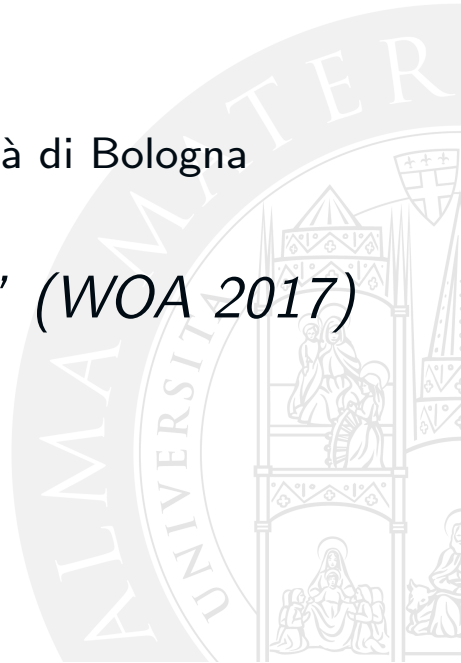
Micro-intelligence for the IoT

Teaching the Old Logic Dog New Programming Tricks

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI), Università di Bologna

Invited Talk @ *18th Workshop "From Objects to Agents" (WOA 2017)*
Scilla, RC, Italy, 16 June 2017



Abstract

New application scenarios for pervasive intelligent systems open novel perspectives for logic-based approaches, in particular when coupled with agent-based technologies and methods. In this explorative talk we provide some examples of how logic programming and its extensions can work as sources of micro-intelligence for the IoT, at both the individual and the collective level, along with an overall architectural view of IoT systems exploiting logic-based technologies.

1. Trends
   - New Scenarios
   - New Technologies
   - New Requirements
2. The Logic Programming Road to Micro-intelligence
   - Faux Pas
   - Steps in the Right Directions
   - LP for the IoIT
3. Further Work & Conclusion

Next in Line...

Focus on...

Knowledge-Intensive Environments (KIE)

Information galore

- ubiquitous *knowledge sources* growing in number and size
- *knowledge prosumers* in socio-technical systems (STS) [Whitworth, 2006]
- application *goals* more and more *depending on information* available in the environment [Mariani and Omicini, 2013]

Pervasive Intelligence

Intelligence everywhere

- computation is everywhere around us
- software components are required to *behave intelligently*, understanding their own goals and the context where they have to work
- integration of software components is supposed to add *social intelligence*, possibly through coordination [Castelfranchi, 1998]
- pervasive computing nowadays calls for ubiquitous intelligence

Internet of Intelligent Things (IoIT)

Intelligent objects in the IoT

- our everyday physical objects should be able to network in the Internet of Things (IoT) [Gubbi et al., 2013, Atzori et al., 2010, Fortino et al., 2014]
- they are required to understand each other, to learn, to understand situations, to understand us [Lippi et al., 2017]
- our *everyday object* should be(come) intelligent in the *Internet of Intelligent Things* (IoIT) [Arsénio et al., 2014]

Focus on...

Agent-Oriented Computing

Agents & multi-agent systems (MAS)

- agents are the most viable abstraction to encapsulate fundamental features such as *control*, *goals*, *mobility*, *intelligence*

  [Zambonelli and Omicini, 2004]

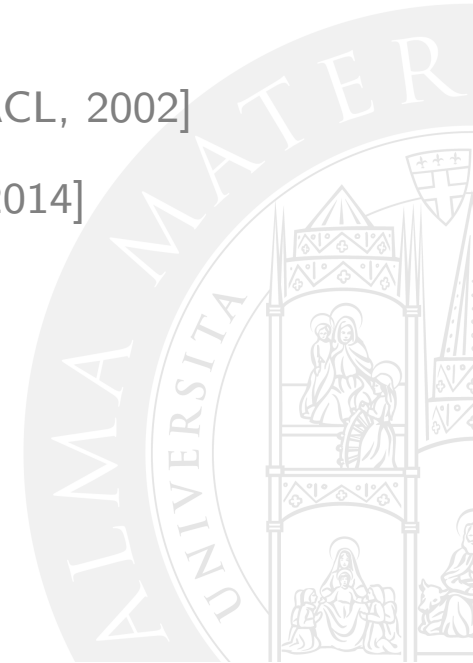
- MAS abstractions such as *society* and *environment* are essential to cope with the complexity of nowadays application scenarios

