment 5, purple molecule, on the top left of the picture).

Furthermore, inter-molecule similarity connections are now linking molecules clustering papers belonging to different but strongly related topics, again according to experts evaluation, besides molecules aggregating papers belonging to the same topic—those tagged with term `GAIA`, as in Figure 6.20. Namely, papers about `SAPERE` and `MoK` have a strong similarity due to many terms, such as "chemical", "coordination", "self-organisation", appearing with similar weights in both documents—the same for `CArtAgO` and `ReSpecT` papers, related by terms such as "coordination", "artefact", "environment", among the many.

However, here time taken to compute similarity for a single aggregation reaction application is $\approx 18$ ms—doubled w.r.t. experiment 1.

**Concept-based similarity** In this latter experiment, the refined approach is further changed by:

- modifying the matching function checking co-occurrences of terms, which has been extended so as to consider WordNet[10]-based *synsets* [MF98] in similarity measure—that is, synonyms, meronyms, hyperonymy, etc
- modifying $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function implement the *concept-based similarity* measure defined in [JS13]

Figure 6.22 shows no substantial improvement w.r.t. experiment 2, but only a different molecules configuration, suggesting that adopting more complex approaches, such as as concept-based similarity and synsets-based matching, may be not worth the investment in computational complexity and execution time—depending on the application domain, of course. In fact, consider this $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ implementation takes $\approx 57$ ms to perform a single comparison, way more than both previous experiments—mostly due to WordNet lookup.

Nevertheless, it should be noted that the lack of improvement may be largely depending on the application domain where $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ has been put to test: all the papers involved in the experiment are from the computer science and engineering area, where synsets are not so meaningful and scarcely represented within WordNet.

**Technical notes** Experiments have been run on an Intel Core i7-720QM (1.6 GHz, 6MB L3 cache) machine with 4GB of DDR3 RAM, using Sabayon-Linux kernel 3.9.11 64 bit. Java VisualVM[11] has been used for profiling execution time, even in sub-processes. Slightly more than a hundred papers have been considered, that is, all papers published on APICe between 2010 and 2013. This apparently small number is motivated by the fact that one should imagine to let $\mathcal{M}o\mathcal{K}$ operate for a few hours, within a research collaboration scenario, in which researchers from a few different research teams in the same department share a $\mathcal{M}o\mathcal{K}$ platform for retrieving potentially relevant scientific papers.

[10]`http://wordnet.princeton.edu`

[11]`http://visualvm.java.net`

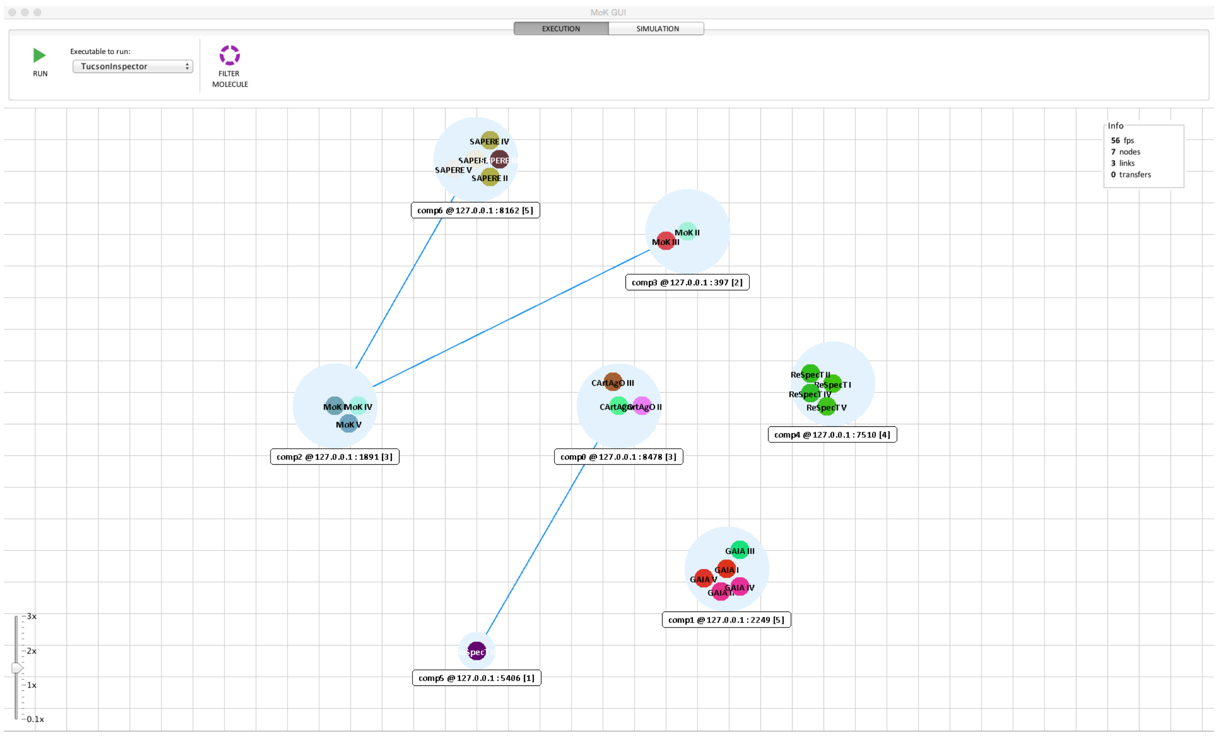


Figure 6.22: Clustering result for concept-based similarity measure.

**Discussion of results** The purpose of the experiments just described is not that of rigorously evaluating performance of the similarity measures, nor of the clustering capabilities of *MoK* aggregation reaction. Instead, the aim of the section is that of assessing feasibility of the chemical-inspired coordination-based approach to clustering promoted by *MoK*—not only aggregation, but also the interplay between diffusion and reinforcement support clustering within *MoK*, as seen in Subsection 6.3.2.

In particular, the aim is to show that some sort of *semantic coordination patterns*, such as clustering of similar information, may be achieved successfully without resorting to full-fledged semantic reasoning facilities—e.g., ontology-based entailment and the like.

The motivation behind this approach is that *MoK* is expected to be deployed within knowledge-intensive socio-technical systems, that is, business environments within which information is produced in massive amounts and at a very fast pace. Therefore, being capable of detecting and reifying similarity patterns as soon as new information is available, may be crucial for the sustainability of the knowledge-management platform at hand.

Accordingly, the focus of the experiments has been about simplicity and efficiency of the approach, rather than on correctness and efficacy.

# Chapter 7

# The $\mathcal{M}$olecules $\mathcal{o}$f $\mathcal{K}$nowledge Technology

In this chapter an overview of the current state of $\mathcal{MoK}$ technology is provided, that is, a report on the extent to which the $\mathcal{MoK}$ model has been actually implemented into a working system. Accordingly, Section 7.1 discusses the $\mathcal{MoK}$ prototype implemented on top of the TuCSoN coordination infrastructure, whereas Section 7.2 describes an envisioned full-fledged $\mathcal{MoK}$ ecosystem, whose building blocks are currently under development.

## 7.1 Prototype on TuCSoN

In this section a prototype implementation of the $\mathcal{M}$olecules $\mathcal{o}$f $\mathcal{K}$nowledge model upon the TuCSoN coordination infrastructure [OZ99] is described, with an emphasis on the *middleware* layer, thus on implementation of $\mathcal{MoK}$ compartments and their chemical-like computational machinery.

In particular, TuCSoN is used to distribute across a network of connected devices a number of $\mathcal{MoK}$ compartments, implemented as suitably programmed ReSpecT tuple centres [Omi07]. There, information atoms and molecules are packaged into TuCSoN first-order logic tuples and manipulated both by catalysts (TuCSoN agents) and $\mathcal{MoK}$ reactions (ReSpecT specifications and *chemical law tuples*).

Accordingly, Subsection 7.1.1 overviews how $\mathcal{MoK}$ main abstractions are mapped on TuCSoN and ReSpecT ones, while Subsection 7.1.2 briefly describes how $\mathcal{MoK}$ compartments are implemented on top of ReSpecT tuple centres. Then, Subsection 7.1.3 further details $\mathcal{MoK}$ compartment implementation, whereas Subsection 7.1.4 reports on early evaluation of the prototype within a simulated citizen journalism scenario.

**Motivation** The TuCSoN infrastructure [OZ99] is an ideal candidate for the implementation of a $\mathcal{MoK}$-like middleware, most of all thanks to *(i)* its situated architecture, enabling definition of a *topology* of distinct, *situated computational loci* – thus, providing a notion of *locality*, too –, and *(ii)* programmability of the ReSpecT tuple centres [Omi07] it distributes over the network.

Doing so makes it possible to straightforwardly map MoK compartments to ReSpecT tuple centres, suitably programmed so as to execute the Gillespie algorithm for stochastic simulation of a chemical solution [Gil77]. These tuple centres are spread over a network according to an *application-specific* topology, defined again in ReSpecT—ultimately providing the notion of *neighbourhood*.

As a side note, in the following the ReSpecT implementation of the Gillespie algorithm is referred to as "the chemical engine" (of a MoK compartment). Also, terms "MoK reaction" and "chemical law" may be used interchangeably, being the latter the implementation of the former within the prototype.

### 7.1.1 Main Abstractions

In the ReSpecT specification implementing MoK compartments' chemical engine, three are the main abstractions upon which the Gillespie simulation algorithm is based—thus MoK model abstractions mapped to: *reactants*, *chemical laws*, *neighbourhoods*. Being TuCSoN first-order logic tuples the only data structure ReSpecT can manipulate, each abstraction is actually implemented as a different kind of tuple.

**Reactants** Each reactant is represented as a TuCSoN tuple with the following syntax:

```
reactant(Kind, Amount)
```

where term `Kind` is a placeholder for the actual reactant, and term `Amount` indicates its quantity, resembling *relevance*—e.g., concentration or multiplicity. The term "reactant" here is used to denote atoms, molecules, enzymes, traces, and also seeds.

**Chemical laws** The chemical laws exploited by the chemical engine to implement MoK reactions are essentially of three sorts: *ordinary*, *temporal*, and *instantaneous*.

*Ordinary laws* are those whose scheduling is driven by a *rate expression*, determining the likelihood they are actually executed, as well as time taken to complete execution. Besides this, in order to be executed, they need the correct amount of reactants to be present in the tuple centre where they are installed, at the time they are scheduled.

These laws are reified as TuCSoN tuples having the following structure:

```
law([Reactants], Rate, [Products])
```

where:

- term `Reactants` represents the set of reactants necessary for the reaction to be selected for scheduling, and *consumed* by the reaction if successfully applied

- term *Rate* is the rate expression of the reaction, that is, the (stochastic) likelihood that a specific law is actually selected for execution among many *competing* ones
- term *Products* represents the set of reactants produced by the reaction as a result of its successful execution

*Temporal laws* are those with no execution rate associated, but to be executed according to a dynamically configurable time interval—still, if and only if all the necessary reactants are available, in the right amount, when the period expires.

These laws are reified as TuCSoN tuples as follows:

```
timedLaw([Reactants], Period, [Products])
```

where terms *Reactants* and *Products* retain their usual meaning, while term *Period* indicates the time interval according to which the reaction should be executed.

*Instantaneous laws* are those having no associated rate or time interval, dictating their scheduling. Instead, they are scheduled for execution as soon as the required reactants are available within the tuple centre where they are installed.

Their representation as TuCSoN tuples is the same as other laws, provided term *Rate/Period* is specified by the constant `inf`:

```
law([Reactants], inf, [Products])
```

**Neighbourhood** The term "neighbourhood" denotes the set of tuple centres which are reachable from a given tuple centre, according to an application-specific *reachability criterion.*

Usual network-based reachability, in fact, is of no help in the context of a TuCSoN-based middleware, because TuCSoN tuple centres can communicate with any other tuple centre belonging to any other TuCSoN node, provided it has a known network address. Thus, without some form of reachability constraint, any MoK compartment would be able to interact with any other compartment within a given MoK system.

Accordingly, MoK administrators are free to define application-specific neighbourhoods through a special kind of tuple *neighbour*, to be put in a tuple centre for each reachable tuple centre. The tuple is structured as follows:

```
neighbour(Id, Address, Port)
```

where:

- term *Id* represents the *locally unique* identifier of the MoK compartment in the neighbourhood—equal to the name of the ReSpecT tuple centre acting as MoK compartment
- term *Address* is the IP address of the TuCSoN node hosting the neighbour tuple centre

- term ***Port*** is the TCP port number where the mentioned TuCSoN node is listening to incoming requests

Once that neighbourhood relationships are defined, a special kind of tuple – or better, of tuple wrap – is provided to indicate the chemical engine that the wrapped tuple has to be sent to a (*uniformly probabilistically* chosen) neighbour: ***firing***.

Firing tuples have the following structure:

`firing(`***Reactant***`)`

where ***Reactant*** is a placeholder for the actual reactant to be sent to the selected neighbour among the pool of available ones.

## 7.1.2 The Chemical Engine Logic

The *logic* of the chemical engine implemented by ReSpecT specifications may be conceptually split in two distinct stages, to be iterated until no more *triggerable laws* can be found:

1. selection of the chemical law to schedule for execution
   - *a.* match reactant *templates* against available reactants, to collect *triggerable* laws
   - *b.* compute *effective* rates for all the triggerable laws
   - *c.* randomly select a triggerable law, *stochastically* chosen according to the effective rates just computed
2. execution of the selected chemical law, following *Gillespie algorithm* for chemical solution dynamics simulation [Gil77]
   - *a.* instantiate products
   - *b.* update reactants and products quantity in the space
   - *c.* enqueue firing reactants ready to be moved (if any)
   - *d.* update the state of the system—e.g., Gillespie exponential decay

**Reactants matching** (1.*a*) In step 1.*a*, the engine verifies, for each chemical law, whether the required reactants are available within the local compartment, that is, whether the local ReSpecT tuple centre contains *at least* one reactant tuple matching each of the reactant templates specified in each chemical law, with at least the desired amount. Then, for each chemical law whose above condition is satisfied, the engine marks the law as *triggerable*, that is, suitable to be scheduled and executed.

According to the $\mathcal{M}o\mathcal{K}$ model, matching of reactant templates against actual reactants should be based on the $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ similarity function, so as to leverage some sort of *semantic* capabilities. In this prototype implementation, however, $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ is simply an improved version of Prolog syntactical *unification*, considering also simple Java regular expressions and WordNet synsets [MF98].

Also, according to the $\mathcal{M}o\mathcal{K}$ model, the matching process should be influenced by the *relevance* of reactants in the compartment, that is, by some value representing the (relative) amount of a given reactant (w.r.t. all the other matching reactants available in the compartment). This guarantees that the more relevant a given reactant is, the *more likely* it will win matching.

Nevertheless, in this prototype implementation, relevance is approximated by *multiplicity* of the tuple representing the reactant, and matching consequently affected by this multiplicity—still probabilistically[1].

**Effective rates** (1.*b*) In step 1.*b*, the engine should compute the *effective rate* of $\mathcal{M}o\mathcal{K}$ triggerable reactions based on their *nominal* rate. The former is the rate as dynamically computed according to the system state – e.g., reactants multiplicity –, actually driving *probabilistic selection* of the chemical law to be executed. The latter is the rate expression as defined within the chemical law implementing a $\mathcal{M}o\mathcal{K}$ reaction.

