# Twenty Years of Coordination Technologies: State-of-the-Art and Perspectives

Giovanni Ciatto[1], Stefano Mariani[2(✉)], Maxime Louvel[3], Andrea Omicini[1], and Franco Zambonelli[2]

[1] Alma Mater Studiorum–Università di Bologna, Cesena, Italy
{giovanni.ciatto,andrea.omicini}@unibo.it
[2] Università di Modena e Reggio Emilia, Reggio Emilia, Italy
{stefano.mariani,franco.zambonelli}@unimore.it
[3] Bag-Era, Montbonnot-Saint-Martin, France
maxime.louvel@bag-era.fr

**Abstract.** Since complexity of inter- and intra-systems interactions is steadily increasing in modern application scenarios (e.g., the IoT), coordination technologies are required to take a crucial step towards maturity. In this paper we look back at the history of the COORDINATION conference in order to shed light on the current status of the coordination technologies there proposed throughout the years, in an attempt to understand success stories, limitations, and possibly reveal the gap between actual technologies, theoretical models, and novel application needs.

**Keywords:** Coordination technologies · Middleware · Survey

## 1 Scope, Goal, and Method

Complexity of computational systems, as well as their impact on our everyday life, is constantly increasing, along with the growing complexity of *interaction*—inter- and intra-systems. Accordingly, the role of *coordination models* should expectedly grow, along with the relevance of *coordination technologies* within ICT systems: instead, this is apparently not happening—*yet*.

Then, it is probably the right time – now, after twenty years of the COORDINATION conference – and the right place – the COORDINATION conference itself – to take a step back and reflect on what happened to coordination models, languages, and (above all) *technologies* in the last two decades. That is why in this paper we survey all the technologies presented and discussed at COORDINATION, examine their stories and their current status, and try to provide an overall view of the state-of-the art of coordination technologies as emerging from twenty years of work by the COORDINATION community. The main goal is to


G. Di Marzo Serugendo and M. Loreti (Eds.): COORDINATION 2018, LNCS 10852, pp. 51–80, 2018.
https://doi.org/10.1007/978-3-319-92408-3_3

provide a sound basis to answer questions such as: Are coordination technologies ready for the industry? If not, what is currently missing? Which archetypal models lie behind them? Which are the research areas most/least explored? And what about the target application scenarios?

### 1.1 Structure and Contribution of the Paper

Section 2 first provides an overview of the data about papers published in the conference throughout the years (Subsect. 2.1), as collected from the official SpringerLink website and its companion BookMetrix service, with the aim of emphasising trends concerning *(i)* the number of *papers* published in each volume, *(ii)* the number of *citations* generated by each volume, *(iii)* the number of *downloads* generated by each volume, *(iv)* the *most cited* paper of each volume, and *(v)* the *most downloaded* paper of each volume.

Then, the scope of our analysis narrows down to those papers bringing a technological contribution, in the sense of describing a *software artefact* offering an API exploitable by other software to coordinate its components. Accordingly, Subsect. 2.2 provides an overview of all the technologies born within the COORDINATION conference series. For each one, the reference model implemented, and the web URL where to retrieve the software – if any – are given.

Then, a brief description of all the software for which no working implementation could be found is reported for the sake of completeness, whereas technologies still available are thoroughly described in Subsect. 2.3. There, each selected technology was downloaded and tested to clearly depict its *health status*:

- date of last update to the source code (or project web page, if the former is not available)
- whether the software appears to be actively developed, in maintenance mode, or discontinued
- whether suitable documentation is available
- whether the source code is publicly available
- whether the build process of the software artefact is reproducible
- whether the software artefact, once built, executes with no errors

For the latter two items, in case of failures, an explanation of the problem and, if needed, the steps undertaken in the attempt to overcome it, are provided too. In particular, the latter test is not meant to measure performance, or, to provide a benchmark for comparisons: its purpose is to assess whether the technology is *usable*, that is, executable on nowadays software platforms and by nowadays programming languages. For instance, a library requiring an obsolete third-party libraries that hinders smooth deployment is considered not usable. Accordingly, each technology is tested either running provided example code, or developing a minimal working example of usage of the API.

Section 3 discusses the data collected so as to deliver insights about: *(i)* the *evolution* of technologies as they are stemming from a few archetypal models (Fig. 5), *(ii)* the *relationships* between the selected technologies, as a *comparison* of their features (Fig. 6), and *(iii)* the *main goal* and *reference scenario* of

each technology (Fig. 7). Also, a general discussion is provided, reporting about success stories, peculiarities, and opportunities.

Finally, Sect. 4 concludes the paper by summarising the results of the survey and providing some perspectives for the future of coordination technologies.

### 1.2 Method

The scope of this survey is indeed the COORDINATION conference series. There, we focus on coordination *technologies* intended as software implementing a given coordination model, language, mechanism, or approach with the goal of providing coordination *services* to other software applications. In other words, our focus is on technologies implementing some form of *coordination middleware or API*—analysed in Subsect. 2.2. We nevertheless include in our overview other technologies produced within COORDINATION (Subsect. 2.1), such as *simulation* frameworks, *model-checking* tools, and proof-of-concept implementations of *process algebras*—which are only described in short.

Starting from the COORDINATION conference proceedings available online from SpringerLink[1], the survey proceeds as follows:

1. for each conference year, papers describing a coordination-related technology were gathered manually into a Google Spreadsheet
2. for each collected paper, we checked whether the paper was actually proposing some software package—papers failing the test are omitted
3. for each paper passing the test, we verified the health *status* of the technology—as described in Subsect. 1.1
4. then, for each paper featuring at least a *usable* distribution – meaning a downloadable version of the software – the corresponding software was downloaded and tested—i.e., installation & basic usage

## 2 The Survey

Although the focus of this paper are coordination technologies, we believe an overview of the whole conference proceedings is due to give context to the survey itself. Accordingly, Subsect. 2.1 summarises and analyses all the data officially available from Springer—concerning, for instance, citations and downloads of each volume and paper. Then, Subsect. 2.2 accounts for all the coordination technologies mentioned in COORDINATION papers, regardless of their actual availability, while Subsect. 2.3 reports about the core of this survey: the status of the coordination technologies nowadays publicly available.

### 2.1 Overview

The COORDINATION conference series has been held 19 times since its first edition in 1996 in Cesena (Italy), and generated as many conference proceedings volumes—all available online (See footnote 1). Data about the number of

[1] http://link.springer.com/conference/coordination.

published papers, the number of *citations* and *downloads* per year of each volume, as well as the *most cited* and *most download* paper have been collected from SpringerLink and its companion service BookMetrix[2]—and are reported in Table 1 (last checked February 9th, 2018). Highest values for each column are emphasised in bold.

**Table 1.** Overall data directly available online from Springer regarding the COORDINATION conference series. To compute citations (downloads) per year, the number of citations (downloads) was divided by the number of years the publications is available since. MCP stands for "Most Cited Paper" whereas MDP stands for "Most Downloaded Paper".

| Edition | No. of papers | Citations/year | Downloads/year | MCP | MDP |
|---|---|---|---|---|---|
| 1996 | 34 | 3.32 | 140.00 | 16 | 124 |
| 1997 | 31 | 2.86 | 140.48 | 14 | 149 |
| 1999 | 32 | 2.05 | 205.26 | 6 | 154 |
| 2000 | 27 | 0.11 | — | 6 | 158 |
| 2002 | **35** | 2.81 | 301.88 | 7 | 180 |
| 2004 | 23 | 4.07 | 197.86 | 19 | 146 |
| 2005 | 19 | 3.00 | 261.54 | 9 | 214 |
| 2006 | 18 | 6.25 | 312.50 | **22** | 297 |
| 2007 | 17 | 8.27 | 341.82 | 14 | 308 |
| 2008 | 21 | 10.70 | 391.00 | 13 | 227 |
| 2009 | 15 | 7.44 | 370.00 | 13 | 259 |
| 2010 | 12 | 3.25 | 507.50 | 6 | 536 |
| 2011 | 14 | 4.00 | 538.57 | 6 | **675** |
| 2012 | 18 | 5.50 | 1081.67 | 6 | 523 |
| 2013 | 17 | 8.20 | 1314.00 | 7 | 547 |
| 2014 | 12 | 10.50 | 792.50 | 10 | 299 |
| 2015 | 15 | 6.67 | 1453.33 | 11 | 336 |
| 2016 | 16 | **14.00** | **2355.00** | 4 | 350 |
| 2017 | 14 | 2.00 | 1930.00 | 1 | 245 |
| Avg. | 20.53 | 5.53 | 701.94 | 10 | 301.42 |
| Std. Dev. | 7.57 | 3.60 | 658.47 | 5.42 | 160.16 |

The trend over time of the number of papers, the citations of the volumes, and their downloads, are plotted in Figs. 1 and 2, respectively, along with their trend line. A few significant trends can be spotted in spite of the high variability between different editions of the conference. For the number of published papers,

[2] http://www.bookmetrix.com/.

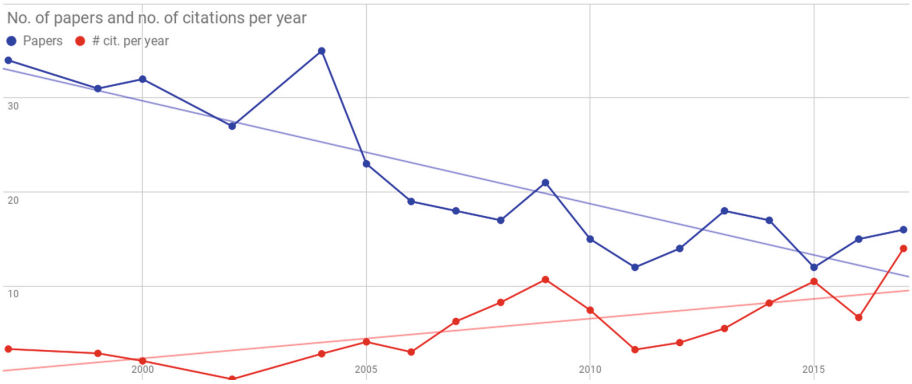


**Fig. 1.** Number of papers in the volume and number of citations per year (computed as described in text) of the volume.

