Logic Programming as a Service (LPaaS)

Distributed Systems / Technologies
Sistemi Distribuiti / Tecnologie

Roberta Calegari Andrea Omicini
roberta.calegari@unibo.it andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2017/2018

Next in Line...

Context I

Internet of Intelligent Things: Intelligent objects in the IoT

- our everyday physical objects should be able to network in the 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* [Arsénio et al., 2014]

Micro-intelligence for the IoT

- micro-level feature, influenced by the *Things* vision in IoT
- ability to abstract, reason, plan, solve, and learn capabilities
- *situatedness* feature, as the capability to interact with and act on the environment.

Context II

New opportunities for IoT apps and services [Zambonelli, 2015]

Devices and people collaborate as a superorganism: *situation-aware* dense ecosystem where *infrastructures* should

- be customisable
- be self-managing
- govern interaction
- encapsulate intelligence

⬇

Micro-intelligence as distributed situated intelligence

- spread light-weight, context-aware, effective *intelligence chunks* where and when needed
- locally satisfy the specific reasoning needs of the application at hand

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

LP languages and technologies → long-respected reputation in supporting intelligence: originally conceived for single solvers and later extended towards concurrency and parallelism

- declarativeness and explicit knowledge enable knowledge sharing at the most adequate level of abstraction
- supporting modularity and separation of concerns [Oliya and Pung, 2011] specially valuable in open and dynamic distributed systems [Niezen, 2013]
- its sound and complete semantics naturally enables intelligent agents to reason and infer new information
- LP extensions or logic-based computational models – like meta-reasoning about situations [Loke, 2004] or labelled variables systems [Calegari et al., 2016] – could be incorporated to enable complex behaviours tailored to the situated components

LP Re-interpretation

LP as a *situated service*

Overall LP has the potential to fully support pervasive computing scenarios once it is suitably *re-interpreted* along three main lines:

architecture beyond (originally monolithic) structure of LP systems

- → unsuitable for distributed contexts such as IoT mobility/cloud ecosystems grounded upon the service-oriented paradigm
- → Everything as a Service (XaaS): promote maximum availability and interoperability, any resource of any sort should be accessible *as a service* [Erl, 2005]

situatedness enabling logic theories, queries, and resolutions to be context-aware w.r.t. the (computational) environment, space, and time

interaction re-think interaction patterns used by clients to query logic engines, which should lean towards on-demand computation

LPaaS Motivations

Logic-based micro-intelligence approach

- enabling people to benefit from ubiquitous information access exploiting *contextual knowledge*
- multiple, distributed Prolog engines
  - intelligence providers
  - technology integrators

⇓

Logic Programming as a Service (LPaaS)
as the evolution of LP in parallel, concurrent, and distributed scenarios

Next in Line. . .

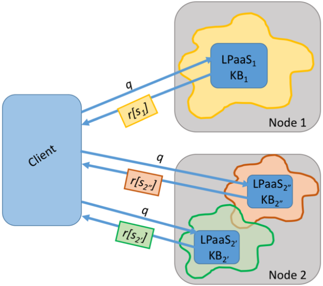
Focus on. . .

LPaaS Vision I

LP as a *situated service*: key features

- encapsulation
- statelessness
- locality
- situatedness: *space*, *time*, *context*

- preservation, with re-contextualisation, of the SLD resolution process
- stateless client-server interaction
- time-sensitive computation
- space-sensitive computation



LPaaS Vision II

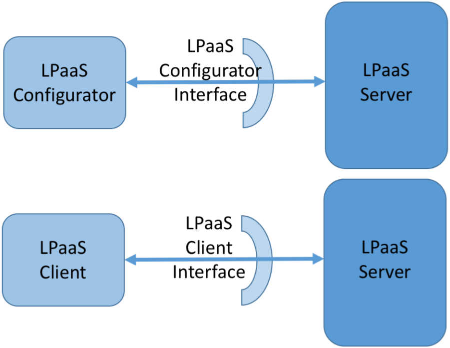
Distributed logic engines

- lightweight and interoperable LP engines distributed even on resource-constrained devices [Denti et al., 2013]
- *multiple logic theories* scattered around, encapsulated in each engine, and associated to individual computational devices and *things* in the IoT
- each logic theory *situated*, representing 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]

LPaaS Architecture

Logic Programming as a Service (LPaaS)

- provides an abstract view of an LP inference engine in terms of service
- promotes interoperability, encapsulation, and situatedness
- promotes context-awareness



Focus on. . .