According to the $\mathcal{M}o\mathcal{K}$ model, the nominal rate of a reaction can be specified by an *arbitrary rate expression*, so as to enable a wide range of stochastic emergent behaviours by going beyond the usual *law of mass action*—as seen in Section 2 of Chapter 6.

Nevertheless, in this prototype implementation, nominal rates ar assumed to be continuos values, which are then automatically multiplied for the product of the concentrations of the reactants to be consumed. This way, rate expressions in the prototype are restricted to be a simple parametrisation of the law of mass action [Car08], which can be adjusted at run-time. Ultimately, this means the range of behaviours achievable by the prototype is a subset of those obtainable with the $\mathcal{M}o\mathcal{K}$ model—see, e.g., simulations in Section 2 of Chapter 6.

**Law selection** (1.*c*) In step 1.*c*, the chemical engine applies the Gillespie chemical solution dynamics simulation algorithm to choose the chemical law to schedule for execution [Gil77].

Conceptually, selection and execution can be accomplished as a single process proceeding as follows:

- each triggerable law is conceptually associated to a timer, whose value is (uniformly) probabilistically set at a random value between 0 and the inverse of the effective rate of the reaction
- then, timers are started in parallel so that a *critical race* begins among the competing chemical laws

[1]Probabilistic sampling of matching reactants is implemented on top of the *uniform coordination primitives* described in Chapter 3.

- finally, the chemical law whose associated timer expires first, is executed with a stochastic, *exponentially distributed delay*, based on the summation of the concentrations of all reactants considered

However, in the prototype, selection and execution are two distinct steps. While execution is discussed below, selection proceeds as follows: based on the effective rates computed as described above, the chemical engine (uniformly) probabilistically chooses a random reaction to schedule for execution.

In this prototype, the role of timers is played by *spawned activities*, which are TuCSoN implementation of LINDA `eval` primitive [Gel85]—see the official TuCSoN documentation for details[2].

**Products instantiation** (2.*a*) In step 2.*a*, the products of the chemical law selected for execution are *instantiated* based on the matching reactants actually sampled from the MoK compartment. This means, if any product contained variables corresponding to other variables specified in reactant *templates*, Prolog-based unification (and propagation) ensures they now contains ground terms.

Accordingly, for each product specified in the scheduled reaction, the engine checks whether it contains *unbound* variables; if so, the engine then iterates over reactants looking for the corresponding variable, bound to an actual value; finally, *forward propagation* is undertaken, to bind that value to the corresponding product variable.

According to the MoK model, a product may specify *arbitrary expressions* over reactants as its own *instantiation rules*. This is necessary to, e.g., support complex forms of information aggregation, such as filtering, merging, etc.

Nevertheless, in this prototype implementation, products are restricted to be composed by Prolog values or variables, simple arithmetic expressions involving variables and values, simple string operators such as concatenation or substitution (built-in in tuProlog [DOR01], the Prolog engine exploited in TuCSoN), or simple list operations such as replacement and concatenation (again, built-in in tuProlog).

This way, a fair degree of expressiveness regarding products is provided, while the implementation is kept simple, thus efficiency high.

**Compartment update** (2.*b*) In step 2.*b*, reactants and products are, respectively, *withdrawn* from the compartment and *injected* into it, that is, the corresponding tuples consumed and put, correspondingly, into the ReSpecT tuple centre acting as MoK compartment.

This is done as a single atomic operation, despite implying multiple `in` and `out` LINDA operations, thanks to the transactional execution semantics of ReSpecT specifications [DNO98].

[2]`http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide`

**Firing queue** (2.*c*) In step 2.*c*, the chemical engine prepares *firing tuples* for being sent to a *uniformly* probabilistically chosen neighbour, among those compartments in the neighbourhood of the one executing the reaction.

Namely, the TuCSoN tuples representing products of, e.g., a *MoK* diffusion reaction, are wrapped as firing tuples, then put into a *firing queue* waiting for their target ReSpecT tuple centre to be defined. Then, the ReSpecT program implementing the chemical engine retrieves the set of *application-specific neighbours*, and randomly picks one according to a uniform probability distribution[3]: this is the tuple centre receiving the firing tuple currently at the top of the firing queue.

This step is executed solely in case the chemical law implementing *MoK* diffusion reaction is the one being executed.

**Engine update** (2.*d*) In step 2.*d*, the engine takes care of updating its state, that is, the state of the simulation according to Gillespie algorithm [Gil77]. Among the many operations to be done, encompassing cleaning up temporary tuples, updating tuples tracking systems properties, and the like, one deserves special attention: enforcement of reaction *execution delay*. According to Gillespie in fact, besides the critical race among triggerable reactions, also the execution step involves time accounting.

In particular, between actual execution of a reaction and the next scheduling step, a *Poisson-distributed time delay* has to expire. The reason for doing this is to faithfully emulate the physical nature of a chemical solution, enabling well-founded formal investigation of asymptotical *convergence* to a desired *emergent* behaviour—as done, e.g., in Section 2 of Chapter 6.

**Final remarks** The whole chemical engine is implemented as a mixture of ReSpecT reactions and Prolog predicates, and the codebase is publicly available under LGPL license from TuCSoN repository in branch `MoK-proto`[4]—ReSpecT specification `mok_engine.rsp` in package `alice.tucson.service.config`.

The chemical engine may get stopped, e.g., to save computational resources, if, at anytime, no more triggerable laws can be found in the compartment. This may happen, e.g., because, for each chemical law, at least one reactant is missing, in the desired amount, from the compartment. The engine resumes itself as soon as at least one chemical law becomes triggerable, checking reactants availability condition anytime a reactant is put into the compartment.

[3]Probabilistic selection of the neighbour, based on uniform distribution, is implemented on top of the *uniform primitives* described in Chapter 3.

[4]`http://bitbucket.org/smariani/tucson/branch/MoK-proto`

### 7.1.3 Spotlight on Engine Implementation

In this section some details regarding ReSpecT implementation of the chemical engine behind $\mathcal{M}o\mathcal{K}$ compartments are provided.

$\mathcal{M}o\mathcal{K}$ abstractions are mapped to TuCSoN tuples, so as to be conveniently handled by ReSpecT specifications implementing $\mathcal{M}o\mathcal{K}$ reactions. For instance, a $\mathcal{M}o\mathcal{K}$ enzyme may be represented by the following tuple:

```
reactant(enzyme(species(Sp),strength(St),reactant(R),context(Ctx)),C)
```

where:

- term `Sp` denotes the species of the enzyme
- `St` is the strength of the reinforcement brought by the enzyme, according to its species
- `R` is the reactant actually targeted by the enzyme
- `Ctx` is meant to store any contextual information regarding the action reified by the enzyme, potentially relevant for coordination purpose
- `C` is the concentration of the enzyme

Then, an enzyme may be exploited by $\mathcal{M}o\mathcal{K}$ reinforcement reaction, which can be encoded as a TuCSoN tuple as follows:

```
law([reactant(enzyme(Sp,St,R1,Ctx),C1),reactant(R2,C2)],
                            Rate,
[reactant(enzyme(Sp,St,R1,Ctx),C1),reactant(R2,C2 + St)])
```

meaning that the relevance of reactant `R2` matching the enzyme's target `R1`, according to $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$, is increased by `St`.

Another interesting example is that of the TuCSoN tuple implementing $\mathcal{M}o\mathcal{K}$ decay reaction:

```
law([reactant(R,1)],Rate,[])
```

meaning that 1 unit of concentration of reactant `R` is destroyed.

As a last example, consider $\mathcal{M}o\mathcal{K}$ diffusion:

```
law([reactant(R,C)],Rate,[firing(reactant(R,C))])
```

meaning that, at a pace given by `Rate` – whose effective value is computed at run-time – an amount `C` of reactant `R` is made available for diffusion.

At this point, a legitimate question may arise: who takes care of selecting the compartment target of diffusion? How the choice is made?

Here is were ReSpecT comes to the rescue. The above tuples are nothing more that a convenient way of representing $\mathcal{M}o\mathcal{K}$ reactions within the TuCSoN framework, that is, as first-order logic Prolog terms. Then, suitable ReSpecT specifications, besides those implementing the chemical engine, takes care of interpreting and executing the computation corresponding to this representation.

In the case of diffusion, e.g., an ad-hoc ReSpecT reaction probabilistically picks a `neighbour` tuple, exploiting uniform primitives thoroughly described in Chapter 3, so as to select the destination compartment based on a uniform probability distribution. Then, it sends the `firing` tuple by relying on TuCSoN communication facilities.

Another example of the need for ReSpecT is while matching reactant templates, used in the left-hand side of chemical laws, against the actual reactant tuples present in the local tuple centre. There, in fact, matching is based on $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ similarity measure, thus tuProlog engine [DOR01] cannot be used as is.

Accordingly, then, suitable ReSpecT reactions are dedicated to perform $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$-based matching, by exploiting the already mentioned TuCSoN *spawned activities*. This way, the kind of comparisons evaluated in Section 4 of Chapter 6 may be conveniently performed in parallel.

In next section, an early evaluation of the $\mathcal{M}o\mathcal{K}$ prototype so far described is presented.

### 7.1.4 Early Evaluation: $\mathcal{M}o\mathcal{K}$-News

While $\mathcal{M}o\mathcal{K}$ is a general-purpose model for knowledge self-organisation, it can be tailored on specialised application domains, by refining the notion of atom – which in turn impacts the notion of molecule, too –, and suitably defining $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ correlation function exploited in $\mathcal{M}o\mathcal{K}$ reactions. Since journalism and news management provide a prominent example of a knowledge-intensive socio-technical system, it has been chosen as a case study for $\mathcal{M}o\mathcal{K}$.

So, in the remainder of this section $\mathcal{M}o\mathcal{K}$ is specialised for the news management case study, starting from the standards for knowledge representation in the news management domain, and moving towards the definition of the $\mathcal{M}o\mathcal{K}$-News model for self-organisation of news.

**Knowledge representation for news management** The IPTC (International Press Telecommunications Council[5]) is a consortium of the world major news agencies, news publishers and news industry vendors. IPTC develops and maintains technical standards

[5] `http://iptc.org`

for improved news management—used, among the many, by the Italian ANSA, the American Thomson Reuters, and the English BBC. In order to specialise the MoK model and make it domain-specific, two of the most relevant standards for news are taken as a reference for atoms refinement: *NewsML*[6] and *NITF*[7].

NewsML is a news sharing format (or, tagging language) orthogonal w.r.t. media-type, aimed at conveying not only the core news *content*, but also the data that describe the content in an abstract way; namely, the *metadata*. In order to ease syntactical and semantical interoperability, NewsML adopts XML as the first implementation language for its standards, and maintains sets of *controlled vocabularies*, collectively branded as *NewsCodes*[8], to represent concepts describing and categorising news objects in a consistent manner—similarly to what domain-specific ontologies do.

By standardising on NewsCodes, news deliverers can ensure a common understanding of news content, as well as a large degree of interoperability between content from different providers. Also, NewsML allows knowledge workers to shape their own vocabularies, by defining concepts, and structuring them within proper knowledge containers.

NewsML provides journalists all the above features through four main abstractions:

**news item** it vehicles both the news' content and its metadata, hence, information reporting about what has just happened, as well as information about the news lifecycle—who owns the copyright, the topics covered, the target audience, distribution rights, etc.

**concept item** news are about events, persons, organisations, and the like. This information is worth to be remembered – and referred to – along with the news content to better identify, recognise, categorise it. Thus, a data structure collecting all this worth-to-be-remembered information is needed: the concept item, indeed

**package item** it is meant to organise and convey a structured set of News Items—such as "Top 10 news of the week" and the like

**knowledge item** it is a container for a whole taxonomy of Concept Items, acting like an ontology, which enables distribution of basic knowledge about all the terms the News Item refers to

Every element above is actually represented as an XML document (examples available at `http://www.iptc.org/std/NewsML-G2/`): within it, a journalist can use all the NewsML – and NITF, too – tags to enrich content and metadata of a news.

The News Industry Text Format, too, adopts XML to enrich the content of news articles, supporting the identification and description of a number of features typical in journalism, among which the most notable are:

[6]`http://iptc.org/standards/newsml-g2/`

[7]`http://iptc.org/standards/nitf/`

[8]`http://iptc.org/standards/newscodes/`

**who** owns the copyright to the item, who may republish it, and who it is about

**what** subjects, organisations, and events it covers

**when** it happened, was reported, issued, and revised

**where** it was written, where the action took place, and where it may be released

**why** it is newsworthy, based on the editor analysis of the metadata

NewsML, in fact, provides no support to any form of inline tagging adding information to the plain text, for instance, with the purpose of simplifying the work of a text mining algorithm usable to automatically process the document.

Therefore, NITF and NewsML are complementary standards. In fact, they perfectly combine to provide a comprehensive and coherent framework supporting management of the whole news lifecycle: comprehensive, given that the latter cares about news overall structure, including metadata, whereas the former focusses on their internal meaning, to make it unambiguous; coherent, because they both exploit the same IPTC abstractions—NITF makes usage of NewsCodes, too.

To give some hints about the capabilities of NITF, among its most used inline tags there are:

- *<person>* wraps personal names, both living people and fictitious. It could contain the *<function>* tag if the tagged person goes along with its public role
- *<org>* identifies organisational names. An inner tag (*<orgid>*) allows special codes to be added, such as codes from the Standard Industry Classification[9] list, or NewsCodes
- *<location>* identifies geographic locations and significant places. It contains either mere text or structured information including *<sublocation>*, *<city>*, *<region>*, *<state>*, and *<country>*
- *<event>* should be limited to newsworthy events, that is, events that carry news value for a journalist. Factors of news value are, for instance, significance, proximity, prominence of the involved persons, consequences, unusualness, human interest, timeliness
- *<object.title>* could tag anything that no other tag could wrap

NITF tags like the ones above (and many others) can be spread throughout the text of a web document to better characterise its most relevant terms semantics.[10]

[9] http://www.sec.gov/info/edgar/siccodes.htm
[10] http://www.iptc.org/site/News_Exchange_Formats/NITF/Examples/.

**$\mathcal{M}o\mathcal{K}$-News** Once that news representation standards are understood, they can be used with the aim of specialising the $\mathcal{M}o\mathcal{K}$ model for the news management scenario.

There, journalists usually gain the knowledge they need to create news from diverse and sparse sources of information. Assuming that sources provide journalists with the required *raw information* already formatted according to the afore-mentioned IPTC standards – as in real-world news agencies typically happens – a simple yet effective mapping between models can be devised out.

In fact, a $\mathcal{M}o\mathcal{K}$ atom has a clear counterpart in NewsML and NITF standards, that is, the *tag*: tags – along with their content – can easily be seen as the atoms that altogether compose the news-substance of a news story. As a result, a $\mathcal{M}o\mathcal{K}$-coordinated news management system would contain *<newsItem>* atoms, *<person>* atoms, *<subject>* atoms, etc.—that is, virtually one kind of atom for each NewsML/NITF tag.