  [Omicini and Mariani, 2013]

- agent-oriented models and technologies are gaining momentum for embedding decentralised intelligence [Singh and Chopra, 2017]

Semantic Technologies

- Semantic Web to effectively share information between humans and software agents
- ontologies
- standard inter-agent communication languages [FIPA ACL, 2002]
- conversational technologies and systems [Nishida et al., 2014]



Everything as a Service (XaaS)

- in order to promote maximum availability and interoperability, any resource of any sort should be accessible *as a service*
- possibly, an *intelligent* one
- through standard network operations

. . . and Many Others

- ubiquitous artificial intelligence
- Big Data
- intelligente sensing
- . . .

Focus on...

Observability

- *observability* is essential for understanding execution of distributed systems [Tanenbaum and van Steen, 2007]—as in the case of stateless communication in REST
- *mutual observation* is a pre-condition for many forms of cognitive agent coordination—e.g., behavioural implicit communication (BIC) [Castelfranchi et al., 2010]
- inspectability an essential feature for *online engineering* [Fredriksson and Gustavsson, 2004]

Malleability

- the ability of changing / adapting at run time is a pre-condition for most forms of self-adaptive / self-organising systems

e.g. malleability of coordination artefacts allows for self-organisation patterns in MAS [Viroli et al., 2005]

e.g. malleability of a Prolog engine allow for adaptive intelligent middleware [Piancastelli et al., 2008]

Understandability

We need explanations

- it is not enough for systems to work
- we need to understand them—otherwise, it is not different from magic

e.g. decision support systems need not just to propose solutions, but also to motivate them

e.g. the problem with deep learning

e.g. engineering systems with self-organisation techniques mandates some for of system predictability

Formalisability

We need to be sure

- it is not enough for systems to work
- we need to know how they will behave—otherwise, it could be dangerous, or, unacceptable, or, ...

e.g. we like to add features of *self-organising systems* to our systems—but, we need to foresee their evolution over time, at least in general terms

e.g. complex systems are generally *unpredictable*, however we would need at least partial predictability of their behaviour when we engineer them

Norm Compliance

Systems & norms

- if computational systems are pervasive, and affect every aspects of our everyday life, they are subject of all the *norms* ruling human individual and social behaviour
- *norm compliance* should be enforced by computational systems, starting from the earliest phases of software engineering
- *normative reasoning* should become part of the intelligent behaviour of components and systems
- accountability, traceability, and responsibility should be part of each component and every system

Widespread Intelligence

Component- & system level

- every single component is expected to behave and interact intelligently
- every group of interacting components – e.g., agent societies – should integrate so as to produce intelligent collective behaviours
- overall system behaviour—ok, intelligent

! of course, the diverse levels are connected, but do not imply each other

Next in Line...

Focus on...

Why Logic Programming for Intelligent Systems? I

General features of Logic Programming (LP)

- computation as deduction
- declarative and procedural interpretation match
- reasoning about what is true rather than about what to do
- logic theories as programs and knowledge bases
- programming with relations and inferences
- non-typed tree structures for uniform data representation
- unification as a non-directional mechanism for message passing
- single-assignment variables, step-by-step refinement
- reasoning with incomplete information
- non determinism
- interactive computational model
- ...

Why Logic Programming for Intelligent Systems? II

Overall

LP languages and technologies

represent in principle the most natural candidates for

injecting intelligence

within computational systems

Why Logic Programming for Distributed Systems? I

Early days

- since the early days of logic programming, features like
  - high-level languages
  - non-determinism
  - referential transparency

  made the potential for exploitation of parallelism in the execution of logic programs clearly emerge [Gupta et al., 2001]
- at the same time, the articulation of logic computation, with the frequent occurrence of
  - heavy computational load
  - non-trivial computational structures

  made the computational techniques developed for logic languages relevant for the general scenario of concurrent / parallel computation

Why Logic Programming for Distributed Systems? II

Or- & and-parallelism

- logic languages like Prolog allow for parallel evaluation
  - the proof tree could be explored parallel / concurrently
  - even more, the potential computational complexity of the proof tree actually *encourages* parallel exploration
- basically, Prolog supports two sorts of parallelism
  - *and-parallelism*
  - *or-parallelism*

fitting distributed computing

Logic Programming: Issues I

Why did LP never *really* take off?