Service Description

Logic resolution process provided as a service

The LPaaS service

- implements SLD resolution [Robinson, 1965]
- encapsulates the logic theory / Knowledge Base (KB)
- is configured with the set of goals that the client can ask to be proven

Service initialised at deployment-time → dynamic re-configuration (when needed) only by the Configurator

Main features of the LP service

- *stateful* vs. *stateless* interaction
- *dynamic* vs. *static* KB

Focus on. . .

LPaaS Configurator Interface

LPaaS methods

observational methods to provide configuration and contextual information about the service

usage methods to query the service for triggering computations and reasoning, and for asking solutions

```
setConfiguration(+ConfigurationList)
getConfiguration(-ConfigurationList)
resetConfiguration()

setTheory(+Theory)
getTheory(-Theory)
setGoals(+GoalList)
getGoals(-GoalList)
```

LPaaS Client Interface – Static KB

STATIC KNOWLEDGE BASE		
Stateless		**Stateful**

	`getServiceConfiguration(-ConfigList)`
	`getTheory(-Theory)`
	`getGoals(-GoalList)`
	`isGoal(+Goal)`	

		`setGoal(template(+Template))` `setGoal(index(+Index))`

`solve(+Goal, -Solution)`
`solveN(+Goal, +NSol, -SolutionList)`
`solveAll(+Goal, -SolutionList)`
		`solve(-Solution)`
		`solveN(+N, -SolutionList)`
		`solveAll(-SolutionList)`

`solve(+Goal, -Solution, within(+Time))`
`solveN(+Goal, +NSol, -SolutionList, within(+Time))`
`solveAll(+Goal, -SolutionList, within(+Time))`
		`solve(-Solution, within(+Time))`
		`solveN(+NSol, -SolutionList, within(+Time))`
		`solveAll(-SolutionList, within(+Time))`

`solveAfter(+Goal, +AfterN, -Solution)`
`solveNAfter(+Goal, +AfterN, +NSol, -SolutionList)`
`solveAllAfter(+Goal, +AfterN, -SolutionList)`		

		`solve(-Solution, every(@Time))`
		`solveN(+N, -SolutionList, every(@Time))`
		`solveAll(-SolutionList, every(@Time))`
		`pause()`
		`resume()`

	`reset()` `close()`	

LPaaS Client Interface – Dynamic KB

DYNAMIC KNOWLEDGE BASE	
Stateless	**Stateful**
getServiceConfiguration(-ConfigList) getTheory(-Theory, ?Timestamp) getGoals(-GoalList) isGoal(+Goal)	getServiceConfiguration(-ConfigList) getTheory(-Theory, ?Timestamp) getGoals(-GoalList) isGoal(+Goal)
	setGoal(template(+Template)) setGoal(index(+Index))
solve(+Goal, -Solution, ?Timestamp) solveN(+Goal, +NSol, -SList, ?TimeStamp) solveAll(+Goal, -SList, ?TimeStamp)	solve(-Solution, ?Timestamp) solveN(+N, -SolutionList, ?TimeStamp) solveAll(-SolutionList, ?TimeStamp)
solve(+Goal, -Solution, within(+Time), ?TimeStamp) solveN(+Goal, +NSol, -SList, within(+Time), ?TimeStamp) solveAll(+Goal, -SList, within(+Time), ?TimeStamp)	solve(-Solution, within(+Time), ?TimeStamp) solveN(+NSol, -SList, within(+Time), ?TimeStamp) solveAll(-SList, within(+Time), ?TimeStamp)
solveAfter(+Goal, +AfterN, -Solution, ?TimeStamp) solveNAfter(+Goal, +AfterN, +NSol, -SList, ?TimeStamp) solveAllAfter(+Goal, +AfterN, -SList, ?TimeStamp)	
	solve(-Solution, every(@Time), ?TimeStamp) solveN(+N, -SList, every(@Time), ?TimeStamp) solveAll(-SList, every(@Time), ?TimeStamp) pause() resume()
reset() close()	reset() close()