Accordingly, a generic $\mathcal{M}o\mathcal{K}$ atom of the form $\texttt{atom(}\mathit{src}\texttt{,}\mathit{Content}\texttt{,}\mathit{Meta\text{-}info}\texttt{)}_c$ may become a specialised $\mathcal{M}o\mathcal{K}$-News atom as follows:

$$\mathit{atom(src,Content,sem(Tag,\ Catalog))}_c$$

where:

*src* ::= *news source uri*
*Content* ::= *news content*
*Meta-info* ::= sem(*Tag*,*Catalog*)
*Tag* ::= *NewsML tag* | *NITF tag*
*Catalog* ::= *NewsCode uri* | *ontology uri*

Here, the content of an atom is mostly given by the pair *<Content, Tag>*, where *Tag* could be either a metadata tag drawn from NewsML or an inline description tag taken from NITF. The precise and unambiguous semantics of the news content (*Content*) can be specified thanks to the *Catalog* information, which could be grounded in either NewsML or NITF standards in the form of NewsCodes, or instead be referred to a custom ontology defined by the journalist.

$\mathcal{M}o\mathcal{K}$ molecules and reactions – actually, any other $\mathcal{M}o\mathcal{K}$ abstraction – are both syntactically and semantically affected by such a domain-specific mapping of the $\mathcal{M}o\mathcal{K}$ model. Here, in fact, molecules can be re-interpreted as *ever-growing news stories*, and reactions as *autonomous news manipulators*. Technically (syntactically), this happens because they are specified by non-terminal symbols whose productions are the terminal symbols above instantiated.

Thus, e.g., $\mathcal{M}o\mathcal{K}$ *aggregation* reaction can now be interpreted as relating pieces of different news stories based on semantical relationships, either between NewsML/NITF tags or their tagged content. This *semantic matchmaking* capability may be straightforwardly assigned to $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function, suitably extended beyond content-based similarity solely.

*Reinforcement* reaction now increases relevance of those news pieces which are more frequently accessed, while, on the contrary, *decay* reaction makes obsolete, stale, or no longer considered information fade away as time passes by—recall $\mathcal{M}o\mathcal{K}$ seeds guarantee later recovery, if needed.

*Diffusion* then, is responsible of moving news toward the potentially interested journalists, leading to both *self-organisation* and *adaptation* of the spatial configuration of news pieces, within the network of $\mathcal{M}o\mathcal{K}$ compartments used as news stories repositories.

Another fundamental aspect of the $\mathcal{M}o\mathcal{K}$ model perfectly suits the news management scenario: the notion of *concentration*. The concentration $c_t$ of a tuple $t$ in a tuple space $TS$ is defined in $\mathcal{M}o\mathcal{K}$-News as the number $\sharp t$ of occurrences of the tuple over the number $\sharp TS$ of tuples in $TS$: so, $c_t ::= \frac{\sharp t}{\sharp TS}$. The three most characterising facets of a news are its *novelty*, *relevance*, and *usefulness*, that is, respectively: *(i)* how much new it is if compared to the actual environment and to time passing, *(ii)* how interesting it is perceived by other knowledge workers and target audience, and *(iii)* how useful it is to anyone according to some criteria—e.g., economic revenues, or new know-how it could provide. Concentration easily models all the three facets: the more a news – atom, molecule – is fresh, interesting, and useful to someone, the greater its concentration will be—hence, it will more frequently affect the dynamics of the $\mathcal{M}o\mathcal{K}$ system, according to the biochemical coordination metaphor implemented by $\mathcal{M}o\mathcal{K}$ compartments.

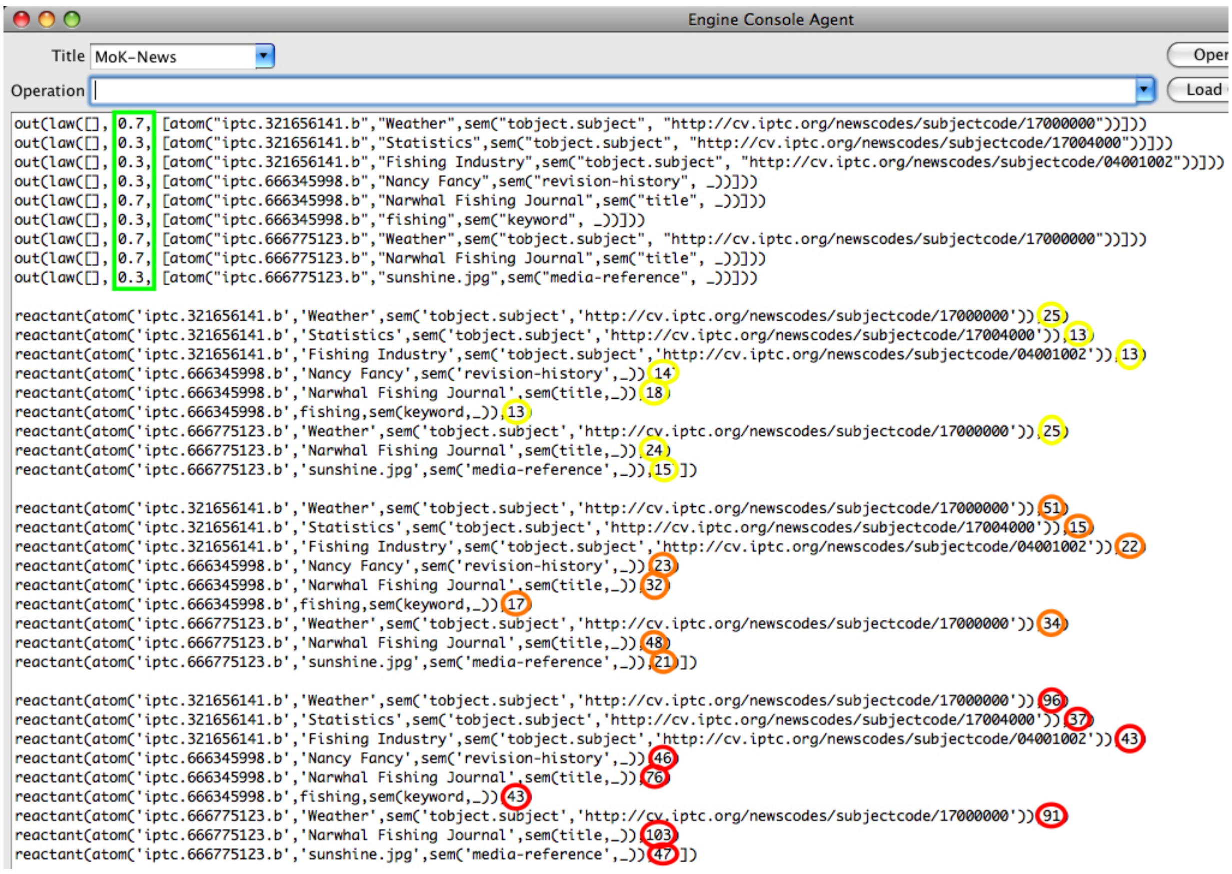
The specialised $\mathcal{M}o\mathcal{K}$ model just presented, called $\mathcal{M}o\mathcal{K}$-News, is evaluated against the $\mathcal{M}o\mathcal{K}$ prototype middleware described in Section 7.1. In next section, experimental results are reported and discussed.

**$\mathcal{M}o\mathcal{K}$-News experiments** In what follows, NITF-tagged documents are exploited as $\mathcal{M}o\mathcal{K}$ seeds (thus the sources of information), injecting news pieces atoms in compartments. There, some of the $\mathcal{M}o\mathcal{K}$-News model features are tested, as inherited from $\mathcal{M}o\mathcal{K}$. In particular, atoms aggregation into molecules, and their propagation toward interested news prosumers.

**Atoms injection** First of all, Figure 7.1 shows how atoms are injected by seeds into their compartment through $\mathcal{M}o\mathcal{K}$ injection reaction. It should be noted that the screenshot shows the triggerable laws with their effective rate, thus, chemical laws implementing $\mathcal{M}o\mathcal{K}$ reactions as they are after step 1.$b$ of the simulation algorithm detailed in Subsection 7.1.2.

The point is that each piece of news can be injected with different rates – as in the depicted scenario – and different concentrations, too, enabling attribution of different *relevance* to different news atoms according to either prosumers' preferences – in case of manual extraction – or automated criteria—in case of text processing by, for instance, ad-hoc NITF XML parsers.

**Molecules generation** Now, its is expected that molecules begin to self-assembly. Figure 7.2 shows exactly atoms aggregating into molecules, through a chemical law matching



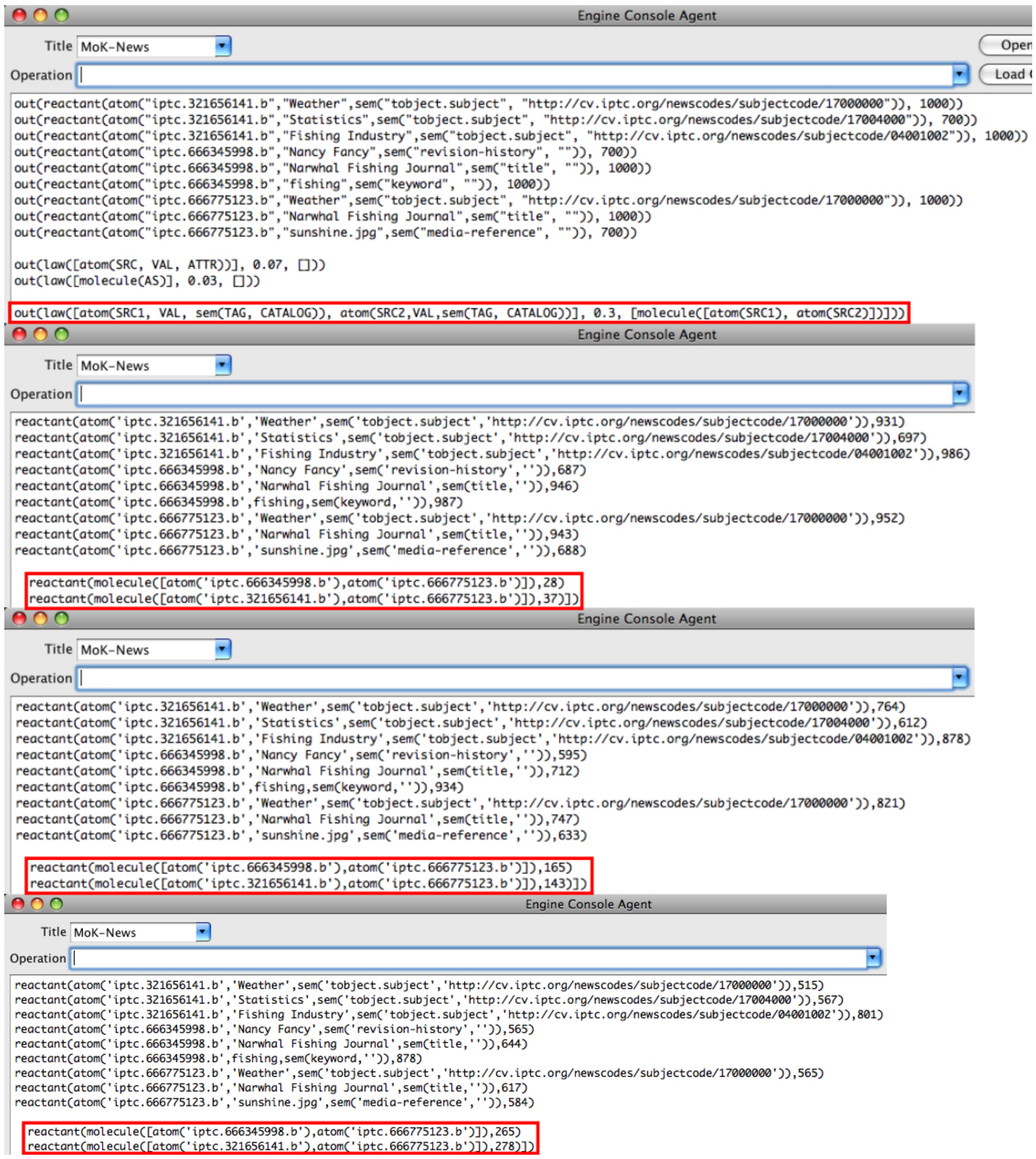
```
out(law([], 0.7, [atom("iptc.321656141.b","Weather",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/17000000"))]))
out(law([], 0.3, [atom("iptc.321656141.b","Statistics",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/17004000"))]))
out(law([], 0.3, [atom("iptc.321656141.b","Fishing Industry",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/04001002"))]))
out(law([], 0.3, [atom("iptc.666345998.b","Nancy Fancy",sem("revision-history", _))]))
out(law([], 0.7, [atom("iptc.666345998.b","Narwhal Fishing Journal",sem("title", _))]))
out(law([], 0.3, [atom("iptc.666345998.b","fishing",sem("keyword", _))]))
out(law([], 0.7, [atom("iptc.666775123.b","Weather",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/17000000"))]))
out(law([], 0.7, [atom("iptc.666775123.b","Narwhal Fishing Journal",sem("title", _))]))
out(law([], 0.3, [atom("iptc.666775123.b","sunshine.jpg",sem("media-reference", _))]))

reactant(atom('iptc.321656141.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),25)
reactant(atom('iptc.321656141.b','Statistics',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17004000')),13)
reactant(atom('iptc.321656141.b','Fishing Industry',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/04001002')),13)
reactant(atom('iptc.666345998.b','Nancy Fancy',sem('revision-history',_)),14)
reactant(atom('iptc.666345998.b','Narwhal Fishing Journal',sem(title,_)),18)
reactant(atom('iptc.666345998.b',fishing,sem(keyword,_)),13)
reactant(atom('iptc.666775123.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),25)
reactant(atom('iptc.666775123.b','Narwhal Fishing Journal',sem(title,_)),24)
reactant(atom('iptc.666775123.b','sunshine.jpg',sem('media-reference',_)),15])

reactant(atom('iptc.321656141.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),51)
reactant(atom('iptc.321656141.b','Statistics',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17004000')),15)
reactant(atom('iptc.321656141.b','Fishing Industry',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/04001002')),22)
reactant(atom('iptc.666345998.b','Nancy Fancy',sem('revision-history',_)),23)
reactant(atom('iptc.666345998.b','Narwhal Fishing Journal',sem(title,_)),32)
reactant(atom('iptc.666345998.b',fishing,sem(keyword,_)),17)
reactant(atom('iptc.666775123.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),34)
reactant(atom('iptc.666775123.b','Narwhal Fishing Journal',sem(title,_)),48)
reactant(atom('iptc.666775123.b','sunshine.jpg',sem('media-reference',_)),21])

reactant(atom('iptc.321656141.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),96)
reactant(atom('iptc.321656141.b','Statistics',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17004000')),37)
reactant(atom('iptc.321656141.b','Fishing Industry',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/04001002')),43)
reactant(atom('iptc.666345998.b','Nancy Fancy',sem('revision-history',_)),46)
reactant(atom('iptc.666345998.b','Narwhal Fishing Journal',sem(title,_)),76)
reactant(atom('iptc.666345998.b',fishing,sem(keyword,_)),43)
reactant(atom('iptc.666775123.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),91)
reactant(atom('iptc.666775123.b','Narwhal Fishing Journal',sem(title,_)),103)
reactant(atom('iptc.666775123.b','sunshine.jpg',sem('media-reference',_)),47])
```

Figure 7.1: The set of triggerable laws for atoms injection, with their effective rate [MO12].

similar information pieces coming from different sources—the aim is thus to aggregate different news stories talking about the same subject.
In particular, the depicted instantiation of the aggregation chemical law requires atoms to have similar `Content`, `Tag` and `Catalog` fields, being $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function defined as described in Subsection 7.1.2 as a slight extension to Prolog unification.

