Mini-ME matchmaker and reasoner for the Semantic Web of Things

Floriano Scioscia, Michele Ruta, Giuseppe Loseto, Filippo Gramegna, Saverio Ieva,
Agnese Pinto, Eugenio Di Sciascio
Polytechnic University of Bari, Italy

{floriano.scioscia, michele.ruta, giuseppe.loseto, filippo.gramegna, saverio.ieva, agnese.pinto, eugenio.disciascio}@poliba.it

ABSTRACT

The Semantic Web of Things (SWoT) aims to support smart semantics-enabled applications and services in pervasive contexts. Due to architectural and performance issues, most Semantic Web reasoners are often impractical to be ported: they are resource consuming and are basically designed for standard inference tasks on large ontologies. On the contrary, SWoT use cases generally require quick decision support through semantic matchmaking in resource-constrained environments. This paper describes Mini-ME (the Mini Matchmaking Engine), a mobile inference engine designed from the ground up for the SWoT. It supports Semantic Web technologies and implements both standard (subsumption, satisfiability, classification) and non-standard (abduction, contraction, covering, bonus, difference) inference services for moderately expressive knowledge bases. In addition to an architectural and functional description, usage scenarios and experimental performance evaluation are presented on PC (against other popular Semantic Web reasoners), smartphone and embedded single-board computer testbeds.

Keywords: Ubiquitous Computing, Semantic Web, Internet of Things, Knowledge Representation, Mobile Reasoning.

INTRODUCTION

Semantic Web technologies have been acknowledged to promote interoperability and intelligent information processing in ubiquitous computing. Scenarios include supply chain management (Giannakis & Louis, 2016), ubiquitous commerce (Liu, 2013; De Virgilio, Di Sciascio, Ruta, Scioscia, & Torlone 2011), peer-to-peer resource discovery (Ruta, Di Sciascio, & Scioscia, 2011; Ruta, Scioscia, Ieva, Capurso & Di Sciascio, 2017) and so on. The ever-increasing computational resources and communications effectiveness of mobile devices enable ubiquitous processing and exchange of rich and structured information for context-aware resource discovery and decision support. The Semantic Web and the Internet of Things paradigms are converging more and more toward the so–called *Semantic Web of Things* (SWoT) (Ruta, Scioscia & Di Sciascio, 2012; Pfisterer *et al.*, 2011). It enables semantic-enhanced pervasive computing by embedding intelligence into ordinary objects and environments through a plethora of heterogeneous micro-devices conveying short information seeds.
Such a vision requires increased flexibility and autonomy of ubiquitous knowledge-based systems in information encoding, management, dissemination and discovery. User agents running on mobile personal devices should be able to discover dynamically the best available resources according to user's profile and preferences, in order to support her current tasks through unobtrusive and context-dependent suggestions. Reasoning and query answering are particularly critical issues, stimulating the need for further specialized inference services in addition to classical ones (like *subsumption* and *satisfiability* check). Furthermore, mobile computing platforms (e.g., smartphones, tablets) are still constrained by

hardware/software limitations with respect to typical setups for Semantic Web reasoning engines. In fact, architectural and performance issues affect the porting of current OWL-based reasoners, designed for the Semantic Web, to mobile devices (Bobed, Yus, Bobillo, & Mena, 2015).

This chapter describes *Mini-ME* (*the Mini Matchmaking Engine*) (Scioscia *et al.*, 2014b), a compact matchmaker and reasoner for the $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) Description Logic (DL). It is aimed to semantic matchmaking for resource/service discovery in mobile and ubiquitous contexts, although it is also a general-purpose Semantic Web inference engine. Optimized non-standard inference services allow a fine-grained categorization and ranking of matching resources w.r.t. a request, providing both a distance metric and a logic-based explanation of the outcomes. Mini-ME is suitable to a widespread class of applications where large sets of low-complexity component resources can be aggregated to build composed services with growing semantic complexity. This is fit for the computational and power supply limitations of resource providers in ubiquitous contexts and to their short storage availability. An "agile" service discovery architectures able to select, assemble and orchestrate on the fly many elementary components is more manageable and effective in mobile and pervasive applications.

Mini-ME uses the *OWL API* (Horridge & Bechhofer, 2011) to parse and manipulate Knowledge Bases in all supported syntaxes of Web Ontology Language (OWL) version 2 (World Wide Web Consortium [W3C], 2012). It exploits structural inference algorithms on unfolded and CNF (Conjunctive Normal Form) normalized concept expressions for efficient computations also on resource-constrained platforms. Mini-ME implements both standard reasoning tasks for Knowledge Base (KB) management *(subsumption, satisfiability, classification)* and non-standard inference services for semantic-based resource discovery and ranking *(abduction, contraction, covering, bonus, difference)*. The reasoner is developed in Java, with Android as the main target computing platform. Furthermore, reasoner and GUI (Graphical User Interface) plug-ins have been developed for the Protégé ontology editor (Musen, 2015). Mini-ME has already been employed in prototypical testbeds on mobile and embedded devices for ubiquitous and pervasive computing scenarios.

The chapter is organized as follows. The next section recalls needed theoretical background. Then, a functional and architectural description of the system is given, followed by three usage scenarios, respectively in ontology engineering with Protégé, mobile semantic augmented reality and semantic sensor networks. In the last part of the chapter some experimental evaluation is provided for both standard and non-standard inferences on PC, mobile and embedded computing platforms. Finally, relevant related work and main novel contribution are discussed, before closing remarks.

BACKGROUND

In what follows relevant reasoner background is presented and particularly the logic language it supports is described followed by an overview of the allowed inference services. This should make self-contained the architectural and implementation details given later on.

Description Logics

Description Logics - also known as Terminological languages, Concept languages – are a family of logic formalisms for Knowledge Representation in a decidable fragment of First Order Logic (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2002). The basic DLs syntax elements are: *concepts* (a.k.a. classes), denoting sets of objects; *roles* (a.k.a. properties), relating pairs of concepts; *individuals* (a.k.a. instances), denoting special named elements in concepts. A *semantic interpretation* is a pair $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$, consisting of a *domain* Δ and an *interpretation function* $\cdot^{\mathcal{I}}$ mapping every concept to a subset of Δ, every role to a subset of $\Delta \times \Delta$, and every individual to an element of Δ. Basic elements can be combined using *constructors* to compose concept and role *expressions*. Each DL has a different set of constructors: the conjunction of concepts, usually denoted as $\sqcap$, is used in every DL; some DLs also include disjunction $\sqcup$ and complement $\neg$. Roles can be combined with concepts using *existential* and *universal*

quantifications; other constructs may involve counting, such as number restrictions. Semantics of expressions is given by defining the interpretation function over each construct.
In DL-based reasoners, an *ontology* $\mathcal{T}$ (a.k.a Terminological Box or TBox) is composed by a set of assertions in the form $A \sqsubseteq D$ (*inclusion* a.k.a. *subsumption*) or $A \equiv D$ (*definition* a.k.a. *equivalence*), with A and D concept expressions. Particularly, a *simple*-TBox is an acyclic TBox such that: (i) A is always an atomic concept; (ii) if A appears in the left-hand side (lhs) of a concept definition assertion, then it cannot appear also in the lhs of any concept inclusion assertion. Mini-ME supports the $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) DL, which has polynomial computational complexity for standard and non-standard inferences in simple-TBoxes, whose depth of concept taxonomy is bounded by the logarithm of the number of axioms in it (see (Ruta *et al.*, 2011) for further explanation). This DL fragment has been selected because it grants good worst-case complexity and memory efficiency of non-standard inference algorithms for semantic matchmaking. Syntax and semantics of $\mathcal{ALN}$ DL constructs are summarized in Table 1.