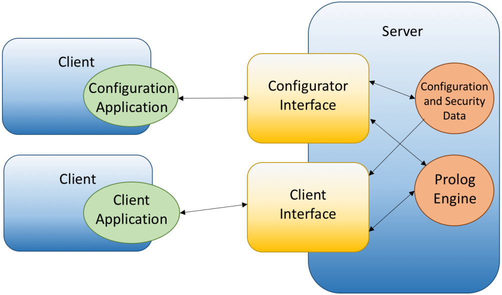
Next in Line...

Focus on. . .

Architecture

The LPaaS RESTful WS Architecture [Fielding and Taylor, 2002]

- reused and adapted patterns commonly used for the REST architectural style
- introduced a novel architecture supporting embedding Prolog engines into Web Services
- client applications interacting via HTTP requests and JSON objects



Technology

Exploiting a plurality of common technologies

- Business Logic realised on the J2EE framework [J2EE, 2017], exploiting EJB [EJB, 2017]
- database interaction implemented on top of JPA [Java Persistence API, 2013]
- service interfaces exploit the EJB architecture—also accessible as RESTful WS → JAX-RS Java Standard (Jersey) [Jersey, 2017]
- security based on jose.4.j [jose.4.j, 2017]
- application deployment → Payara Application Server [Payara, 2017]
- Prolog engine implemented on top of the tuProlog system [Denti et al., 2001]

Focus on. . .

The Smart Bathroom Case Study I

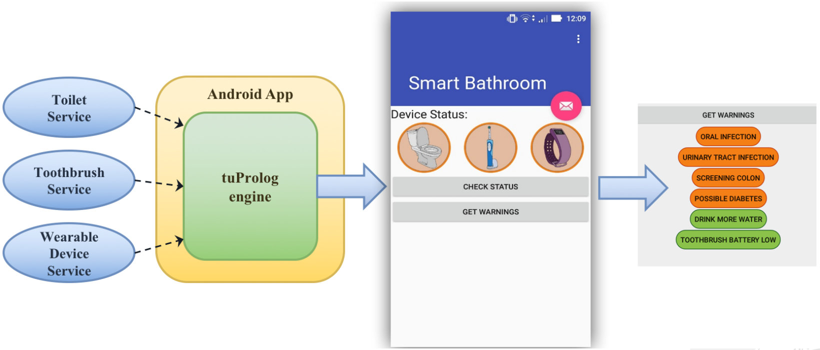
Testbed Scenario: monitor physiological functions

- deduce symptoms and diseases
- sensors collect data and undertake reasoning on tuProlog-LPaaS
- solutions available through a dedicated tuProlog Android app

Three tuProlog-enabled LPaaS services processing data by

- toilet sensors analysing biological products, like temperature, volume or glucose sensors (Toilet Server)
- nano sensors integrated into the toothbrush (Toothbrush Server)
- ultrasonic bathtubs, pressure sensing toilet seats and other devices to monitor people's cardiovascular health (Personal Server).

The Smart Bathroom Case Study II



Collected data may trigger different alerts: urgent ones, such as presence of streptococcus infection, positive diabetes tests, etc. and normal ones

Prototype Screenshots

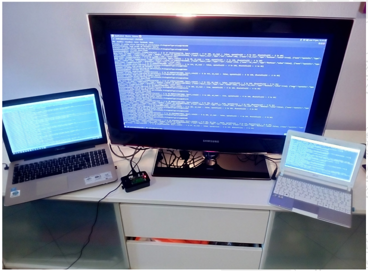
KB extraction of LPaaS Personal Server and LPaaS Toothbrush Server

```
%%LPaaS Personal Server KB
disease('possible diabetes') :- measurement('glucose'), not(disease('diabetes')).
disease('diabetes') :- measurement('glucose'), measurement('ketones').
symptoms('high sodium') :- measurement(sodium(X)), X > 200.
symptoms('dehydration') :- measurement(water(X)), X > 1028.
symptoms('whiteBloodCells') :- measurement('whiteBloodCells').
warning('drinkMoreWater') :- symptoms('dehydration').
warning('limitSodiumIntake') :- symptoms('high sodium').

%%LPaaS Toothbrush Server KB
disease(infections(X)) :- symptoms(infection(X)).
symptoms(infection('streptococco infection')) :- measurement(bacteria(X, Y)), X >100, Y='streptococco'.
warning('toothbrushLowBattery') :- measurement(battery(X)), X < 16.
```