**Smart migration** Finally, Figure 7.3 and Figure 7.4 show how smart migration of news pieces can be realised within $\mathcal{M}o\mathcal{K}$-News. The depicted scenario is quite simple: a generator $\mathcal{M}o\mathcal{K}$-News compartment stores a collection of different news sources – talking about weather, baseball and finance – and diffuses them to the neighbour compartments "economics" and "sports"—belonging to journalists/consumers interested to that particular topic.

Despite diffusion being implemented to be equiprobable towards each neighbour compartment, every user tends to attract within her compartment the most appealing knowledge chunks. What makes this *emergent self-organisation* phenomena happen is the



```
out(reactant(atom("iptc.321656141.b","Weather",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/17000000")), 1000))
out(reactant(atom("iptc.321656141.b","Statistics",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/17004000")), 700))
out(reactant(atom("iptc.321656141.b","Fishing Industry",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/04001002")), 1000))
out(reactant(atom("iptc.666345998.b","Nancy Fancy",sem("revision-history", "")), 700))
out(reactant(atom("iptc.666345998.b","Narwhal Fishing Journal",sem("title", "")), 1000))
out(reactant(atom("iptc.666345998.b","fishing",sem("keyword", "")), 1000))
out(reactant(atom("iptc.666775123.b","Weather",sem("tobject.subject", "http://cv.iptc.org/newscodes/subjectcode/17000000")), 1000))
out(reactant(atom("iptc.666775123.b","Narwhal Fishing Journal",sem("title", "")), 1000))
out(reactant(atom("iptc.666775123.b","sunshine.jpg",sem("media-reference", "")), 700))

out(law([atom(SRC, VAL, ATTR)], 0.07, []))
out(law([molecule(AS)], 0.03, []))

out(law([atom(SRC1, VAL, sem(TAG, CATALOG)), atom(SRC2,VAL,sem(TAG, CATALOG))], 0.3, [molecule([atom(SRC1), atom(SRC2)])]))
```

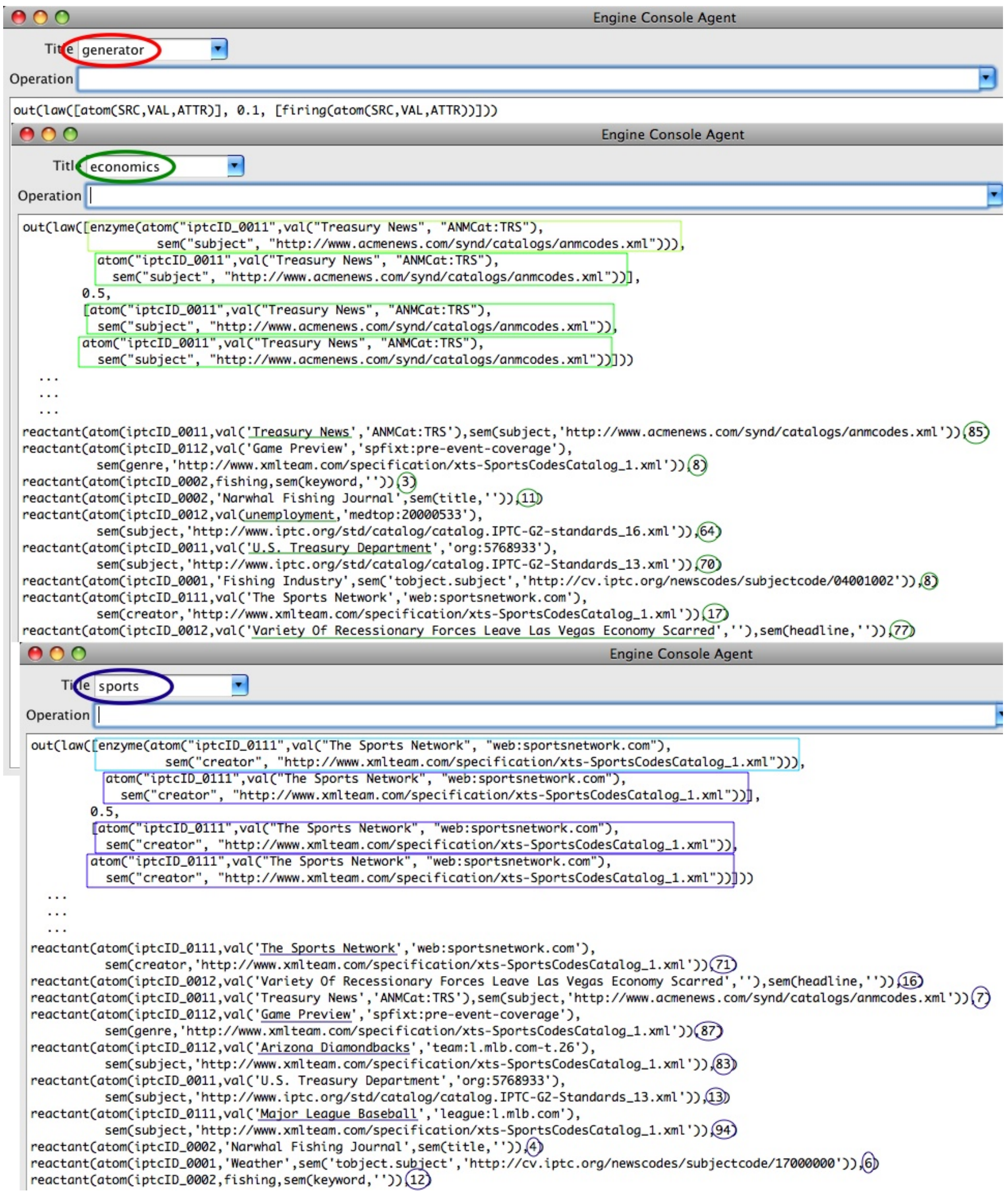
```
reactant(atom('iptc.321656141.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),931)
reactant(atom('iptc.321656141.b','Statistics',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17004000')),697)
reactant(atom('iptc.321656141.b','Fishing Industry',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/04001002')),986)
reactant(atom('iptc.666345998.b','Nancy Fancy',sem('revision-history','')),687)
reactant(atom('iptc.666345998.b','Narwhal Fishing Journal',sem(title,'')),946)
reactant(atom('iptc.666345998.b',fishing,sem(keyword,'')),987)
reactant(atom('iptc.666775123.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),952)
reactant(atom('iptc.666775123.b','Narwhal Fishing Journal',sem(title,'')),943)
reactant(atom('iptc.666775123.b','sunshine.jpg',sem('media-reference','')),688)

  reactant(molecule([atom('iptc.666345998.b'),atom('iptc.666775123.b')]),28)
  reactant(molecule([atom('iptc.321656141.b'),atom('iptc.666775123.b')]),37)])
```

```
reactant(atom('iptc.321656141.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),764)
reactant(atom('iptc.321656141.b','Statistics',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17004000')),612)
reactant(atom('iptc.321656141.b','Fishing Industry',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/04001002')),878)
reactant(atom('iptc.666345998.b','Nancy Fancy',sem('revision-history','')),595)
reactant(atom('iptc.666345998.b','Narwhal Fishing Journal',sem(title,'')),712)
reactant(atom('iptc.666345998.b',fishing,sem(keyword,'')),934)
reactant(atom('iptc.666775123.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),821)
reactant(atom('iptc.666775123.b','Narwhal Fishing Journal',sem(title,'')),747)
reactant(atom('iptc.666775123.b','sunshine.jpg',sem('media-reference','')),633)

  reactant(molecule([atom('iptc.666345998.b'),atom('iptc.666775123.b')]),165)
  reactant(molecule([atom('iptc.321656141.b'),atom('iptc.666775123.b')]),143)])
```

```
reactant(atom('iptc.321656141.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),515)
reactant(atom('iptc.321656141.b','Statistics',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17004000')),567)
reactant(atom('iptc.321656141.b','Fishing Industry',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/04001002')),801)
reactant(atom('iptc.666345998.b','Nancy Fancy',sem('revision-history','')),565)
reactant(atom('iptc.666345998.b','Narwhal Fishing Journal',sem(title,'')),644)
reactant(atom('iptc.666345998.b',fishing,sem(keyword,'')),878)
reactant(atom('iptc.666775123.b','Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),565)
reactant(atom('iptc.666775123.b','Narwhal Fishing Journal',sem(title,'')),617)
reactant(atom('iptc.666775123.b','sunshine.jpg',sem('media-reference','')),584)

  reactant(molecule([atom('iptc.666345998.b'),atom('iptc.666775123.b')]),265)
  reactant(molecule([atom('iptc.321656141.b'),atom('iptc.666775123.b')]),278)])
```

Figure 7.2: Atoms aggregating into molecules [MO12].

(self-)balanced cooperation among two mechanisms typical of self-* systems: positive and negative feedback.
Positive feedback is enacted by *MoK*reinforcement reaction, that, in this implementation,



```
out(law([atom(SRC,VAL,ATTR)], 0.1, [firing(atom(SRC,VAL,ATTR))]))
```

```
out(law([enzyme(atom("iptcID_0011",val("Treasury News", "ANMCat:TRS"),
                 sem("subject", "http://www.acmenews.com/synd/catalogs/anmcodes.xml"))),
          atom("iptcID_0011",val("Treasury News", "ANMCat:TRS"),
            sem("subject", "http://www.acmenews.com/synd/catalogs/anmcodes.xml"))],
        0.5,
        [atom("iptcID_0011",val("Treasury News", "ANMCat:TRS"),
          sem("subject", "http://www.acmenews.com/synd/catalogs/anmcodes.xml")),
        atom("iptcID_0011",val("Treasury News", "ANMCat:TRS"),
          sem("subject", "http://www.acmenews.com/synd/catalogs/anmcodes.xml"))]))
  ...
  ...
  ...
reactant(atom(iptcID_0011,val('Treasury News','ANMCat:TRS'),sem(subject,'http://www.acmenews.com/synd/catalogs/anmcodes.xml')),85)
reactant(atom(iptcID_0112,val('Game Preview','spfixt:pre-event-coverage'),
         sem(genre,'http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml')),8)
reactant(atom(iptcID_0002,fishing,sem(keyword,'')),3)
reactant(atom(iptcID_0002,'Narwhal Fishing Journal',sem(title,'')),11)
reactant(atom(iptcID_0012,val(unemployment,'medtop:20000533'),
         sem(subject,'http://www.iptc.org/std/catalog/catalog.IPTC-G2-standards_16.xml')),64)
reactant(atom(iptcID_0011,val('U.S. Treasury Department','org:5768933'),
         sem(subject,'http://www.iptc.org/std/catalog/catalog.IPTC-G2-Standards_13.xml')),70)
reactant(atom(iptcID_0001,'Fishing Industry',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/04001002')),8)
reactant(atom(iptcID_0111,val('The Sports Network','web:sportsnetwork.com'),
         sem(creator,'http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml')),17)
reactant(atom(iptcID_0012,val('Variety Of Recessionary Forces Leave Las Vegas Economy Scarred',''),sem(headline,'')),77)
```

```
out(law([enzyme(atom("iptcID_0111",val("The Sports Network", "web:sportsnetwork.com"),
                 sem("creator", "http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml"))),
          atom("iptcID_0111",val("The Sports Network", "web:sportsnetwork.com"),
            sem("creator", "http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml"))],
        0.5,
        [atom("iptcID_0111",val("The Sports Network", "web:sportsnetwork.com"),
          sem("creator", "http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml")),
        atom("iptcID_0111",val("The Sports Network", "web:sportsnetwork.com"),
          sem("creator", "http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml"))]))
  ...
  ...
  ...
reactant(atom(iptcID_0111,val('The Sports Network','web:sportsnetwork.com'),
         sem(creator,'http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml')),71)
reactant(atom(iptcID_0012,val('Variety Of Recessionary Forces Leave Las Vegas Economy Scarred',''),sem(headline,'')),16)
reactant(atom(iptcID_0011,val('Treasury News','ANMCat:TRS'),sem(subject,'http://www.acmenews.com/synd/catalogs/anmcodes.xml')),7)
reactant(atom(iptcID_0112,val('Game Preview','spfixt:pre-event-coverage'),
         sem(genre,'http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml')),87)
reactant(atom(iptcID_0112,val('Arizona Diamondbacks','team:l.mlb.com-t.26'),
         sem(subject,'http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml')),83)
reactant(atom(iptcID_0011,val('U.S. Treasury Department','org:5768933'),
         sem(subject,'http://www.iptc.org/std/catalog/catalog.IPTC-G2-Standards_13.xml')),13)
reactant(atom(iptcID_0111,val('Major League Baseball','league:l.mlb.com'),
         sem(subject,'http://www.xmlteam.com/specification/xts-SportsCodesCatalog_1.xml')),94)
reactant(atom(iptcID_0002,'Narwhal Fishing Journal',sem(title,'')),4)
reactant(atom(iptcID_0001,'Weather',sem('tobject.subject','http://cv.iptc.org/newscodes/subjectcode/17000000')),6)
reactant(atom(iptcID_0002,fishing,sem(keyword,'')),12)
```

Figure 7.3: Atoms diffusing toward interested prosumers' compartments [MO12]

takes an atom and the relative enzyme to produce two copy of the atom—thus increasing concentration by 1. Negative feedback stems from MoK decay reaction, which destroys 1 unit of atoms and molecules concentration.