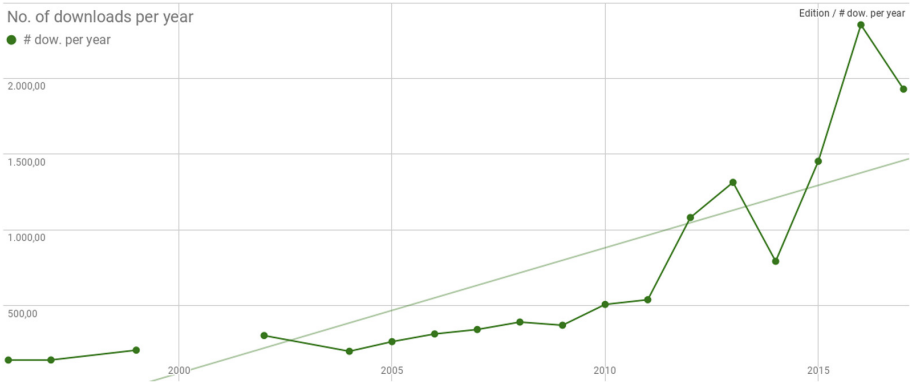


**Fig. 2.** Number of downloads per year (computed as described in text) of the volume.

the trend is clearly *descending*: the first five editions featured an average of 32 papers, whereas the latest five an average of 15. As far as the number of citations per year generated by each volume of the proceedings is concerned, a few oscillations can be observed:

- a first phase (from the 1st edition to the 4th) shows a *decreasing* number of citations, from 3.32 down to 0.11 (the *all-time-low*)
- then, in a second phase (from the 5th to the 10th edition) the number of citations *increases*, up to 10.70 in 2008
- finally, after a brief fall in 2009 and 2010, the number of citations per year kept increasing up to the *all-time-high* of 2016 (14.00)

For the number of downloads per year, two phases can be devised out in Fig. 2:

- in the first period, from the 1st edition to the 13th (2011), the trend is quite *stable*, oscillating between 140 and 538.57

– in the second one instead, from 2012 up to latest edition, there is a *sharp increase* up to the *all-time-high* of 2355.00 in 2016

Finally, Figs. 3 and 4 show the most cited paper and the most downloaded paper per year, respectively. Besides noting *(i)* the highly *irregular* trend regarding the most cited papers, oscillating from 6 to 22 through approximately three epochs[3] (few citations during 1996–2002, more citations during 2003–2009, few again during 2010–2017), and *(ii)* the *increasing* number of downloads in recent years: in the four years between 2010 and 2013 the most downloaded papers combined generated more downloads than the most downloaded papers of *all* the previous years combined (2281 vs. 2216), it could be interesting to check how many of such papers are related to technology, if any.

Overall, in the 19 editions of COORDINATION held until now, the most cited/downloaded paper is about technology – in the broadest acceptation of

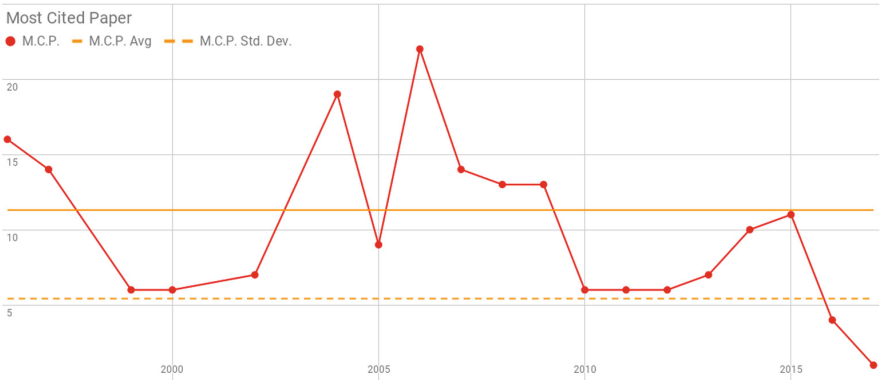


**Fig. 3.** Most cited paper per year with average values & standard deviation.

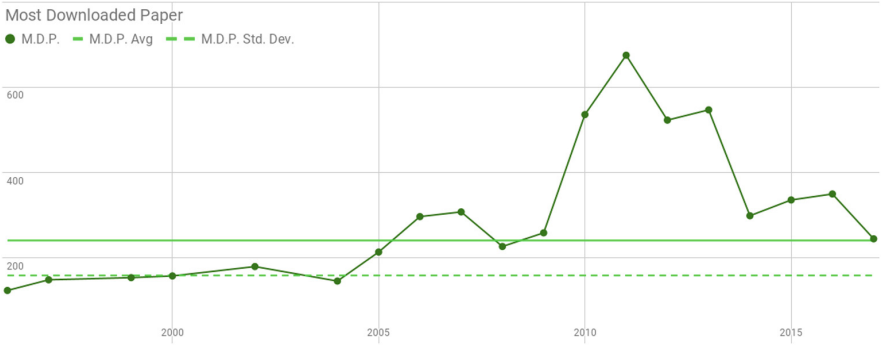


**Fig. 4.** Most downloaded paper per year with average values & standard deviation.

[3] Excluding the most recent editions, which had less time to generate citations.

the term – in *slightly less then a half* of them: 7 papers amongst the most cited ones, and 8 amongst the most downloaded ones.

By extending the analysis to all the papers published in the proceedings, instead, out of all the 390 papers published, only 47 (just 12.05%) – based on authors' inspection of the papers – convey a technological contribution. And, such an estimate is somehow optimistic, since we counted papers just for merely *mentioning* a technology, with no means to access it—see Table 2.

### 2.2 Analysis of Technologies

Table 2 provides an overview of the coordination technologies born within the COORDINATION conference series throughout the years. Only those technologies passing test §2 in Sect. 1.2 are included, that is, those technologies actually delivering some form of *coordination services* to applications—i.e. in the form of a *software library* with suitable API. For each technology, the original paper is referenced, the *model* taken as reference for implementation indicated – if any – and the URL to the technology web page hosting the software given—if any is still reachable. Technologies whose corresponding software is still available – that is, those passing test §3 in Sect. 1.2 – are further discussed in Subsect. 2.3; those with no working software found are briefly described in the following.

*The Early Days.* The first few years of COORDINATION (1996–2000) saw a *flourishing* of successful technologies: some of the ideas introduced back then are still alive and healthy. For instance, $\mathcal{ACLT}$ [35] adopted *first-order logic terms* as LINDA tuples, an intuition shared by the $\mu^2Log$ model and its language, MultiBinProlog [31]. Also, $\mathcal{ACLT}$ allowed agents to *dynamically program* tuple spaces via a specification language, enabling definition of computations to be executed in response to some *events* generated by interacting processes. Both features influenced the TuCSoN model and infrastructure [28], one of the few technologies to be still actively maintained nowadays.

Similarly, the IWIM coordination model and its corresponding language, MANIFOLD [5], were introduced back in 1996 and survived until present days by evolving into Reo [6]. IWIM came by recognising a dichotomy between *exogenous* and *endogenous* coordination, and exploiting *channel composition* as a means to build increasingly complex coordination patterns.

Finally, Moses [4] was presented to the COORDINATION community as an infrastructure reifying the *Low Governed Interaction* (LGI) model. The technology is still alive and inspectable from its homepage, even if apparently no longer maintained. Analogously, the Piccola composition language presented in [2] clearly relies on a coordination technology which reached stability and robustness, even if it seems to be no longer maintained, too.

Besides these success stories, many other papers at that time proposed a technology, but either they only mentioned the technology without actually providing a reference to a publicly available software, or such a reference is no longer reachable (i.e. the link is dead and no reference to the software have been found on the web). For instance:

**Table 2.** Overview of the coordination technologies presented at COORDINATION. "Name" denotes the technology, whereas "Model" makes explicit the model taken as reference for the implementation. The last column points to the web page where the software is available – if any – and provides for additional notes.

| Name | Year | Model | (Closest) Web page & notes |
|---|---|---|---|
| Manifold [5] | 1996 | IWIM [5] | http://projects.cwi.nl/manifold<br>*no link to implementation* |
| Sonia [12] | 1996 | LINDA + access control | *no implementation found* |
| Laura [92] | 1996 | service-oriented LINDA | *no implementation found* |
| MultiBinProlog [31] | 1996 | $\mu^2 Log$ [31] | http://cseweb.ucsd.edu/~goguen/courses/230/pl/art.html<br>*dead links* |
| MESSENGERS [42] | 1996 | Navigational Programming [42] | http://www.ics.uci.edu/~bic/messengers<br>*dead links* |
| $\mathcal{ACLT}$ [35] | 1996 | LINDA + programmable tuple spaces | *evolved into* TuCSoN |
| Blossom [46] | 1997 | LINDA + coordination patterns | *no implementation found* |
| Bonita [84] | 1997 | asynch LINDA | *no implementation found* |
| Berlinda [93] | 1997 | LINDA | *no implementation found* |
| SecOS [22] | 1999 | LINDA | *no implementation found* |
| Messengers [95] | 1999 | CmPS + mobility [52] | http://osl.cs.illinois.edu/software/<br>*no mention of "Messengers"* |
| MJada [82] | 1999 | OO LINDA | http://www.cs.unibo.it/cianca/wwwpages/macondo/<br>*no reference to MJada* |
| STL++ [87] | 1999 | ECM [87] | *no implementation found* |
| Clam [86] | 1999 | IWIM [5] | *no implementation found* |
| TuCSoN [28] | 1999 | *novel* (many extensions to LINDA) | http://tucson.unibo.it/ |
| Truce [53] | 1999 | *novel* (protocols + roles) | *no implementation found* |
| CoLaS [29] | 1999 | *novel* (protocols + roles) | *no implementation found* |
| OpenSpaces [38] | 2000 | OO LINDA | *no implementation found* |
| Piccola [2] | 2000 | *novel* | http://scg.unibe.ch/research/piccola |
| Moses [4] | 2000 | LGI [4] | http://www.moses.rutgers.edu |
| Scope [63] | 2000 | LINDA + mobility + space federation | *no implementation found* |
| $P\epsilon\omega$ [6] | 2002 | IWIM [5] | http://reo.project.cwi.nl/reo<br>*evolved into Reo* |
| SpaceTub [94] | 2002 | LINDA | *no implementation found* |
| O'Klaim [15] | 2004 | OO LINDA + mobility | http://music.dsi.unifi.it/xklaim<br>*evolved into X-Klaim* |