Table 1 Syntax and semantics of $\mathcal{ALN}$ constructs and simple TBoxes

Name	Syntax	Semantics
Top	$\top$	Δ^I
Bottom	$\bot$	$\emptyset$
Intersection	$C \sqcap D$	$C^I \bigcap D^I$
Atomic negation	$\neg A$	$\Delta^I - A^I$
Universal quantification	$\forall R.C$	$\{ d_1 \mid \forall d_2 : (d_1, d_2) \in R^I \rightarrow d_2 \in C^I \}$
Number restriction	$(\geq n\ R)$	$\{ d_1 \mid \#\{ d_2 \mid (d_1, d_2) \in R^I \} \geq n \}$
	$(\leq n\ R)$	$\{ d_1 \mid \#\{ d_2 \mid (d_1, d_2) \in R^I \} \leq n \}$
Inclusion	$A \sqsubseteq D$	$A^I \subseteq D^I$
Definition	$A \equiv D$	$A^I = D^I$

According to the World Wide Web Consortium (W3C) Recommendation for the OWL 2 ontology language, an OWL 2 ontology is basically an RDF (Resource Description Framework) (W3C, 2014) graph referring to the OWL 2 vocabulary (W3C, 2012), serialized in RDF/XML syntax or in one of the other optional syntaxes.

Inference Services

Given a TBox $\mathcal{T}$ and a concept expression C, the **unfolding** procedure recursively expands references to axioms in $\mathcal{T}$ within the concept expression itself. In this way $\mathcal{T}$ is not needed anymore when executing subsequent inferences. On the other hand, **normalization** translates the unfolded concept expression in a canonical (*normal*) form through pre-defined substitution rules preserving its semantics. This makes subsequent inference algorithms applicable to any concept expression. When loading a KB, Mini-ME performs a preprocessing in order to execute unfolding and normalization in *Conjunctive Normal Form* (CNF) (Ruta *et al.*, 2011). Any $\mathcal{ALN}$ concept expression C can be reduced to CNF as: $C \equiv C_{CN} \sqcap C_{LT} \sqcap C_{GT} \sqcap C_{\forall}$, where C_{CN} is the conjunction of (possibly negated) atomic concept names, C_{LT} (respectively C_{GT}) is the conjunction of $\leq$ (resp. $\geq$) number restrictions (no more than one per role), and $C_{\forall}$ is the conjunction of universal quantifiers (no more than one per role; fillers are recursively in CNF). Normalization preserves semantic equivalence w.r.t. models induced by the TBox; furthermore, CNF is unique up to commutativity of conjunction operator (Ruta *et al.*, 2011). The normal form of an unsatisfiable concept is simply the bottom concept $\bot$.

Mini-ME was designed as a semantic matchmaker, *i.e.* a tool to find the best resources for a given request, when both resource and request descriptions are satisfiable concept expressions w.r.t. a common ontology. The following standard reasoning services are currently supported:

- **Concept Satisfiability** (a.k.a. consistency) **check**. In a semantic matchmaking framework, given a request R and a supplied resource S as concept expressions w.r.t. a common TBox $\mathcal{T}$, satisfiability allows to determine whether there is a partial (disjoint) match or not, by checking whether $\mathcal{T} \vDash S \sqcap D \sqsubseteq \bot$ holds or not. Due to CNF properties, satisfiability check is trivially performed during normalization.
- **Subsumption check**: subsumption determines whether the resource is a full (subsume) match for the request or not, by checking whether $\mathcal{T} \vDash S \sqsubseteq D$ holds or not. The classic structural subsumption algorithm is exploited, reducing the procedure to a set containment test (Baader *et al.*, 2002).

Further non-standard inference services were also implemented, allowing to (i) provide explanation of outcomes beyond the trivial "positive/negative" answer of satisfiability and subsumption tests, (ii) enable a logic-based relevance ranking of a set of available resources w.r.t. a specific query (Ruta *et al.*, 2011) and (iii) aggregate resources in order to satisfy complex requests:

- **Concept Contraction** (Colucci *et al.*, 2007): given a request R and a supplied resource S, if they are not compatible with each other, Contraction determines which part of R is conflicting with S. If one retracts conflicting requirements G (*Give up*) from R, a concept K (*Keep*) is obtained, representing a contracted version of the original request, such that $K \sqcap S$ is satisfiable w.r.t. $\mathcal{T}$. The solution G to Contraction represents "why" $R \sqcap S$ are not compatible.
- **Concept Abduction** (Colucci *et al.*, 2007): whenever R and S are compatible, but S does not imply R, Abduction allows to determine what should be hypothesized in S in order to completely satisfy R, also enabling a logic-based relevance ranking of a resource w.r.t. a given request (Ruta *et al.*, 2011). The solution H (*Hypothesis*) to Abduction represents "why" the subsumption relation $\mathcal{T} \vDash S \sqsubseteq R$ does not hold. H can be interpreted as *what is requested in R and not specified in S*.
- **Concept Covering**: many ubiquitous scenarios require that relatively large number of low-complexity resources are aggregated in order to satisfy an articulated request. To this aim, a further non-standard reasoning task based on the solution of *Concept Covering Problem* (CCoP, formally defined in (Ragone *et al.*, 2007) has been defined. It allows to: (i) cover (*i.e.*, satisfy) features expressed in a request as much as possible, through the conjunction of one or more instances of a KB – seen as elementary building blocks – and (ii) provide explanation of the uncovered part of the request itself. Given a concept expression R (request) and a set of instances $S = \{S_1, S_2, \ldots, S_n\}$ (available resources), where R and $S_1, S_2, \ldots, S_n$ are satisfiable in the reference ontology $\mathcal{T}$, *Concept Covering* aims to find a pair (S_c, H) where $S_c \subseteq S$ includes concepts in S covering R w.r.t. $\mathcal{T}$ as much as possible and H is the residual part of R not covered by concepts in S_c.
- **Concept Bonus**: a resource S could contain features not requested in R – possibly because the requester did not know or consider them – which could be useful in a query refinement process to improve matchmaking outcome. For this purpose, the Bonus non-standard inference service (Ruta *et al.*, 2011) extracts a concept B from S, which denotes something the resource provides even though the request did not ask for it.
- **Concept Difference**: finally, advanced multi-agent knowledge sharing and integration scenarios (Scioscia, Ruta & Di Sciascio, 2015) require ways to subtract information in an expression from another one. This is accomplished by means of the Concept Difference reasoning service (Teege, 1994).

Since S is an approximated match of R, it would be useful to evaluate how good the approximation is. Based on the uniqueness of CNF, a *norm* for concept expressions can be computed by "counting" the number of conjuncts in it (Ruta *et al.*, 2011). Hence, numerical *penalty functions* can be defined based on the norm of expressions G and H_K, which allow to evaluate the goodness of match approximation as well as to rank several resources w.r.t. a request. Also for Difference many valid solutions exist, but a maximality criterion should be adopted in this case (*i.e.*, subtract as much as possible).
In order to use Mini-ME in more general knowledge-based applications, the following reasoning services over ontologies were also implemented:

- **Coherence:** it is a simplified check w.r.t. Satisfiability, because it does not process individuals (the difference is discussed *e.g.* in (Moguillansky *et al.*, 2010). Since CNF normalization allows to identify unsatisfiable concepts, it is sufficient to normalize every concept during ontology parsing to detect unsatisfiabilities in the ontology.
- **Classification:** ontology classification computes the overall concept taxonomy induced by the subsumption relation, from $\top$ to $\bot$ concept. In order to reduce the subsumption tests, the following optimizations introduced in (Baader, Hollunder, Nebel, Profitlich & Franconi, 1994) were implemented: enhanced traversal top search, enhanced traversal bottom search, exploitation of told subsumers.

MINI-ME: SYSTEM OUTLINE AND FEATURES

Currently Mini-ME stills at 2.0 version, available at http://sisinflab.poliba.it/swottools/minime/. In what follows, a description of the matchmaker tool is provided, including architectural and implementation details.

General Architecture

The Mini-ME architecture is sketched as UML diagram in Figure 1. Main components are outlined hereafter:

- **Android Service**: implements a service (*i.e.*, a background daemon) any Android application can invoke to use the engine;
- **OWL API** (Horridge & Bechhofer, 2011): provides support for parsing and manipulating the OWL 2 language expressions;
- **MicroReasoner**: reasoner implementation, exposing fundamental KB operations (load, parse), as well as inference tasks;
- **KB Wrapper**: implements KB management functions (creation of internal data structures, normalization, unfolding) and inference procedures on ontologies (Classification and Coherence check);
- **Data Structures**: in-memory data structures for concept manipulation and reasoning; the inference procedures on concept expressions (Concept Satisfiability, Subsumption, Abduction, Contraction, Covering, Difference) are implemented at this level.