The result of the competition between opposite feedbacks, is the *unstable equilibrium*

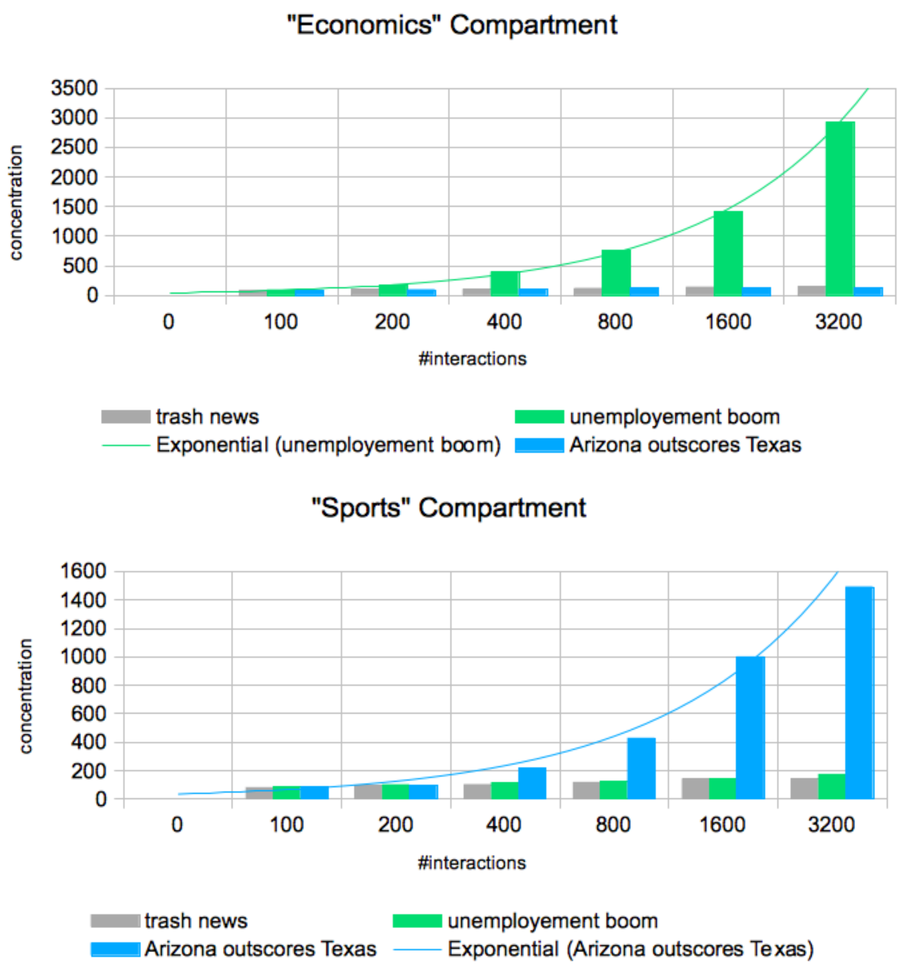


Figure 7.4: The stochastic equilibrium between diffusion, reinforcement and decay laws, makes a smart diffusion pattern appear by emergence [MO13b].

depicted in Figure 7.4, where the peripheral compartments “economics” and “sports” are mainly populated by compartment-related atoms – thanks to the positive feedback – whereas wrong knowledge chunks concentrations are maintained relatively low thanks to both decay and absence of enzymes.

As a last note, in the experiments enzyme injection as a consequence to user actions is simulated by injection laws similar to those depicted in Figure 7.1.

## 7.2 MoK Ecosystem

Besides the MoK prototype implemented upon the TuCSoN infrastructure, a comprehensive MoK *ecosystem* is currently under development[11]. The aim of this implementation effort is that of providing of *full-fledged platform* comprising not only the MoK core middleware, as described so far, but also all the satellite capabilities MoK demands, such as:

- automatic *information retrieval* and *extraction*
- automatic *semantic enrichment* and representation of unstructured text
- document-oriented persistent *storage layer* and graph-based in-memory representation layer
- networking facilities such as automatic *discovery* of compartments for dynamic reconfiguration of network topology, mechanisms for *adaptive routing* of information, etc.
- automatic *knowledge inference* and *discovery*, either ontology-based or not
- suitable *user interface* assisting knowledge inference and discovery through BIC inspired mechanisms in the spirit of [RJMK14]

For these reasons, the MoK ecosystem architecture depicted in Figure 7.5 below is envisioned, made up of the following logical layers:

**information harvesting** providing searching facilities for information retrieval, as well as text mining related techniques for information extraction—e.g., *part of speech* tagging [Abn97]

**knowledge representation & persistency** devoted to knowledge representation formats and languages, in particular, concerning technologies for building *ubiquitous knowledge bases* on resource-constrained devices [RSDR11]. Also, memorisation of data, information, and knowledge, is considered here, both persistent and in-memory—e.g., *document-oriented* DB for persistent storage and *on-memory graph* DB for run-time manipulation of MoK abstractions

**networking & communication** providing both low-level networking (discovery services, heartbeat, point-to-point and multicast data transfer, etc.) and high-level communication services (gossiping, routing, etc.)

[11]Code publicly available under LGPL license at `http://bitbucket.org/smariani/mok` and `http://bitbucket.org/smariani/mok-projects`.

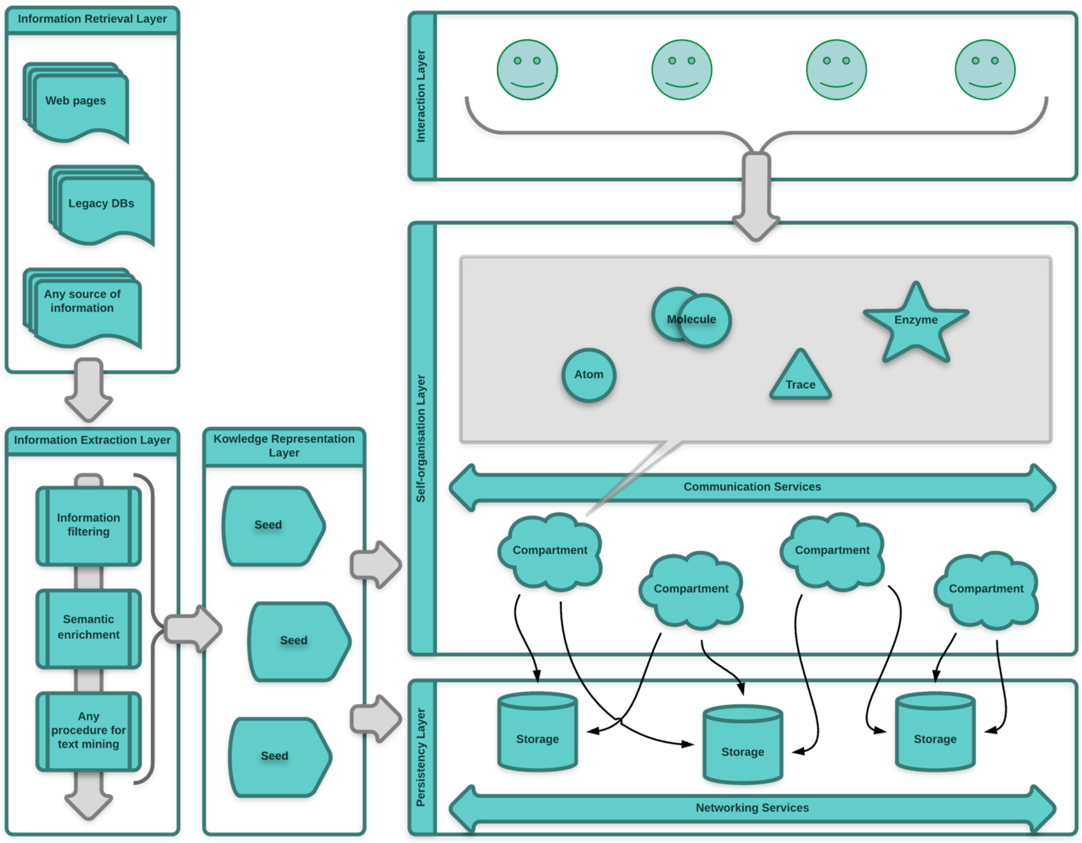


Figure 7.5: $\mathcal{M}o\mathcal{K}$ ecosystem architecture

**self-organisation** devoted to chemical-inspired self-organisation mechanisms ($\mathcal{M}o\mathcal{K}$ reactions), semantics-related aspects ($\mathcal{F}_{\mathcal{M}o\mathcal{K}}$-based matching), as well as users' interactions exploitation ($\mathcal{M}o\mathcal{K}$ perturbation actions)

**user interaction** providing to users the means to interact with the system, and to the IT platform the ability of reifying them, promoting usage for coordination purposes

In the upcoming sections, an overview of two of the software modules implementing the above layers is provided, which, although still under development, have reached the "working prototype" state—the information harvesting layer and the networking and communication layer.

Although the self-organisation layer exists in the state of a working prototype, it will not be described because it offers the same functionalities of the chemical engine implemented on top of TuCSoN in the context of the $\mathcal{M}o\mathcal{K}$ middleware prototype described in Section 7.1.

Implementation is different, namely not on top of TuCSoN, but the design rationale is similar to the mentioned prototype: for each compartment, a chemical engine is in charge of scheduling and executing $\mathcal{M}o\mathcal{K}$ reactions. In particular, w.r.t. the prototype on TuCSoN:

- reactants matching is based on $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$, considering *cosine similarity* [Hua08], *synsets* [MF98], and allowing reactant templates to specify *regular expressions* [Hab04]
- reaction rates are now mathematical expressions providing the required expressiveness for supporting the custom kinetic rates seen in Section 6.2 of Chapter 6
- firing of reactants now follows the API provided by the communication module described in Subsection 7.2.2

### 7.2.1 Information Harvesting Layer

Two software modules are currently available as working prototypes, while still under development, to offer the functionalities expected for this layer: the first one is a *distributed information retrieval system*, comprising a very simple search engine and a web crawling module; the second is an *information extraction component*, composed by a Natural Language Processing (NLP) service and a module for building $\mathcal{M}o\mathcal{K}$ atoms.

**Information retrieval system** The first component of the information retrieval system is a very simple custom *search engine* built on top of the *Google Custom Search* platform[12]. The platform lets developers build their own custom search facilities by providing API for, e.g., choosing which websites to search in, ranking results, etc.

Within the context of $\mathcal{M}o\mathcal{K}$ ecosystem prototype, the Google service is used to provide *personalised search* within the APICe website[13]. The choice is made to support deployment of $\mathcal{M}o\mathcal{K}$ demos involving academic papers collaborative management. The search service is nothing dissimilar to a normal Google search, except it only targets APICe publications space.

The second component is a *web crawler* based on *crawler4j* library[14], providing Java API for implementing a multi-threaded crawler.

Crawling is quite straightforward: given a set of pages retrieved through a keyword-based search by the above described custom search engine, the crawler starts processing them in parallel with the aim of *(i)* downloading their textual content, and *(ii)* acquiring further links to other pages containing the same keywords. Depth of the search may be configured, as well as of other performance parameters, likewise the opportunity of searching for *synsets* of the given keywords too, which is based on WordNet [MF98].

Since crawler4j library does not provide means to distributed crawlers over an infrastructure of networked hosts, a dedicated *distribution layer* has been implemented. A

[12]`http://developers.google.com/custom-search/`
[13]`http://apice.unibo.it`
[14]`http://github.com/yasserg/crawler4j`

server component waits on a well-known network address for crawlers registration. As soon as a custom search provides some results, the server components distributes the retrieved pages to the registered crawlers. Then, crawlers begin to process pages, sending to the server components the texts snippets and further links found—if related to the original search query. Finally, the server component stores the received snippets in a remote DB – for resilience reasons – and further dispatches the new links to the crawlers.

**Information extraction system** The first component of the information extraction system is a *NLP module* implemented on top of the *Apache OpenNLP* library[15]. In particular, two functionalities provided by the library are especially welcome for MoK: *part of speech* (POS) tagging and *named entity recognition*.

Part of speech tagging is useful to give a first degree of semantics to snippets of text by, e.g., recognising verbs, subjects of a dialogue, etc., based mostly on grammatical rules. Named entity recognition, instead, is useful to give another bit of semantics to unstructured text, by locating and classifying elements of text based on pre-defined *categories*, e.g., names of persons, organisations, locations, expressions of times, quantities, etc.

The component just described fetches text snippets from the information retrieval system presented in previous paragraph—from the DB fed by the server component as soon as crawlers finish processing. Then, it applies POS and named entity recognition algorithms to feed the second component of this module: the *MoK atoms builder*.

As the name suggests, the component is in charge of *(i)* implementing the data structure representing MoK atoms, as well as *(ii)* the mechanisms automatically filling this data structure based on information provided by both the NLP module – such as tagged sentences – and the information retrieval component—e.g., the URL of the web page where the text comes from.

The component as a whole is widely configurable to, e.g., generate different kinds of atoms based on NLP outcomes—e.g. person atoms, organisation atoms, sentence atoms, etc. Also, it can be *trained* to learn new tagging models for both POS and named entity recognition.

**Remark** The experiments described in Section 6.4 in previous chapter have been implemented by exploiting this module for retrieving scientific papers from the APICe online repository.

### 7.2.2 Networking & Communication Layer

As for previous layer, two software modules are currently available as working prototypes, while still under development, to offer the functionalities expected for this layer: the first one is an *asynchronous networking module* meant to provide low-level API for both data

[15]`http://opennlp.apache.org`

transfer and network topology maintenance; the second one is an *adaptive communication module* providing higher-level API meant to directly support MoK diffusion reaction implementation.

**Asynchronous network module** The networking module is developed on top of the *Netty NIO framework*[16] for *channel-based*, *event-driven asynchronous* communication.

As its name implies, Netty is an asynchronous communication framework based on Java NIO framework, which promotes an event-driven approach to networking by providing the following main components:

**channel** actually carries out communications by providing API for opening, binding to, connecting to, closing channels, as well as for reading from and writing data to channels

**channel handler** interceptor of channel operations and events

**event loop** allows management of channel operations

Each channel is registered to one event loop, which processes all events from that channel within a single thread. An event loop may serve multiple channels—one thread per each. Changes in the state of channels, as well as events generated by channels are intercepted by a dedicated channel handler.

MoK networking module is implemented on top of the above described abstractions, to provide its own, higher-level ones. In particular:

**compartment** the main abstraction of the module, it is meant to represent the communication capabilities of MoK compartments – e.g., diffusion of atoms and molecules – and to manage network-related properties of the compartment, which may change at run-time—such as its network address, its neighbourhood, wether forwarding of received packet is enabled or not, etc.

**neighbourhoods** compartments are grouped in neighbourhoods, which define the boundaries for MoK diffusion

**membrane** each compartment is associated to other in its neighbourhood by membranes, actually enabling and taking care of diffusion of atoms and molecules. Also membranes have application-specific properties to track, which may change at run-time—e.g., diffusion likelihood, distance to connected compartments, etc.

A compartment should first of all *join* a MoK system to receive its neighbourhood in the form of a set of membranes. Then, in order to communicate with the compartments in its neighbourhood, it utilises the appropriate membranes, which provide methods for

[16]`http://netty.io/index.html`

*point-to-point*, *multicast*, and *broadcast* data transfer. Each data transfer is marked by a *globally unique ID*, enabling implementation of *retransmission* mechanisms, and necessary to avoid *flooding* the network when compartments forward received atoms and molecules.

Each networking operation within the framework is asynchronous and follows a call-back model, thus, any compartment invokes operations with a "fire-and-forget" semantics, to be later notified upon their completion—either successful or not. Accordingly, notifications for compartment joining the neighbourhood, reception of atoms and molecules, failures, etc. are available.

The networking module also includes *hotspot* components, meant to support *openness* of the MoK system, and network *topology* (re)configuration and maintenance. Hotspots are passive components reachable on a well-known address, on which they listen to incoming connection requests by MoK compartments.

As soon as the request arrives, the hotspot computes the neighbourhood the newcomer should be assigned to, based on the properties of the requesting compartment – e.g., its geographical location, average latency, application-specific preferences – and its own configured *policies* for neighbourhood association. Then, it sends the neighbourhoods to the joining compartment, and notifies the compartments therein about the newcomer.

It should be noted that joining the MoK system by contacting multiple hotspots is perfectly fine, and enables complex topologies to be built, since each hotspots may have different neighbourhood association policies and different *partial views* of the network.

