OULU 2016
C 581

ACTA UNIVERSITATIS OULUENSIS

Xiang Su

LIGHTWEIGHT DATA AND KNOWLEDGE EXCHANGE FOR PERVASIVE ENVIRONMENTS

*UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING*

ACTA UNIVERSITATIS OULUENSIS
C Technica 581

XIANG SU

LIGHTWEIGHT DATA AND KNOWLEDGE EXCHANGE FOR PERVASIVE ENVIRONMENTS

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in Auditorium IT116, Linnanmaa, on 21 September 2016, at 12 noon

UNIVERSITY OF OULU, OULU 2016

Supervised by
Professor Jukka Riekki

Reviewed by
Doctor Ora Lassila
Associate Professor Kerry Taylor

Opponent
Professor Johan Lilius

ISBN 978-952-62-1334-7 (Paperback)
ISBN 978-952-62-1335-4 (PDF)

ISSN 0355-3213 (Printed)
ISSN 1796-2226 (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2016

Su, Xiang, Lightweight data and knowledge exchange for pervasive environments.
University of Oulu Graduate School; University of Oulu, Faculty of Information Technology and Electrical Engineering
Acta Univ. Oul. C 581, 2016
University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

Pervasive environments are physical spaces saturated with devices collecting data, controlling the environment, and interacting with users. These environments support human users in their everyday tasks so that the users can focus on their own tasks and access services and resources whenever and wherever they want. Such environments are also called smart spaces. Knowledge-based systems would enable realizing a variety of intelligent applications for pervasive environments. Generally, such systems recognize the situation in the environment from sensor data and utilize automated reasoning techniques to respond to the situation and the needs of the users.

However, building knowledge-based systems for pervasive environments presents challenges. This dissertation focuses on the challenge of data and knowledge representations. Knowledge-based systems utilize expressive knowledge representations that are verbose and require sufficient resources in order to use them. Most devices in pervasive environments cannot handle these representations as the devices have limited resources for computation, storage, and communication. The main aim of this dissertation is to tackle this challenge. That is, on the one hand, pervasive environments demand data and knowledge representations that do not require many resources from the resource-constrained devices; and on the other hand, the representations should be compatible with the knowledge-based systems. Specifically, a general solution is required that enables many applications to use the same data with minimal effort from application developers.

This dissertation presents a novel representation, Entity Notation (EN), to tackle these challenges. EN is designed as a general lightweight representation for data and knowledge. EN expresses entities, their properties, and property values. This structure resembles the triple structure of Resource Description Framework (RDF) and Web Ontology Language (OWL). Hence, sensor data in EN syntax can be transformed into common knowledge models in a straightforward manner and utilized with ease by knowledge-based systems. EN Schema is designed for transferring advanced knowledge models. Moreover, EN also offers an approach to shorten the format with templates and prefixes. This way, EN can be utilized by resource-constrained devices and environments. Our evaluation verifies that small devices can utilize EN to transfer data and knowledge to devices realizing intelligent functions, such as inference. Moreover, the expressive power of EN is comparable with the alternative representations. Finally, resource consumption is verified by prototypes. Based on the evaluation, we can conclude that EN can facilitate harnessing the full potential of pervasive environments.

Keywords: Entity Notation, knowledge-based systems, ontology, reasoning, Resource Description Framework, resourceconstrained devices, semantics

Su, Xiang, Kevyen datan ja tietämyksen esitystavat jokapaikan tietotekniikan sovelluksille.
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Tieto- ja sähkötekniikan tiedekunta
Acta Univ. Oul. C 581, 2016
Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä

Kaikkialla läsnäolevat ympäristöt ovat fyysisiä tiloja täynnä laitteita, jotka keräävät dataa, ohjaavat ympäristöä ja ovat vuorovaikutuksessa käyttäjien kanssa. Nämä ympäristöt tukevat ihmisiä päivittäisissä tehtävissä siten, että ihmiset voivat keskittyä tehtäviinsä sekä käyttää erilaisia palveluja ja resursseja ajanhetkestä ja paikasta riippumatta. Tällaisia ympäristöjä kutsutaan myös älykkäiksi tiloiksi. Tietämysjärjestelmät mahdollistavat monia sovellusskenaarioita näihin ympäristöihin. Nämä järjestelmät tunnistavat ympäristössä vallitsevan tilanteen sensoridatan avulla ja hyödyntävät automaattista päättelyä reagoidakseen tilanteeseen ja käyttäjien tarpeisiin.

Tietämysjärjestelmien kehitys näihin ympäristöihin on kuitenkin haasteellista. Tämä väitöskirja keskittyy datan ja tiedon esitystavan haasteisiin. Tietämysjärjestelmät käyttävät ilmaisuvoimaisia tietämyksen esitysmalleja, joiden monimuotoisuus puolestaan edellyttää riittäviä resursseja. Monet laitteet eivät pysty käsittelemään näitä esitysmalleja koska niillä ei ole riittäviä laskenta-, kommunikaatio- ja tallennusresursseja. Väitöskirjan päätavoite on ratkaista tämä haaste: Toisaalta dataa ja tietämyksen esitysmalleja on voitava käsitellä niukoilla resursseilla; toisaalta esitysmallien on oltava yhteensopivia erilaisten tietämysjärjestelmien kanssa. Erityisesti tarvitaan yleisratkaisu, joka voidaan yhdistää useaan sovellukseen vähäisellä sovelluskehitystyöllä.