Figure 1. UML component diagram

Mini-ME 2.0 was developed in Java using Android SDK Tools, Revision 25, corresponding to Android Platform version 6.0 (API level 23). It was tested with previous version, however, resulting compatible with all devices running Android 2.1 or later. Mini-ME can be used as:

1. *Android Service* by Android applications;
2. library by calling public methods of the MicroReasoner component directly.

In the last case, it runs unmodified on Java *Standard* Edition runtime environment (version 6 or later) and supports all OWL 2 syntaxes accepted by the OWL API parser.

Inference Core Framework

Standard Java Collection Framework objects are used to define the low-level data structures package (mentioned before), composed of the following classes:

- **Item**: it represents a named concept expression. When parsing an ontology, the KB Wrapper component builds a Java HashMap object containing all concepts in the TBox as String-Item pairs. Each concept is unfolded, normalized and stored in the HashMap with its name as key and Item instance as value;
- **SemanticDescription**: models a concept expression in CNF as aggregation of C_{CN}, C_{LT}, C_{GT}, $C_{\forall}$ components, each one stored in a different Java ArrayList. Methods implement inference services;
- **Concept**: superclass of all concept types. Subclasses are AtomicConcept, UniversalRestriction and CardinalityRestriction, which is further extended by GreatherThanRestriction and LessThanRestriction. The *equals* method, inherited from java.lang.Object, has been overridden in order to properly implement a logic-based comparison;
- **Abduction** and **Contraction**: represent the result returned by Concept Abduction and Concept Contraction, respectively. Abduction contains a SemanticDescription as Hypothesis, while Contraction contains two SemanticDescription as Give Up and Keep. Furthermore, they both contain a penalty score induced by the inference procedure;
- **Composition**: represents the result returned by the Concept Covering service. It contains a vector of Item elements as covering set and a further one as uncovered part of the request.

Mini-ME exploits structural algorithms for standard and non-standard inference services on (unfolded and normalized) concept expressions. The careful optimization of the implementation of both algorithms and data structures enables efficient computations even on resource-constrained systems such as mobile and embedded devices. In particular, the following algorithms can be invoked through the *MicroReasoner* interface:

- **abduce**, implements the Concept Abduction service by means of a recursive procedure, reported in Figure 2;
- **contract**, performs the Concept Contraction service with a recursive algorithm, detailed in Figure 3;
- **cover**; exploits the above Abduction algorithm to solve a CCoP starting from a request and a set of compatible services. The related flow chart is in Figure 4.

Figure 2. Concept Abduction algorithm

Figure 3. Concept Contraction algorithm

Figure 4. Concept Covering algorithm

USE CASES

Mini-ME provides different tools – such as consistency check and classification tasks –assisting users in knowledge bases engineering in both classical and ubiquitous Semantic Web scenarios. Furthermore, it offers resource discovery capabilities through standard and non-standard inference services to build

applications and services for several application scenarios. In particular, by running on mobile and embedded devices, Mini-ME allows leveraging semantics also in pervasive Semantic Web of Things contexts, such as u-commerce, u-learning, u-healthcare, home and building automation, navigation and driving assistance systems, VANETs (Vehicular Ad-hoc NETworks), WSSANs (Wireless Semantic Sensor and Actor Networks), and many more. To focus just on few clear examples, this section illustrates how Mini-ME can be exploited in the following use cases: (i) knowledge engineering with an ontology editor; (ii) a mobile augmented reality explorer to discover points of interest; (iii) a cooperative WSSANs framework.

In the Semantic Web: Protégé Plugins

Mini-ME has been integrated in the *Protégé* ontology editor (Musen, 2015) through the implementation of an OWL reasoner plugin. Standard reasoning tasks are accessible through the Protégé user interface in the Reasoning menu. The entry point is the *MinimeReasoner* class, extending *AbstractProtegeOWLReasonerInfo*, which manages the synchronization between the knowledge base and the reasoner in case of changes to the currently loaded KB. The class also references the reasoner factory, responsible for the creation of the Mini-ME instance.

Figure 5. Protégé plugin for non-standard inferences

Moreover, an additional Protégé plugin was developed to:

- exploit non-standard inferences through a user-friendly GUI;
- support users during design and development of ontologies for pervasive and ubiquitous scenarios.

The existing DL Query plugin (http://github.com/protegeproject/dlquery) was adopted as guideline. The proposed plugin is a Tab Widget and it consists of the following components, highlighted in Figure 5:

A. *OWLIndividualsList* and *OWLIndividualsTypes* tabs, showing all KB instances with related descriptions;

B. *OWLAssertedClassHierarchy* and *OWLClassDescription* tabs, containing the general taxonomy along with the description of selected classes;

C. an input box used to select the inference task, the request R and –in case of abduction, contraction, bonus and difference– the resource annotation S through a simple list containing the IRI of all individuals defined within the KB. For Concept Covering, a subset of KB individuals can be checked through the Individuals List panel as available resources;

D. the results area shows the output of the chosen inference service. Particularly, Figure 5 shows a CCoP resolution: component individuals and the uncovered part of the request are displayed.

Due to plugin interface changes between Protégé versions 4 and 5, two different versions were developed to be used with Protégé 4.3 and 5.1.

In the Ubiquitous Semantic Web: Mobile Augmented Reality Explorer

Semantic-based technologies can support articulated and meaningful descriptions of locations and Points of Interest (POIs). The use of metadata (annotations) endowed with formal machine-understandable meaning enables advanced location-based resource discovery through proper inferences. The Mini-ME engine powers a novel discovery approach in Mobile Augmented Reality (MAR), implemented for Android (Ruta *et al.*, 2014). The overall architecture of the proposed ubiquitous POI discovery framework is depicted in Figure 6. It consists of the following components:

- The OpenStreetMap (OSM) server working as cartography provider. OSM map entities are semantically enriched in a way that best fits location-based resource discovery.
- A general method and an editor (Scioscia, Binetti, Ruta, Ieva, & Di Sciascio, 2014) for annotating maps, so allowing a collaborative crowd-sourced enhancement of basic OpenStreetMap cartography. The standard OWL 2 languages are exploited to create and share POI annotations, based on ontologies providing the conceptual vocabulary to express them.
- A MAR client providing the following features: (i) discovery of most relevant POIs w.r.t. user's annotated profile, via a logic-based matchmaking; (ii) visualization of POI annotations and examination of discovery results, through a fully visual user interface.

In order to allow users to store semantic annotations in a POI description retaining backward compatibility, new tags have been introduced complying with the basic key-value pair structure of OSM element tags (Scioscia *et al.*, 2014a).

Figure 6. Architecture of the MAR System

Figure 7. User interface of the semantic MAR explorer: (a) UI, (b) Abduction results, (c) Contraction results

The client application –shown in Figure 7a– was developed using Android SDK Tools, Revision 23, corresponding to Android Platform version 4.2.2 (API level 17). The system adopted a modified version of the Android Augmented Reality framework. Given POI target coordinates (latitude, longitude and altitude), it collects the azimuth and inclination angle between the device and the target from gyroscope and compass, in order to calculate where the device is pointing and its degree of tilt. Using this information, the system decides if and where a POI marker should be displayed within the viewfinder image on the screen.
In the proposed AR POI discovery framework, the user profile plays the role of request R. It consists of a concept expression including personal information like interests and hobbies. The profile can be either composed by browsing visually the ontology modeling the reference domain (Scioscia *et al.*, 2014a), or imported from other applications and services. Available resources are the annotated OSM POIs in the user's area, referred to the same ontology as the user profile. They are extracted from OSM server and cached in the MAR client. Several resource domains (cultural heritage, shopping, accommodation, etc.) can be explored by simply switching to the proper reference ontology. Hence the proposed system works as a general-purpose location-based discovery facilitator. Exploiting the embedded Mini-ME matchmaker, the application executes a semantic matchmaking between the user profile and the annotations of POIs in her surroundings (enclosed into semantic-enhanced OSM map). Figure 8 sketches the resource discovery process.