The system is built on the following network:

- toilet server: on Raspberry Pi 3 (Ubuntu Mate Arm)
- toothbrush server: on Lubuntu laptop
- personal server: on Windows 10 laptop
- client: tablet Lenovo A10 (Android 5.0.1)
- client: desktop application on Windows 10



Next in Line...

Focus on. . .

Agents & multi-agent systems (MAS)

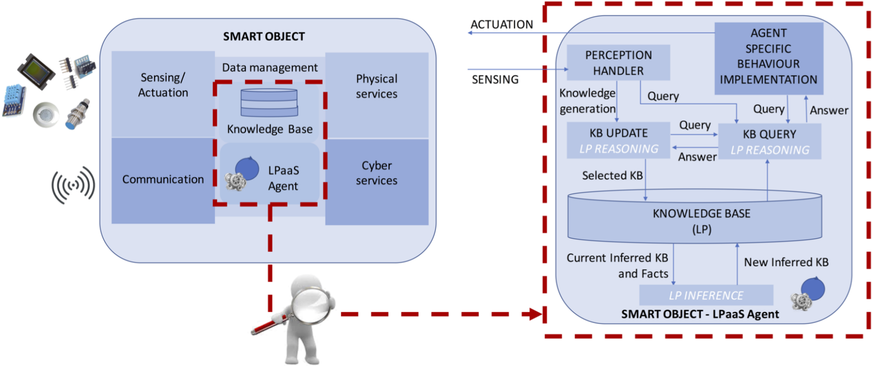
- agents are the most viable abstraction to encapsulate fundamental features such as *control*, *goals*, *mobility*, *intelligence*
  [Zambonelli and Omicini, 2004]
- MAS abstractions such as *society* and *environment* are essential to cope with the complexity of nowadays application scenarios
  [Omicini and Mariani, 2013]
- agent-oriented models and technologies are gaining momentum for embedding decentralised intelligence [Singh and Chopra, 2017]

LPaaS for MAS: Model & Architecture I

The LPaaS for MAS

- agents on Smart Objects (SO): *autonomy*, *situatedness*, *sociality*, *mobility*
- resource-constrained devices $\rightarrow$ *intelligence* is a challenge
- whenever local intelligence cannot be available (i.e. memory constraints, CPU constraints limiting efficiency,...) $\rightarrow$ SO may request to another, "more intelligent" one, to perform some inferences on its behalf

LPaaS for MAS: Model & Architecture II



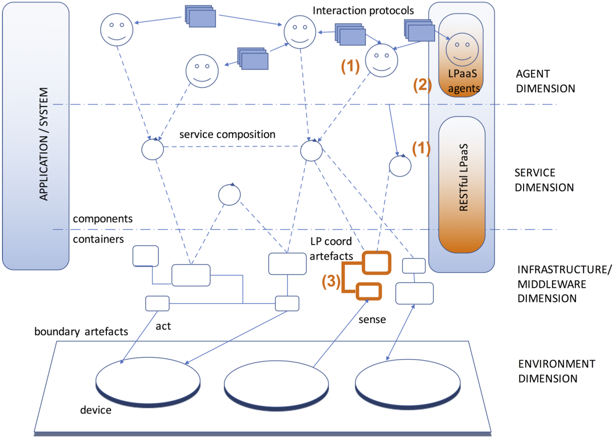
LPaaS for MAS

LPaaS for MAS: Model & Architecture III

Overview of a LPaaS multi-agent system

- bottom layer → physical/computational environment lives (*boundary artefacts* [Omicini et al., 2006] for representation and interactions with the rest of the MAS)
- middleware infrastructure → common API and services to application-level software – i.e. *coordination artefacts* [Omicini et al., 2006]
- on the top of the middleware → application/system as a whole lives, in LPaaS MAS view as a mixture of services – possibly RESTful, as for LPaaS as a WS – and agents