Tämä väitöskirja esittää ratkaisuksi uuden datan ja tietämyksen esitystavan, Entity Notation -mallin (EN). EN on suunniteltu yleiseksi ja kevyeksi tiedon ja tietämyksen esitystavaksi. EN ilmaisee entiteettejä sekä niiden ominaisuus-arvopareja. Tämä rakenne muistuttaa RDF-kuvauskieltä sekä OWL-ontologiakieltä. Täten sensoridata EN-kielessä voidaan muuttaa suoraviivaisesti yleisiksi tietämysmalleiksi ja hyödyntää helposti tietämysjärjestelmissä. EN Schema on suunniteltu tietämysmallien siirtämiseen. EN tarjoaa myös tavan lyhentää muotoa mallineilla ja etuliitteillä. Näin EN-esitystapaa voidaan hyödyntää resurssirajoitteisissa laitteissa ja ympäristöissä. Tehdyt kokeet osoittavat, että pienet laitteet voivat käyttää EN-esitystapaa tiedon ja tietämyksen siirtämiseen älykkäitä toimintoja toteuttaviin laitteisiin. Lisäksi EN-esitystavan ilmaisuvoimaisuus on verrattavissa vaihtoehtoisiin esitystapoihin. Prototyyppien avulla tarkistettiin resurssien tehokas käyttö. Kokeiden perusteella voidaan todeta, että EN-esitystapa helpottaa läsnäolevien ympäristöjen täyden potentiaalin hyödyntämistä.

Asiasanat: ontologia, päättely, resurssirajoitteiset laitteet, semantiikka, tietämysjärjestelmät

To my whole family

Acknowledgements

First and foremost, my very special thanks to my supervisor, Professor Jukka Riekki. I would not be here without his guidance, trust, and support. He can always point out new directions of research in a couple of words. Jukka gave me lots of freedom in doing my research work the way I wished. I am indebted to him for his constant encouragement, patience, and invaluable guidance throughout this research.

I also wish to thank Dr. Ora Lassila and Associate Professor Kerry Taylor for reviewing this dissertation and providing me very helpful and constructive comments. I would like to thank Professor Johan Lilius for serving as the opponent in the doctoral defence.

The Department of Computer Science and Engineering offers an excellent working environment for carrying out this research. Especially, I would like to thank Dr. Janne Haverinen for his great help in measuring resource consumption of sensors, and Dr. Ekaterina Gilman for all the discussions and nice cooperation in research and some student projects. Great thanks are also expressed to my other colleagues Mika Rautainen, Susanna Pirttikangas, Yuhong Li, Mikko Perttunen, Iván Sánchez, Marta Cortés, Oleg Davidyuk, Timo Saloranta, Mika Oja, Mikko Polojärvi, Mikko Pyykkönen, Teemu Leppänen, Mikko Kauppila, Marko Jurmu, Szymon Sasin, Jiehan Zhou, Meirong Liu, and Pingjiang Li for constructive discussions. Office staff and computer administrators are also acknowledged for keeping everything running so smoothly. Moreover, during this research, I made a one-year visit to Department of Computer Science and Engineering, Helsinki University of Technology (now Aalto University). I would like to thank the hosting from Professor Sasu Tarkoma. Thanks also go to Jaakko Kangasharju, Pin Nie, and Zhihua Jin for interesting discussions and comments. I also wish to thank Professor Jukka K. Nurminen, Johanna Nieminen, Ari Keränen, Markus Koskimies, and Ilari Maarala for their great contribution in the Internet of Things ICT SHOK program.

I gratefully acknowledge the financial support from Infotech Oulu graduate school, NOKIA Foundation, HPY Research Foundation, Tauno Tönning Foundation, Finnish Foundation of Electronics Engineers, Walter Ahlström Foundation, Tekniikan Edistämis-säätiö, and University of Oulu Graduate School funding for finalizing this dissertation. Part of the research was carried out in Knowledge Environment for Interacting ROBOt

SWARMs (ROBOSWARM) project supported by the Sixth Framework Programme for Research and Technological Development, Pervasive Service Computing project funded by the Ubiquitous Computing and Diversity of Communication (MOTIVE) programme of the Academy of Finland, and Internet of Things (IoT) program funded by DIGILE and Tekes.

Finally and most importantly, I owe eternal gratitude to my splendid wife and our four lovely children, Yun, Yan, Xuan, and Ziyuan. Our kids bring us lots of happiness. My parents Jinsheng Su and Xiaoling Li have been always there for supporting and encouraging me, no matter what choices I have made. I can always feel their love and understanding in these years in Finland, 6,631 km far away from my hometown.

Oulu, June 2016

Abbreviations

AmI	*Ambient Intelligence*
CNL	*Controlled Natural Language*
CoAP	*Constrained Application Protocol*
CONON	*CONtext ONtology*
CPU	*Central Processing Unit*
CWM	*Closed World Machine*
DL	*Description Logic*
EBNF	*Extended Backus-Naur Form*
EN	*Entity Notation*
EXI	*Efficient XML Interchange*
FOAF	*Friend Of A Friend*
FOL	*First Order Logic*
GPS	*Global Positioning System*
GSN	*Global Sensor Networks*
HDT	*RDF Header-Dictionary-Triples*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
IRI	*Internationalized Resource Identifier*
Jess	*Java Expert System Shell*
JSON	*JavaScript Object Notation*
JSON-LD	*JSON for Linked Data*
M2M	*Machine to Machine*
MCU	*Microcontroller*
MIME	*Multipurpose Internet Mail Extensions*
N3	*Notation 3*
OGC	*Open Geospatial Consortium*
OWL	*Web Ontology Language*
PC	*Personal Computer*
PML	*Product Markup Language*
PoI	*Point of Interest*
RDBMS	*Relational Database Management System*

RDF	*Resource Description Framework*
RDFa	*Resource Description Framework in Attributes*
RDFS	*RDF Schema*
RFCOMM	*Radio Frequency Communication*
RFID	*Radio-frequency Identification*
RIF	*Rule Interchange Format*
RIF-BLD	*RIF Basic Logic Dialect*
RSS	*RDF Site Summary*
RuleML	*Rule Markup Language*
SIOC	*Semantically Interconnected Online Communities*
SenML	*Media Types for Sensor Markup Language*
SOUPA	*Standard Ontology for Ubiquitous and Pervasive Applications*
SRAM	*Static random-access memory*
SSL	*Secure Sockets Layer*
SSN	*Semantic Sensor Network*
SWRL	*Semantic Web Rule Language*
TCP	*Transmission Control Protocol*
Turtle	*Terse RDF Triple Language*
UCS	*Universal Coded Character Set*
URN	*Uniform Resource Name*
UUID	*Universally Unique Identifier*
W3C	*World Wide Web Consortium*
WAP	*Wireless Application Protocol*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition Language*

Contents

1 Introduction

1.1 Background and motivation

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it" [1]. Mark Weiser described his vision of ubiquitous computing, now also called pervasive computing, in 1991. The essence of that vision was the creation of pervasive environments that are saturated with computing and communication capabilities and interact with human users.