Figure 8. POI matchmaking process in the MAR system

The semantic description concerning each POI is stored as an attribute of its marker. A score is finally associated to each POI, estimating the result of the matchmaking between the user profile and the POI annotation. The overall resource score is computed with the utility function:

$$f(R, POI) = 100 * \left[1 - \frac{penalty(R, POI)}{penalty(R, \top)} * \left(1 + \frac{distance(User\_GPS, POI\_GPS)}{max\_distance} \right) \right]$$

where penalty(R, POI) is the semantic distance between profile R and POI; this value is normalized dividing by penalty(R,⊤), which is the distance between R and the universal concept and depends only on assertions in the ontology. Geographical distance (normalized by user-specified maximum range) is combined as weighting factor. The purposes of the utility function are: (i) to weight the result of semantic matching

according to distance and (ii) to translate the score to a more user-friendly scale. Of course nearer resources are preferred, but in case of a full match penalty(R, POI) = 0 hence f(R, POI) = 100 regardless of distance. By touching a marker, the user can see its features, which are presented as icons around a wheel shape, in order to provide a clear and concise description, as shown in the central portion (A) of Figure 7b. The View result panel (B) in Figure 7b lists all missing features w.r.t. user profile (C), computed through Concept Abduction. In case of incompatibility, the same left-hand menu shows the Concept Contraction outcome: properties the POI satisfies and incompatible elements (Figure 7c-(D)). Overall, the user can quickly identify what POI resources are most relevant to her needs, by looking at the POI marker color, at the matchmaking result shown in the score panel and – if interested – by exploring POIs features. Simple operations on the device touchscreen allow effortless information acquisition and management.
The proposed AR framework has been integrated in an indoor/outdoor navigation solution for people with motion disabilities, such as wheelchair users (Ruta, Scioscia, Ieva, De Filippis, & Di Sciascio, 2015). Notable features included: annotation of accessibility information in the OSM cartography; a built-in routing engine with support for both outdoor and indoor positioning and navigation in multi-level buildings. A functional prototype was developed for Android mobile devices specifically devoted to assist users with physical disabilities.

In the Semantic Web of Things: Cooperative Semantic Sensor Networks

The Constrained Application Protocol (CoAP) is becoming one of the most widely accepted application-layer protocols for the Web of Things (Bormann, Castellani, & Shelby, 2012). CoAP adopts the CoRE Link Format (Shelby, 2012) specification for resource discovery. This protocol only allows a syntactic string-matching of attributes, lacking any explicit and formal characterization of the resource semantics. To overcome this limit, the Mini-ME engine has been integrated in a framework for collaborative discovery of sensors and actuators in pervasive contexts. The proposed framework integrates the following components: (i) slight backward-compatible extensions to CoAP and CoRE Link Format resource discovery protocol (Ruta *et al.*, 2013); (ii) high-level event detection and annotation through resource-efficient data mining algorithms on raw data gathered by a Semantic Sensor Network (SSN) using the SSN-XG ontology as reference vocabulary (Compton *et al.*, 2012); (iii) non-standard inferences for semantic matchmaking for resource retrieval and ranking of approximate matches.

Figure 9. CoAP-based Sensor Network Architecture

The proposed framework architecture is shown in Figure 9. Sensors deployed in an area communicate with a local sink node, which acts as cluster head. Multiple sinks are connected to a gateway, interfacing the micronetwork toward the outside. Each sensor is characterized not only by data-oriented attributes, but also by a semantic annotation describing its features and functionalities. Sinks are able to: (i) register sensors along with their semantic description as CoAP resources; (ii) support logic-based resource discovery on annotated metadata. For these purposes, sink nodes embed CoAP servers. They also gather and process data for event detection. When an event is recognized, it is annotated and a resource record is updated in the server. Beyond the semantic annotation, the record contains further extra-logical context parameters, such as geographic coordinates and a timestamp. The gateway waits for resource discovery requests from client applications searching for events in the area, and replies on behalf of connected sink nodes.
Modules in the basic framework (Ruta *et al.*, 2013) were improved to support a collaborative sensing process (Ruta *et al.*, 2017b). Communication in SSNs was implemented using a modified version of Californium CoAP library, enabling the semantic-based enhancements of the CoAP protocol.
JOSM SSN plugin. Figure 10 shows the prototype GUI of the SSN plugin for the Java OpenStreetMap (OSM) open source editor. It can be used to perform the following tasks: (i) SSN browsing, showing on the map in (A) the available sensors and sink nodes registered on CoAP gateways; (ii) semantic-based sensor discovery, for customizing a semantic-based CoAP request (by specifying reference location, maximum discovery range, inference task to perform and relevance threshold) visually through panel (B) and sending

it to look for sensors in the area; (iii) SSN scenario generation to create random configurations for large-scale SSN simulations, through the panel (C) shown in Figure 4.6, which extends the *OSM to Rescue plugin* (Gobelbecker & Dornhege, 2009).

Figure 10. JOSM plugin for CoAP-based SSNs

CoAP Mobile Node. An Android-based client was developed using Android SDK Tools (Revision 21.1, corresponding to Android Platform version 4.2.2, API level 17) and tested on a Samsung GT-i9250 Galaxy Nexus smartphone. It was devised to support in-the-field communication with SSNs and to perform:

- SSN browsing and sensor discovery, where the user can select a gateway node and view all connected sensors or devices filtered-out by a semantic-based discovery. Each sensor can be also queried to retrieve data it measures;
- Collaborative sensing: when a mobile node (*e.g.*, an Android smartphone) queries a CoAP gateway, it can also act as information source, connected to the gateway temporarily. It can provide data coming from both its embedded micro-devices (e.g., accelerometer, gyroscope) and external sensing peripherals available through wired or wireless connections.

A client application can compose a discovery request and query a SSN gateway to find the set of most suitable sensors having an OWL semantic description referred to a shared ontology. The gateway carries out semantic matchmaking by solving a CCoP, in order to find the set of resources which together satisfy the request to the maximum extent. In case of a partial cover, the response includes both the semantic description of the uncovered part (H) of the request and the percentage of request still not covered. Furthermore, exploiting the proxy support built into CoAP, the gateway has the possibility to forward the uncovered part as a new request towards other SSN nodes in the area of interest, searching for more resources to satisfy missing features. In this way, each semantic-enabled gateway can start a collaborative and multi-hop resource discovery.

PERFORMANCE EVALUATION

An experimental campaign was performed with Mini-ME on PC, mobile and embedded systems, for standard and non-standard inference services. This allows to assess effectiveness of the proposed reasoner, both in comparison with other popular Semantic Web inference engines and w.r.t. requirements of SWoT applications. Followed evaluation criteria and obtained results are outlined hereafter.

Standard inferences on Personal Computers

Mini-ME was compared on PC with: *FaCT++* (Tsarkov & Horrocks, 2006) version 1.6.3, OWL API 3.4 (http://owl.man.ac.uk/factplusplus/); *HermiT* (Glimm, Horrocks, Motik, Stoilos, & Wang, 2014) version 1.3.8 (http://hermit-reasoner.com/) and *Pellet* (Sirin, Parsia, Cuenca-Grau, Kalyanpur & Katz, 2007) version 2.3.1 (http://clarkparsia.com/pellet/). All reasoners were tested via the OWL API on a PC testbed (Intel Core i7 CPU 860 at 2.80 GHz –4 cores/8 threads– with 8 GB DDR3 SDRAM (1333 MHz) memory, 1 TB SATA (7200 RPM) hard disk, 64-bit Microsoft Windows 7 Professional and 32-bit Java 7 SE Runtime Environment, build 1.7.0_03-b05). The reference dataset was taken from the 2012 OWL Reasoner Evaluation workshop (http://www.cs.ox.ac.uk/isg/conferences/ORE2012/materials.html): it is composed of 214 OWL ontologies with different size, expressiveness and syntax. For each reasoning task, two tests were performed: (i) correctness of results and turnaround time; (ii) memory usage peak. For turnaround time, each test was repeated four times and the average of the last three runs was taken. For memory tests, the final result was the average value of three runs.