LPaaS for MAS: Model & Architecture IV



LPaaS-based MAS

Technology

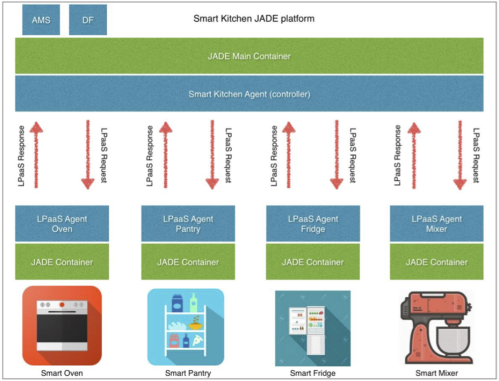
Exploiting a plurality of common technologies

- implemented on top of the JADE middleware [JADE API, 2017], which facilitates the development of interoperable, open, and heterogeneous multi-agent systems by relying on the FIPA standard [O'Brien and Nicol, 1998]
- communication between JADE agents occurs via ACL (Agent Communication Language) messages [FIPA ACL, 2002]
- security using JADE-S (Secure JADE) [Poggi et al., 2001]
- exploiting tuProlog [Denti et al., 2001] as the LPaaS Prolog engine

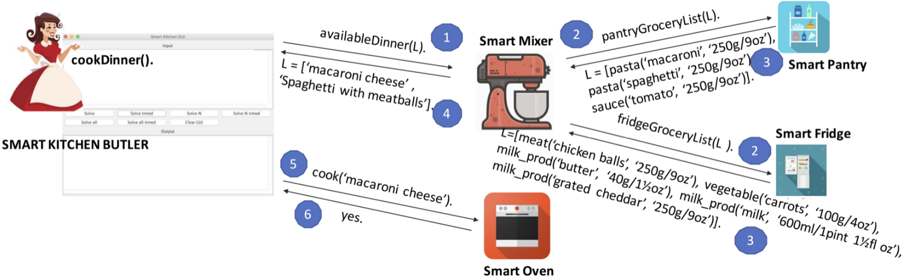
Focus on. . .

The Smart Kitchen Case Study I

- four IoT devices (a fridge, a pantry, a mixer, an oven) supply information to clients, exploiting LPaaS, about food supply and users' preferences
- fridge and pantry capable of monitoring the quantity of food, and collecting historical data on user's habits (i.e. most eaten food, preferred meals) while oven identifies and cook food
- mixer manages the recipe instructions, interacting with both fridge – to check ingredients availability – and with oven—to check its ability to cook that food, synthesise control instructions



The Smart Kitchen Case Study II



Next in Line...

tuProlog in a Nutshell

http://tuprolog.unibo.it

- light-weight Prolog system for distributed applications and infrastructures [Denti et al., 2001]
- intentionally designed around a minimal core
- can be either statically or dynamically *configured* by loading/unloading libraries of predicates
- natively supports multi-paradigm programming, providing a clean, seamless integration model between Prolog and mainstream object-oriented languages
- runs on most known platforms and devices (Java, .NET, Android, iOS)
- interoperability → also supports JSON serialisation natively, ensuring the interoperability required by a WS

Next in Line. . .

