Logic Programming as a Service (LPaaS): Intelligence for the IoT

Roberta Calegari[1] Enrico Denti[1] Stefano Mariani[2]
Andrea Omicini[1]
{roberta.calegari, enrico.denti, andrea.omicini}@unibo.it
stefano.mariani@unimore.it

[1]Dipartimento di Informatica, Scienza e Ingegneria—Università di Bologna
[2]Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia

Talk @ *14th IEEE International Conference on Networking, Sensing and Control (ICNSC 2017)*
Calabria, 16 May 2017

Next in Line...

Context

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

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

Motivations

IoT scenarios

- mobility/pervasive/cloud (eco)systems
- infrastructure, platforms, and software *as a service*
- enabling people to benefit from ubiquitous information access exploiting *contextual knowledge*

Logic-based languages

- multiple, distributed Prolog engines
  - intelligence providers
  - technology integrators

Logic Programming as a Service (LPaaS)

Next in Line...

Focus on...

LPaaS Vision I

Evolution of LP in parallel, concurrent, and distributed scenarios

re-interpreting the notion of *distribution* of LP in today's context

service perspective $\rightarrow$ further underlines the role of *situatedness*

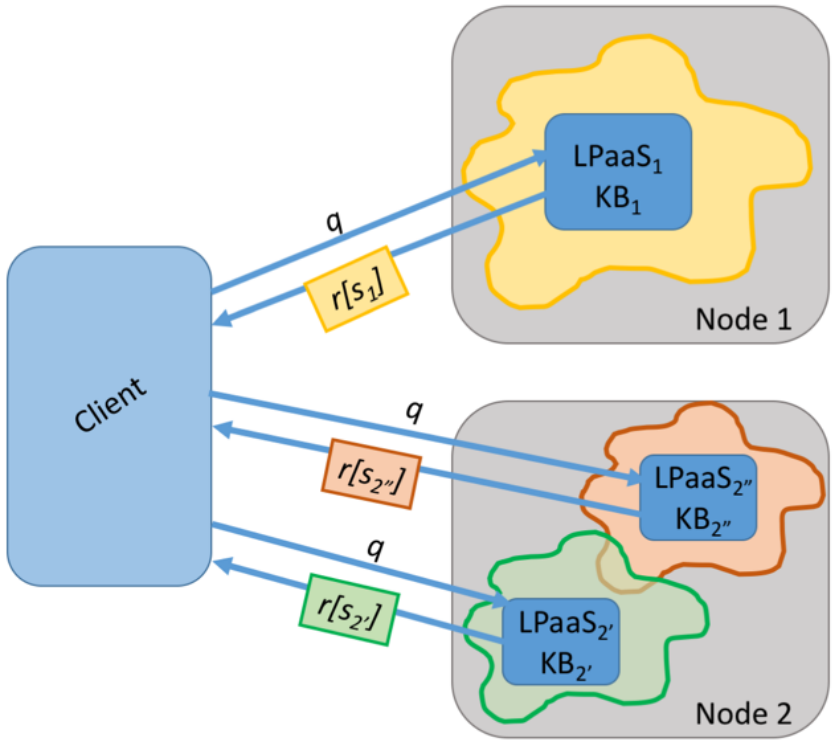
LP as a situated service

Key features

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

LPaaS Vision II

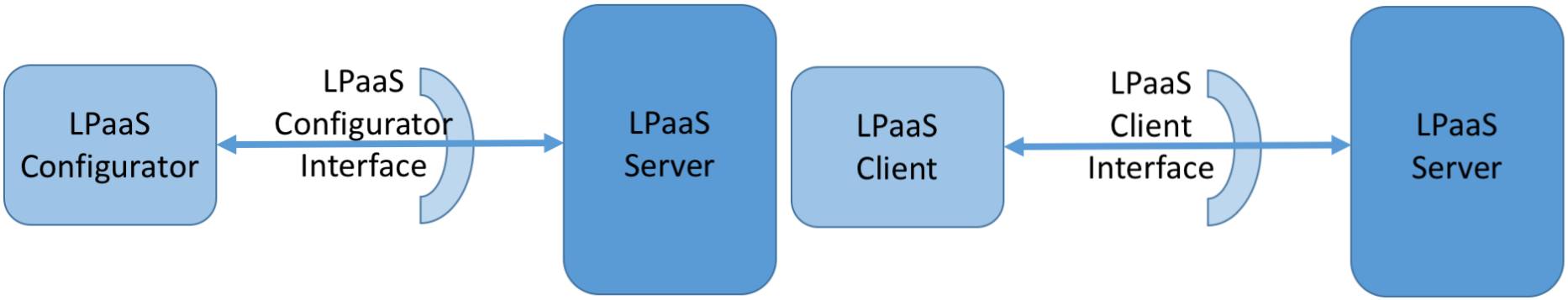
- preservation, with re-contextualisation, of the SLD resolution process
- stateless client-server interaction
- time-sensitive computation
- space-sensitive computation