- computational costs too high, machinery too huge
- not really suited for programming in the large—intrinsic modularity (predicates) does not really scales up, modules are not enough
- a novel programming paradigm requires huge efforts by programmers
- "no types" makes it difficult (also) to deal with domain-specific applications
- how to deal with non determinism?
- distance from mainstream programming paradigms potentially harms conceptual integrity
- troublesome integration with mainstream technologies
- ...

Logic Programming: Issues II

LP for distributed systems?

- mostly, distributed computing for LP—to share the huge computational load
- distributed computing is *not* the same as distributed systems, at least according to their historical meanings—per se, LP does not work for distributed systems
- interactive computational model works locally—one user querying one engine
- the universal notion of consistency of logic theory does not cope well with the incompleteness and inconsistency intrinsically implied by the needs of distributed pervasive systems
- LP technology does not always integrate well with available tools and technologies for distributed pervasive systems

Focus on...

Ubiquitous Declarative Languages and Technologies

Declarative languages and technologies

- SQL and its derivatives are everywhere in databases and knowledge sources of any sort
- XML and its applications are the de facto standard for the Web as well as for most of the novel application scenarios at any level
- even the shift in object-oriented programming from classes to interfaces in some sense represents a shift towards declarativeness
- declarative models and technologies play an essential roles in agent-oriented computing [Omicini and Zambonelli, 2004]

New LP Technologies

Essential features

- minimal and configurable engines running without heavy computational costs [Denti et al., 2001]
- multi-platform technology, including resource-constrained devices [Denti et al., 2013]
- interoperability with widespread languages, technologies, and standards
- clean model integration for conceptual integrity [Denti et al., 2005]

e.g. tuProlog `http://tuprolog.unibo.it`

Distributing Logic Programs

Concurrent logic processes

- processes represented by logic *processes* in concurrent systems
  - logic *agents* in the acceptation of coordination models [Ciancarini, 1996]
- there, concurrent / distributed systems are first of all collection of logic engines running in a concurrent / distributed way [Robertson, 2004]

e.g. Shared Prolog [Brogi and Ciancarini, 1991], $\mathcal{ACLT}$ [Denti et al., 1996]

$\rightarrow$ moving LP towards *distributed systems* and MAS [Baldoni et al., 2010]

Logic-based Coordination

Logic tuple spaces

- the space of interaction is built around *logic tuple spaces*
  [Ciancarini, 1994, Denti et al., 1996]
- allowing for heterogeneous distributed processes to be coordinated
- blackboards and programmable logic tuple spaces promote logic-based coordination of distributed systems

e.g. Shared Prolog [Brogi and Ciancarini, 1991], ReSpecT [Omicini and Denti, 2001]

Focus on...

Micro-intelligence for the IoT

- IoT scenarios (and CPS as well) mandates for distributed & situated micro-intelligence
  - huge numbers of small unit of computation
  - situated within a spatially-distributed environment
  - promoting the local exploitation of high-level symbolic languages
  - with inferential capabilities

? what role could the LP models and technologies play in such scenarios?

? what LP for the IoIT? [Arsénio et al., 2014]

Distributing Intelligence through Logic Engines I

Distributed logic engines

- lightweight and interoperable Prolog engines could be distributed even on resource-constrained devices [Denti et al., 2013]
- *multiple logic theories* are then scattered around, encapsulated in each engine, and associated to individual computational devices and *things* in the IoT
- each logic theory is *situated*, and represents what is true *locally*, according to a simple paraconsistent interpretation
- LP *resolution process* is local to each theory / engine, so it is both standard and consistent [Robinson, 1965]

Distributing Intelligence through Logic Engines II

Our example: tuProlog

tuProlog `http://tuprolog.unibo.it`

- is a light-weight Prolog system for distributed applications and infrastructures [Denti et al., 2001]
- is intentionally designed around a minimal core
- can be either statically or dynamically *configured* by loading/unloading libraries of predicates
- natively supports multi-paradigm programming [Denti et al., 2005], providing a clean, seamless integration model between Prolog and mainstream object-oriented languages
- runs on most known platforms and devices