Classification. The input of the classification task was the whole ontology dataset. For each test, one of the following possible outcomes was recorded:

- *Correct*, the computed taxonomy matched with the reference classification (if available in the dataset) or results of all the reasoners were the same. In this case the total time taken to load and classify the ontology was also recorded;
- *Parsing Error*, the ontology could not be parsed by the OWL API due to syntax errors;
- *Failure*, the classification task failed because the ontology contained unsupported logic language constructors;
- *Out of memory*, the reasoner generated an exception due to memory constraints;
- *Timeout*, the task did not complete within the timeout threshold (set to 60 minutes).

Mini-ME correctly classified 83 out of 214 ontologies; 71 were discarded due to parsing errors, 58 presented unsupported language constructors, and the timeout was reached in 2 cases. Pellet classified correctly 130 ontologies, HermiT 128, FaCT++ 122. The lower "score" of Mini-ME is due to the presence of General Concept Inclusions, cyclic TBoxes or unsupported logic constructors. Parsing errors occurred in the OWL API library and were therefore common to all reasoners.
Performance was measured only for the 73 ontologies correctly classified by all reasoners, dividing them in five categories, according totheir number of concepts:

- Extra Small (XS): fewer than 10 concepts; 13 ontologies were in this group;
- Small (S): between 10 and 100 concepts; 9 ontologies;
- Medium (M): between 100 and 1000 concepts; 25 ontologies;
- Large (L): between 1000 and 10000 concepts; 22 ontologies;
- Extra Large (XL): more than 10000 concepts; 4 ontologies.

Figure 11 compares the classification times of each reference reasoner w.r.t. the ontology categories. Pellet, HermiT and FaCT++ exhibited a similar trend, with the first two reasoners faster than the other engines for large ontologies. Mini-ME was very competitive for small-medium ontologies (up to about 1200 classes), but less for large ones. This can be considered as an effect of the Mini-ME design, which is optimized to manage elementary TBoxes.

Figure 11. Classification test on PC

For what concerns the *class satisfiability* test, the adopted dataset consisted of 107 ontologies. Nevertheless, only the 69 ontologies Mini-ME correctly classified in the previous test were considered: for each of them, the dataset specifically indicated one or more classes to be checked. As reported in Figure 12, performances were basically similar, with times differing only for few microseconds and no reasoner consistently faster or slower. Moreover, no correlation between time and ontology size is revisable, whereas time is in direct ratio with the complexity of class description and to its depth in the taxonomy (data not shown).

Figure 12. Class Satisfiability on PC

Figure 13 shows the ontology satisfiability test results. For this task, all reasoners presented performance similar to the ones reported in Figure 11 for classification. In fact, the ontology satisfiability test implies loading, classifying and checking consistency of all concepts in the ontology with the first two steps requiring the larger time. Outcomes of all reasoners were the same, except for ontologies with IDs 199, 200, 202, 203 in the dataset. In contrast to Pellet, HermiT and FaCT++, Mini-ME checks ontology coherence regardless of the ABox. The above ontologies included an unsatisfiable class (GO_0075043) with no instances, therefore the ontology was reported as incoherent by Mini-ME, but as satisfiable by the other reasoners.

Figure 13. Ontology Satisfiability on PC

Finally, Figure 14 reports on memory usage peak during classification, which was verified as the most intensive task. For small-medium ontologies, used memory was roughly similar for all reasoners. Mini-ME provided good results, with slightly lower memory usage than Pellet and HermiT and on par with FaCT++. Also for large ontologies Mini-ME results were comparable with the other reasoners, but in this case FaCT++ had the best performance.

Figure 14. Memory usage test on PC

Standard inferences on mobile and single-board computers

Using the same ontology dataset, results obtained on an Android smartphone (Samsung Galaxy Nexus GT-I9250 with dual-core ARM Cortex A9 CPU at 1,2 GHz, 1 GB RAM, 16 GB storage memory, and Android version 4.2.2) and on a Raspberry Pi Model B (equipped with a Broadcom BCM2835 system on a chip, with an ARM1176JZF-S CPU at 700 MHz, 512 MB RAM and Java 7 SE Runtime Environment build 1.7.0_40-b43) were compared to the above outcomes gathered for PC tests.
Results computed by Mini-ME on the mobile platforms were in all cases the same as on the PC. On the mobile and single-board devices, 75 ontologies out of 214 were correctly classified, 55 were discarded due to parsing errors, 56 had unsupported language constructors, 26 generated out-of-memory exceptions (these included ontologies correctly classified on PC or not classified due to parsing errors or unsupported constructors) and 2 of them reached the timeout. Figure 15 shows a comparison between the classification turnaround times on PC and on the mobile/embedded platforms. Data refer to the 73 ontologies correctly classified by Mini-ME on all devices.
Times were roughly an order of magnitude higher on both Android and Raspberry Pi devices. Absolute values for small-medium ontologies were under 1 second, so they can be deemed as acceptable in pervasive contexts. Furthermore, it can be noticed that the turnaround time increased linearly w.r.t. number of classes both on PC and on mobile platforms, thus confirming that Mini-ME has predictable behavior regardless of the reference platform. Similar considerations apply to class and ontology satisfiability tests: the turnaround time comparisons are reported in Figure 16 and Figure 17, respectively.

Figure 15. Classification on mobile and single-board computer (PC as baseline)

Figure 16. Class Satisfiability on mobile and single-board computer (PC as baseline)

Figure 17. Ontology Satisfiability on mobile and single-board computer (PC as baseline)

Memory allocation peak during the classification task was reported in Figure 18. For the Android platform, data were obtained exploiting the Android logging system, which provides a mechanism for collecting and viewing system debug output, including heap memory data and garbage collector calls. For small-medium ontologies, the required memory was roughly stable. Instead, for large ontologies the used memory increased according to the total number of classes. Moreover, in every test memory usage on Android and Raspberry Pi was significantly lower than on PC. This is due to the harder memory constraints on smartphones and single-board computes, imposing to have as much free memory as possible at any time. Furthermore, the tested smartphone exhibited better memory performance than the Raspberry Pi: indeed, the Android Dalvik virtual machine performs more frequent and aggressive garbage collection w.r.t. Java SE virtual machine. This reduces memory usage, but on the other hand can be responsible for a significant portion of the PC-smartphone turnaround time gap that was found.
Finally, a comparison was carried out using the five ontologies tested in (Bobed *et al.*, 2015). Mini-ME did not classify Wine and Pizza ontologies correctly because of unsupported language constructors, DBpedia produced a runtime error due to the presence of General Concept Inclusions, whereas GO and

NCI gave an out-of-memory exception even using the *android:largeHeap="true"* attribute in the application manifest (the other reasoners produced the same error).

Figure 18. Memory usage test on mobile and single-board computer (PC as baseline)

Non-standard inferences

A performance evaluation was carried out also for non-standard inferences using the same PC testbed and mobile platforms adopted for the previous tests. In this case the goal was to compare the general performance trends on the three different platforms, regardless of the specific PC, smartphone and Raspberry Pi configuration. The test performed both unfolding and normalization over a 557 kB knowledge base. 100 request/resource pairs were randomly generated starting from the ontology defined in (Ruta *et al.*, 2011), with an average size of 4.2 kB - and finally it was executed abduction and contraction between each pair. Every task was repeated four times and the average turnaround time of the last three runs was taken. Figure 19 reports time results (in microseconds) in case of PC, Android smartphone and Raspberry Pi board testbeds, respectively.

Figure 19. Non-standard inference tests

For each request/resource pair, Mini-ME executed a compatibility check; in case, abduction was performed, otherwise contraction was run, followed by abduction with the compatible part of the request. Notice that the computational time basically varies depending on the complexity of the semantic descriptions. Results for mobile and single-board tests were referred to the ones for PC in order to highlight non-standard inferences exhibit similar trends. Processing times are reported in a logarithmic scale: despite the performance gap between PC and mobile platforms, reasoning tasks maintained an acceptable computational load also on the latter. Times were roughly an order of magnitude higher on the mobile devices. This is due not only to their limited computational capabilities, but also – as said above – to the more frequent garbage collection by the Android Dalvik virtual machine. This trend was even more apparent on the Raspberry Pi, which is characterized by lower computational resources than the Android reference platform. On all platforms, all request/resource pairs show slightly variable memory peak values due to the similar structure of their semantic descriptions. For this reason, Figure 20 reports only on the average value of memory peak. Non-standard inferences on the mobile device required 11.32 MB on average with a standard deviation of 27 kB, on Raspberry Pi the average was 11.64 MB with a standard deviation of 5 kB, whereas on PC the average was 23.88 MB with a standard deviation of 4 kB.