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



Each LP server node exposes its services concurrently to multiple clients, via interfaces: Configurator Interface and Client Interface

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)`		`solve(-Solution)`
`solveN(+Goal, +NSol, -SolutionList)`		`solveN(+N, -SolutionList)`
`solveAll(+Goal, -SolutionList)`		`solveAll(-SolutionList)`
`solve(+Goal, -Solution, within(+Time))`		`solve(-Solution, within(+Time))`
`solveN(+Goal, +NSol, -SolutionList, within(+Time))`		`solveN(+NSol, -SolutionList, within(+Time))`
`solveAll(+Goal, -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)`	
	`setGoal(template(+Template))`
	`setGoal(index(+Index))`
`solve(+Goal, -Solution, ?Timestamp)`	`solve(-Solution, ?Timestamp)`
`solveN(+Goal, +NSol, -SList, ?TimeStamp)`	`solveN(+N, -SolutionList, ?TimeStamp)`
`solveAll(+Goal, -SList, ?TimeStamp)`	`solveAll(-SolutionList, ?TimeStamp)`
`solve(+Goal, -Solution, within(+Time), ?TimeStamp)`	`solve(-Solution, within(+Time), ?TimeStamp)`
`solveN(+Goal, +NSol, -SList, within(+Time), ?TimeStamp)`	`solveN(+NSol, -SList, within(+Time), ?TimeStamp)`
`solveAll(+Goal, -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()`	

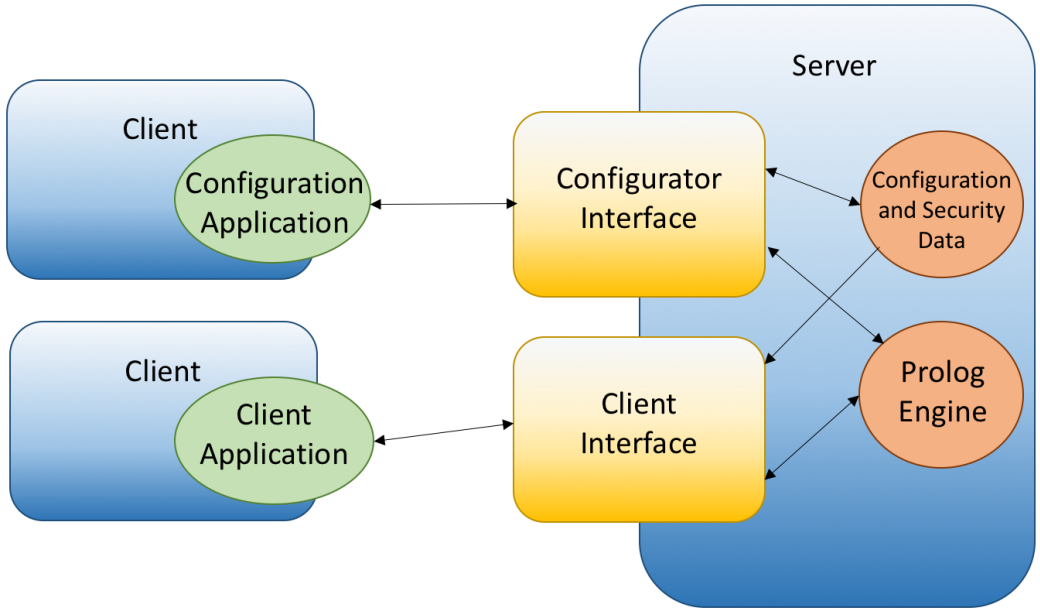
Next in Line...