Hotspots are also in charge of *gracefully* releasing connections when compartments leave the MoK system, that is, of informing compartments in the neighbourhood about the leaving one. However, even in the case of an abrupt disconnection, e.g. due to a crash, the system is guaranteed to keep a consistent (partial) view over the network: as soon as compartments attempt to communicate with the no longer available peer, disconnection is detected and promptly communicated to the hotspot, which propagates information to the interested compartments—those in the neighbourhood of the leaving one.

Besides handling join/leave of compartments, and topology-related aspects, hotspots take no role in communication between compartments. In fact, after the join phase, communication among compartments is *completely p2p*.

**Adaptive communication module** The communication module is developed on top of the networking module just described, thus, exploits its abstractions and services to provide *direct support* to MoK communication requirements. In particular, the module provides an API to *(i)* carry out *MoK diffusion* – that is, sharing of atoms, molecules, or traces between compartments in the same neighbourhood – according to a few different modalities, and *(ii)* undertake *search actions* to seek for information within the whole space of networked MoK compartments, despite the actual compartment where the action takes place.

As regards the search action[17], two modalities are supported:

- one takes as input parameter a set of *keywords* and searches matching information by exploiting the $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ function, here implemented as the *cosine similarity* measure [Hua08] extended to consider *sysnsets* [MF98]—many other parameters can be specified, e.g., whether to restrict searching to the current neighbourhood

- one takes as input parameter an atom or a molecule, and searches information matching that piece of information—still based on $\mathcal{F}_{\mathcal{M}o\mathcal{K}}$ similarity measure

Whichever is the chosen modality, a few features are supported by the current implementation which are worth to be mentioned:

- whenever a compartment receives a search request, regardless of the fact it can provide matching information or not, it tracks that the requesting compartment is looking for information matching the given keywords / atom / molecule

- whenever the compartment where the search action took place receives a reply from a remote compartment, within its neighbourhood or not[18], it tracks that the compartment has matching information (for that keywords / atom / molecule)

- whenever a compartment receives a search request, regardless of the fact it can provide matching information or not, it forwards the request to its neighbours—flooding is automatically avoided by the underlying networking module described in previous paragraph

- replies to search actions imply movement of a fraction of the matching atoms and molecules concentration, not their copy

As regards $\mathcal{M}o\mathcal{K}$ diffusion, four modalities for transmitting information are currently supported:

**random** the diffusing item (atom, molecules, or trace) is sent to a neighbour compartment chosen at random, each compartment having the same probability

**probabilistic** the diffusing item is sent to a neighbour compartment chosen at random, each compartment having a probability depending on properties of the membrane connecting it to the sender compartment—e.g., their distance, according to application-specific metrics

[17]It should be noted that this is not the kind of search for information sources ($\mathcal{M}o\mathcal{K}$ seeds) described in Subsection 7.2.1, but the `harvest` action modelling $\mathcal{M}o\mathcal{K}$ catalysts behaviour (thus one of those described in Section 6.3), aimed at acquiring information already present within the $\mathcal{M}o\mathcal{K}$ ecosystem.

[18]In $\mathcal{M}o\mathcal{K}$, actions are reified into enzymes which release traces, that, in this case, are meant to spread the search through neighbourhoods recursively.

**broadcast** the diffusing item is sent to all the compartments in the neighbourhood

**focussed** the diffusing item is sent to a specific compartment

Both diffusion and search generate *gradients*, which are used to route *(i)* search replies, *(ii)* further search requests, and *(iii)* focussed diffusions. More in general, forwarding of data packets is implemented on top of gradients *dynamically created* by the above described actions, providing *adaptive routing lists* to every networked compartment.

Routing lists are data structures tracking compartments connecting a source compartment to a destination compartment. Routing lists are incrementally and cooperatively filled by the compartments themselves, which add their reference to the list prior to passing it while forwarding data packets.

Routing lists may provide more alternatives to reach a given destination: strictly worse alternatives are eliminated – e.g., those with more compartments to traverse – whereas equivalent ones are chosen probabilistically, so as to support *graceful degradation* of routing performance in case of topology changes—e.g., compartments disconnections breaking a routing path.

Spreading of search requests is actually implemented on top of diffusion, where the items diffused are not atoms nor molecules, but traces (deposited by the `harvest` enzyme, in turn released by the homonym action).

All the features described so far, are meant to cooperatively support *adaptive communications* within *MoK*. Compartments track each other requests to improve their performance, e.g., by attracting information which is more frequently requested by neighbour compartments. Complementarily, compartments track each other replies to improve their efficiency, e.g., by autonomously switching diffusion modality to focussed if a compartment becomes recognised as being the best source for a given kind of information.

# Part III

# Conclusion & Outlook

# Chapter 8

# Conclusion

In this thesis, coordination issues posed by knowledge-intensive socio-technical systems have been deeply analysed, and novel approaches to deal with them have been accordingly proposed.

In particular, a chemical-inspired approach to coordination for self-organising systems has been proposed and evaluated, which relies on artificial chemical reactions with custom kinetic rates as coordination laws. An architectural and linguistic approach to coordination for situated pervasive systems has been proposed and thoroughly described, which relies on a situated, distributed, programmable, and tuple-based coordination medium and language to implement situated coordination laws. An approach to user-driven coordination for socio-technical systems has been proposed and described, inspired to behavioural implicit communication, and relying on the notions of tacit message and perturbation action to drive the coordination processes enacted by the coordination medium.

As a summation of the above proposals, the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge model and technology for self-organisation of knowledge in knowledge-intensive socio-technical systems has been defined, designed, implemented, and evaluated.

The rationale driving the apparently distant research efforts above described has been the following: engineering efficient and effective coordination for large-scale, data-intensive systems with "humans in the loop" is a very difficult task, which should be approached in a holistic way by considering both the model, the architecture, and the language level of a potential computational solution. The choice to start the research work from nature-inspired coordination models, in particular, chemical-inspired ones, is motivated by their well-proven ability to deal with distribution, decentralisation, unpredictability, and scale in a simple yet expressive way. Then, the need for a suitable coordination infrastructure supporting chemical-inspired approaches naturally stems from such a choice, while the need to readily account for users behaviour since the very foundation of the proposed coordination framework stems from the kind of systems mainly targeted by this thesis—that is, socio-technical ones.

Accordingly, the main contributions brought by this thesis to advance the field of coordination models, languages, and infrastructures, in the context of socio-technical systems, are:

- a novel coordination model for decentralised, data-driven coordination, based on a chemical metaphor for both representing and enacting coordination laws, and on the behavioural implicit communication framework for steering the emergent self-organisation toward the ever-changing users goals
- a novel infrastructure and language supporting design and implementation of the aforementioned model, and in general of situated coordination policies within multi-agent systems
- a novel software ecosystem providing not only the core coordination functionalities of the aforementioned model, but also those satellite services necessary for a real-world deployment—such as information harvesting components, networking services, storage.

# Chapter 9

# Outlook

In present IT systems, it is apparent the need for efficient and smart ways of preserving, managing, and analysing the astonishing amount of raw data, derived information, and high-level knowledge that any socio-technical system produces everyday.

Big data like approaches are more or less the state of art, mostly because they are good in finding hidden patterns in the data they are fed with. Nevertheless, they mostly neglect "humans in the loop", being the role of human users confined to analysis of ready-to-use results, produced by algorithms which are completely user-neutral and goal-independent. They also require ever-increasing computational power to scale up with the complexity of the problem at hand, which is something not all the organisations can afford.

For the above reasons, it may be appropriate to promote a paradigm shift toward self-organisation of information and knowledge, where:

- user-driven adaptability of information mining and knowledge discovery activities is the main concern to deal with, and the foremost goal to pursue
- techniques supporting the mentioned activities natively account for users goals, as well as take advantage of the properties of the business domain at hand
- the overall performance seamlessly scales up and down naturally, as a result of exploitation of local information and fully decentralised mechanisms, rather than of *post-hoc* technical solutions

All the above is witnessed by the latest H2020 calls for proposals, increasingly demanding citizen-inclusive policy making, governance participation, user-centric knowledge sharing platforms, and the like—see, e.g., calls H2020-SC6-CO-CREATION-2016-2017, H2020-EINFRA-2016-2017, and H2020-FETPROACT-2016-2017.

In particular, as far as the contributions of this thesis are concerned, many advancements may still be made, and will certainly be the subject of the author's future research activity:

- simulation of composite bio-inspired patterns is undoubtedly interesting, both to evaluate their expressiveness, and to evaluate the extent to which artificial chemical reactions are composable
- further formal investigation of uniform primitives expressiveness, too, is interesting for better framing the relative expressiveness of probabilistic coordination languages
- improvement of $\mathcal{M}o\mathcal{K}$ prototype on TuCSoN is already under development, in particular as far as the chemical engine is concerned
- instantiation of the $\mathcal{M}o\mathcal{K}$ model within different business domains, as well as deployment of $\mathcal{M}o\mathcal{K}$ prototype therein surely helps in assessing its generality
- development of the missing parts of $\mathcal{M}o\mathcal{K}$ ecosystem is already in process, and represents a necessary ingredient for real-world deployment of the $\mathcal{M}$olecules $o$f $\mathcal{K}$nowledge technology

# Bibliography

[AAS06] Rui Alves, Fernando Antunes, and Armindo Salvador. Tools for kinetic modeling of biochemical networks. *Nature biotechnology*, 24(6):667–672, 2006.

[Abn97] S. Abney. *Corpus-Based Methods in Language and Speech Processing*, chapter Part-of-Speech Tagging and Partial Parsing, pages 118–136. Springer Netherlands, Dordrecht, 1997.

[ACC+98] James Allan, Jamie Callan, W Bruce Croft, Lisa Ballesteros, Donald Byrd, Russell Swan, and Jinxi Xu. Inquery does battle with trec-6. *NIST SPECIAL PUBLICATION SP*, pages 169–206, 1998.

[Ack00] Mark S Ackerman. The intellectual challenge of cscw: the gap between social requirements and technical feasibility. *Human–Computer Interaction*, 15(2-3):179–203, 2000.

[BB06] Jacob Beal and Jonathan Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *Intelligent Systems, IEEE*, 21(2):10–19, 2006.

[BDPW02] Antonio Brogi, Alessandra Di Pierro, and Herbert Wiklicky. Linear embedding for a quantitative comparison of language expressiveness. *Electronic Notes in Theoretical Computer Science*, 59(3):207–237, 2002.

[BDU+12] Jacob Beal, Stefan Dulman, Kyle Usbeck, Mirko Viroli, and Nikolaus Correll. Organizing the aggregate: Languages for spatial computing. *CoRR*, abs/1202.5509, 2012.

[Bea10] Jacob Beal. A basis set of operators for space-time computations. In *Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 91–97, Washington, DC, USA, 2010. IEEE Computer Society.

[BGLZ05] Mario Bravetti, Roberto Gorrieri, Roberto Lucchi, and Gianluigi Zavattaro. Quantitative information in the tuple space coordination model. *Theoretical Computer Science*, 346(1):28–57, 23 November 2005.

[BGZ00] Nadia Busi, Roberto Gorrieri, and Gianluigi Zavattaro. On the expressiveness of linda coordination primitives. *Information and Computation*, 156(1):90–121, 2000.

[Bha01] Ganesh D. Bhatt. Knowledge management in organizations: Examining the interaction between technologies, techniques, and people. *Journal of Knowledge Management*, 5(1):68–75, 2001.

[BHW07] Rafael H. Bordini, Jomi F. Hübner, and Michael J. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason.* John Wiley & Sons, Ltd, October 2007.

[BLY01] Jonsson Bengt, Kim G. Larsen, and Wang Yi. Probabilistic extensions of process algebras. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, chapter 11, pages 685–710. Elsevier Science B.V., 2001.

[BMS11] Jacob Beal, Olivier Michel, and Ulrik Pagh Schultz. Spatial computing: Distributed systems that take advantage of our geometric world. *ACM Trans. Auton. Adapt. Syst.*, 6(2):11:1–11:3, June 2011.

[BPR99] Fabio Luigi Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE–a FIPA-compliant agent framework. In *4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-99)*, pages 97–108, London, UK, 19–21 April 1999. The Practical Application Company Ltd.

[Bra08] Mario Bravetti. Expressing priorities and external probabilities in process algebra via mixed open/closed systems. *Electronic Notes in Theoretical Computer Science*, 194(2):31–57, 16 January 2008.

[Bro86] Rodney A. Brooks. Achieving artificial intelligence through building robots. Technical Report AIM-899, Massachussets Institute of Technology (MIT), May 1986.

[Car08] Luca Cardelli. On process rate semantics. *Theoretical computer science*, 391(3):190–215, 2008.

[Cas98] Cristiano Castelfranchi. Modelling social action for AI agents. *Artificial Intelligence*, 103(1-2):157–182, August 1998.

[Cas06] C Castlefranchi. From conversation to interaction via behavioral communication: For a semiotic design of objects, environments, and behaviors. *Theories and practice in interaction design*, pages 157–79, 2006.

[Cas12] Cristiano Castelfranchi. Goals, the true center of cognition. In Fabio Paglieri, Luca Tummolini, Rino Falcone, and Maria Miceli, editors, *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, volume 20 of *Tributes*, chapter 41, pages 837–882. College Publications, London, December 2012.

[CC95] Rosaria Conte and Cristiano Castelfranchi, editors. *Cognitive and Social Action*. Routledge, 1995.

[CH09] Federica Ciocchetta and Jane Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33–34):3065 – 3084, 2009. Concurrent Systems Biology: To Nadia Busi (1968–2007).

[Cia96] Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2):300–302, June 1996.

[CLZ00] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. MARS: A programmable coordination architecture for mobile agents. *IEEE Internet Computing*, 4(4):26–35, July/August 2000.

[CM01] Adam Cheyer and David Martin. The Open Agent Architecture. *Autonomous Agents and Multi-Agent Systems*, 4(1-2):143–148, March 2001.

[CO09] Matteo Casadei and Andrea Omicini. Situated tuple centres in ReSpecT. In Sung Y. Shin, Sascha Ossowski, Ronaldo Menezes, and Mirko Viroli, editors, *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, volume III, pages 1361–1368, Honolulu, Hawai'i, USA, 8–12 March 2009. ACM.

[COZ00] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Multiagent system engineering: The coordination viewpoint. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *LNAI*, pages 250–259. Springer, 2000.

[CPT10] Cristiano Castelfranchi, Giovanni Pezzullo, and Luca Tummolini. Behavioral implicit communication (BIC): Communicating with smart environments via our practical behavior and its traces. *International Journal of Ambient Computing and Intelligence*, 2(1):1–12, January–March 2010.

[CV13] Matteo Casadei and Mirko Viroli. Toward approximate stochastic model checking of computational fields for pervasive computing systems. In Jeremy Pitt, editor, *Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, pages 199–204. IEEE CS, April 2013. 2012 IEEE Sixth International Conference (SASOW 2012), Lyon, France, 10-14 September 2012. Proceedings.

[CVG09] Matteo Casadei, Mirko Viroli, and Luca Gardelli. On the collective sort problem for distributed tuple spaces. *Science of Computer Programming*, 74(9):702–722, 2009.