Not surprisingly, this vision was ahead of the time when articulated. However, recent advances in sensing technology, embedded computing, human-computer interaction, and wireless communication bring pervasive environments closer to our daily lives. Such pervasive environments contain a large amount of devices, such as sensors, actuators, and mobile phones, to perceive the environment, store and process data, pursue goals, and perform actions to fulfill the needs of human users.

Two trends can be observed regarding the capabilities of these networked devices. On the one hand, some devices, such as networked sensor nodes, are composed of simple components. They can contain only sensing electronics, a modest microcontroller (MCU), a transmission chip, and an energy source. These sensors are small and easy to embed in physical environments, but have limited resources for computation, storage, and communication. On the other hand, mobile devices have better resources, which allow pushing to mobile devices the functionality, seen only on Personal Computers (PCs) and server machines before. For example, modern smart phones have resources for knowledge processing and inference.

Pervasive environments facilitate the development of various applications and systems that support human users in their daily lives. Building knowledge-based systems is a potential way to enable advanced functions. When information can be expressed in well-structured knowledge models, intelligent decisions with justifications can be provided. A knowledge-based system here refers to a computer program that reasons and uses a knowledge base to solve problems, and here a knowledge base refers to technology used to store complex structured and unstructured information [2]. In general, such systems facilitate realizing pervasive environments, in which "massively distributed devices operate collectively while embedded in the environment using information and intelligence that is hidden in the interconnection network" [3], which is also the main vision of Ambient Intelligence (AmI).

Hence, applications facilitating the daily life of their users can be built by distributing knowledge components into physical devices in the environment. Compared with deploying knowledge-based systems on a centralized system or Cloud, this approach enables harnessing the computing capabilities of the devices in the environments and decreasing the amount of communication. Moreover, users have better opportunities to control privacy of their own data, when data is stored locally. However, distributing knowledge systems introduces several challenges, including knowledge exchange among system nodes, knowledge discovery in the environment, and aggregating knowledge to support inference at different levels. Solutions are needed to access, disseminate, and utilize knowledge in a flexible fashion in highly dynamic pervasive environments.

From these challenges, we focus in this dissertation on data and knowledge exchange. Generally, data is a set of values of qualitative or quantitative variables; in other words, data are individual pieces of information [4]. We focus on data captured by sensors and human-computer interfaces. Knowledge is the "justified true belief"; it forms meaning from data and enables computer systems to extract facts from data. The knowledge that computer systems utilize for reasoning has to be evident and coherent. These features allow knowledge-based systems to reason with judgmental, imprecise, and qualitative knowledge, as well as reason with formal knowledge of established theories [2].

Semantic Web is an extension of the current Web and aims at a common framework for sharing and reusing data across heterogeneous applications and systems. Semantic Web technologies offer powerful representation facilities and reasoning techniques, and facilitate data and knowledge modelling, querying, reasoning, service discovery, privacy, and provenance. Semantic Web technology-based knowledge processing is a good candidate for implementing intelligent functionality for pervasive environments. For example, it supports rule-based reasoning and Description Logic (DL)-based reasoning, knowledge reuse and knowledge sharing mechanisms. However, applying Semantic Web technologies in pervasive environments is challenging, especially as these environments contain resource-constrained devices. This is because common Semantic Web technologies require a considerable amount of resources and such resources are not available in constrained environments.

In this dissertation, we tackle this challenge at data and knowledge interchange level. Most popular representations for data and knowledge are developed by Semantic Web communities and standardization organizations, such as World Wide Web Consortium

(W3C). These models and their representations, including RDF [5] and Web Ontology Language 2 (OWL 2) [6], and their serializations such as RDF in eXtensible Markup Language (XML) and Terse RDF Triple Language (Turtle) [7], have mainly been designed for editing, storing, and utilizing knowledge, and as such are not optimal for resource-constrained devices in pervasive environments. Meanwhile, data formats utilized by pervasive systems, such as Generic Sensor Format [8] and JavaScript Object Notation (JSON) [9] cannot be transformed into knowledge representations in a straightforward manner. Hence, there is a need to develop a data and knowledge representation that can be both handled by resource-constrained devices and transformed into Semantic Web representations. [1]

1.2 Objectives and scope

The main objective of this research is to study a data and knowledge representation that both complies with Semantic Web technologies and can be used by resource-constrained devices in pervasive environments. Such a representation would facilitate building a large set of mobile and pervasive applications and services that utilize both resource-constrained devices and Semantic Web technologies.

The detailed objectives of this research are the following:

The first objective is to design a lightweight representation that can be utilized by resource-constrained devices.With a lightweight representation, we minimize the resource usage of CPU, memory, bandwidth, and energy consumption for encoding and decoding this representation. Hence, any simple sensor in pervasive environments could be able to connect with knowledge-based systems using minimal computation power and energy, and deliver messages to other system components in a resource-efficient manner. This would enable a large number of advanced functions. For example, applications could reason for the data produced by the sensors, reuse knowledge, and coordinate the sensors with actuators.

The second objective is to design a representation that advances interoperability among knowledge-based components in mobile devices, servers, and other systems deployed in pervasive environments. Such a representation facilitates transferring both raw sensor data and ontology knowledge. Different data formats and Semantic Web

[1]When this work was started, there were no knowledge representations targeted to resource-constrained environment. Some parallel efforts, such as JSON for Linked Data (JSON-LD) [10] started during this work. We discuss these efforts later in this dissertation.

languages have been designed separately. However, we expect to design a uniform solution to express similar semantics for more than one language. Our aim is that the representation developed in this dissertation enables mapping to different types of ontology knowledge.