Dealing with Situatedness & Domain-specific Scenarios I

- pervasive applications typically demands for domain-specific approaches
- multiple, heterogeneous devices in the IoT usually have specific application goals and manage specific sorts of information
- situatedness require reactivity to environment change

! LP is general enough to deal with whatever, but never specific enough to do it well

- many specific logics exist to deal with many diverse application scenarios—e.g., S4 modal logic for spatial computing
- LP can be extended to capture different logic and domains

Dealing with Situatedness & Domain-specific Scenarios II

Our example: Labelled Variables in Logic Programming

In Labelled Variables in Logic Programming (LVLP) [Calegari et al., 2016a]

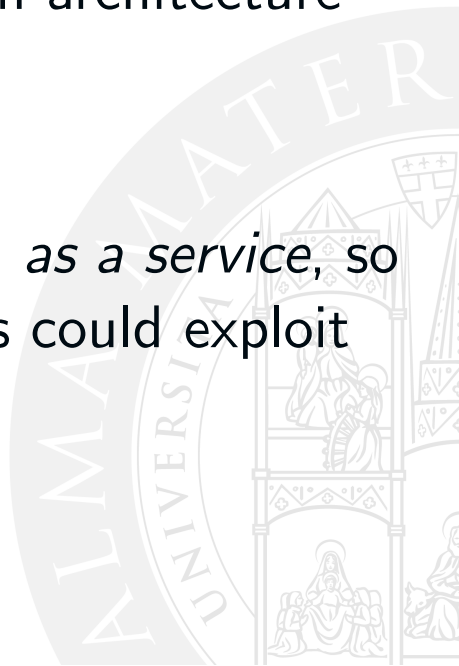
- logic variable can be associated to a *label*
- labels are *domain-specific*
- each logic engine can be configured with its own set of domain-specific label, along with the specific functions to interfere with the logic resolution process, without losing declarativeness
- labels allows for a customisable computational model (domain-specific *labelled system*) bound to the standard LP computation

e.g. a wardrobe could host a LVLP engine whose labelled system represent its content, and helps in dress selection [Calegari et al., 2016a]

Making LP Available in Distributed Systems I

- LP promotes interactive programming, even though one user / one engine sort
- in order to meet the needs of nowadays distributed systems, a more *standard* approach should be adopted, in terms of both architecture and technology
- possibly subsuming standard LP interaction

e.g. providing logic engines (theories, inference, resolution) *as a service*, so that standard distributed components and applications could exploit them



Making LP Available in Distributed Systems II

Our example: Logic Programming as a Service

Logic Programming as a Service (LPaaS) [Calegari et al., 2016b]

- provides an abstract view of an LP inference engine in terms of service
- promotes interoperability, encapsulation, and situatedness, as well as context-awareness
- each logic engine exposes its services concurrently to multiple clients, via two interfaces (*client* and *configurator*)
- preserving the standard resolution process
- providing for both *stateless* and *stateful* client-server interaction
- promoting time-sensitive and space-sensitive computation

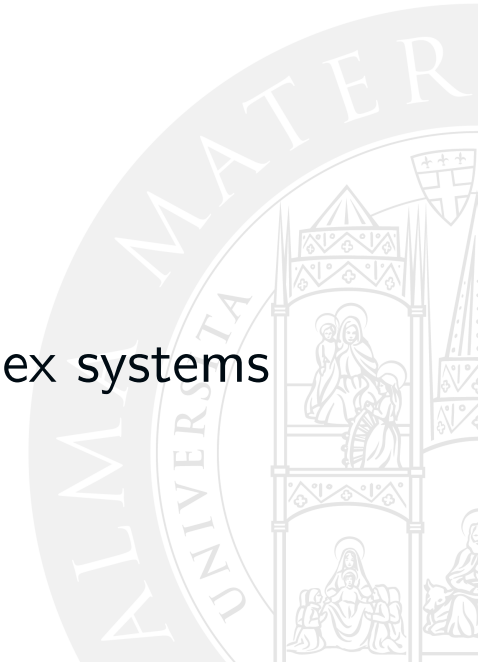
Logic-based Coordination Artefacts I

Coordination artefacts

- encapsulate coordination policies for distributed systems
- can inject *social intelligence* within computational systems [Castelfranchi, 1998]
- can be observable and malleable [Omicini et al., 2008]