[DB10] Marco Dorigo and Mauro Birattari. Ant colony optimization. In Claude Sammut and GeoffreyI. Webb, editors, *Encyclopedia of Machine Learning*, pages 36–39. Springer US, 2010.

[dBP94] Frank S. de Boer and Catiuscia Palamidessi. Embedding as a tool for language comparison. *Information and Computation*, 108(1):128–157, 1994.

[DC15] Enrico Denti and Roberta Calegari. Butler-ising HomeManager: A pervasive multi-agent system for home intelligence. In Stephane Loiseau, Joaquim Filipe, Beatrice Duval, and Jaap Van Den Herik, editors, *7th International Conference on Agents and Artificial Intelligence 2015 (ICAART 2015)*, pages 249–256, Lisbon, Portugal, 10–12 January 2015. SCITEPRESS – Science and Technology Publications.

[Den14] Enrico Denti. Novel pervasive scenarios for home management: the butlers architecture. *SpringerPlus*, 3(52):1–30, January 2014.

[DM91] Yves Demazeau and Jean-Pierre Müller. From reactive to intentional agents. *Decentralized A.I.*, 2:3–10, 1991.

[DMY$^+$09] Menggao Dong, Xinjun Mao, Junwen Yin, Zhiming Chang, and Zhichang Qi. Sade: A development environment for adaptive multi-agent systems. In Jung-Jin Yang, Makoto Yokoo, Takayuki Ito, Zhi Jin, and Paul Scerri, editors, *Principles of Practice in Multi-Agent Systems*, volume 5925 of *Lecture Notes in Computer Science*, pages 516–524. Springer Berlin Heidelberg, 2009.

[DNLKM06] Rocco De Nicola, Diego Latella, Joost-Pieter Katoen, and Mieke Massink. StoKlaim: A stochastic extension of Klaim. Technical Report 2006-TR-01, Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" (ISTI), 2006.

[DNO97] Enrico Denti, Antonio Natali, and Andrea Omicini. Programmable coordination media. In David Garlan and Daniel Le Métayer, editors, *Coordination Languages and Models*, volume 1282 of *LNCS*, pages 274–288. Springer-Verlag, 1997.

[DNO98] 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.

[DOR01] Enrico Denti, Andrea Omicini, and Alessandro Ricci. tuProlog: A light-weight Prolog for Internet applications and infrastructures. In I.V. Ramakrishnan, editor, *Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 184–198. Springer, 2001. 3rd International Symposium (PADL 2001), Las Vegas, NV, USA, 11–12 March 2001. Proceedings.

[DPHW03] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Quantitative relations and approximate process equivalences. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 498–512. Springer, 2003.

[DPHW04] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic KLAIM. In Rocco De Nicola, Gian-Luigi Ferrari, and Greg Meredith, editors, *Coordination Models and Languages*, volume 2949 of *LNCS*, pages 119–134. Springer Berlin / Heidelberg, 2004.

[DPHW05] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic Linda-based coordination languages. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *3rd International Conference on Formal Methods for Components and Objects (FMCO'04)*, volume 3657 of *LNCS*, pages 120–140. Springer, Berlin, Heidelberg, 2005.

[Dra67] Alvin W. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill College, 1967.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, July 2004.

[DWH07] Tom De Wolf and Tom Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In *Engineering Self-Organising Systems*, pages 28–49. Springer, 2007.

[FGMR10] Giancarlo Fortino, Alfredo Garro, Samuele Mascillaro, and Wilma Russo. Using event-driven lightweight DSC-based agents for MAS modelling. *International Journal of Agent-Oriented Software Engineering*, 4(2):113–140, April 2010.

[FM96] Jacques Ferber and Jean-Pierre Müller. Influences and reaction: A model of situated multiagent systems. In Mario Tokoro, editor, *2nd International Conference on Multi-Agent Systems (ICMAS-96)*, pages 72–79, Tokio, Japan, December 1996. AAAI Press.

[FMDMSA11] Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, and Josep Lluis Arcos. Infrastructureless spatial storage algorithms. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 6(2):15, 2011.

[FMMSM$^+$12] JoseLuis Fernandez-Marquez, Giovanna Marzo Serugendo, Sara Montagna, Mirko Viroli, and JosepLluis Arcos. Description and composition of bio-inspired design patterns: a complete overview. *Natural Computing*, pages 1–25, 2012.

[FMSM12] Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, and Sara Montagna. Bio-core: Bio-inspired self-organising mechanisms core. In *Bio-Inspired Models of Networks, Information, and Computing Systems*, volume 103 of *Lecture*

*Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 59–72. Springer Berlin Heidelberg, 2012.

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

[GH94] Stephen Gilmore and Jane Hillston. The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In Günter Haring and Gabriele Kotsis, editors, *Computer Performance Evaluation Modelling Techniques and Tools*, volume 794 of *Lecture Notes in Computer Science*, pages 353–368. Springer Berlin Heidelberg, 1994.

[Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, December 1977.

[Gra59] Pierre-Paul Grassé. La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80, March 1959.

[GVCO07] Luca Gardelli, Mirko Viroli, Matteo Casadei, and Andrea Omicini. Designing self-organising MAS environments: The collective sort case. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 254–271. Springer, May 2007.

[GVO06] Luca Gardelli, Mirko Viroli, and Andrea Omicini. On the role of simulations in engineering self-organising MAS: The case of an intrusion detection system in TuCSoN. In Sven A. Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, editors, *Engineering Self-Organising Systems*, volume 3910 of *LNAI*, pages 153–168. Springer, 2006. 3rd International Workshop (ESOA 2005), Utrecht, The Netherlands, 26 July 2005. Revised Selected Papers.

[GVO09] Luca Gardelli, Mirko Viroli, and Andrea Omicini. Combining simulation and formal tools for developing self-organizing MAS. In Adelinde M. Uhrmacher and Danny Weyns, editors, *Multi-Agent Systems: Simulation and Applications*, Computational Analysis, Synthesis, and Design of Dynamic Systems, chapter 5, pages 133–165. CRC Press, June 2009.

[Hab04] Mehran Habibi. *Java regular expressions: taming the java. util. regex engine.* Springer, 2004.

[Her02] Holger Hermanns. *Interactive Markov chains: and the quest for quantified quality.* Springer-Verlag, 2002.

[HJD07] Kasper Hallenborg, Ask Just Jensen, and Yves Demazeau. Reactive agent mechanisms for manufacturing process control. In *2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology Workshops (WI-IATW '07)*, pages 399–403, Washington, DC, USA, 2007. IEEE Computer Society.

[HKBR99] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999.

[HNP05] Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. In *Ldv Forum*, volume 20, pages 19–62, 2005.

[HP01] Oltea Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous pi-calculus. *CoRR*, cs.PL/0109002, 2001.

[HS96] Scott E Hudson and Ian Smith. Techniques for addressing fundamental privacy and disruption tradeoffs in awareness support systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 248–257. ACM, 1996.

[Hua08] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

[Hut95] Edwin Hutchins. *Cognition in the Wild.* MIT press, 1995.

[JS13] Ms Aruna Jadhav and Subhash K Shinde. A concept based mining model for nlp using text clustering. In *International Journal of Engineering Research and Technology*, volume 2. ESRSA Publications, 2013.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[Leu01] Anton Leuski. Evaluating document clustering for interactive information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 33–40. ACM, 2001.

[LHKK96] Krista Lagus, Timo Honkela, Samuel Kaski, and Teuvo Kohonen. Self-organizing maps of document collections: A new approach to interactive exploration. In *KDD*, volume 96, pages 238–243, 1996.

[MAC$^+$07] Emanuela Merelli, Giuliano Armano, Nicola Cannata, Flavio Corradini, Mark d'Inverno, Andreas Doms, Phillip Lord, Andrew Martin, Luciano Milanesi, Steffen Möller, Michael Schroeder, and Michael Luck. Agents in bioinformatics, computational and systems biology. *Briefings in Bioinformatics*, 8(1):45–59, 2007.

[Mar13a] Stefano Mariani. Analysis of the Molecules of Knowledge model with the Bio-PEPA Eclipse plugin. AMS Acta Technical Report 3783, Alma Mater Studiorum–Università di Bologna, Bologna, Italy, 20 September 2013.

[Mar13b] Stefano Mariani. Parameter engineering vs. parameter tuning: the case of biochemical coordination in MoK. In Matteo Baldoni, Cristina Baroglio, Federico Bergenti, and Alfredo Garro, editors, *From Objects to Agents*, volume 1099 of *CEUR Workshop Proceedings*, pages 16–23, Turin, Italy, 2–3 December 2013. Sun SITE Central Europe, RWTH Aachen University. XIV Workshop (WOA 2013). Workshop Notes.

[Mar14] Stefano Mariani. On the "local-to-global" issue in self-organisation: Chemical reactions with custom kinetic rates. In *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014*, Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, pages 61 – 67, London, UK, September 2014. IEEE. Best student paper award.

[MAYA01] Gregoris Mentzas, Dimitris Apostolou, Ronald Young, and Andreas Abecker. Knowledge networking: a holistic solution for leveraging corporate knowledge. *Journal of knowledge management*, 5(1):94–107, 2001.

[MC94] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[MF98] George Miller and Christiane Fellbaum. Wordnet: An electronic lexical database, 1998.

[MMTZ06] Marco Mamei, Ronaldo Menezes, Robert Tolksdorf, and Franco Zambonelli. Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52(8):443–460, 2006.

[MO12] Stefano Mariani and Andrea Omicini. Self-organising news management: The *Molecules of Knowledge* approach. In Jeremy Pitt, editor, *Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, pages 235–240. IEEE CS, 2012. 2012 IEEE Sixth International Conference (SASOW 2012), Lyon, France, 10-14 September 2012. Proceedings.

[MO13a] Stefano Mariani and Andrea Omicini. Event-driven programming for situated MAS with ReSpecT tuple centres. In Matthias Klusch, Matthias Thimm, and

Marcin Paprzycki, editors, *Multiagent System Technologies*, volume 8076 of *LNAI*, pages 306–319. Springer, 2013. 11th German Conference (MATES 2013), Koblenz, Germany, 16-20 September 2013. Proceedings.

[MO13b] Stefano Mariani and Andrea Omicini. MoK: Stigmergy meets chemistry to exploit social actions for coordination purposes. In Harko Verhagen, Pablo Noriega, Tina Balke, and Marina de Vos, editors, *Social Coordination: Principles, Artefacts and Theories (SOCIAL.PATH)*, pages 50–57, AISB Convention 2013, University of Exeter, UK, 3–5 April 2013. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour.

[MO13c] Stefano Mariani and Andrea Omicini. Molecules of Knowledge: Self-organisation in knowledge-intensive environments. In Giancarlo Fortino, Costin Bădică, Michele Malgeri, and Rainer Unland, editors, *Intelligent Distributed Computing VI*, volume 446 of *Studies in Computational Intelligence*, pages 17–22. Springer, 2013.

[MO13d] Stefano Mariani and Andrea Omicini. Probabilistic embedding: Experiments with tuple-based probabilistic languages. In *28th ACM Symposium on Applied Computing (SAC 2013)*, pages 1380–1382, Coimbra, Portugal, 18–22 March 2013. Poster Paper.

[MO13e] Stefano Mariani and Andrea Omicini. Probabilistic modular embedding for stochastic coordinated systems. In Christine Julien and Rocco De Nicola, editors, *Coordination Models and Languages*, volume 7890 of *LNCS*, pages 151–165. Springer, 2013. 15th International Conference (COORDINATION 2013), Florence, Italy, 3–6 June 2013. Proceedings.

[MO13f] Stefano Mariani and Andrea Omicini. Promoting space-aware coordination: ReSpecT as a spatial-computing virtual machine. In *Spatial Computing Workshop (SCW 2013)*, AAMAS 2013, Saint Paul, Minnesota, USA, May 2013.

[MO13g] Stefano Mariani and Andrea Omicini. Space-aware coordination in ReSpecT. In Matteo Baldoni, Cristina Baroglio, Federico Bergenti, and Alfredo Garro, editors, *From Objects to Agents*, volume 1099 of *CEUR Workshop Proceedings*, pages 1–7, Turin, Italy, 2–3 December 2013. Sun SITE Central Europe, RWTH Aachen University. XIV Workshop (WOA 2013). Workshop Notes.

[MO13h] Stefano Mariani and Andrea Omicini. TuCSoN on cloud: An event-driven architecture for embodied / disembodied coordination. In Rocco Aversa, Joanna Kolodzej, Jun Zhang, Flora Amato, and Giancarlo Fortino, editors, *Algorithms and Architectures for Parallel Processing*, volume 8286 of *LNCS*, pages 285–294. Springer International Publishing Switzerland, December 2013. 13th International Conference (ICA3PP-2013), Vietri sul Mare, Italy, 18-20 December 2013. Proceedings, Part II.

[MO14a] Stefano Mariani and Andrea Omicini. Coordination in situated systems: Engineering mas environment in TuCSoN. In Giancarlo Fortino, Giuseppe Di Fatta, Wenfeng Li, Sergio Ochoa, Alfredo Cuzzocrea, and Mukaddim Pathan, editors, *Internet and Distributed Computing Systems*, volume 8729 of *Lecture Notes in Computer Science*, pages 99–110. Springer International Publishing, September 2014. 7th International Conference on Internet and Distributed Computing Systems (IDCS 2014), Calabria, Italy, 22-24 September 2014, Proceedings.

[MO14b] Stefano Mariani and Andrea Omicini. Coordination mechanisms for the modelling and simulation of stochastic systems: The case of uniform primitives. *SCS M&S Magazine*, IV:6–25, December 2014. Special Issue on "Agents and Multi-Agent Systems: From Objects to Agents".

[MO14c] Stefano Mariani and Andrea Omicini. TuCSoN coordination for MAS situatedness: Towards a methodology. In Corrado Santoro and Federico Bergenti, editors, *WOA 2014 – XV Workshop Nazionale "Dagli Oggetti agli Agenti"*, volume 1260 of *CEUR Workshop Proceedings*, pages 62–71, Catania, Italy, 24–26 September 2014. Sun SITE Central Europe, RWTH Aachen University.

[MO15a] Stefano Mariani and Andrea Omicini. Anticipatory coordination in socio-technical knowledge-intensive environments: Behavioural implicit communication in MoK. In Marco Gavanelli, Evelina Lamma, and Fabrizio Riguzzi, editors, *AI*IA 2015, Advances in Artificial Intelligence*, volume 9336 of *Lecture Notes in Computer Science*, chapter 8, pages 102–115. Springer International Publishing, 23–25 September 2015. XIVth International Conference of the Italian Association for Artificial Intelligence, Ferrara, Italy, September 23–25, 2015, Proceedings.

[MO15b] Stefano Mariani and Andrea Omicini. Coordinating activities and change: An event-driven architecture for situated MAS. *Engineering Applications of Artificial Intelligence*, 41:298–309, May 2015. Special Section on Agent-oriented Methods for Engineering Complex Distributed Systems.

[MU00] Naftaly H. Minsky and Victoria Ungureanu. Law-Governed interaction: A coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(3):273–305, 2000.

[MZ09] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications: The TOTA approach. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 18(4):15:1–15:56, July 2009.