The third objective is to design a representation that can be generally utilized by distributed systems. We do not restrict the type of information. Different types of data, from raw sensor measurements to structured knowledge, can be expressed and transferred in pervasive environments. Although we focus on pervasive environments and sensor data in this dissertation, the designed representation should be usable to exchange information between any two components of a distributed system. That is, the representation should be suitable for different applications and systems. The information should always be identifiable in a unique fashion. This is an important feature for general scalable systems when all possible uses of the information cannot be specified in advance. Researchers in several related research domains, including Ambient Intelligence (AmI), Internet of Things (IoT), machine to machine (M2M) communications, sensor networks, and mobile robotics could take advantage of this representation to enable numerous intelligent applications.

To summarize the objectives, we study a lightweight and general data and knowledge representation that advances interoperability. We limit the scope of this dissertation as follows: First, we have examined W3C Semantic Web standards, and consider RDF 1.1 and OWL 2 as essential models for developing knowledge interoperability. Hence, we select RDF 1.1 and OWL 2 ontology as the knowledge models that the representation developed in this dissertation has to support. That is, we emphasize interoperability with RDF 1.1 and OWL 2.

Second, we do not design new protocols. When a device sends data to another system component, it describes the data using the representation developed in this dissertation and passes the data as payload for the protocol performing the actual delivery. The receiver decomposes the data into the knowledge representation it uses. The protocol delivering the messages can be, for example, Hypertext Transfer Protocol (HTTP), Transmission Control Protocol (TCP), or Constrained Application Protocol (CoAP) [11].

Third, we do not design new ontologies. Although a representation supporting semantics facilitates transferring data in pervasive environments, it is not all that is needed. In addition, the meaning encoded in the messages needs to be shared by all entities producing and consuming the information. Ontologies are needed for this purpose. The existing ontologies, like Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [12], OntoSensor [13], and Semantic Sensor Network (SSN) Ontology [14], are good alternatives. From these, SSN Ontology developed by the W3C Semantic Sensor Networks Incubator Group is a well-designed upper level ontology for describing sensors and observations. It has the advantage that it is currently being developed through the standard processes of the Open Geospatial Consortium (OGC) and the W3C.

1.3 Contributions

This dissertation presents contributions in three areas. First, we design Entity Notation (EN) as a lightweight data and knowledge representation that can be handled by resource-constrained devices, transferred over low power communication links with limited bandwidth, and transformed into knowledge representations in a straightforward fashion. EN produces short packets which in their simplest form are only seventeen bytes long and efficiently decrease the amount of processing power and communication bandwidth required from the sensors and their radio interfaces. At the same time, EN can be transformed into other syntaxes of RDF 1.1 in a straightforward manner. In short, EN is proposed as a candidate solution for bridging the gap between resource-constrained sensors and other components of pervasive environments, such as knowledge-based systems, at the data interchange level. EN is also a practical and useful data representation with necessary expressive power and human-readable syntax.

Second, we design EN Schema for representing meta-data. EN Schema is designed as a syntax for OWL 2 and it provides a simple solution for knowledge exchange in pervasive environments. EN and EN Schema together facilitate ontology transfer among devices and applications in pervasive environments. Knowledge can be decomposed into small entities, individual packets that can be communicated separately and assembled in an unambiguous fashion. This results in a distributed description that can be transferred even one small message at a time. Individual packets for one ontology can be distributed to different devices at different times, and be transferred via different routes. Individual packets produced by different devices can be transferred incrementally to compose large ontologies. This feature enables transferring knowledge in a very flexible fashion in highly dynamic environments.

Third, we implement EN and EN Schema in several pervasive systems, verify their usefulness, and suggest solutions for their optimization for different pervasive systems.

We compare the expressive power of EN with other representations. EN and EN Schema are evaluated in different resource usage aspects, including computing, memory and communication usage, and energy consumption. Our evaluation shows that they have sufficient expressive power and their compactness is comparable with the best of other techniques. These prototypes, comparisons, and evaluations show that EN and EN Schema are practical solutions.

1.4 Research methodology and history

This work contains both theoretical and constructive research [15]. The theoretical part includes examining the state-of-the-art data formats and knowledge representations utilized in pervasive and distributed systems and designing syntaxes for EN and EN Schema. The constructive part of this research includes implementing prototypes and evaluating them. Moreover, some evaluation and analysis are performed based on well-known data sets and ontologies.

The work in this dissertation was performed in 2006 - 2015. This research was supported by Infotech Oulu graduate school from January 2007 to January 2010, and was investigated also in three research projects: Interacting ROBOt SWARMs (ROBOSWARM) project supported by Sixth Framework Programme for Research and Technological Development, Pervasive Service Computing project funded by the Ubiquitous Computing and Diversity of Communication (MOTIVE) programme of the Academy of Finland and Internet of Things program supported by DIGILE and Tekes.

We started this research with the idea of connecting resource-constrained devices with knowledge-based systems. We firstly designed a representation while considering some underlying protocol issues. We called it Entity Protocol at the beginning. After about one year, we learnt from our implementations that it is a better idea to separate representation from protocols. We focused on designing a representation which can be payload for protocols, such as HTTP and CoAP, although we still include some packet negotiation strategy issues in our research.