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, 2017]
- 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...

tuProlog engine

- open-source, light-weight
- minimal and easily deployable
- dynamically configurable
- inter-operable
- multi-paradigm and multi-language
- multi-platform (Java, .NET, Android, iOS)

Minimality

tuProlog core contains only the Prolog engine essentials → other feature implemented via *libraries*

Interoperability

tuProlog also supports JSON serialisation natively, ensuring the interoperability required by a WS

Next in Line...

The Smart Bathroom Case Study I

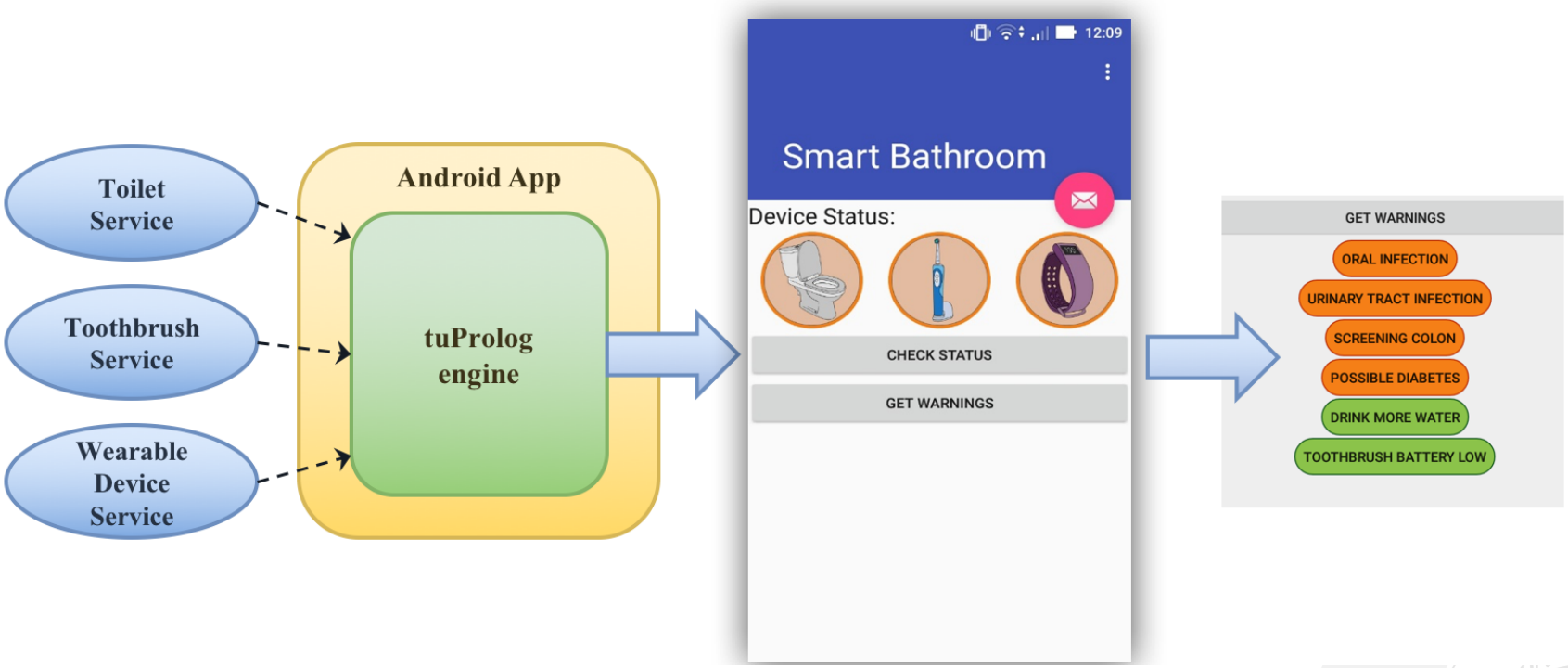
Testbed Scenario: monitor physiological functions