[Nag04] Radhika Nagpal. A catalog of biologically-inspired primitives for engineering self-organization. In *Engineering Self-Organising Systems*, pages 53–62. Springer, 2004.

[Nar96] B.A. Nardi. *Context and Consciousness: Activity Theory and Human-computer Interaction.* MIT Press, 1996.

[NM09] Cynthia Nikolai and Gregory Madey. Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2, 2009.

[NOV11] Elena Nardini, Andrea Omicini, and Mirko Viroli. Description spaces with fuzziness. In Mathew J. Palakal, Chih-Cheng Hung, William Chu, and W. Eric Wong, editors, *26th Annual ACM Symposium on Applied Computing (SAC 2011)*, volume II: Artificial Intelligence & Agents, Information Systems, and Software Development, pages 869–876, Tunghai University, TaiChung, Taiwan, 21–25 March 2011. ACM.

[OBS96] Vicki L O'Day, Daniel G Bobrow, and Mark Shirley. The social-technical design circle. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 160–169. ACM, 1996.

[OD01a] Andrea Omicini and Enrico Denti. Formal ReSpecT. *Electronic Notes in Theoretical Computer Science*, 48:179–196, June 2001.

[OD01b] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.

[Ode02] James J. Odell. Objects and agents compared. *Journal of Object Technology*, 1(1):41–53, May–June 2002.

[OM13] Andrea Omicini and Stefano Mariani. Coordination for situated MAS: Towards an event-driven architecture. In Daniel Moldt and Heiko Rölke, editors, *International Workshop on Petri Nets and Software Engineering (PNSE'13)*, volume 989 of *CEUR Workshop Proceedings*, pages 17–22. Sun SITE Central Europe, RWTH Aachen University, 6 August 2013.

[Omi02] Andrea Omicini. Towards a notion of agent coordination context. In Dan C. Marinescu and Craig Lee, editors, *Process Coordination and Ubiquitous Computing*, chapter 12, pages 187–200. CRC Press, Boca Raton, FL, USA, October 2002.

[Omi07] Andrea Omicini. Formal ReSpecT in the A&A perspective. *Electronic Notes in Theoretical Computer Science*, 175(2):97–117, June 2007.

[Omi12] Andrea Omicini. Agents writing on walls: Cognitive stigmergy and beyond. In Fabio Paglieri, Luca Tummolini, Rino Falcone, and Maria Miceli, editors, *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, volume 20 of *Tributes*, chapter 29, pages 543–556. College Publications, London, December 2012.

[Omi13a] Andrea Omicini. Nature-inspired coordination for complex distributed systems. In *Intelligent Distributed Computing VI*, pages 1–6. Springer, 2013.

[Omi13b] Andrea Omicini. Nature-inspired coordination models: Current status, future trends. *ISRN Software Engineering*, 2013, 2013.

[OO02] Sascha Ossowski and Andrea Omicini. Coordination knowledge engineering. *The Knowledge Engineering Review*, 17(4):309–316, December 2002.

[OO03] Andrea Omicini and Sascha Ossowski. Objective versus subjective coordination in the engineering of agent systems. In Matthias Klusch, Sonia Bergamaschi, Peter Edwards, and Paolo Petta, editors, *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*, pages 179–202. Springer, 2003.

[OPRV09] Andrea Omicini, Michele Piunti, Alessandro Ricci, and Mirko Viroli. Agents, intelligence, and tools. In Max Bramer, editor, *Artificial Intelligence: An International Perspective*, volume 5640 of *LNAI: State-of-the-Art Survey*, chapter 9, pages 157–173. Springer, 2009.

[Orl92] Wanda J Orlikowski. The duality of technology: Rethinking the concept of technology in organizations. *Organization science*, 3(3):398–427, 1992.

[ORV$^+$04a] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. 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 286–293, New York, USA, 19–23 July 2004. ACM.

[ORV$^+$04b] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Marco Cioffi, and Giovanni Rimassa. Multi-agent infrastructures for objective and subjective coordination. *Applied Artificial Intelligence*, 18(9-10):815–831, 2004.

[ORV05] 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.

[ORV08] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, December 2008.

[ORZ06] 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.

[OV11] Andrea Omicini and Mirko Viroli. Coordination models and languages: From parallel computing to self-organisation. *The Knowledge Engineering Review*, 26(1):53–59, March 2011.

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

[OZ04] Andrea Omicini and Franco Zambonelli. MAS as complex systems: A view on the role of declarative approaches. In João Alexandre Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni, editors, *Declarative Agent Languages and Technologies*, volume 2990 of *LNAI*, pages 1–17. Springer, May 2004. 1st International Workshop (DALT 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.

[OZKT01] Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications.* Springer-Verlag, March 2001.

[Par97] H. Van Dyke Parunak. "Go to the ant": Engineering principles from natural agent systems. *Annals of Operation Research*, 75(0):69–101, January 1997. Special Issue on Artificial Intelligence and Management Science.

[Par06] H. Van Dyke Parunak. A survey of environments and mechanisms for human-human stigmergy. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for Multi-Agent Systems II*, volume 3830 of *LNCS*, pages 163–186. Springer, 2006.

[PBS02] 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*, volume 1, pages 449–450, New York, NY, USA, 15–19 July 2002. ACM.

[PCBB07] Julien Pauty, Paul Couderc, Michel Banatre, and Yolande Berbers. Geo-Linda: a geometry aware distributed tuple space. In *Advanced Information Networking and Applications*, pages 370–377, 2007. 21st International Conference (AINA '07), 21–23 May 2007, Niagara Falls, ON, CA. Proceedings.

[PCF07] Michele Piunti, Cristiano Castelfranchi, and Rino Falcone. Anticipatory coordination through action observation and behavior adaptation. In *Proceedings of AISB*, 2007.

[PMR99] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. LIME: Linda meets mobility. In *21st International Conference on Software Engineering (ICSE'99)*, pages 368–377, New York, NY, USA, 16–22 May 1999. ACM Press.

[PMV13] Danilo Pianini, Sara Montagna, and Mirko Viroli. Chemical-oriented simulation of computational systems with Alchemist. *Journal of Simulation*, 2013.

[POS13] PP González Pérez, A Omicini, and M Sbaraglia. A biochemically inspired coordination-based model for simulating intracellular signalling pathways. *Journal of Simulation*, 7(3):216–226, 2013.

[Rao96] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John W. Perram, editors, *Agents Breaking Away*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), Eindhoven, The Netherlands, 22-25 January 1996, Proceedings.

[RCD01] Davide Rossi, Giacomo Cabri, and Enrico Denti. Tuple-based technologies for coordination. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 4, pages 83–109. Springer, January 2001.

[RJMK14] Tuukka Ruotsalo, Giulio Jacucci, Petri Myllymäki, and Samuel Kaski. Interactive intent modeling: Information discovery beyond search. *Commun. ACM*, 58(1):86–92, December 2014.

[ROD03] Alessandro Ricci, Andrea Omicini, and Enrico Denti. 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, April 2003. 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17 September 2002. Revised Papers.

[ROV$^+$07] Alessandro Ricci, Andrea Omicini, Mirko Viroli, Luca Gardelli, and Enrico Oliva. Cognitive stigmergy: Towards a framework based on agents and artifacts. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNCS*, pages 124–140. Springer, May 2007. 3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

[RSDR11] Michele Ruta, Floriano Scioscia, Eugenio Di Sciascio, and Domenico Rotondi. Ubiquitous knowledge bases for the semantic web of things. In *First Internet of Things International Forum*, nov 2011.

[RVO07] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 67–86. Springer, May 2007.

[RVO08] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. The A&A programming model and technology for developing agent environments in MAS. In Mehdi

Dastani, Amal El Fallah Seghrouchni, Alessandro Ricci, and Michael Winikoff, editors, *Programming Multi-Agent Systems*, volume 4908 of *LNCS*, pages 89–106. Springer, April 2008.

[RWJ+95] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *NIST SPECIAL PUBLICATION SP*, 109:109, 1995.

[Sch01] Michael Schumacher. *Objective Coordination in Multi-Agent System Engineering. Design and Implementation*, volume 2039 of *LNCS*. Springer, April 2001.

[SFH+03] Giovanna Di Marzo Serugendo, Noria Foukia, Salima Hassas, Anthony Karageorgos, Soraya Kouadri Mostéfaoui, Omer F Rana, Mihaela Ulieru, Paul Valckenaers, and Chris Van Aart. *Self-organisation: Paradigms and applications*. Springer, 2003.

[Sha91] Ehud Shapiro. Separating concurrent languages with categories of language embeddings. In *23rd Annual ACM Symposium on Theory of Computing (STOC'91)*, pages 198–208, New York, NY, USA, 1991. ACM.

[SS00] C Simone and K Schmidt. Mind the gap! towards a unified view of cscw. In *Fourth International Conference on Design of Cooperative Systems (COOP2000), Sophia-Antipolis (Fr)*, 2000.

[Suc87] Lucy A. Suchman. Situated actions. In *Plans and Situated Actions: The Problem of Human-Machine Communication*, chapter 4, pages 49–67. Cambridge University Press, New York, NYU, USA, 1987.

[SW04] Kjeld Schmidt and Ina Wagner. Ordering systems: Coordinative practices and artifacts in architectural design and planning. *Computer Supported Cooperative Work (CSCW)*, 13(5-6):349–408, 2004.

[SZ01] 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, 2001.

[TCR+05] Luca Tummolini, Cristiano Castelfranchi, Alessandro Ricci, Mirko Viroli, and Andrea Omicini. "Exhibitionists" and "voyeurs" do it better: A shared environment approach for flexible coordination with tacit messages. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for Multi-Agent Systems*, volume 3374 of *LNAI*, pages 215–231. Springer, February 2005.

[TM04] Robert Tolksdorf and Ronaldo Menezes. Using Swarm Intelligence in Linda Systems. In Andrea Omicini, Paolo Petta, and Jeremy Pitt, editors, *Engineering Societies in the Agents World IV*, volume 3071 of *LNCS*, pages 49–65. Springer, June 2004. 4th International Workshops (ESAW 2003), London, UK, 29-31 October 2003. Revised Selected and Invited Papers.

[TRDMS11] A.-E. Tchao, M. Risoldi, and G. Di Marzo Serugendo. Modeling self-* systems using chemically-inspired composable patterns. In *Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on*, pages 109 –118, oct. 2011.

[Tur39] Alan Mathison Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 2(1):161–228, 1939.

[V$^{+}$78] Lev Vigotsky et al. Mind in society, 1978.

[VBC11] Mirko Viroli, Jake Beal, and Matteo Casadei. Core operational semantics of Proto. In Mathew J. Palakal, Chih-Cheng Hung, William Chu, and W. Eric Wong, editors, *26th Annual ACM Symposium on Applied Computing (SAC 2011)*, volume II: Artificial Intelligence & Agents, Information Systems, and Software Development, pages 1325–1332, Tunghai University, TaiChung, Taiwan, 21–25 March 2011. ACM.

[VC09] Mirko Viroli and Matteo Casadei. Biochemical tuple spaces for self-organising coordination. In John Field and Vasco T. Vasconcelos, editors, *Coordination Languages and Models*, volume 5521 of *LNCS*, pages 143–162. Springer, Lisbon, Portugal, June 2009.

[VCMZ11] Mirko Viroli, Matteo Casadei, Sara Montagna, and Franco Zambonelli. Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems*, 6(2):14:1–14:24, June 2011.

[VCO09] Mirko Viroli, Matteo Casadei, and Andrea Omicini. A framework for modelling and implementing self-organising coordination. In Sung Y. Shin, Sascha Ossowski, Ronaldo Menezes, and Mirko Viroli, editors, *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, volume III, pages 1353–1360, Honolulu, Hawai'i, USA, 8–12 March 2009. ACM.

[VGS$^{+}$13] Panagiotis Vlacheas, Raffaele Giaffreda, Vera Stavroulaki, Dimitris Kelaidonis, Vassilis Foteinos, George Poulios, Panagiotis Demestichas, Andrey Somov, Abdur Rahim Biswas, and Klaus Moessner. Enabling smart cities through a cognitive management framework for the Internet of things. *IEEE Communications Magazine*, 51(6):102–111, 2013.

[VHR$^{+}$07] Mirko Viroli, Tom Holvoet, Alessandro Ricci, Kurt Schelfthout, and Franco Zambonelli. Infrastructures for the environment of multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):49–60, July 2007.

[VO06] Mirko Viroli and Andrea Omicini. Coordination as a service. *Fundamenta Informaticae*, 73(4):507–534, 2006. Special Issue: Best papers of FOCLASA 2002.

[VOR07] Mirko Viroli, Andrea Omicini, and Alessandro Ricci. Infrastructure for RBAC-MAS: An approach based on Agent Coordination Contexts. *Applied Artificial Intelligence: An International Journal*, 21(4–5):443–467, April 2007. Special Issue: State of Applications in AI Research from AI*IA 2005.

[VPB12] Mirko Viroli, Danilo Pianini, and Jacob Beal. Linda in space-time: an adaptive coordination model for mobile ad-hoc environments. In Marjan Sirjani, editor, *Coordination Languages and Models*, volume 7274 of *LNCS*, pages 212–229. Springer-Verlag, June 2012.

[VSS95] Rob J. Vanglabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.

[WG03] Peter Wegner and Dina Goldin. Computation beyond Turing machines. *Communications of the ACM*, 46(4):100–102, April 2003.

[Whi06] Brian Whitworth. Socio-technical systems. *Encyclopedia of human computer interaction*, pages 533–541, 2006.

[WOO07] Danny Weyns, Andrea Omicini, and James J. Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, February 2007.

[ZCF$^+$11] Franco Zambonelli, Gabriella Castelli, Laura Ferrari, Marco Mamei, Alberto Rosi, Giovanna Di Marzo Serugendo, Matteo Risoldi, Akla-Esso Tchao, Simon Dobson, Graeme Stevenson, Yuan Ye, Elena Nardini, Andrea Omicini, Sara Montagna, Mirko Viroli, Alois Ferscha, Sascha Maschek, and Bernhard Wally. Self-aware pervasive service ecosystems. *Procedia Computer Science*, 7:197–199, December 2011.

[ZOA$^+$15] Franco Zambonelli, Andrea Omicini, Bernhard Anzengruber, Gabriella Castelli, Francesco L. DeAngelis, Giovanna Di Marzo Serugendo, Simon Dobson, Jose Luis Fernandez-Marquez, Alois Ferscha, Marco Mamei, Stefano Mariani, Ambra Molesini, Sara Montagna, Jussi Nieminen, Danilo Pianini, Matteo Risoldi, Alberto Rosi, Graeme Stevenson, Mirko Viroli, and Juan Ye. Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive and Mobile Computing*, 17:236–252, February 2015. Special Issue "10 years of Pervasive Computing" In Honor of Chatschik Bisdikian.

[ZV08] Franco Zambonelli and Mirko Viroli. Architecture and metaphors for eternally adaptive service ecosystems. In *Intelligent Distributed Computing, Systems and Applications*, volume 162/2008 of *Studies in Computational Intelligence*, pages

23–32. Springer, September 2008. 2nd International Symposium on Intelligent Distributed Computing (IDC 2008), Catania, Italy, 18–19 September 2008. Proceedings.

# List of Figures

# List of Tables