The first version of EN that could be transformed into RDF statements, along with a simulator and a gesture recognition prototype, was published as a conference paper [16]. This early design of EN was utilized in a simulator [16][17] and a context-aware map prototype [18]. In this early design, we included information about type of RDF objects with separate tokens as mandatory information in complete EN packets. However, when we later designed EN Schema for transferring ontologies, we noticed that information on type of RDF objects is often included in property information in practice. That is, the type of a RDF object is described in well defined working ontologies. To simplify the complete EN packet format, we consider property type information as optional information from complete EN packets and made corresponding modifications to related prototypes. The new EN syntax including all EN features for sensors was published in Personal and Ubiquitous Computing Journal [19], the first version of ambient social interaction prototype with new EN syntax was published in 2011 International AmI conference [20], and a detailed evaluation against other data formats and approaches was published in Concurrency and Computation: Practice and Experience Journal [21]. Furthermore, the latest results of latency evaluation against other data formats with real traffic scenario was published in 4th International Conference on the Internet of Things [22] and accepted by IEEE IoT Journal [23]. The design and evaluation of the ambient social interaction prototype was published in Journal of Ambient Intelligence and Smart Environments [24]. Meanwhile, we investigated transferring ontologies with EN and EN Schema packets. The first results, including early design, preliminary implementation and evaluation, were published in [25]. The latest version complying with OWL 2 is included in this dissertation. Moreover, we studied distributing intelligence in pervasive environments based on the capabilities of the devices [26] and the role of EN in bridging the gap of Semantic Web and networked sensors [27].

The work presented in this dissertation only includes the latest design of EN and the prototypes using this design. The latest design has been slightly modified based on [19]. Hence, some design considerations and evaluation results differ from some earlier publications. This is mainly because changing the representation of type information of RDF objects had an effect on the results.

During the time this research was in progress, we applied EN to different application domains. Hence, the research results have been published at different forums: robotics conferences, ubiquitous and context-aware computing conferences, sensor network conferences, AmI conferences and journals, and IoT conferences and journals.

1.5 Dissertation structure

The rest of this dissertation is organized as follows: Chapter 2 presents the key concepts and provides a literature review about data and knowledge exchange for pervasive environments. Chapter 3 introduces the design details of EN. This chapter presents the formal description of the alternative formats, and how EN can be transformed into

RDF statements. Furthermore, we discuss packet negotiation and chaining operations, supported data structures, and main limitations. Chapter 4 introduces EN Schema for expressing OWL 2 ontologies. Chapter 5 presents the evaluations. We analyze resource usage by transferring standard RDF data sets and some ontologies. Moreover, we evaluate EN in a simulator and in several prototypes. Finally, Chapter 6 concludes this dissertation by presenting general analysis, discussing how the research objectives were met, summarizing the contributions, discussing open issues, and sketching possible future work.

2 Data and knowledge representations for pervasive environments

This chapter presents an overview of data and knowledge representations for pervasive environments. We start with clarifying concepts about data and knowledge representations, then present different representations for connecting devices to knowledge-based systems and modelling pervasive environments, and finally present the state of the art of utilizing semantic representations in pervasive systems.

2.1 Knowledge representations and reasoning

Building knowledge-based systems is a feasible way for developing a variety of intelligent applications for pervasive environments. A knowledge-based system is a computer program that reasons and uses a knowledge base to solve complex problems [2]. One core issue is representing information about the world in a form that computer programs can utilize. Each reasoning technique is normally associated with a certain knowledge representation. Perttunen et al. [28] surveyed different knowledge representations and reasoning methods, including logic programming, ontology-based representation, case-based representation, and reasoning about uncertainty. They identified ontology-based representation as the most widely used and promising approach. Ontology-based representations meet the requirements set by pervasive systems, such as distributed knowledge creation and composition, partial validation, and precise and traceable formality [29]. Hence, ontology-based reasoning offers good possibilities to apply artificial intelligence methods and tools in pervasive environments.

Hence, we consider ontology-based representations and reasoning techniques as the most important approaches for building knowledge-based systems in pervasive environments. Gruber et al. [30] originally define ontology as an explicit specification of a conceptualisation. Studer et al. [31] define ontology as:

> "An ontology is a formal, explicit specification of a shared conceptualisation. A 'conceptualisation' refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. 'Explicit' means that the type of concepts used, and the constraints on their use are explicitly defined. For example, in medical

> domains, the concepts are diseases and symptoms, the relations between them are causal and a constraint is that a disease cannot cause itself. 'Formal' refers to the fact that the ontology should be machine readable, which excludes natural language. 'Shared' reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group."

For modelling pervasive environments, an ontology can be developed as a set of precise descriptive statements about pervasive environments themselves and entities inside pervasive environments. Precise statements prevent misunderstandings and ensure other components, such as reasoners, to behave in a predictable manner.

Description logics (DL) are a family of knowledge representation formalisms based on First-Order Logic (FOL) and well-structured computational properties. DL-based knowledge bases are usually divided into two parts: TBox and ABox [32]. TBox is a "terminological component" and it contains statements describing a system in terms of controlled vocabularies. ABox is an "assertion component" and it contains TBox-compliant statements about that vocabulary. In practice, TBox includes a set of concepts and properties for these concepts. ABox includes assertions on individuals. Taking the example of pervasive systems, ABox includes specific data of devices and system components, such as sensor measurement data [33], and TBox knowledge includes a class taxonomy and property definitions for describing pervasive environments. Such knowledge is also known as data schema for providing a data modelling vocabulary. Together, TBox and ABox statements make up a knowledge base which enables intelligent functionality for pervasive systems.

A combination of ontologies and reasoners is important. Reasoners are software components able to infer new logical relations from information stored in knowledge bases. Ontologies focus on specifying knowledge and DL reasoning typically focuses on subsumption and realization. Subsumption determines sub-concept/super-concept relationships for concepts in the TBox, whereas realization computes whether a given individual necessarily belongs to a particular concept [34]. In general, reasoners verify the consistency of knowledge models and make implicit knowledge explicit. There are many different techniques that can be applied for reasoners, in addition to DL, First Order Logic (FOL) and probability theory, for example.

RDF is a family of W3C specifications for representing information about resources in the World Wide Web. It was originally designed for representing metadata about Web resources, but has become a general method for conceptual description and modeling of information. RDF Schema (RDFS) [35] provides an extension of the basic RDF, and declares basic classes describing RDF and providing basic elements when describing the terms used in RDF. Most RDFS constructs are included in the more expressive OWL 2.