(*continued*)

**Table 2.** (*continued*)

| Name | Year | Model | (Closest) Web page & notes |
|---|---|---|---|
| Limone [40] | 2004 | LINDA + mobility + spaces federation | http://mobilab.cse.wustl.edu/projects/limone |
| CRIME [65] | 2007 | LIME [34] | http://soft.vub.ac.be/amop/crime/introduction |
| TripCom [89] | 2007 | Triple Space Computing [39] | http://sourceforge.net/projects/tripcom |
| CiAN [88] | 2008 | *novel* | http://mobilab.cse.wustl.edu/Projects/CiAN/Home/Home.shtml |
| Smrl [1] | 2008 | PEPA [45] | http://groups.inf.ed.ac.uk/srmc/download.html |
| CaSPiS [18] | 2008 | IMC [17] | http://sourceforge.net/projects/imc-fi |
| LeanProlog [91] | 2008 | *novel* | http://www.cse.unt.edu/~tarau/research/LeanProlog |
| JErlang [75] | 2010 | JOIN-CALCULUS [41] | http://github.com/jerlang/jerlang |
| Session Java [68] | 2011 | Session Types [50] | http://www.doc.ic.ac.uk/~rhu/sessionj.html |
| WikiRecPlay/InFeed [80] | 2012 | BPM | *no implementation found* |
| Statelets [57] | 2012 | *novel* | http://sourceforge.net/projects/statelets |
| IIC [77] | 2012 | Reo [6] | http://github.com/joseproenca/ip-constraints |
| LINC [58] | 2015 | LINDA [44] | *implementation not available for commercial reasons* *see* http://bag-era.fr/index_en.html#about |
| RepliKlaim [3] | 2015 | Klaim [19] | http://sysma.imtlucca.it/wp-content/uploads/2015/03 |
| Logic Fragments [30] | 2014 | SAPERE [97] | *no implementation found* |

**Sonia** [12] — a LINDA-like approach supporting *human workflows*, therefore stressing aspects such as understandability of the tuple and template languages, time-awareness and timeouts, and security by means of access control

**Laura** [92] — a language attempting to steer LINDA towards *service-orientation*, where tuples can represent (formal descriptions of) service requests, offers, or results, thus enabling loosely coupled agents to cooperate by means of Linda-like primitives

**MESSENGERS** [42] — following the *Navigational Programming* methodology [42], where strongly-mobile agents – a.k.a. *Messengers* – can migrate between nodes. Here, coordination is seen as "invocation [of distributed computations] and exchange of data" and it "is managed by groups of Messengers propagating autonomously through the computational network"

**Blossom** [46] — a LINDA variant focusing on safety, which is provided by supporting a *type system* for tuples and templates, and a taxonomy of access patterns to tuple spaces, aimed at supporting a sort of "least privilege" principle w.r.t. access rights of client processes

**Bonita** [84] — another LINDA-like technology – as its successor WCL [85] – focusing on *asynchronous* primitives and distribution of tuple spaces, which can also migrate closer to their users

**Berlinda** [93] — providing a *meta-model* – along with a Java implementation – for instantiating different LINDA-like models

**SecOS** [22] — a LINDA variant focusing on *security* and exploring the exploitation of (a)symmetric key encryption

**Messengers** [95] — not to be confused with [42] despite its name, which focusses on *message exchange* by means of migrating actors

**MJada** [82] — an extension of the Jada language [25], focusing on coordinating concurrent (possibly distributed) Java agents by means of LINDA-like tuple spaces with an extended primitive set and *object-oriented tuples*

**Clam** [86] — a coordination language based on the IWIM model [5]

**Truce** [53] — a scripting language aimed at describing *protocols* to which agents must comply by enacting one or more *roles*

**CoLaS** [29] — a model and its corresponding language providing a framework where a number of *participants* can join interaction *groups* and play one or more *roles* within the scope of some coordination *protocol*. In particular, CoLaS focuses on the enforcement of coordination rules by validating and constraining participants behaviour

*The Millenials.* After year 2000, technologies are *less* present amongst COORDINATION papers, but not necessarily less important. For instance, Reo made its first appearance in 2002 [6], its name written in Greek ($P\epsilon\omega$). Reo provides an *exogenous* way of governing interactions between processes in a concurrent and possibly distributed system. Its strength is due to its *sound semantics*, enabling researchers to formally verify system evolution, as well as to the availability of *software tools*. The technology is indeed still alive and actively developed.

Recent implementations are more easily available on the web. Out of 22 coordination technologies, just 6 were not found on the web during the survey:

**OpenSpaces** [38] — focussing on the harmonisation of the LINDA model with the OOP paradigm and, in particular, with the inheritance mechanism

**Scope** [63] — analogously to Lime, it provides multiple distributed tuple spaces cooperating by means of local interactions when some process attempts to access a tuple, thus providing a sort of federated view on the tuple space

**SpaceTub** [94] — successor of Berlinda, it aims at providing a meta-framework where other LINDA-like frameworks can be reproduced

**WikiRecPlay/InFeed** [80] — a pair of tools (browser extensions, no longer available) aimed at extracting and manipulating information from web applications to record them and later *replay*, enabling the definition of sequences of activities that can be *synchronised* with each other. The goal here is to augment *social software* with coordination capabilities.

**LINC** [58] — a coordination environment implementing the basic LINDA primitives – `out`, `in`, `rd` – in a setting in which each tuple space (called bag) could implement the primitives differently (still preserving semantics), a convenient

opportunity when dealing with *physical devices* (i.e. in the case of deployment to IoT scenarios) or *legacy systems* (i.e. databases). It provides *transactions* to alleviate to developers the burden of rolling back actions in the case of failures, and a chemical-reaction model inspired to Gamma [11] for enacting *reaction rules*. Several tools [59] are provided to help developers debug the rules, and to generate rules from high level specifications. The LINC software is nevertheless not publicly available because it is exploited by the Bag-Era company. Accordingly, it is not further analysed in Subsect. 2.3, but it is included in Sect. 3 as an example of industrial success.

**Logic Fragments** [30] — a *chemical-based* and programmable coordination model likewise SAPERE [97] – to which it is inspired – enriched with a *logic-based* one through the notion of Logic Fragments, which are combinations of logic programs defining on-the-fly, ad-hoc chemical reactions – similar to SAPERE eco-laws – that apply on matching tuples to manipulate other tuples or to produce new Logic Fragments. The aim is to guarantee data consistency, favour knowledge representation of partial information, and support constraints satisfaction, thanks to verification of *global properties* enabled by the logic nature of the framework.

All the others are still publicly available, thus further analysed in next section.

For instance, the O'Klaim language presented in [15] evolved into the X-Klaim project [16] which is still alive, even if apparently no longer maintained. Similar considerations can be made for Limone [40] and CRIME [65], which both stem from the idea of *opportunistic federation* of transient tuple spaces introduced by LIME [66], and improve it with additional features such as lightweightness and orientation to ambient-programming.

Analogously, the CiAN [88] model and middleware, targeting the coordination of distributed workflows over *M*obile *A*d-hoc *Net*works (MANETs), comes with a mature implementation, although no longer maintained. An extension to Session Java [51] is proposed in [68] to explicitly tackle synchronisation issues such as freedom from deadlock via multi-channel session primitives, Whereas the implementation was discontinued in 2011[4], the source code is still available from GoogleCode archive. JErlang [75], a Java-based implementation of Erlang extended with constructs borrowed from the JOIN-CALCULUS [41], appears to be no longer maintained too as explicitly stated in its home page[5], although a couple of implementations are still available and (partially) working.

Also RepliKlaim [3], an implementation of KLAIM [19] aimed at optimising performance and reliability through *replication* of tuples and tuple spaces, received updates until 2015 as far as we know, thus appears to be discontinued. Likewise, 2015 is the year when both Statelets [57] and IIC [77] received their last known update: the former is a programming model and language aimed at integrating *social context* and *network effects*, derived from social networks analysis, as well as *semantic* relationships amongst shared artefacts in, i.e. groupware

[4] Year of latest commit: https://code.google.com/archive/p/sessionj.
[5] http://jerlang.org/.

applications, into a single and coherent coordination model, while the latter proposes *Interactive Interaction Constraints* (IIC) as a novel framework to ground *channel-based* interaction – *à la* Reo – upon *constraints satisfaction*, interpreting the process of coordinating components as the execution of a constraints solver.

Next section briefly focuses on those technologies—that is, coordination technologies that can be actually installed and used nowadays—step §4 in Sect. 1.2.

### 2.3 Analysis of Selected Technologies

Table 3 overviews the *working technologies* we were able to somewhat successfully test, that is, only those technologies listed in Table 2 which successfully surpassed test §4 described in Sect. 1.2—a software artefact exists and is still working.

It is worth noting that, w.r.t. Table 2, a few technologies are not included in this section despite the corresponding software is available from the reference web page therein referenced. The reason is:

- Smrl requires ancient software to run—that is, an old version of Eclipse requiring in turn an ancient version of the Java runtime (1.4)
- CaSPiS [18] (or better, JCaSPiS, namely the Java-based implementation of CaSPiS) was not found anywhere—neither in the author personal pages, nor in their account profiles on Github, nor in the web pages of the SENSORIA project mentioned in the paper. Nevertheless, the IMC model and framework allegedly grounding its implementation is still accessible[6]. Then we proceeded to download it looking for the CaSPiS code, without success. It is worth to be mentioned, anyway, that the IMC framework code appears to be broken, since compilation fails unless a restricted/deprecated Java API is used[7], and even in the case of instructing the compiler to allow for it[8] the attempt to run any part of the software failed without informative error messages—just generic Java exceptions.
- LeanProlog is not usable as a coordination technology as defined in Sect. 1.2: it is a Prolog engine with low-level mechanisms for handling multi-threading, and provides no API for general purpose coordination
- Session Java, as explicitly stated in its home page, requires an ancient version of the Java runtime to run, that is, 1.4
- Statelets is explicitly tagged as being in "pre-alpha" development stage, and, upon inspection, revealed to be only partially developed

TuCSoN. Although TuCSoN [28] appeared at COORDINATION in 1999, its roots date back to the first edition of the conference, as the $\mathcal{ACLT}$ model [35].