Figure 20. Memory usage peak for non-standard inferences

RELATED WORK

When processing semantic-based information to infer entailed implicit knowledge, painstaking optimization is needed to achieve acceptable performance for adequately expressive languages (Baader, Hollunder, Nebel, Profitlich, & Franconi, 1994; Horrocks & Patel-Schneider, 1999). This is specifically true in case of logic-based matchmaking for mobile and ubiquitous computing, which are characterized by resource limitations affecting not only processing, memory and storage capabilities, but also energy consumption. Most mobile inference engines currently provide only rule processing for entailments materialization in a KB (Koch, Meyer, Dignum, & Rahwan, 2006; Tai, Keeney, & O'Sullivan, 2011; Kim, Park, Hyun, & Lee, 2010; Motik, Horrocks, & Kim, 2012), so resulting unsuitable to support applications requiring non-standard inference tasks and extensive reasoning over ontologies (Motik *et al.*, 2012). More expressive languages could be used by adapting tableaux algorithms –whose variants are implemented in reasoners running on PCs– to mobile computing platforms, but an efficient implementation of reasoning services is still an open problem. Several techniques (Horrocks & Patel-Schneider, 1999) allow increasing expressiveness or decrease running time at the expense of main

memory usage, which is precisely the most constrained resource in mobile systems. Latest trends in performance optimization involve combining different types of reasoning methods, selecting the best algorithms based on the logical expressiveness of a knowledge base or the required inference service. Reasoners following this approach include *MORe* (Armas-Romero, Cuenca-Grau, Horrocks & Jiménez-Ruiz, 2013), *PAGOdA* (Zhou, Nenov, Cuenca-Grau & Horrocks, 2015) and *Konclude* (Steigmiller, Liebig & Glimm, 2014), which also exploits parallelism afforded by multi-core computing systems to achieve top performance in OWL standard inference services (Parsia, Matentzoglu, Gonçalves, Glimm & Steigmiller, 2015).

Focusing on ubiquitous contexts, semantic-based resource discovery was early investigated in (Avancha, Joshi, & Finin, 2002). There, the need for discovery mechanisms more powerful than string-matching was clearly pointed out for the first time. The issue of approximate matches –lacking exact ones– was discussed, but no formal frameworks were given. In (von Hessling, Kleemann, & Sinner, 2004) semantic user profiles were introduced to increase accuracy in matching services in a mobile environment. If there was no intersection between user interests and service offers, authors concluded the user was not interested in the service; a complete and integrated solution for matching degree calculation was not provided. *Pocket KRHyper* (Sinner & Kleemann, 2005) was the first reasoning engine specifically designed for mobile devices. It supported the $\mathcal{ALCHIR}+$ DL and was built as a Java ME (Micro Edition) library. Pocket KRHyper was exploited in a DL-based matchmaking framework between user profiles and descriptions of mobile resources/services (Kleemann & Sinner, 2005). However, frequent "out of memory" errors strongly limited the size and complexity of manageable logic expressions. To overcome performance constraints, tableaux optimizations to reduce memory consumption were introduced in (Steller & Krishnaswamy, 2008) and implemented in *mTableau*, a modified version of Java Standard Edition Pellet reasoner (Sirin *et al.*, 2007). Comparative performance tests were executed on a PC, showing faster turnaround times than both unmodified Pellet and *Racer* (Haarslev & Müller, 2001) reasoners.

As pointed out by Matentzoglu *et al.* (2015), most reasoners are currently based on the Java programming language. This is not surprising, since the main OWL frameworks, such as the OWL API (Horridge & Bechhofer, 2011) and *Jena* (McBride, 2002), are implemented in Java. Nevertheless, the Java SE technology is not expressly tailored to the current generation of handheld devices. In fact, other relevant inference engines cannot run on common mobile platforms, since they rely on Java class libraries incompatible with most widespread mobile OS (*e.g.*, Android). Bobed *et al.* (2015) recently ported five Semantic Web reasoners to the Android platform (Pellet, *CB* (Kazakov, 2009), Hermit (Glimm *et al.*, 2014) and *JFact*, a Java port of Fact++ (Tsarkov & Horrocks, 2006)), albeit with significant rewriting or restructuring effort in some cases. Similarly, in (Kazakov & Klinov, 2013) the *ELK* reasoner (Kazakov, Krötzsch, & Simancík, 2014) was optimized and evaluated on Android.

Nevertheless, all ported systems were designed mainly for batch jobs over large ontologies and/or expressive languages. This makes mobile device usage less suitable due to computation and memory constraints. The non-standard services of Mini-ME are more useful in ubiquitous scenarios, where mobile agents must provide quick decision support and/or on-the-fly organization in intrinsically unpredictable contexts. Moreover, it can be observed that the above systems, like the matchmaking framework proposed in (Kleemann & Sinner, 2005), only support standard inference services such as satisfiability and subsumption, which provide only binary "yes/no" answers. Consequently, they can only distinguish among *full (subsume)*, *potential (intersection-satisfiable)* and *partial (disjoint)* match types, as defined in (Colucci *et al.*, 2007) and (Li & Horrocks, 2004) respectively. Analogously, in the HTTP-based ubiquitous infrastructure by (Pfisterer *et al.*, 2011), queries allow only exact matches with facts derived from a support knowledge base. Non-standard inferences like abduction and contraction are needed to enable approximate matches, semantic ranking and explanations of outcomes (Colucci *et al.*, 2007). For this reason, initial efforts have been made to port the OWL API to iOS (Ruta, Scioscia, Di Sciascio & Bilenchi, 2016), in order to support cross-platform SWoT applications and services.

In latest years, the bad worst-case complexity of OWL language stimulated a different approach to implement reasoning tools. It was based on simplifying both the underlying logic languages and admitted

KB axioms, so that structural algorithms could be adopted, while maintaining sufficient expressiveness for broad application areas. In (Baader, Brandt, & Lutz, 2005), the basic $\mathcal{EL}$ DL was extended to $\mathcal{EL}^{++}$, a language deemed suitable for various applications, characterized by very large ontologies with moderate expressiveness. A structural classification algorithm was also devised, which allowed high-performance $\mathcal{EL}^{++}$ ontology classifiers such as *CEL* (Baader, Lutz, & Suntisrivaraporn, 2006), *Snorocket* (Lawley & Bousquet, 2010) and ELK. OWL 2 profiles definition complies with this perspective, focusing on language subsets of practical interest for important application areas rather than on fragments with significant theoretical properties. In a parallel effort motivated by similar principles, in (Ruta, Di Noia, Di Sciascio, Piscitelli, & Scioscia, 2008) an early approach was proposed to adapt non-standard logic-based inferences to pervasive computing contexts. By limiting expressiveness to the $\mathcal{AL}$ language, acyclic, structural algorithms were adopted reducing standard and non-standard inference tasks to set-based operations. KB management and reasoning were then executed through a data storage layer, based on a mobile RDBMS (Relational DBMS). Such an approach was further investigated in (Ruta, Scioscia, Di Noia, & Di Sciascio, 2009) and (Ruta *et al.*, 2011), by increasing the expressiveness to $\mathcal{ALN}$ DL and allowing larger ontologies and more complex descriptions, through the adoption of both mobile OODBMS (Object-Oriented DBMS) and performance optimized data structures. Finally, in (Ruta, Scioscia, & Di Sciascio, 2010) expressiveness was extended to $\mathcal{ALN}$(D) DL with fuzzy operators. The above tools were designed to run on Java Micro Edition devices and were adopted in several case studies in ubiquitous computing, employing semantic matchmaking over moderately expressive KBs. The reasoning engine presented here recalls lessons learned in those previous efforts, and aims to provide a standards-compliant implementation of most common inferences (both standard and non-standard) for Android, the most widespread mobile platform worldwide.