LPaaS Benefits

- ubiquitous intelligence for pervasive scenarios

→ distribute reasoning and inference capabilities amongst the components of IoT system, balancing the computational requirements to best suit the deployment scenario

- situated reasoning:

→ enable reasoning and inferential processes to be context-aware w.r.t. the (possibly ever-changing) environment where the process takes place

→ rely mostly on locally available information reduces the bandwidth consumption and the need for reliable communications

- other benefits when coupling LPaaS with MAS

→ *goal-orientedness*: LPaaS agents may in fact exploit LPaaS to reason about their own goals, the plans and actions needed to achieve them, and the effects brought by—which is something only rational agents (such as BDI ones [Rao, 1996]) usually do

Future Work

- further tests in pervasive deployment scenarios are required, mainly in the IoT landscape—e.g., testing directly LPaaS tuProlog over Bluetooth Low Energy connections
- deal with space-awareness and mobility need to be further investigated, for instance by exploring the idea to opportunistically federate LP engines by need as a form of dynamic service composition
- architecting a specialised logic-based middleware
- integrating LPaaS with Labelled LP [Calegari et al., 2016] for domain-specific logic-based computation
- integration with databases as distributed knowledge base of the system → handling replication and consistency of data scattered in connected devices arise
- integration with sensor devices to have LPaaS always working the most up-to-date perception of the environment properties of interest for the application at hand.

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.

Calegari, R., Denti, E., Dovier, A., and Omicini, A. (2016).
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.

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

References II

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 *Lecture Notes in Computer Science*, pages 184–198. Springer Berlin Heidelberg.
3rd International Symposium (PADL 2001), Las Vegas, NV, USA, 11–12 March 2001. Proceedings.

EJB (2017).
Home Page.
`http://www.oracle.com/technetwork/java/javaee/ejb/`.

Erl, T. (2005).
Service-Oriented Architecture: Concepts, Technology, and Design.
Prentice Hall / Pearson Education International, Upper Saddle River, NJ, USA.

Fielding, R. T. and Taylor, R. N. (2002).
Principled design of the modern Web architecture.
ACM Transactions on Internet Technology, 2(2):115–150.

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

References III

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.

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.

J2EE (2017).
Home Page.
`http://docs.spring.io/autorepo/docs/spring-framework/1.2.x/reference/`.

JADE API (2017).
Home Page.
`http://jade.tilab.com/doc/api/`.

Java Persistence API (2013).
Home Page.
`http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html`.

References IV

Jersey (2017).
Home Page.
http://jersey.java.net/.

jose.4.j (2017).
Home Page.
http://bitbucket.org/b_c/jose4j/.

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.

Loke, S. W. (2004).
Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective.
The Knowledge Engineering Review, 19(3):213–233.

Niezen, G. (2013).
Ontologies for interaction: Enabling serendipitous interoperability in smart environments.
Journal of Ambient Intelligence and Smart Environments, 5(1):135–137.

References V

O'Brien, P. D. and Nicol, R. C. (1998).
FIPA — towards a standard for software agents.
BT Technology Journal, 16(3):51–59.

Oliya, M. and Pung, H. K. (2011).
Towards incremental reasoning for context aware systems.
In Abraham, A., Lloret Mauri, J., Buford, J. F., Suzuki, J., and Thampi, S. M., editors, *Advances in Computing and Communications: First International Conference, ACC 2011, Kochi, India, July 22-24, 2011. Proceedings, Part I*, volume 190 of *Communications in Computer and Information Science*, pages 232–241. Springer, Berlin, Heidelberg.

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. (2006).
Agens Faber: Toward a theory of artefacts for MAS.
Electronic Notes in Theoretical Computer Sciences, 150(3):21–36.
1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.

References VI

Payara (2017).
Home Page.
http://www.payara.fish.

Poggi, A., Rimassa, G., and Tomaiuolo, M. (2001).
Multi-user and security support for multi-agent systems.
In Omicini, A. and Viroli, M., editors, *WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, Modena, Italy. Pitagora Editrice Bologna.

Rao, A. S. (1996).
AgentSpeak(L): BDI agents speak out in a logical computable language.
In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away*, volume 1038 of *LNCS*, pages 42–55. Springer.
7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), Eindhoven, The Netherlands, 22-25 January 1996, Proceedings.

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

References VII

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.

Zambonelli, F. (2015).
Engineering self-organizing urban superorganisms.
Engineering Applications of Artificial Intelligence, 41(C):325–332.

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.

Logic Programming as a Service (LPaaS)

Distributed Systems / Technologies
Sistemi Distribuiti / Tecnologie

Roberta Calegari Andrea Omicini
roberta.calegari@unibo.it andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2017/2018