TuCSoN is a coordination model adopting LINDA as its core but extending it in several ways, such as by adopting nested tuples (expressed as first-order logic terms), adding primitives (i.e. *bulk* [83] and *uniform* [60]), and replacing tuple

[6] https://sourceforge.net/projects/imc-fi/.
[7] A class uses a deprecated API, and another one requires breaking access restrictions.
[8] See https://goo.gl/pdWCsx.

**Table 3.** Overview of the working coordination technologies presented at COORDINATION. Column "Health" denotes the status of the software, for instance whether it is still actively developed, only in maintenance mode, or actually discontinued, column "Build" is filled whenever source code is available and denotes whether build steps (i.e. compilation into binaries and dependencies resolution) were successful, column "Deployment" indicates whether the software has been successfully executed. It is worth to emphasise that LINC has been left out since it is part of commercial solutions sold by the Bag-Era company, thus no further inspection of the software was possible.

| Name | Last update | Health | Documentation | Source code | Build | Deployment |
|---|---|---|---|---|---|---|
| TuCSoN | 2017 | Actively developed | Available | Available | Successful | Successful |
| Moses | 2017 | Actively developed/maintained | Available | Unavailable | — | Successful |
| JErlang | 2017 | Discontinued | Poor | Available | Failed | — |
| IIC | 2015 | Discontinued | Poor | Available | Failed | Successful |
| Reo | 2013 | Actively developed | Available | Available | Successful | Partially successful |
| TripCom | 2009 | Discontinued | Partially available | Available | Successful | Successful |
| CiAN | 2008 | Discontinued | Available | Available | Successful | Successful |
| Piccola | 2006 | Discontinued | Available | Java only No Smalltalk | Successful | Successful |
| CRIME | 2006 | Discontinued | Unavailable | Unavailable | — | Successful |
| Klava | 2004 | Discontinued | Poor | Available | Successful | Successful |
| X-Klaim | 2004 | Discontinued | Available | Available | Failed | — |
| Limone | 2004 | Discontinued | Unavailable | Available | Failed | — |
| RepliKlaim | —[a] | —[a] | Unavailable | Available | Successful | Successful |

[a] There is no publicly available code repository, thus no information about latest commits.

spaces with *tuple centres* [71] programmable in the ReSpecT language [70]. It comes with a Java-based implementation providing *coordination as a service* [96] in the form of a Java library providing API and a middleware runtime, especially targeting distributed Java process but open to rational agents implemented in tuProlog [36]. The TuCSoN middleware is *publicly available* from its home page[9], which provides both the binaries (a ready-to-use Java jar file) and a link to the *source code* repository. From there, also *documentation* pages are available, in the form of a usage guide and a few tutorials providing insights into specific features. Finally, a few related sub-projects are therein described too, such as TuCSoN4Jade [62] and TuCSoN4*Jason* [61], which are both Java libraries aimed at integrating TuCSoN with Jade [13] and *Jason* [21] agent runtimes, respectively, by wrapping TuCSoN services into a more convenient form which best suites those developers accustomed to programming in those platforms.

As far as technology is concerned, TuCSoN is still *actively* developed, being the latest commit in 2017, when also the latest related publication has been produced—an extension to the ReSpecT language and toolchain exploited to program tuple centres in TuCSoN [27]. Also, it is actively *exploited* as the infrastructural backbone for other projects – e.g., the smart home logic-based platform Home Manager [23] – and industrial applications—e.g., the Electronic Health Record solution [37]. Nevertheless, TuCSoN is the results of many years of active development by many different people with many different goals. Thus, despite some success stories, TuCSoN would require some substantial refactoring and refinement before it can become a truly commercially-viable product.

*Moses.* Moses [4] is the technology implementing the *Law Governed Interaction* (LGI) coordination model [64], which aims at controlling the interaction of agents interoperating on the Internet. In LGI, each agent interacts with the system by means of a *controller*, that is, a component exposing a fixed set of primitives allowing agents to exchange messages with other agents. The controller is in charge of intercepting invocations of primitives by interacting agents to check if they are allowed according to the *law* currently adopted by that controller.

Laws are shared declarative specifications dictating how the controller should react when it intercepts events of interest. Laws are expressed either in a Prolog-like language or as Java classes. Each controller has its own state which can be altered by reactions to events and can influence the effect of future reactions. Non-allowed activities are technically prohibited by the controller which takes care of aborting the forbidden operation—for instance, by not forwarding a message to the intended receiver if some conditions are met.

The project home page[10] is well-organised and provides a number of resources focussed on Moses/LGI such as reference papers, manuals, tutorials, JavaDoc, examples. The page also provides an archive with the compiled versions of the *Moses middleware*, the latest one dating back to 2017—suggesting that the

[9] http://tucson.unibo.it.
[10] http://www.moses.rutgers.edu/index.html.

project is *actively maintained and/or developed*, and representing another success story born within the COORDINATION series. We were able to *successfully* compile and execute the code: however, no source code is provided, and some portion of the web page, such as the JavaDoc, are not updated w.r.t. the current Moses implementation. Finally, Moses still bounds to *deprecated technologies* such as Java Applets, which we believe may hinder its adoption.

*JErlang.* JErlang [75] is an extension of the Erlang language for concurrent and distributed programming featuring *joins* as the basic synchronisation construct—as borrowed from the Join-Calculus [41]. The web page mentioned in the paper[11] is no longer accessible; by searching JErlang and the authors' names on the web, a GitHub repository with the same broken reference popped up[12], apparently tracking the development history of the JErlang technology. There, however, JErlang is described as an implementation of Erlang/OTP on the JVM. Also, another apparently very similar technology is therein referenced: Erjang.

Anyway, JErlang installation and usage instructions are nowhere to be found, and, when trying to build the project through the provided Maven pom.xml file, the build fails due to many errors related to obsolete dependencies—which we were not able to fix. Instead, Erjang GitHub repository – with no clue about its links to the paper – provides installation instructions, however building fails due to a Java compilation failure for a "bad class file" error[13]. We feel then free to declare the implementation as discontinued.

*IIC. Interactive Interaction Constraints* (IIC) [77] is a sort of "spin-off" of Reo introduced in 2013 [77]. The original approach of implementing Reo connectors as interaction constraints is extended to allow interaction to take place also *between rounds* of constraints satisfaction. This extends the expressive reach of IIC beyond Reo, and makes the whole process of constraints satisfaction *transactional* w.r.t. observable behaviour.

The IIC software is distributed as a Scala library providing an handy syntax which eases definition of Reo-like connectors. The Scala library *source code* is distributed by means of a GitHub repository[14] where the latest commit dates back to 2015. The library ships with a SBT configuration, allegedly supporting automatic building. Nevertheless, we were not able to reproduce the compilation process since the provided SBT configuration depends on an *ancient SBT version*. Therefore, we consider IIC a *no longer maintained* but *still usable* full-fledged coordination technology.

*Reo.* Reo was firstly introduced to the COORDINATION community in [6], its name in Greek letters ($P\epsilon\omega$). Similarly to the IWIM model, Reo adopts a

[11] https://www.doc.ic.ac.uk/~susan/jerlang/.
[12] Second link in "See also" section at https://github.com/jerlang/jerlang.
[13] Actual error is: "class file contains malformed variable arity method: [...]".
[14] http://github.com/joseproenca/ip-constraints.

paradigm for exogenous coordination of concurrent and possibly distributed software components. According to the Reo model, *components* are the entities to be coordinated, representing the computations to be performed, while *connectors* are the abstraction reifying coordination rules. The only assumption Reo makes about components is that they have a unique name and a well-defined interface in the form of a set of input ports and output ports. Conversely, connectors are composed by *nodes* and *channels*, or other connectors. A number of coordination schemes can be achieved by combining the different sorts of nodes and channels accordingly. This allows to formally specify *how*, *when*, and upon which conditions data may flow from the input to the output ports of components.

Diverse research activities originated from Reo throughout the years, mostly aimed at *(i)* analysing the formal properties of both Reo connectors and *constraints automata* [10], which are the computational model behind Reo semantics; and *(ii)* supporting web services orchestration [54], composition, and verification [55] by means of code generation and verification tools.

Several technologies are available from the Reo tools homepage[15], collectively branded as the Extensible Coordination Tools (ECT). They consist of various Eclipse IDE plugins, such as a graphical designer for Reo connectors, and a code generator which automatically converts the graphical description into Java sources in which developers may inject applicative logic. Nevertheless, the generated code comes with no explicit support for distribution.

According to their home page, ECT are allegedly compatible with any Eclipse version starting from 3.6; while we were not able to reproduce its installation in that version (due to a dependency requiring an higher version of Eclipse), we succeeded in installing it on Eclipse version 4.7 (the latest available), but the code generator appears *buggy and unstable* – thus hindering further testing – because of several non-informative error messages continuously appearing when trying to use the Reo model designer—which is a required step for code generation.

The ECT source code is available from a Google Code repository[16]—last commit dating back to 2013. In [78] a novel implementation is proposed, named Dreams, implemented in Scala and aimed at closing the gap between Reo and distributed systems. Nevertheless, its binary distribution seems *unavailable* and no documentation is provided describing how to compile or use it, thus we were not able to further test this novel Dreams framework.

*TripCom.* TripCom [89] is essentially a departure from the LINDA model where the tuple space abstraction is brought towards the Semantic Web vision [47] and web-based semantic interoperability in general. The former is achieved by employing the Resource Description Framework (RDF) – that is, a representation of semantic information as a triple "subject-predicate-object" – as the tuple representation language, and by considering tuple spaces as RDF triplets containers. Also, LINDA primitives have been consequently re-thought under a semantics-oriented perspective—that is, by adopting an ad-hoc templating

[15] http://reo.project.cwi.nl/reo/wiki/Tools.
[16] https://code.google.com/archive/p/extensible-coordination-tools/source.

language enabling expression of semantic relationships. The latter is achieved by making triple spaces accessible on the web as SOAP-based web-services.