SUMMARY OF CONTRIBUTIONS

As the chapter pointed out, the theoretical work on the mobile and embedded reasoner named Mini-ME provided several contributions to the state of the art on reasoning for the Semantic Web (of Things). They can be summarized as in what follows:

- Mini-ME was the first mobile reasoner expressly designed and implemented for the Android mobile platform, albeit supporting also Java SE;
- The engine includes both standard reasoning services for KB manipulation and non-standard inference tasks for semantic matchmaking, resource discovery and decision support applications;
- Logical language choice and architectural and implementation optimizations make Mini-ME efficient on both conventional computer architectures and mobile and embedded systems, as demonstrated by an extensive experimental campaign;
- Mini-ME supports a wide range of Semantic Web (of Things) scenarios effectively; this chapter focused particularly on (i) knowledge engineering and KB editing, (ii) mobile resource discovery and (iii) autonomous Wireless Semantic Sensor Networks.

PROJECT DESCRIPTION AND PERSPECTIVES

The paper presented a prototypical reasoner devised for the ubiquitous computing. It supports Semantic Web technologies through the OWL API and implements both standard and non-standard reasoning tasks. Developed in Java, it targets the Android platform but also runs on Java SE and on embedded devices like Raspberry Pi. Experiments were performed both on PCs and smartphones and evidenced: (i) correctness of implementation, (ii) competitiveness with state of-the-art reasoners in standard inferences, (iii) acceptable performance on target mobile devices.

The work on Mini-ME started in 2011 leveraging both theoretical and practical results reached in 10 years of research on the semantic-based matchmaking and reasoning. This effort produced several publications in international journals, conferences and workshops. In recent years, numerous case studies and prototypes have been developed for different ubiquitous scenarios to prove the feasibility and benefits of non-standard inferences, including ubiquitous commerce (Di Noia *et al.*, 2008), ambient intelligence and infomobility services (Ruta, Scioscia, Di Noia, & Di Sciascio, 2010), home and building automation (Ruta, Scioscia, Loseto, & Di Sciascio, 2014), up to semantic blockchain systems (Ruta *et al.*, 2017a). A selection of applications can be viewed at http://sisinflab.poliba.it/swottools/. A widespread exploitation of Mini-ME is ongoing in a project funded under the Apulia Region Cluster research program in the healthcare field. Furthermore, some large Italian enterprises have shown their interest for the engine, due to its flexibility and scalability, for an extensive application in their products or services. The intimate connection between theoretical research, engineering innovation and practical problem-oriented solutions is at the heart of the research activities of the Information Systems Laboratory at Polytechnic University of Bari, and in this perspective Mini-ME is being continuously developed and improved as a key enabling technology. Besides further performance optimization leveraging peculiarities of Android Dalvik (for Android versions up to 4.4) and ART (for versions 5.0 and above) runtimes, ongoing work includes a port to the iOS platform (also compatible with macOS) in the Swift programming language. This is an opportunity to refine architecture and design, as well as to include further optimizations for improving reasoning efficiency, which could be backported to the Java version. Further improvements under investigation include: support for ABox management; implementation of further reasoning tasks; the support for more expressive languages, *e.g.*, with $\mathcal{EL}^{++}$ extension of abduction and contraction algorithms.

REFERENCES

Avancha, S., Joshi, A., & Finin, T. (2002). Enhanced Service Discovery in Bluetooth. *IEEE Computer*, 35(6), 96–99.

Armas-Romero, A., Cuenca-Grau, B., Horrocks, I., & Jiménez-Ruiz, E. (2013). MORe: a Modular OWL Reasoner for Ontology Classification. In Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE), 61-67.

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the EL envelope. In *Proceedings of 19th International Joint Conference on Artificial Intelligence*, 364–399.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2002). *The Description Logic Handbook*. Cambridge, United Kingdom: Cambridge University Press.

Baader, F., Hollunder, B., Nebel, B., Profitlich, H., & Franconi, E. (1994). An empirical analysis of optimization techniques for terminological representation systems. Applied Intelligence, 4(2), 109–132.

Baader, F., Lutz, C., & Suntisrivaraporn, B. (2006). CEL – a polynomial-time reasoner for life science ontologies. *Automated Reasoning*, 287–291.

Bobed, C., Yus, R., Bobillo, F., & Mena, E. (2015). Semantic reasoning on mobile devices: Do Androids dream of efficient reasoners? *Web Semantics: Science, Services and Agents on the World Wide Web*, 35, 167-183.

Bormann, C., Castellani, A.P., & Shelby, Z.(2012). CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Computing*, 16(2), 62-67.

Colucci, S., Di Noia, T., Pinto, A., Ragone, A., Ruta, M., & Tinelli, E. (2007). A Non-Monotonic Approach to Semantic Matchmaking and Request Refinement in E-Marketplaces. Int. Jour. of Electronic Commerce, 12(2), 127–154.

Compton, M., Barnaghi, P., Bermudez, L., Garcìa-Castro, R., Corcho, O., Cox, S. Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Le Phuoc, D., Lefort, L., Leggieri M., Neuhaus, H., Nikolov. A., Page, K., Passant, A., Sheth, A., & Taylor, K. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17, 25–32.

De Virgilio, R., Di Sciascio, E., Ruta, M., Scioscia, F., & Torlone, R. (2011). Semantic-based rfid data management. In *Unique Radio Innovation for the 21st Century*, 111-141.

Di Noia, T., Di Sciascio, E., Donini, F. M., Ruta, M., Scioscia, F., & Tinelli, E. (2008). Semantic-based Bluetooth-RFID interaction for advanced resource discovery in pervasive contexts. *International Journal on Semantic Web and Information Systems*, 4(1), 50-74.

Giannakis, M., & Louis, M. (2016). A multi-agent based system with big data processing for enhanced supply chain agility. *Journal of Enterprise Information Management*, 29(5), 706-727.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3), 245-269.

Gobelbecker, M., & Dornhege, C. (2009). Realistic Cities in Simulated Environments - An OpenStreetMap to Robocup Rescue Converter. In *Proceedings of 4th International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED 2009)*.

Haarslev, V., & Müller, R. (2001). Racer system description. *Automated Reasoning*, 701–705.

Horridge, M., & Bechhofer, S. (2011). The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1), 11-21.

Horrocks, I., & Patel-Schneider, P. (1999). Optimizing description logic subsumption. *Journal of Logic and Computation*, 9(3), 267–293.

Kazakov, Y. (2009). Consequence-driven reasoning for Horn SHIQ ontologies. In *International Joint Conference on Artificial Intelligence*, 9, 2040–2045.

Kazakov, Y., & Klinov, P. (2013). Experimenting with ELK Reasoner on Android. In *Proceedings. of 2nd International Workshop on OWL Reasoner Evaluation (ORE)*, 68–74.

Kazakov, Y., Krötzsch, M., & Simancík, F. (2014). The Incredible ELK. *Journal of Automated Reasoning*, 53(1), 1–61.

Kim, T., Park, I., Hyun, S., & Lee, D. (2010). MiRE4OWL: Mobile Rule Engine for OWL. In Proceedings of the 34th IEEE Annual Computer Software and Applications Conference Workshops (CompSACW), 317–322.

Kleemann, T., & Sinner, A. (2006). User Profiles and Matchmaking on Mobile Phones. In *Proceedings of 16th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2005)*, 135-147. Springer.

Koch, F., Meyer, J.-J. C., Dignum, F., & Rahwan, I. (2006). Programming deliberative agents for mobile services: the 3APL-M platform. In *Programming multi-agent systems*, 222–235. Springer.

Lawley, M. J., & Bousquet, C. (2010). Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In Proceedings of 6th Australasian Ontology Workshop (IAOA'10), 45-49.

Li, L., & Horrocks, I. (2004). A software framework for matchmaking based on Ssemantic Wweb technology. *International Journal of Electronic Commerce*, 8(4), 39–60.

Liu, Q. (2013). U–commerce research: a literature review and classification. *International Journal of Ad Hoc and Ubiquitous Computing*, 12(3), 177–187.

Matentzoglu, N., Leo, J., Hudhra, V., Sattler, U., & Parsia, B. (2015). A Survey of Current, Stand-alone OWL Reasoners. In *Proceedings of the 4th OWL Reasoner Evaluation (ORE) Workshop*, 68-79.

McBride, B. (2002). Jena: A Semantic Web toolkit. *IEEE Internet Computing*, 6(6), 55-59.