OWL is a standard Semantic Web language based on DL [36]. The main building blocks of OWL are concepts representing sets of objects, roles representing relationships between objects, and individuals representing specific objects. With OWL, complex concepts can be described through constructors that define the conditions on concept membership. OWL 2 is a revised extension of OWL, which is now commonly called OWL 1. OWL 2 extends OWL 1 with qualified cardinality restrictions, property chains, etc. Moreover, OWL 2 provides support for defining properties to be reflexive, irreflexive, transitive, and asymmetric, and to define disjoint pairs of properties. Three profiles, namely OWL 2 EL, OWL 2 QL, and OWL 2 RL, have been developed for balancing expressive power and reasoning efficiency, targeting different application scenarios. OWL 2 EL is suitable for applications utilizing ontologies to define very large numbers of classes and properties. OWL 2 QL can be tightly integrated with Relational Database Management Systems (RDBMSs) and can hence benefit from relational database technology. OWL 2 RL is suitable for applications that require scalable reasoning without sacrificing too much expressive power [37].

One of the earliest formalisms combining OWL and rules is the Semantic Web Rule Language (SWRL) [38]. Syntactically, SWRL extends the syntax of OWL with additional constructs to form Horn-style rule axioms. SWRL rules can be described with XML syntax. This syntax is based on Rule Markup Language (RuleML) and OWL/XML.

An alternative format, Rule Interchange Format (RIF) [39], became W3C recommendation in 2013. RIF does not provide a general rule language, instead it enables rule exchange amongst diverse rule systems and reasoning engines as different systems use various rule formats and formal logic paradigms. RIF ensures compatibility between RDF and OWL ontologies. RIF Basic Logic Dialect (RIF-BLD) maps different logic rules from the original format to RIF XML serialization syntax and it can be also used as a rule language. Jena is a Java framework for building Semantic Web applications, and it includes a number of predefined reasoners, including transitive reasoner, RDFS rule reasoner, OWL, OWL Mini, OWL Micro Reasoners, and generic user defined rule reasoner [40].

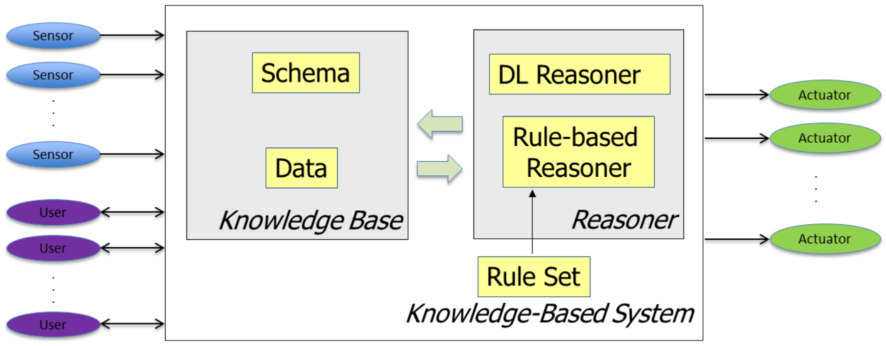


Fig. 1. Conceptual diagram of a knowledge-based system.

Figure 1 presents an overview of a knowledge-based system for pervasive environments. The system is viewed from the information flow perspective, and that perspective is the focus of this dissertation. The components of a knowledge-based system and its relations with sensors, users, and actuators in pervasive environments are shown in the figure. A reasoner with rules on top of knowledge base is essential for building knowledge-based systems. User-defined rules allow expressing richer semantic relations that lie beyond the expressive power of OWL 2 and couple ontological and rule knowledge.

Open world assumption facilitates building knowledge-based systems, because the systems do not need to model complete information about the world. This assumption is well suited for pervasive systems, where complete models are infeasible and even unavailable, and information is often incomplete due to sensor inaccuracies and partial observations.

2.2 Semantics for resource-constrained devices

Semantic technologies have been successfully applied to many domains, and pervasive computing is an important one of them. The development of pervasive applications can be facilitated by encoding the meaning of the data in the messages sent by devices, but the constrained resources of these devices challenge the common Semantic Web solutions for doing this.

Many proprietary formats have been defined for different pervasive systems, such as Generic Sensor Format [8], Unisens [41], and Unified Transportation Sensor Data Format [42]. However, sensor specific data formats introduce dependencies into the system, and even set up a barrier to utilizing components from other vendors and developers. For this reason, we do not consider proprietary formats, but only open formats supporting interoperability.

What can be achieved if we add semantics to the information produced by the large quantity of sensors deployed in pervasive environments? Berners-Lee et al. pointed out in their landmark article about the Semantic Web, that "developments will usher in significant new functionality as machines become much better able to process and understand the data" [43]. We see this significant new functionality possible when devices send data directly in a syntax that contains semantics in addition to the raw data. Since the meaning of the data is encoded in the message, the receiver of the message can utilize the data in a straightforward and general fashion. The receiver does not need device-specific knowledge, but can process data from all devices in a similar way. However, since devices in pervasive environments are often small and equipped with modest computing, communication, memory, and energy resources, these devices introduce challenges not present in the common scenarios of Semantic Web.

Here, we present Semantic Web models and representations and briefly discuss using these representations in pervasive environments. Research on Semantic Web has produced well established specifications for formal knowledge representations. These knowledge representations support logical reasoning to infer new information from existing assertions and rules. Standard syntaxes of Semantic Web models are potential candidates for representing sensor data. Among them, RDF is the most widely used data model for representing semantic data. RDF represents data as triples in the form (Subject, Predicate, Object). A triple denotes that a subject has a property whose value is the object. Data in pervasive environments usually originates from devices, humans, and other entities in the physical world. It refers to attributes of phenomena and to relations among these entities. One way of semantically representing sensor data, like a measurement made by a device, is denoting the device as the subject, the measured quantity as the predicate, and the measured value as the object. This is unambiguous, as a unique device is the subject, and the knowledge related to this device can be easily described. For example, "Sensor 1" is the subject, "Temperature" is the property, and "25" is the value. The unit of measurement, for example "Celsius", can be defined separately. The data measured by a device can be associated with entities that have relations with the device. For example, when Alice is carrying a positioning device and the device detects the current location to be "Campus", we can form a triple where "Alice" is the subject, "isLocated" is the property, and "Campus" is the value.