The implementation is hosted on a SourceForge repository[17] and it is apparently *discontinued*, provided that the last commit dates back to 2009, and the home page lacks any sort of presentation or reference to publications or documentation. Nevertheless, the available source code appears well engineered and is *well documented*. It can be easily compiled into a `.war` file and then deployed on a Web Server (i.e. Apache Tomcat).

Once deployed, the web service is accessible via HTTP – making it is virtually interoperable with any programming language and platform – and can be tested by means of a common web browser. Additionally, the service exposes a WSDL description of the API needed to use it, which implies that a client library (aka stub) may be automatically generated using standard tools for service-oriented architectures. Nevertheless, this WSDL description is the only form of documentation when it comes to actually interact with the web-service.

*CiAN. Collaboration in Ad hoc Networks* (CiAN) [88] is a Workflow Management System (WfMS) enabling users to schedule and execute their custom workflow over MANETs. It comes with a reference architecture and a middleware. The middleware keeps track of the workflow state in a distributed way, and takes into account routing of tasks' input/output data, on top of a dynamic network topology where nodes communication is likely to be opportunistic.

Workflows in CiAN are modelled as directed graphs whose vertices represent *tasks*, and edges represent the data-flow from a task to its successors: when a task is completed, a result value is transferred through its outgoing edges. Conditions may be specified within task definitions stating, for instance, weather a task should wait for all its inputs or just for one of them.

Users can encode their workflow descriptions via a XML-based language to be endowed to an *initiator* singleton node, distributing the workflow to a number of *coordinator* nodes in charge of allocating tasks to the available *worker* nodes.

While the middleware is implemented in Java, tasks logic can be implemented virtually by means of any language since CiAN only assumes the application logic to interact with the middleware by means of the SOAP protocol, which provides great interoperability. Both the middleware's source code and its compiled version are distributed through CiAN website[18], together with detailed documentation and some runnable examples. The source code can be easily compiled, and both the obtained binaries and those publicly available can be run smoothly. The code is well documented and engineered. Nevertheless, the source code and documentation both date back to 2008: we therefore consider the project to be mature and usable, but no longer maintained.

*Piccola.* Piccola [2] is in its essence a *composition language*. It provides simple yet powerful abstractions: *forms* as immutable, prototype-like, key-value objects;

[17] https://sourceforge.net/projects/tripcom.
[18] http://mobilab.cse.wustl.edu/Projects/CiAN/Software/Software.shtml.

*services* as functional forms which can be invoked and executed; *agents* as concurrent services; and *channels* as inter-agent communication facilities. Virtually *any* interaction mechanism can be built by properly composing these abstractions, such as shared variables, push and pull streams, message-passing, publish-subscribe, and so on.

Nevertheless, a limitation is due to the fact that not solely the coordination mechanisms are to be programmed with the Piccola language, but *also* the coordinated entities. There is thus no possibility of integration with mainstream programming languages, which is a severe limitation for adoption. Additionally, even if Piccola comes with networking capabilities *virtually* enabling deployment to a distributed setting, there is no middleware facility available and no opportunity with integration with others is given, which is another factor likely to hinder Piccola adoption within the scope of distributed programming and coordination.

Piccola home page[19] is still available and collects a number of useful resources such as documentation pages and implementation. This comes in two flavours: JPiccola, based on Java, which reached version 3.7, and SPiccola, based on Smalltalk, which reached version 0.7. Source code is provided for the Java implementation only, which *correctly compiles and executes*.

Nevertheless, the project appears to be *discontinued*, given that the last commit on the source repository dates back to 2006.

*CRIME.* CRIME adheres to the *Fact Spaces* model, a variant of LINDA which absorbs transient federation of tuple space from Lime [66] for implementing mobile *fact spaces*—tuple spaces where tuples are logic facts and each tuple space is indeed a logic theory. Federated fact spaces are therefore seen as distributed knowledge bases.

In this sense, CRIME has some similarities with TuCSoN, which exploits first-order logic tuples both as the communication items and as the coordination laws. In this context, LINDA `out` and `in` primitives collapse into logic facts assertions and retractions, respectively.

Suspensive semantics is not regarded as being essential within the scope of the *Fact Spaces* model, since the focus is about programming fact spaces to react to information insertion/removal (or appearance/disappearance in case of transient federation). Accordingly, users can register arbitrary logic rules by means of a Prolog-like syntax. The head of such rules represent propositions which may be proved true (activated) or unknown (deactivated) given the current knowledge base by evaluating the body of the rule. Users can then plug arbitrary application logic reacting to (de)activation of these rules.

Implementation of CRIME is available on the project home page[20] and consists of an archive shipping pre-compiled Java classes with no attached source code. The software is apparently *no longer maintained*: the web page has been updated last in 2010, and the archive dates back to 2006. Nevertheless, the archive provides a number of example applications which have been tested and

[19] http://scg.unibe.ch/research/piccola.
[20] http://soft.vub.ac.be/amop/crime/introduction.

are still *correctly working*. No support is provided to application deployment and *no documentation* has been found describing how to deploy CRIME to an actual production environment.

*Klava-⋆.* With notation Klava-⋆ we refer to the family of models and technologies stemming from KLAIM [19] – such as O'Klaim [15] and MoMi [14] – which nowadays evolved into the X-Klaim/Klava framework [20].

X-Klaim consists of a domain-specific language and its compiler, which produces Java code by accepting X-Klaim sources as input. The produced code exploits the Klava library in turn, that is, the Java library implementing the middleware corresponding to the KLAIM model.

The overall framework explicitly targets code mobility, thus allowing both processes and data to migrate across a network. To do so, X-Klaim and Klava provide a first-class abstraction known as *locality*. Localities are of two sorts: either *physical*, such as network *nodes* identifiers, or *logical*, such as symbolic references to network nodes having a local semantics. Each locality hosts its own tuple space, and the processes therein interacting. The LINDA primitives supported by Klava are always explicitly or implicitly related to the tuple space hosted on a specific locality. Furthermore, processes are provided with primitives enabling them to migrate from a locality to another in a *strong* manner, that is, along with their execution state.

Both X-Klaim and Klava are distributed by means of the KLAIM Project home page[21], providing well detailed *documentation*. For what concerns X-Klaim, its C++ *source code* – dating back to 2004, date of the last edit, visible right below the title – is *publicly available* along with a self-configuring script meant to ease compilation. Nevertheless, we were not able to reproduce the compilation process on modern Linux distributions, seemingly due to some missing (and undocumented) dependency. No clues about how to fix the self-configuration process when it fails is provided, neither we were able to find some sort of documentation explicitly enumerating the compilation dependencies.

Conversely, the Klava library – actually implementing the coordination middleware – is distributed as a single `.jar` file containing both Java sources and the binaries. The `.jar` file dates back to 2004 likewise for X-Klaim, so it is apparently no longer developed, but further testing showed how the Klava library is still *functioning*, since it is self-contained and targets Java versions 1.4+.

*Limone.* Limone [40] is a model and middleware meant to improve scalability and security in Lime [66] through access control, and explicitly targeting distributed mobile systems and, in particular, agents roaming across ad-hoc networks built on top of opportunistically interconnected mobile devices.

Once two or more devices enter within their respective communication range and thus establish a connection, the agents running on top of them are (potentially) enabled to interact by means of transient sharing of their own tuple spaces. But, for some agents to be actually able to communicate, Limone states they should specify their *engagement* policies. An agent $A$'s engagement policy

[21] http://music.dsi.unifi.it/klaim.html.

determines which agents are allowed to interact with it and to which extent, that is, which primitives are allowed to be invoked. Agents satisfying the policy are registered within $A$'s *acquaintance* list. So, each agent only has to care about its acquaintance list, thus reducing the bandwidth requirements for the middleware.

A reactive programming mechanism completes the picture, enabling agents to inform their peer about their interest in tuples matching a given template, in order to be informed when such tuples becomes available.

The Limone technology is distributed by means of the project web page[22] in the form of a compressed archive containing the Java source code (dated back in 2004) and a `Makefile` for automatic build. Nevertheless, the code strictly requires to be compiled against a Java version *prior to 1.5*, and modern Java compilers do not support such an ancient version[23]. For these reasons, we could not proceed to further test the technology and we consider it to be no longer maintained *nor actually usable*.

RepliKlaim. RepliKlaim [3] is a variant of KLAIM [19] introducing first-class abstractions and mechanisms to deal with *data locality* and *consistency*, so as to give programmers the ability to explicitly account for and tackle these aspects when developing parallel computing applications. Specifically, the idea is to let the programmer specify and *coordinate replication of data*, and operate on replicas with a configurable level of consistency. This enables the programmer to adapt data distribution and locality to the needs of the application at hand, especially with the goal of improving *performance* in terms of concurrency level and data access speed—in spite of latencies due to distribution.

Most of the abstractions and mechanisms, as well as syntax elements and semantics, of RepliKlaim are exactly as in KLAIM, such as data repositories, processes, locations, and many actions. When due, actions are extended to explicitly deal with replication aspects, such as in the case of an `out` primitive putting multiple copies of the same tuple in multiple localities, or an `in` primitive removing all replicas from all locations at once. Also, various *degrees of consistency* among replicas in the same or different locations are achieved depending on whether primitives are synchronous (namely, atomically executed) or asynchronous.

There exists a *prototype* implementation of RepliKlaim on top of Klava, the Java implementation of KLAIM, available for direct download from a URL[24] given in its companion paper [3]. From there, a .rar archive is provided, containing a version of Klava and the *source* files implementing RepliKlaim, which can be easily compiled and run successfully.

Nevertheless, as stated in the paper describing RepliKlaim, its implementation currently relies on encoding its model in the standard KLAIM model, thus, on the practical side the code provided only features examples about how to translate RepliKlaim primitives into Klava. *No higher-level API* directly providing to developers the replica-oriented operations of RepliKlaim is provided.

[22] http://mobilab.cse.wustl.edu/projects/limone.

[23] As stated here: https://docs.oracle.com/javase/9/tools/javac.htm#JSWOR627.

[24] http://sysma.imtlucca.it/wp-content/uploads/2015/03/RepliKlaim-test-examples.rar.