Moguillansky, M., Wassermann, R., Falappa, M. (2010). An argumentation machinery to reason over inconsistent ontologies. Advances in Artificial Intelligence–IBERAMIA 2010, 100–109.

Motik, B., Horrocks, I., & Kim, S. M. (2012). Delta-reasoner: a semantic web reasoner for an intelligent mobile platform. In Proceedings of the 21st International Conference Companion on World Wide Web, 63–72.

Musen, M. A. (2015). The Protégé project: a look back and a look forward. *AI matters*, 1(4), 4-12.

Parsia, B., Matentzoglu, N., Gonçalves, R. S., Glimm, B., & Steigmiller, A. (2015). The OWL reasoner evaluation (ORE) 2015 competition report. *Journal of Automated Reasoning*, 1-28.

Pfisterer, D., Romer, K., Bimschas, D., Hasemann, H., Hauswirth, M., Karnstedt, M., Kleine, O., Kröller, A., Leggieri, M., Mietz, R., Pagel, M., Passant, A., Richardson, R., & Truong, C. (2011). SPITFIRE: toward a Semantic Web of things. *IEEE Communications Magazine*, 49(11), 40–48.

Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F. M., Colucci, S., & Colasuonno, F. (2007). Fully automated web services discovery and composition through concept covering and concept abduction. *International Journal of Web Services Research (JWSR)*, 4(3), 85–112.

Ruta, M., Di Noia, T., Di Sciascio, E., Piscitelli, G., & Scioscia, F. (2008). A semantic-based mobile registry for dynamic RFID-based logistics support. In *Proceedings of the 10th International Conference on Electronic Commerce*, 1-9.

Ruta, M., Di Sciascio, E., & Scioscia, F. (2011). Concept abduction and contraction in semantic-based P2P environments. *Web Intelligence and Agent Systems*, 9(3), 179–207

Ruta, M., Scioscia, F., De Filippis, D., Ieva, S., Binetti, M., & Di Sciascio, E. (2014). A semantic-enhanced augmented reality tool for OpenStreetMap POI discovery. *Transportation Research Procedia. 17th Meeting of the EURO Working Group on Transportation*, 479-488. Elsevier.

Ruta, M., Scioscia, F., Di Noia, T., & Di Sciascio, E. (2009). Reasoning in Pervasive Environments: an Implementation of Concept Abduction with Mobile OODBMS. In Proceedings of *2009 IEEE/WIC/ACM International Conference on Web Intelligence*, 145–148.

Ruta, M., Scioscia, F., Di Noia, T., & Di Sciascio, E. (2010). A hybrid ZigBee/Bluetooth approach to mobile semantic grids. *Computer Systems Science and Engineering*, 25(3), 235–249.

Ruta, M., Scioscia, F., & Di Sciascio, E. (2010). Mobile Semantic-based Matchmaking: a fuzzy DL approach. In *Proceedings of the 7th Extended Semantic Web Conference*, 16–30.

Ruta, M., Scioscia, F., & Di Sciascio, E. (2012). Enabling the Semantic Web of Things: framework and architecture. In *Proceedings of the Sixth IEEE International Conference on Semantic Computing (ICSC 2012)*, 345-347.

Ruta, M., Scioscia, F., Di Sciascio, E., & Bilenchi, I. (2016). OWL API for iOS: early implementation and results. In Proceedings of the OWL: Experiences and Directions (OWLED) and OWL Reasoner Evaluation Workshop (OWLED-ORE), 141-152. Springer.

Ruta, M., Scioscia, F., Ieva, S., Capurso, G., & Di Sciascio, E. (2017). Semantic Blockchain to Improve Scalability in the Internet of Things. *Open Journal of Internet Of Things (OJIOT)*, Special Issue: Proceedings of the International Workshop on Very Large Internet of Things (VLIoT 2017) in conjunction with VLDB 2017, 3(1), 46-61.

Ruta, M., Scioscia, F., Ieva, S., De Filippis, D., & Di Sciascio, E. (2015). Indoor/outdoor mobile navigation via knowledge-based POI discovery in augmented reality. In *Proceedings of the 3rd International Workshop on Data Oriented Constructive Mining and Multi-Agent Simulation and the 7th International Workshop on Emergent Intelligence on Networked Agents (DOCMAS/WEIN-2015)*, 26-30.

Ruta, M., Scioscia, F., Loseto, G., & Di Sciascio, E. (2014). Semantic-based resource discovery and orchestration in Home and Building Automation: a multi-agent approach. *IEEE Transactions on Industrial Informatics*, 10(1), 730–741.

Ruta, M., Scioscia, F., Pinto, A., Di Sciascio, E., Gramegna, F., Ieva, S., & Loseto, G. (2013). Resource annotation, dissemination and discovery in the Semantic Web of Things: a CoAP-based framework. In *Proceedings International Conference on Green Computing and Communication (GreenCom) and Internet of Things (iThings) and Cyber, Physical and Social Computing (CPSCom)*, 527-534.

Ruta, M., Scioscia, F., Pinto, A., Gramegna, F., Ieva, S., Loseto, G., Di Sciascio, E. (2017). Cooperative Semantic Sensor Networks for pervasive computing contexts. In Proceedings of the 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI 2017), 38-43. New York: IEEE.

Scioscia, F., Binetti, M., Ruta, M., Ieva, S., & Di Sciascio, E. (2014). A Framework and a Tool for Semantic Annotation of POIs in OpenStreetMap. *Transportation Research Procedia. 16th Meeting of the EURO Working Group on Transportation*, 1092-1101.

Scioscia, F., Ruta, M., & Di Sciascio, E. (2015). A swarm of Mini-MEs: reasoning and information aggregation in ubiquitous multi-agent contexts. 4th OWL Reasoner Evaluation Workshop (ORE 2015), CEUR Workshop Proceedings, 1207, 15-22.

Scioscia, F., Ruta, M., Loseto, G., Gramegna, F., Ieva, S., Pinto, A., & Di Sciascio, E. (2014). A mobile matchmaker for the Ubiquitous Semantic Web. International Journal on Semantic Web and Information Systems, 10(4), 77-100.

Shelby, Z. (2012). Constrained RESTful Environments (CoRE) Link Format, *Internet Engineering Task Force (IETF) Request For Comments (RFC) 6690*, retrieved June 30th, 2017, from https://tools.ietf.org/html/rfc6690

Sinner, A., & Kleemann, T. (2005, July). KRHyper - In Your Pocket. In *Proceedings of 20th International Conference on Automated Deduction (CADE)*, 452-457.

Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2), 51–53.

Steigmiller, A., Liebig, T., & Glimm, B. (2014). Konclude: system description. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27, 78-85.

Steller, L., & Krishnaswamy, S. (2008). Pervasive Service Discovery: mTableaux Mobile Reasoning. In *International Conference on Semantic systems (I-Semantics)*, 93-101.

Tai, W., Keeney, J., & O'Sullivan, D. (2011). COROR: a composable rule-entailment OWL reasoner for resource-constrained devices. *Rule-Based Reasoning, Programming, and Applications*, 212–226.

Teege, G. (1994). Making the Difference: A Subtraction Operation for Description Logics. Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR94), 540–550.

Tsarkov, D., & Horrocks, I. (2006). FaCT++ description logic reasoner: System description. *Automated Reasoning*, 292–297.

von Hessling, A., Kleemann, T., & Sinner, A. (2004). Semantic User Profiles and their Applications in a Mobile Environment. In Workshop on Aartificial Iintelligence in Mmobile Ssystems (AIMS), 59-63.

World Wide Web Consortium (2012). OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). *W3C Recommendation 11 December 2012*, retrieved June 30th, 2017, from https://www.w3.org/TR/owl2-syntax/

World Wide Web Consortium (2014). RDF 1.1 Concepts and Abstract Syntax. *W3C Recommendation 25 February 2014*, retrieved June 30th, 2017, from https://www.w3.org/TR/rdf11-concepts/

Zhou, Y., Nenov, Y., Grau, B. C., & Horrocks, I. (2015). Ontology-based Query Answering with PAGOdA. In Proceedings of the 4th OWL Reasoner Evaluation (ORE) Workshop, 1-7.