Logic-based coordination artefacts

- represent coordination policies declaratively
- pave the way towards (partial) formalisation of complex systems



Logic-based Coordination Artefacts II

Our example: ReSpecT coordination artefacts

ReSpecT coordination artefacts (LPaaS) [Omicini, 2007]

- are *inspectable* and *malleable*
- can express any Turing-computable coordination policy [Denti et al., 1998]
- there, both communication and coordination theories are first-order logic theories
- can be used within the TuCSoN middleware for MAS coordination [Omicini and Zambonelli, 1999]

Logic-based Middleware I

Distributed logic agents and logic engines

Overall

- distributing logic engines
- configuring them as services
- allowing logic-based agents to roam around
- making agents interact with each other and exploiting logic services
- coordinating logic agents and engines / services

suggest that a LP-based middleware would be of some help for the engineering of intelligent pervasive and IoT applications

Logic-based Middleware II

Our example: work in progress

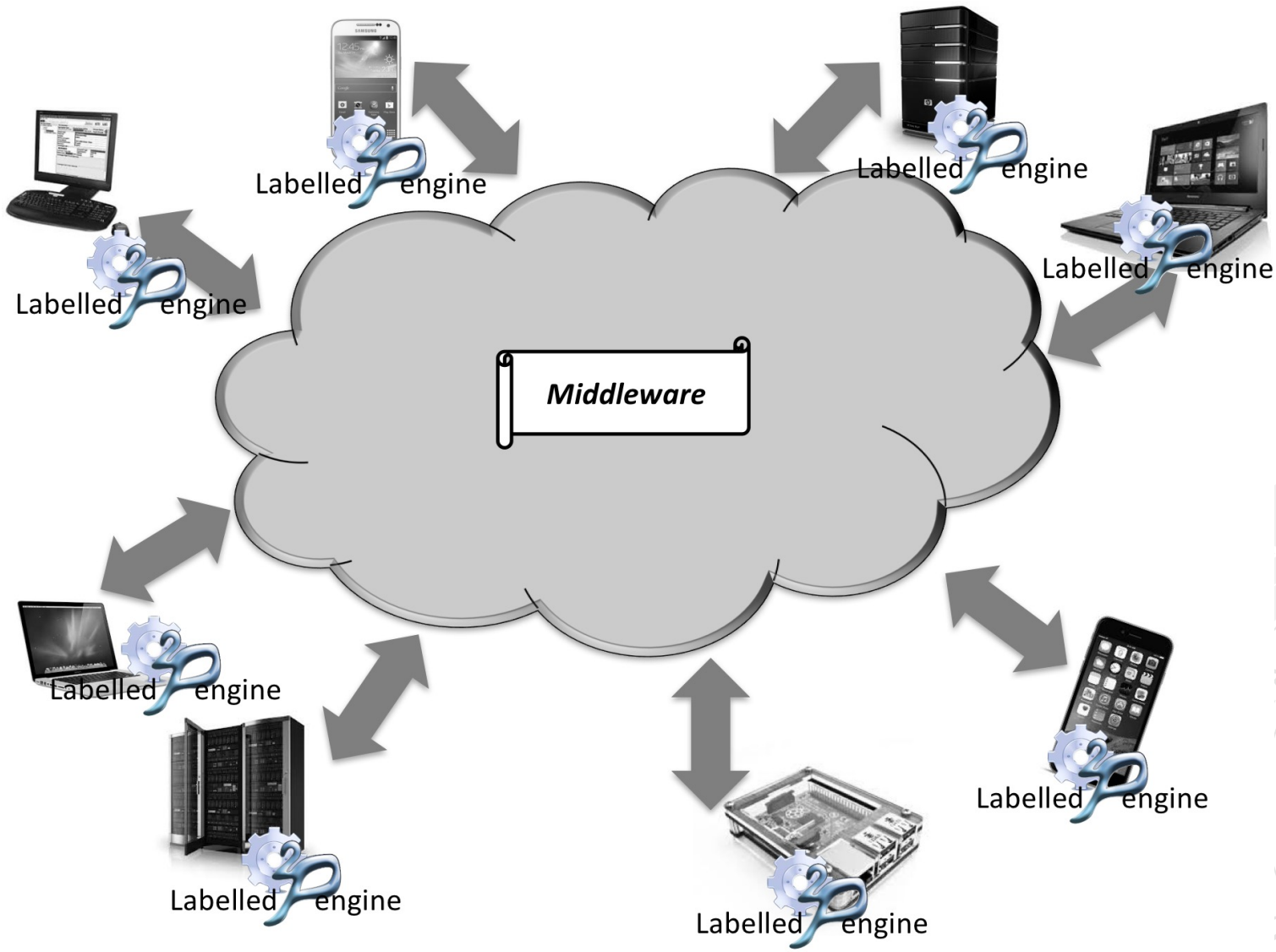
Thesis by Alberto Sita, in cooperation with Roberta Calegari, Enrico Denti, Stefano Mariani