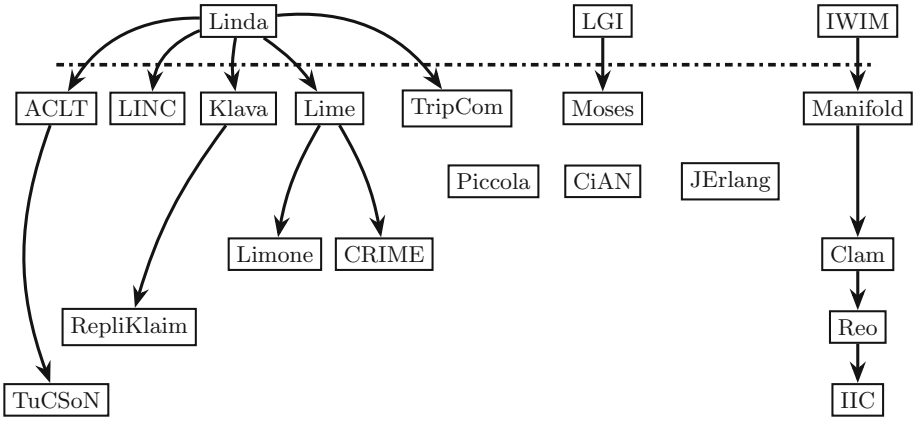


**Fig. 5.** Lines of evolution of selected technologies (below the dashed line), as stemming from a few archetypal coordination model (above the dashed line).

In other words, there exists no RepliKlaim Java library which can be imported to other java projects in order to exploit its provided coordination services.

## 3 Discussion

In this section we aim at providing further insights about the technologies described in Subsect. 2.3, especially to understand *(i)* whether they stem from a common archetypal coordination framework (Fig. 5), *(ii)* their relationships in terms of the features they provide (Fig. 6), and *(iii)* which goal mostly motivated their development and which application scenario they mostly target (Fig. 7).

*A Family Tree.* Figure 5 depicts a sort of "family tree" of the selected coordination technologies, emphasising how they stem from a few archetypal coordination models/languages, and how they are built on each other. It makes thus apparent how most of the technologies still available stem from two archetypal models: LINDA [44] and IWIM [5]. Nevertheless, whereas in the case of LINDA many heterogeneous extensions have been proposed throughout the years, focussing on different features and thus diverging from LINDA in many diverse ways, the evolution of the IWIM model appears much more homogeneous, featuring descendants which "linearly" extend their ancestors' features. Summing up, from LINDA stem the TuCSoN family, the Klaim [19] family – including Klava and RepliKlaim –, the LIME [74] family – with Limone and CRIME –, besides the lone runners LINC and TripCom, whereas from the IWIM root stems the Reo family—completed by Manifold, Clam, and the latest extension IIC.

Apart from these two big family trees, we have the LGI model, along with its implementation, Moses, and a small group of "lone runners" with unique features: Piccola, CiAN, and JErlang. While the former inspired some features

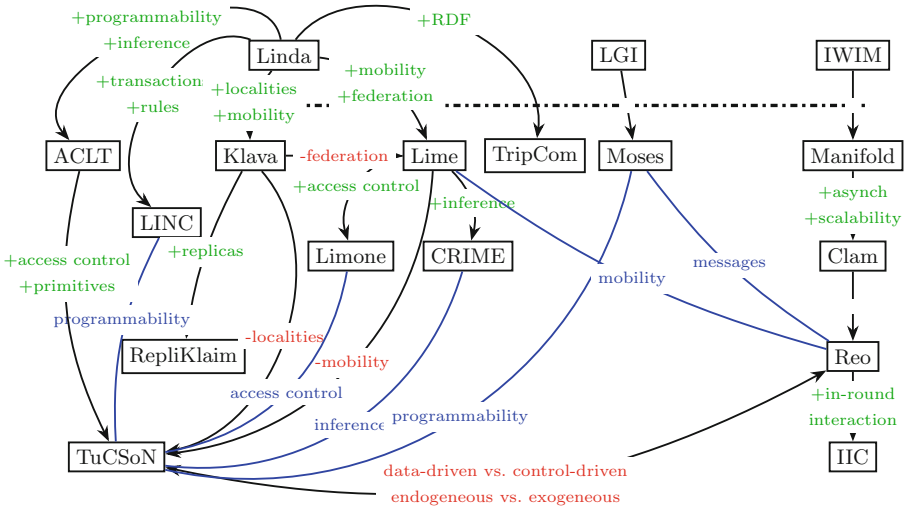


**Fig. 6.** Main differences (in green and red) and similarities (in blue) amongst selected technologies. Arrows indicate what it takes (in green, add something; in red, remove something) to go from one technology (the source) to another (the destination). (Color figure online)

of technologies stemming from other models – for instance, its *programmable laws* inspired essentially any other technology or model having *reactive rules* of some sort, such as LINC –, the latter remained mostly confined to itself.

It is interesting to notice how "the IWIM family" and "the LINDA family" remained well-isolated one from each other over all these years. Whereas this can be easily attributed to the fundamental difference in the approach to coordination they have – data-driven vs. control-driven, as also emphasised in Fig. 6 – it seems odd that nobody tried to somewhat integrate these two extremely successful coordination models, in an attempt to improve the state of art by cherry-picking a few features from both to create a novel, *hybrid* coordination model [69], with "the best of two worlds". To some extent, the TuCSoN model, along with its coordination language, ReSpecT, pursues this path: ReSpecT in fact can be regarded as a data-driven model because coordination is based on availability of tuples, as in LINDA, but, at the same time, coordination policies are enforced by declarative specifications which *control* the way in which the coordination medium behaves, thus, ultimately, how the coordinated components interact—as typical for control-driven models like IWIM.

We believe that the path toward integration could be the key in further perfecting and improving coordination models and languages, by complementing data-driven models elegance and flexibility with control-driven models fine-grained control and predictability.

*Families Marriage.* Figure 6 enriches the family tree just described with relationships indicating *differences* (red and green arrows) and *similarities* (blue arrows) in features provided—notice that w.r.t. Fig. 5 Piccola, CiAN, and JErlang have been removed because they are so unique that no clear relationship may be found with other technologies. As already mentioned for Fig. 5, LINDA has been taken as the common ground for many technologies which are instead very heterogeneous in the aim pursued: if $\mathcal{ACLT}$, TuCSoN, and LINC have a LINDA core enriched with many other features – such as programmability, transactionality, and novel primitives –, the KLAIM family and the LIME one diverge more, by changing the way in which primitives behave – as in the case of localities in KLAIM –, or the way in which the interacting processes see each others' tuple spaces—as for LIME transient federation.

Nevertheless, technologies which may appear as being far apart from each other have interesting similarities, as in the case of the interaction rules of LGI, thus Moses, which strongly resembles $\mathcal{ACLT}$ and TuCSoN reactions, or the fact that both the Reo family and Moses are based on message passing. Or, the fact that both CRIME and TuCSoN rely on logic tuples so as to leverage on the inference capabilities of interacting agents, while Reo and both LIME and KLAIM take into account mobility of processes and coordination abstractions (tuple spaces vs. channels) as a first-class citizen.

It is worth emphasising here that Fig. 6 highlights the features to which more attention has been devoted throughout the years: programmability, access control, and mobility. We believe that these features, possibly extended with scalability and inference capabilities, are crucial for widening applicability of coordination technologies to real-world scenarios. For instance, the Internet of Things (IoT) [9] – along with its variants Web of Things [48] and Internet of Intelligent Things [8] – is a very good fit for testing coordination technologies, and requires precisely the aforementioned features.

*Goals & Preferred Scenarios.* Finally, Fig. 7 relates the selected technologies with the main aim pursued which motivates their extension in a particular direction, along with the applications scenario they best target.

From the description of the selected technologies we gathered, two are the main goals motivating their evolution: *(i)* providing *flexibility* so as to deal with the majority of heterogeneous application scenarios possible, and *(ii)* focussing on first-class abstractions for better supporting *space-awareness* of both the coordination abstractions and the interacting processes.

In fact, TuCSoN/$\mathcal{ACLT}$, LINC, and Moses all provide means to somewhat *program* the coordinative behaviour of the coordination medium, thus aim at making it configurable, adaptable, malleable, even at run-time, and/or provide additional coordination primitives to expand the expressive reach of the coordination technology. The Klaim family, the Reo family, and the Lime family instead, are geared toward some forms of *space-awareness*, be it by promoting mobility or by providing location-sensitive primitives.

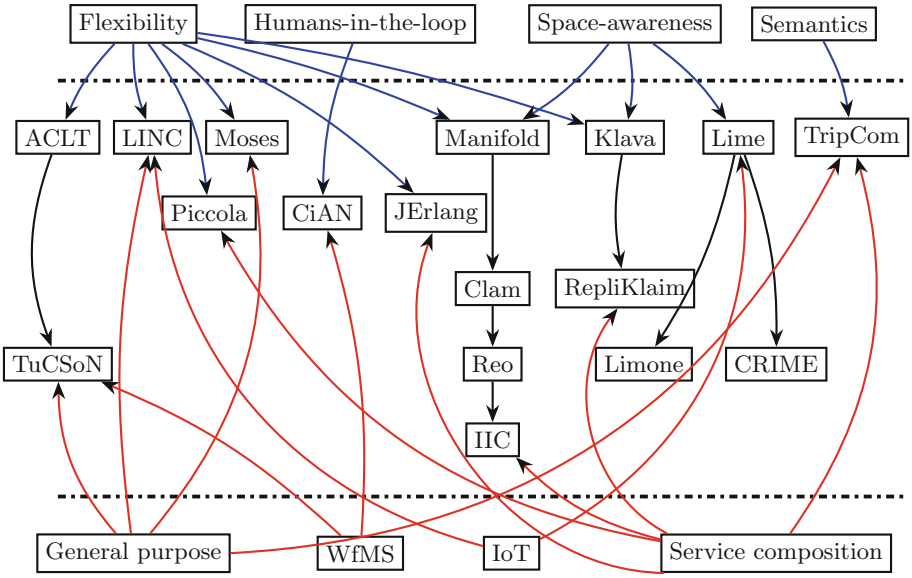


**Fig. 7.** Selected technologies per main *goal* pursued (top, blue arrows) and preferred *application scenario* (bottom, red arrows). (Color figure online)

Besides these, two more main goals can be devised, peculiar to specific technologies: *(iii)* supporting *humans-in-the-loop*, in the case of CiAN, and *(iv)* provide a *semantic* representation of data items, in the case of TripCom.