In this section, we utilize an example of a sensor node, which sends a time stamp value together with temperature, acceleration, and magnetic field values. Other sensor data can be represented in a similar way. The following example presents the sensor data in RDF/XML syntax:

```
<rdf:RDF xml:base="http://ee.oulu.fi/o" xmlns:e="http://ee.oulu.fi/o#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\#">
<e:TempAccMagSensor rdf:ID="tempAccMagSensor01">
  <e:timeStamp>2012-05-18T12:00:00</e:timeStamp>
  <e:accX>618</e:accX>
  <e:accY>319</e:accY>
  <e:accZ>671</e:accZ>
  <e:magX>123</e:magX>
  <e:magY>234</e:magY>
  <e:magZ>345</e:magZ>
  <e:temp>22.5</e:temp>
</e:TempAccMagSensor>
</rdf:RDF>
```

A clear advantage of RDF is that the existing higher level languages RDFS 1.1 and OWL 2 provide standard vocabularies for defining classes and relationships among classes, which enables complex inference. Hence, when resource-constrained devices express data in RDF, these languages facilitate realizing advanced semantic processing. XML is the most widely used language for describing RDF models, and it is the only normative syntax that Semantic Web tools should support. But similarly to other W3C languages, RDF/XML is designed for Web applications and is hence not a perfect solution for systems in pervasive environments.

N3 [44], Turtle [7], and N-Triples [45] are alternative syntaxes for RDF/XML. They are also based on triple structure, but they differ in expressive power. They can all serialize the RDF model in a straightforward manner and are in most cases more lightweight than RDF/XML. N3 is a flexible language with strong expressive power going beyond the RDF model. Turtle is an RDF-compatible subset of N3, while N-Triples has constrained expressive power. N3 and Turtle have shorthand syntaxes. These syntaxes shorten the descriptions, but on the other hand, can require more computing resources when the descriptions are processed. For this reason, they need to be optimized for resource-constrained devices in pervasive environments. The example data is in N3 as follows:

```
@prefix e:<http://ee.oulu.fi/o#>.
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
    e:tempAccMagSensor01
    e:accX "618";
    e:accY "319";
    e:accZ "671";
    e:magX "123";
    e:magY "234";
    e:magZ "345";
    e:temp "22.5";
    e:timeStamp "2012-05-18T12:00:00";
    a e:TempAccMagSensor.
```

RDF, N3, Turtle, and N-Triples are designed to be used by Web applications; hence, resource usage was not the main issue when these languages were designed. Media Types for Sensor Markup Language (SenML) [46], on the other hand, is a sensor data description language for representing simple sensor measurements and device parameters. It is targeted for resource-constrained devices and hence the amount of processing and the size of data were considered when it was designed. A SenML description carries a single base object consisting of attributes and an array of entries. Each entry, in turn, consists of attributes such as a unique identifier for the sensor, the time the measurement was made, and the current value.

SenML enables a straightforward transform from the data structures of programming languages into representations. SenML can be represented in JSON, XML, and Efficient XML Interchange (EXI) [47]. Among them, JSON and EXI are more lightweight and still easy to serialize. The SenML format can be extended with further custom attributes. For example, the Resource Type (rt) attribute can be used to define the meaning of a resource. Other semantic attributes can be defined in a similar way. Finally, additional information can be made available by including in a SenML description a link in the CoRE Link Format [48], but then additional communication is required to fetch that information.

Here is the sensor data example in SenML/JSON format:

```
{"e": [
        {"n": "accX", "v": 618},
        { "n": "accY", "v": 319},
        { "n": "accZ", "v": 671},
        { "n": "magX", "v": 123},
        { "n": "magY", "v": 234},
        { "n": "magZ", "v": 345},
        { "n": "temp", "v": 22.5}],
      "bn": "tempAccMagSensor01",
      "pr": "http://ee.oulu.fi/o#",
      "bt": "3296120023",
      "rt": "TempAccMagSensor"}
```

“bt” is base time and “bn” is base name, it denotes a device ID in this case. “pr” stands for prefix, which can be transformed to xml:base="http://ee.oulu.fi/o" when SenML data are transformed to RDF/XML. The sensor data example is in SenML/XML format as follows:

```
<senml xmlns="urn:ietf:params:xml:ns:senml"
       pr="http://ee.oulu.fi/o#"
       bn="tempAccMagSensor01"
       bt="3296120023"
       rt="TempAccMagSensor">
     <e n="accX" v="618" />
     <e n="accY" v="319" />
     <e n="accZ" v="671" />
     <e n="magX" v="123"/>
     <e n="magY" v="234" />
     <e n="magZ" v="345" />
     <e n=temp" v="22.5" />
</senml>
```

Here is the sensor data example in SenML/EXI:

```
   0x80419cd95b ... 145 bytes ... 0801001000
```

Another recent effort is to serialize RDF in JSON format. Several proposals, including RDF/JSON [49], JSN3 [50], JTriples [51], RDFj [52], and JSON-LD [10], have been presented. They allow an RDF graph to be written in a format compatible with JSON. To achieve this, the essential idea is to introduce: 1) universal identifiers for JSON objects via the use of Internationalized Resource Identifiers (IRIs) [53], 2) a mechanism to serialize a set of RDF triples as a series of nested data structure in JSON, and 3) a mechanism to associate data types with values. The W3C RDF working group compared these formats with examples in [54]. Based on this comparison, JSON-LD is considered as the most promising format and became a W3C recommendation in early 2014. JSON-LD is designed to be completely compatible with JSON and it has a slightly better expressive power than the RDF model. This means that in practice it can be considered to be a JSON serialization for RDF. JSON-LD requires minimal effort from developers to transform normal JSON to JSON-LD. Only two keywords (@context and @id) need to be known for utilizing the basic features. The JSON-LD format was developed contemporaneously with EN, and was published at the final stages of EN development. We compare EN and JSON-LD in Chapter 5.