Overall View I

LP for the IoIT

- distributed logic engines for widespread *micro-intelligence*
- exploited as standard *services* by components of any sorts, including intelligent agents
- situated and *domain-specific* extensions to meet the demand of IoT
- coordinated via logic-based artefacts, encapsulating *social intelligence*
- based on a logic-based *middleware*

Overall View II



Next in Line...

Further Work

- porting tuProlog on most popular resource-constraint devices
- full implementation of LPaaS in tuProlog
- full-fledged logic-based middleware architecture and technology
- testing in real-world applications
- general formalisation of the framework

Conclusion

Micro-intelligence for the IoIT

- the requirements of pervasive and IoIT systems include widespread intelligence
- LP-based languages and technologies have the potential to work as the sources of micro-intelligence encapsulated within devices of any sort, and to make them work together in groups, aggregates, societies
- by promoting observability, malleability, understandability, formalisability, and norm compliance

! provided that LP legacy works as a rich platform for future developments rather than a huge burden

References I

Arsénio, A., Serra, H., Francisco, R., Nabais, F., Andrade, J., and Serrano, E. (2014).
Internet of Intelligent Things: Bringing artificial intelligence into things and communication networks.
In *Inter-cooperative Collective Intelligence: Techniques and Applications*, volume 495 of *Studies in Computational Intelligence*, pages 1–37. Springer Berlin Heidelberg.

Atzori, L., Iera, A., and Morabito, G. (2010).
The Internet of Things: A survey.
Computer Networks, 54(15):2787–2805.

Baldoni, M., Baroglio, C., Mascardi, V., Omicini, A., and Torroni, P. (2010).
Agents, Multi-Agent Systems and Declarative Programming: Who, What, When, Where, Why, How?
In Dovier, A. and Pontelli, E., editors, *A 25 Year Perspective on Logic Programming. Achievements of the Italian Association for Logic Programming, GULP*, LNAI: State-of-the-Art Survey, chapter 10, pages 200–225. Springer.

Brogi, A. and Ciancarini, P. (1991).
The concurrent language, Shared Prolog.
ACM Transactions on Programming Languages and Systems, 13(1):99–123.

References II

Calegari, R., Denti, E., Dovier, A., and Omicini, A. (2016a).
Labelled variables in logic programming: Foundations.
In Fiorentini, C. and Momigliano, A., editors, *CILC 2016 – Italian Conference on Computational Logic*, volume 1645 of *CEUR Workshop Proceedings*, pages 5–20, Milano, Italy. CEUR-WS.
Proceedings of the 31st Italian Conference on Computational Logic.

Calegari, R., Denti, E., Mariani, S., and Omicini, A. (2016b).
Towards logic programming as a service: Experiments in tuProlog.
In Santoro, C., Messina, F., and De Benedetti, M., editors, *WOA 2016 – 17th Workshop "From Objects to Agents"*, volume 1664 of *CEUR Workshop Proceedings*, pages 91–99. Sun SITE Central Europe, RWTH Aachen University.
Proceedings of the 17th Workshop "From Objects to Agents" co-located with 18th European Agent Systems Summer School (EASSS 2016).