About the application scenarios explicitly declared as of particular interest for the technology, the most prominent one is *service composition*, which is especially interesting for Piccola, JErlang, the Reo family, the Klaim family, and TripCom—besides being naturally applicable to all other technologies too. Then, whereas technologies such as LINC and the Lime family are mainly tailored to the *IoT landscape*, being meant to cope with the requirements posed by small, possibly portable, possibly embedded devices with low resources, *Workflow Management* (WfMS) is peculiar to CiAN, while also considered by TuCSoN [79]. Besides these application scenarios, there are many technologies without a specific focus, although they have been applied to many different ones, such as TuCSoN itself, LINC, Moses, and TripCom: these have been associated with the generic "General purpose" scenario.

We believe that the goals and application scenarios just highlighted strengthen our previous consideration that the IoT could be the "killer-app" for coordination technologies. In fact, flexibility (there including programmability and configurability), space-awareness (there including mobility and location-awareness), and semantics (there including interoperability of data representation formats) are all necessary ingredients for any non-trivial IoT deployment: the former helps in dealing with *uncertainty* and *unpredictability* typical of the IoT scenarios, the latter is required for building open IoT systems, and some

form of space-awareness is a common feature of many IoT deployments, from retail to industry 4.0. Also, the fact that service composition has been already thoroughly explored is a great advantage and the perfect starting point for tackling IoT challenges: both the IoT and the Web of Things vision foster a world where connected objects provide and consume services, which can be composed in increasingly high-level ones.

## 4 Conclusion

The main aim of this paper is to provide insights about the state-of-the-art of coordination technologies after twenty years of the COORDINATION conference series, and to stimulate informed discussion about future perspectives. Overall, apart from some notable success stories – i.e. the commercial success of LINC along with the active development of TuCSoN and Reo – most coordination technologies have gone through a rapid and effective development at the time they were presented, then lacked further improvements or even maintenance of its usability, thus never reached a wider audience—i.e. outside the COORDINATION community or in the industry. Obviously, something also happens outside the COORDINATION boundaries. For instance, coordination technologies are surveyed in [73], whereas [81] focuses on tuple-based technologies. However, mostly of the technological developments reported here just happened after those survey were published, in 2001 [72].

Although we acknowledge that researchers are usually mostly concerned with providing scientifically-relevant models rather than production-ready software, we also believe that backing up models and languages with more then proof-of-concept software is crucial to promote wider adoption of both the technology itself and the models, which in turn may provide invaluable feedback to researchers for further developing and tuning models. The next decade will probably tell us more about the actual role of coordination technologies in the development of forthcoming application scenarios: the IoT, for instance, is at the "peak of inflated expectations" according to Gartner's hype cycle for 2017, and is expected to reach the plateau in 2 to 5 years. This means the time is ripe for pushing forward the development of coordination technologies, so as to have them ready when the IoT will be mature enough to actually benefit from their added value.

Besides coordination technologies, we believe the COORDINATION conference is quite healthy: although the number of published papers is decreasing, citations and downloads grows (modulo too recent years), and contributions conveying technological advancements still represent almost a half of all the contributions.

## References


1. Abreu, J., Fiadeiro, J.L.: A coordination model for service-oriented interactions. In: Lea and Zavattaro [56], pp. 1–16
2. Achermann, F., Kneubuehl, S., Nierstrasz, O.: Scripting coordination styles. In: Porto and Roman [76], pp. 19–35
3. Andrić, M., De Nicola, R., Lafuente, A.L.: Replica-based high-performance tuple space computing. In: Holvoet and Viroli [49], pp. 3–18
4. Ao, X., Minsky, N., Nguyen, T.D., Ungureanu, V.: Law-Governed Internet communities. In: Porto and Roman [76], pp. 133–147
5. Arbab, F.: The IWIM model for coordination of concurrent activities. In: Ciancarini and Hankin [24], pp. 34–56
6. Arbab, F., Mavaddat, F.: Coordination through channel composition. In: Arbab and Talcott [7], pp. 22–39
7. Arbab, F., Talcott, C. (eds.): COORDINATION 2002. LNCS, vol. 2315. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46000-4
8. Arsénio, A., Serra, H., Francisco, R., Nabais, F., Andrade, J., Serrano, E.: Internet of intelligent things: bringing artificial intelligence into things and communication networks. In: Xhafa, F., Bessis, N. (eds.) Inter-cooperative Collective Intelligence: Techniques and Applications. SCI, vol. 495, pp. 1–37. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-35016-0_1
9. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
10. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in reo by constraint automata. Sci. Comput. Program. **61**(2), 75–113 (2006)
11. Banâtre, J.-P., Fradet, P., Le Métayer, D.: Gamma and the chemical reaction model: fifteen years after. In: Calude, C.S., PĂun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2000. LNCS, vol. 2235, pp. 17–44. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45523-X_2
12. Banville, M.: Sonia: an adaptation of Linda for coordination of activities in organizations. In: Ciancarini and Hankin [24], pp. 57–74
13. Bellifemine, F.L., Poggi, A., Rimassa, G.: JADE–a FIPA-compliant agentframework. In: 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-99), pp. 97–108 (1999)
14. Bettini, L., Bono, V., Venneri, B.: Coordinating mobile object-oriented code. In: Arbab and Talcott [7], pp. 56–71
15. Bettini, L., Bono, V., Venneri, B.: O'Klaim: a coordination language with mobile mixins. In: De Nicola et al. [33], pp. 20–37
16. Bettini, L., De Nicola, R.: Mobile distributed programming in X-Klaim. In: Bernardo, M., Bogliolo, A. (eds.) SFM-Moby 2005. LNCS, vol. 3465, pp. 29–68. Springer, Heidelberg (2005). https://doi.org/10.1007/11419822_2
17. Bettini, L., De Nicola, R., Falassi, D., Lacoste, M., Lopes, L., Oliveira, L., Paulino, H., Vasconcelos, V.T.: A software framework for rapid prototyping of run-time systems for mobile calculi. In: Priami, C., Quaglia, P. (eds.) GC 2004. LNCS, vol. 3267, pp. 179–207. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31794-4_10
18. Bettini, L., De Nicola, R., Loreti, M.: Implementing session centered calculi. In: Lea and Zavattaro [56], pp. 17–32
19. Bettini, L., Loreti, M., Pugliese, R.: An infrastructure language for open nets. In: 2002 ACM Symposium on Applied Computing (SAC 2002), pp. 373–377. ACM, New York (2002)

20. Bettini, L., Nicola, R.D., Pugliese, R.: X-Klaim and Klava: programming mobile code. Electron. Notes Theor. Comput. Sci. **62**, 24–37 (2002)
21. Bordini, R.H., Hübner, J.F., Wooldridge, M.J.: Programming Multi-Agent Systems in AgentSpeak Using Jason. Wiley, Chichester (2007)
22. Bryce, C., Oriola, M., Vitck, J.: A coordination model for agents based on secure spaces. In: Ciancarini and Wolf [26], pp. 4–20
23. Calegari, R., Denti, E.: Building smart spaces on the home manager platform. ALP Newslett. (2016). https://www.cs.nmsu.edu/ALP/2016/12/building-smart-spaces-on-the-home-manager-platform/
24. Ciancarini, P., Hankin, C. (eds.): COORDINATION 1996. LNCS, vol. 1061. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61052-9
25. Ciancarini, P., Rossi, D.: Jada: coordination and communication for Java agents. In: Vitek, J., Tschudin, C. (eds.) MOS 1996. LNCS, vol. 1222, pp. 213–226. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-62852-5_16
26. Ciancarini, P., Wolf, A.L. (eds.): COORDINATION 1999. LNCS, vol. 1594. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48919-3
27. Ciatto, G., Mariani, S., Omicini, A.: Programming the interaction space effectively with ReSpecT𝕏. In: Ivanović, M., Bădică, C., Dix, J., Jovanović, Z., Malgeri, M., Savić, M. (eds.) IDC 2017. SCI, vol. 737, pp. 89–101. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-66379-1_9
28. Cremonini, M., Omicini, A., Zambonelli, F.: Coordination in context: authentication, authorisation and topology in mobile agent applications. In: Ciancarini and Wolf [26], p. 416
29. Cruz, J.C., Ducasse, S.: A group based approach for coordinating active objects. In: Ciancarini and Wolf [26], pp. 355–370
30. De Angelis, F.L., Di Marzo Serugendo, G.: Logic Fragments: a coordination model based on logic inference. In: Holvoet and Viroli [49], pp. 35–48
31. De Bosschere, K., Jacquet, J.M.: $\mu$2Log: towards remote coordination. In: Ciancarini and Hankin [24], pp. 142–159
32. De Meuter, W., Roman, G.-C. (eds.): COORDINATION 2011. LNCS, vol. 6721. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21464-6
33. De Nicola, R., Ferrari, G.-L., Meredith, G. (eds.): COORDINATION 2004. LNCS, vol. 2949. Springer, Heidelberg (2004). https://doi.org/10.1007/b95570
34. Dedecker, J., Van Cutsem, T., Mostinckx, S., D'Hondt, T., De Meuter, W.: Ambient-oriented programming. In: Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, pp. 31–40. ACM, New York (2005)
35. Denti, E., Natali, A., Omicini, A., Venuti, M.: An extensible framework for the development of coordinated applications. In: Ciancarini and Hankin [24], pp. 305–320
36. Denti, E., Omicini, A., Ricci, A.: tuProlog: a light-weight prolog for internet applications and infrastructures. In: Ramakrishnan, I.V. (ed.) PADL 2001. LNCS, vol. 1990, pp. 184–198. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45241-9_13
37. Dubovitskaya, A., Urovi, V., Barba, I., Aberer, K., Schumacher, M.I.: A multiagent system for dynamic data aggregation in medical research. BioMed Res. Int. **2016** (2016). https://www.hindawi.com/journals/bmri/2016/9027457/
38. Ducasse, S., Hofmann, T., Nierstrasz, O.: Openspaces: an object-oriented framework for reconfigurable coordination spaces. In: Porto and Roman [76], pp. 1–18