- 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

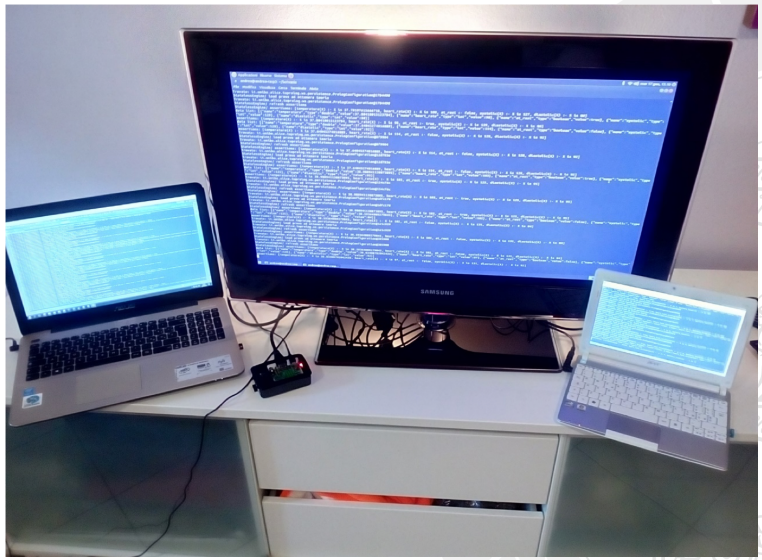
KB extraction of LPaaS Personal Server and LPaaS Toothbrush Server

```
%%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:

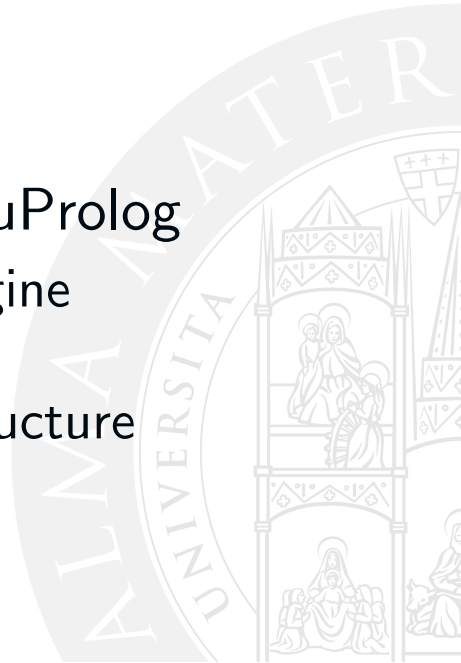
- toilet server: on Raspberry Pi 3 (Ubuntu Mate Arm)
- toothbrush server: on Lubuntu laptop
- personal server: on Windows 10 laptop
- client: tablet Lenovo A10 (Android 5.0.1)
- client: desktop application on Windows 10



Next in Line...

Conclusion

- pervasive and situated IoT scenarios requie a range of diverse degrees of intelligence to support *adaptation* and *self-management*
- LPaaS aims at fitting such a challenging context by providing
  - standard interface
  - both stateful and stateless services
  - either static or dynamic knowledge bases
  - fully configurable and customisable
- first working implementation designed on the top of tuProlog
  - → modular, multi-platform, and multi-language logic engine
  - → situated logic engines
  - → tuProlog as a service based on the usual SOA infrastructure



Further 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
- space-awareness and situatedness 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

References I

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 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.

EJB (2017).
Home Page.

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

References II


J2EE (2017).
Home Page.


Java Persistence API (2017).
Home Page.


Jersey (2017).
Home Page.


jose.4.j (2017).
Home Page.


Payara (2017).
Home Page.


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


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

Logic Programming as a Service (LPaaS): Intelligence for the IoT

Roberta Calegari[1] Enrico Denti[1] Stefano Mariani[2]
Andrea Omicini[1]
{roberta.calegari, enrico.denti, andrea.omicini}@unibo.it
stefano.mariani@unimore.it

[1]Dipartimento di Informatica, Scienza e Ingegneria—Università di Bologna
[2]Dipartimento di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia

Talk @ *14th IEEE International Conference on Networking, Sensing and Control (ICNSC 2017)*
Calabria, 16 May 2017