Castelfranchi, C. (1998).
Modelling social action for AI agents.
Artificial Intelligence, 103(1-2):157–182.

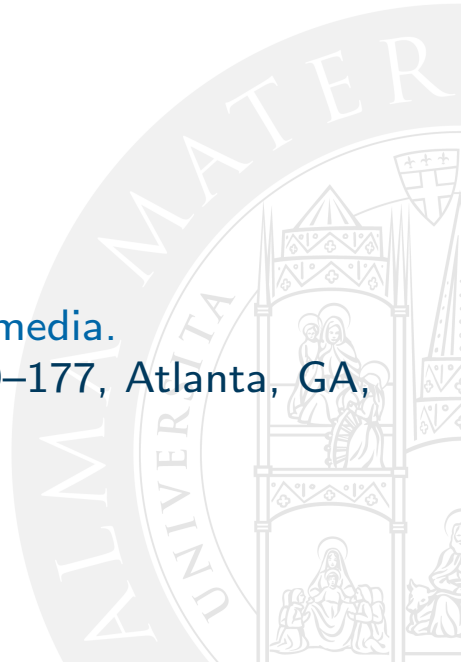
References III

Castelfranchi, C., Pezzullo, G., and Tummolini, L. (2010).
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.

Ciancarini, P. (1994).
Distributed programming with logic tuple spaces.
New Generation Computing, 12(3):251–283.

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

Denti, E., Natali, A., and Omicini, A. (1998).
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. ACM.
Special Track on Coordination Models, Languages and Applications.



References IV


Denti, E., Natali, A., Omicini, A., and Venuti, M. (1996).
Logic tuple spaces for the coordination of heterogeneous agents.
In Baader, F. and Schulz, K. U., editors, *Frontiers of Combining Systems*, volume 3 of *Applied Logic Series*, pages 147–160, Munich, Germany. Kluwer Academic Publishers.


Denti, E., Omicini, A., and Calegari, R. (2013).
tuProlog: Making Prolog ubiquitous.
ALP Newsletter.


Denti, E., Omicini, A., and Ricci, A. (2001).
tuProlog: A light-weight Prolog for Internet applications and infrastructures.
In Ramakrishnan, I., editor, *Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 184–198. Springer.
3rd International Symposium (PADL 2001), Las Vegas, NV, USA, 11–12 March 2001. Proceedings.


Denti, E., Omicini, A., and Ricci, A. (2005).
Multi-paradigm Java-Prolog integration in tuProlog.
Science of Computer Programming, 57(2):217–250.


FIPA ACL (2002).
Agent Communication Language Specifications.
Foundation for Intelligent Physical Agents (FIPA).

References V

Fortino, G., Guerrieri, A., Russo, W., and Savaglio, C. (2014).
Integration of agent-based and Cloud Computing for the smart objects-oriented IoT.
In *2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 493–498.

Fredriksson, M. and Gustavsson, R. (2004).
Online engineering and open computational systems.
In Bergenti, F., Gleizes, M.-P., and Zambonelli, F., editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organization*, pages 377–388. Kluwer Academic Publishers.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013).
Internet of Things (IoT): A vision, architectural elements, and future directions.
Future Generation Computer Systems, 29(7):1645–1660.

Gupta, G., Pontelli, E., Ali, K. A., Carlsson, M., and Hermenegildo, M. V. (2001).
Parallel execution of Prolog programs: a survey.
ACM Transactions on Programming Languages and Systems, 23(4):472–602.

References VI

Lippi, M., Mamei, M., Mariani, S., and Zambonelli, F. (2017).
Coordinating distributed speaking objects.
In *37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*.
IEEE Computer Society.

Mariani, S. and Omicini, A. (2013).
Molecules of Knowledge: Self-organisation in knowledge-intensive environments.
In Fortino, G., Bădică, C., Malgeri, M., and Unland, R., editors, *Intelligent Distributed Computing VI*, volume 446 of *Studies in Computational Intelligence*, pages 17–22.
Springer.
6th International Symposium on Intelligent Distributed Computing (IDC 2012), Calabria, Italy, 24-26 September 2012. Proceedings.