39. Fensel, D.: Triple-space computing: semantic web services based on persistent publication of information. In: Aagesen, F.A., Anutariya, C., Wuwongse, V. (eds.) INTELLCOMM 2004. LNCS, vol. 3283, pp. 43–53. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30179-0_4
40. Fok, C.L., Roman, G.C., Hackmann, G.: A lightweight coordination middleware for mobile computing. In: De Nicola et al. [33], pp. 135–151
41. Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 372–385. ACM (1996)
42. Fukuda, M., Bic, L.F., Dillencourt, M.B., Merchant, F.: Intra- and inter-object coordination with MESSENGERS. In: Ciancarini and Hankin [24], pp. 179–196
43. Garlan, D., Le Métayer, D. (eds.): COORDINATION 1997. LNCS, vol. 1282. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63383-9
44. Gelernter, D.: Generative communication in Linda. ACM Trans. Program. Lang. Syst. (TOPLAS) **7**(1), 80–112 (1985)
45. Gilmore, S., Hillston, J.: The PEPA workbench: a tool to support a process algebra-based approach to performance modelling. In: Haring, G., Kotsis, G. (eds.) TOOLS 1994. LNCS, vol. 794, pp. 353–368. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58021-2_20
46. van der Goot, R., Schaeffer, J., Wilson, G.V.: Safer tuple spaces. In: Garlan and Le Métayer [43], pp. 289–301
47. Hendler, J.A.: Agents and the Semantic web. IEEE Intell. Syst. **16**(2), 30–37 (2001)
48. Heuer, J., Hund, J., Pfaff, O.: Toward the web of things: applying web technologies to the physical world. Computer **48**(5), 34–42 (2015)
49. Holvoet, T., Viroli, M. (eds.): COORDINATION 2015. LNCS, vol. 9037. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19282-6
50. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0053567
51. Hu, R., Yoshida, N., Honda, K.: Session-based distributed programming in Java. In: Vitek, J. (ed.) ECOOP 2008. LNCS, vol. 5142, pp. 516–541. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70592-5_22
52. Jagannathan, S.: Communication-passing style for coordination languages. In: Garlan and Le Métayer [43], pp. 131–149
53. Jamison, W.C., Lea, D.: TRUCE: agent coordination through concurrent interpretation of role-based protocols. In: Ciancarini and Wolf [26], pp. 384–398
54. Jongmans, S.S.T.Q., Santini, F., Sargolzaei, M., Arbab, F., Afsarmanesh, H.: Orchestrating web services using Reo: from circuits and behaviors to automatically generated code. SOCA **8**(4), 277–297 (2014)
55. Kokash, N., Krause, C., de Vink, E.: Reo + mCRL2: a framework for model-checking dataflow in service compositions. Formal Aspects Comput. **24**(2), 187–216 (2012)
56. Lea, D., Zavattaro, G. (eds.): COORDINATION 2008. LNCS, vol. 5052. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68265-3
57. Liptchinsky, V., Khazankin, R., Truong, H.L., Dustdar, S.: Statelets: coordination of social collaboration processes. In: Sirjani [90], pp. 1–16
58. Louvel, M., Pacull, F.: LINC: a compact yet powerful coordination environment. In: Kühn, E., Pugliese, R. (eds.) COORDINATION 2014. LNCS, vol. 8459, pp. 83–98. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43376-8_6

59. Louvel, M., Pacull, F., Rutten, E., Sylla, A.N.: Development tools for rule-based coordination programming in LINC. In: Jacquet, J.-M., Massink, M. (eds.) COORDINATION 2017. LNCS, vol. 10319, pp. 78–96. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59746-1_5
60. Mariani, S., Omicini, A.: Coordination mechanisms for the modelling and simulation of stochastic systems: the case of uniform primitives. SCS M&S Mag. **IV**(3), 6–25 (2014)
61. Mariani, S., Omicini, A.: Multi-paradigm coordination for MAS: integrating heterogeneous coordination approaches in MAS technologies. In: Santoro, C., Messina, F., De Benedetti, M. (eds.) WOA 2016 – 17th Workshop "From Objects to Agents", CEUR-WS.org, vol. 1664, pp. 91–99. Sun SITE Central Europe, 29–30 July 2016
62. Mariani, S., Omicini, A., Sangiorgi, L.: Models of autonomy and coordination: integrating subjective and objective approaches in agent development frameworks. In: Camacho, D., Braubach, L., Venticinque, S., Badica, C. (eds.) Intelligent Distributed Computing VIII. SCI, vol. 570, pp. 69–79. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10422-5_9
63. Merrick, I., Wood, A.: Scoped coordination in open distributed systems. In: Porto and Roman [76], pp. 311–316
64. Minsky, N.H., Leichter, J.: Law-Governed Linda as a coordination model. In: Ciancarini, P., Nierstrasz, O., Yonezawa, A. (eds.) ECOOP 1994. LNCS, vol. 924, pp. 125–146. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59450-7_8
65. Mostinckx, S., Scholliers, C., Philips, E., Herzeel, C., De Meuter, W.: Fact spaces: coordination in the face of disconnection. In: Murphy and Vitek [67], pp. 268–285
66. Murphy, A.L., Picco, G.P., Roman, G.C.: LIME: a coordination model and middleware supporting mobility of hosts and agents. ACM Trans. Softw. Eng. Method. (TOSEM) **15**(3), 279–328 (2006)
67. Murphy, A.L., Vitek, J. (eds.): COORDINATION 2007. LNCS, vol. 4467. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72794-1
68. Ng, N., Yoshida, N., Pernet, O., Hu, R., Kryftis, Y.: Safe parallel programming with session Java. In: De Meuter and Roman [32], pp. 110–126
69. Omicini, A.: Hybrid coordination models for handling information exchange among internet agents. In: Bonarini, A., Colombetti, M., Lanzi, P.L. (eds.) Workshop "Agenti intelligenti e Internet: teorie, strumenti e applicazioni", 7th AI*IA Convention (AI*IA 2000), Milano, Italy, pp. 1–4, 13 September 2000
70. Omicini, A.: Formal ReSpecT in the A&A perspective. Electron. Notes Theor. Comput. Sci. **175**(2), 97–117 (2007)
71. Omicini, A., Denti, E.: From tuple spaces to tuple centres. Sci. Comput. Program. **41**(3), 277–294 (2001)
72. Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R. (eds.): Coordination of Internet Agents: Models, Technologies, and Applications. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-662-04401-8
73. Papadopoulos, G.A.: Models and technologies for the coordination of Internet agents: a survey. In: Omicini et al. [72], chap. 2, pp. 25–56
74. Picco, G.P., Murphy, A.L., Roman, G.C.: LIME: Linda meets mobility. In: 1999 International Conference on Software Engineering (ICSE 1999), pp. 368–377, May 1999
75. Plociniczak, H., Eisenbach, S.: JErlang: Erlang with joins. In: Clarke, D., Agha, G. (eds.) COORDINATION 2010. LNCS, vol. 6116, pp. 61–75. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13414-2_5
76. Porto, A., Roman, G.-C. (eds.): COORDINATION 2000. LNCS, vol. 1906. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45263-X

77. Proença, J., Clarke, D.: Interactive interaction constraints. In: De Nicola, R., Julien, C. (eds.) COORDINATION 2013. LNCS, vol. 7890, pp. 211–225. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38493-6_15
78. Proença, J., Clarke, D., de Vink, E., Arbab, F.: Dreams: a framework for distributed synchronous coordination. In: 27th Annual ACM Symposium on Applied Computing (SAC 2012), pp. 1510–1515. ACM, New York (2012)
79. Ricci, A., Omicini, A., Denti, E.: Virtual enterprises and workflow management as agent coordination issues. Int. J. Coop. Inf. Syst. **11**(3/4), 355–379 (2002)
80. Rossi, D.: A social software-based coordination platform. In: Sirjani [90], pp. 17–28
81. Rossi, D., Cabri, G., Denti, E.: Tuple-based technologies for coordination. In: Omicini et al. [72], chap. 4, pp. 83–109
82. Rossi, D., Vitali, F.: Internet-based coordination environments and document-based applications: a case study. In: Ciancarini and Wolf [26], pp. 259–274
83. Rowstron, A.I.T.: Bulk primitives in Linda run-time systems. Ph.D. thesis, The University of York (1996)
84. Rowstron, A.I.T.: Using asynchronous tuple-space access primitives (bonita primitives) for process co-ordination. In: Garlan and Le Métayer [43], pp. 426–429
85. Rowstron, A.I.T.: WCL: a co-ordination language for geographically distributed agents. World Wide Web **1**(3), 167–179 (1998)
86. Sample, N., Beringer, D., Melloul, L., Wiederhold, G.: CLAM: Composition language for autonomous megamodules. In: Ciancarini and Wolf [26], pp. 291–306
87. Schumacher, M., Chantemargue, F., Hirsbrunner, B.: The STL++ coordination language: a base for implementing distributed multi-agent applications. In: Ciancarini and Wolf [26], pp. 399–414
88. Sen, R., Roman, G.C., Gill, C.: CiAN: a workflow engine for MANETs. In: Lea and Zavattaro [56], pp. 280–295
89. Simperl, E., Krummenacher, R., Nixon, L.: A coordination model for triplespace computing. In: Murphy and Vitek [67], pp. 1–18
90. Sirjani, M. (ed.): COORDINATION 2012. LNCS, vol. 7274. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30829-1
91. Tarau, P.: Coordination and concurrency in multi-engine Prolog. In: De Meuter and Roman [32], pp. 157–171
92. Tolksdorf, R.: Coordinating services in open distributed systems with Laura. In: Ciancarini and Hankin [24], pp. 386–402
93. Tolksdorf, R.: Berlinda: An object-oriented platform for implementing coordination languages in Java. In: Garlan and Le Métayer [43], pp. 430–433
94. Tolksdorf, R., Rojec-Goldmann, G.: The SPACETUB models and framework. In: Arbab and Talcott [7], pp. 348–363
95. Varela, C., Agha, G.: A hierarchical model for coordination of concurrent activities. In: Ciancarini and Wolf [26], pp. 166–182
96. Viroli, M., Omicini, A.: Coordination as a service. Fundamenta Informaticae **73**(4), 507–534 (2006)
97. Zambonelli, F., Omicini, A., et al.: Developing pervasive multi-agent systems with nature-inspired coordination. Pervasive Mob. Comput. **17**, 236–252 (2015)