Nishida, T., Nakazawa, A., Ohmoto, Y., and Mohammad, Y. (2014).
Conversational Informatics. A Data-Intensive Approach with Emphasis on Nonverbal Communication.
Springer Japan, Tokyo.

References VII

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

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

Omicini, A. and Mariani, S. (2013).
Agents & multiagent systems: En route towards complex intelligent systems.
Intelligenza Artificiale, 7(2):153—164.
Special Issue Celebrating 25 years of the Italian Association for Artificial Intelligence.

Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17(3):432–456.
Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

References VIII

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

Omicini, A. and Zambonelli, F. (2004).
MAS as complex systems: A view on the role of declarative approaches.
In Leite, J. A., Omicini, A., Sterling, L., and Torroni, P., editors, *Declarative Agent Languages and Technologies*, volume 2990 of *LNCS*, pages 1–17. Springer.
1st International Workshop (DALT 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.

Piancastelli, G., Benini, A., Omicini, A., and Ricci, A. (2008).
The architecture and design of a malleable object-oriented Prolog engine.
In Wainwright, R. L., Haddad, H. M., Menezes, R., and Viroli, M., editors, *23rd ACM Symposium on Applied Computing (SAC 2008)*, volume 1, pages 191–197, Fortaleza, Ceará, Brazil. ACM.
Special Track on Programming Languages.

References IX

Robertson, D. (2004).
Multi-agent coordination as Distributed Logic Programming.
In Demoen, B. and Lifschitz, V., editors, *Logic Programming. 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004. Proceedings*, volume 3132, pages 416–430. Springer.

Robinson, J. A. (1965).
A machine-oriented logic based on the resolution principle.
Journal of the ACM, 12(1):23–41.

Singh, M. P. and Chopra, A. K. (2017).
The Internet of Things and multiagent systems: Decentralized intelligence in distributed computing.
In *37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*. IEEE Computer Society.

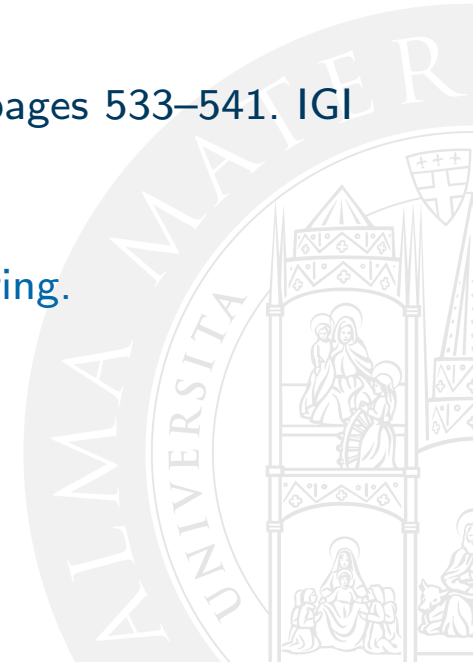
Tanenbaum, A. S. and van Steen, M. (2007).
Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

References X

Viroli, M., Omicini, A., and Ricci, A. (2005).
Engineering MAS environment with artifacts.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *2nd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2005)*, pages 62–77, AAMAS 2005, Utrecht, The Netherlands.

Whitworth, B. (2006).
Socio-technical systems.
In Ghaou, C., editor, *Encyclopedia of Human Computer Interaction*, pages 533–541. IGI Global.

Zambonelli, F. and Omicini, A. (2004).
Challenges and research directions in agent-oriented software engineering.
Autonomous Agents and Multi-Agent Systems, 9(3):253–283.
Special Issue: Challenges for Agent-Based Computing.



URLs

Slides

- On APICe

→ `http://apice.unibo.it/xwiki/bin/view/Talks/MicrointelligenceWoa2017`

- On SlideShare

→ `http://www.slideshare.net/andreaomicini/microintelligence-for-the-iot-teaching-the-old-logic-dog-new-programming-tricks`

Micro-intelligence for the IoT

Teaching the Old Logic Dog New Programming Tricks

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI), Università di Bologna

Invited Talk @ *18th Workshop "From Objects to Agents" (WOA 2017)*
Scilla, RC, Italy, 16 June 2017