The sensor data example can be represented in JSON-LD format as follows:

```
{
  "@context":
  {
    "e": "http://ee.oulu.fi/o#",
    "accX": "e:accX", "accY": "e:accY",
    "accZ": "e:accZ", "magX": "e:magX",
    "magY": "e:magY", "magZ": "e:magZ",
    "temp": "e:temp", "timeStamp": "e:timeStamp"
  },
  "@id": "e:tempAccMagSensor01",
  "@type": "e:TempAccMagSensor",
  "accX": "618", "accY": "319",
  "accZ": "671", "magX": "123",
  "magY": "234", "temp": "22.5",
  "timeStamp": "2012-05-18T12:00:00"
}
```

Contexts describe short-hand terms for JSON-LD, and can be directly embedded in data packets (as in this example) or be referenced. Devices can agree contexts at design time or send full contexts on request between nodes to decrease communication overhead. The JSON-LD packet with referred context is as follows:

```
{
  "@context": "http://ee.oulu.fi/json-ld/contexts",
  "@id": "e:tempAccMagSensor01",
  "@type": "e:TempAccMagSensor",
  "accX": "618", "accY": "319",
  "accZ": "671", "magX": "123",
  "magY": "234", "temp": "22.5",
  "timeStamp": "2012-05-18T12:00:00"
}
```

A device receiving context-referenced JSON-LD data can retrieve context information at http://ee.oulu.fi/json-ld/contexts. When considering resource consumption and compatibility with programming languages, SenML and JSON-LD are potential formats for pervasive environments.

In addition to these formats, several other representations have been suggested for semantic annotations. Semantic Sensor Web [55] enables adding semantic annotations in terms of time, location, and thematic data into the actual sensor data by using Resource Description Framework in Attributes (RDFa) [56]. SemSOS [57] is a similar solution for adding semantic annotations into sensor observations. Finally, semantic extensions are being built for the Product Markup Language (PML)[58], which is an XML-based language for describing physical objects in Electronic Product Code Networks. However, all these XML-based solutions have limitations in supporting semantic interoperability and linking resources to knowledge.

Binary formats for XML like EXI, Wireless Application Protocol (WAP) Binary XML Content Format [59], Fast Infoset [60], and Xebu [61] can be used to transfer data from embedded sensors. These formats are designed to improve parsing performance and reduce data size for the XML representation. They mainly utilize schema-based optimization and token based compression, in which each element, attribute, and so forth, is encoded as a small integer value. W3C recommends EXI, which is a compact representation for the XML Information Set and is intended to simultaneously optimize the performance and utilization of computational resources [62]. Using a relatively simple algorithm, it produces encodings of XML event streams. Its simplified mode of operation called schema informed mode allows embedded devices to work directly with the encoding without the need to work with a full XML parser. In [62], the average compression rate of EXI for 67 test sets reached about 30%, when all optimization techniques, including schema and document analysis, were utilized.

However, binary formats themselves do not support any semantics. Instead, semantic information in RDF/XML and SenML, for example, can be encoded in binary formats to decrease communication load. We have measured the amount of computation required to produce messages in the EXI binary format. These measurements are presented in Chapter 5.

In addition to the binary formats described above, several methods have been developed for compressing XML files, such as XMLPPM [63], XMLZip [64], XML Skeleton Compression [65], and XPress [66]. These methods utilize complex compression algorithms and homomorphic transformation strategies to compress XML. Wilfred Ng et al. [67] tested several XML compression techniques and reported that XMLPPM achieves the best compression ratio, which can reach 8.25% of the original XML file. A large obstacle for adopting these solutions for pervasive environments is that resource-constrained sensors cannot afford these complex algorithms. Moreover, some of these methods lose some content during the compression.

RDF Header-Dictionary-Triples (HDT) [68][69] is a binary format for RDF, especially for large RDF data sets. RDF HDT provides a method for encoding RDF documents in a compact manner and supports splitting large RDF documents into small pieces. RDF graphs are reorganized into Header (optional), Dictionary, and Triples. HDT Dictionary organizes all vocabularies and HDT Triples, which enables the compression of an RDF graph in a compact form. Unique identifiers are assigned to each element in RDF and prefixes are utilized to shorten IRIs. Hasemann et al. [70] reported an approach for sensor nodes to provide sensor data in the RDF HDT format. However, HDT is mainly designed for representing large RDF data sets, not for resource-constrained devices in pervasive environments.

2.3 Representing OWL ontologies

Before the development and standardization of OWL, there were plenty of ontology languages, such as KL-ONE [71], F-logic [72], SHOE [73], DAML-ONT [74], OIL [75], and DAML+OIL [76]. These efforts finally lead to developing OWL, which is a comprehensive ontology language for the Semantic Web. OWL was endorsed by the W3C in 2004, and has been utilized widely in Semantic Web applications and automated reasoners. OWL is primarily concerned with defining terminology that can be used in RDF documents. As an extended version of OWL, OWL 2 was published in 2012. OWL and OWL 2 are high level structural languages based on DL, and can be serialized with various syntaxes.

OWL 2 provides a rich collection of constructs for forming descriptions, and is compatible with existing Web standards. An OWL 2 ontology consists of a set of axioms which place constraints on sets of individuals and on the types of relationships between them. These axioms provide semantics by allowing systems to explicitly infer additional knowledge based on the data provided.

There are different languages for storing, sharing, and editing OWL ontologies. Among them, RDF/XML is the officially recommended exchange syntax, and others have been designed for particular purposes and applications. In this section, we illustrate different representations for OWL with an example of a small ontology. This ontology consists of simple concepts about entities in pervasive environments, including a Sensor class, a Person class, and a Location Sensor class that is a sub-class of Sensor. In addition, a person can own sensors; and hence, sensors can be owned by a person (this is an example of an inverse relation).

OWL 2 provides a bidirectional mapping from the OWL Functional Syntax to RDF Graphs [77]. This means that OWL can then be serialized into any RDF representations such as RDF/XML and N3. Most current OWL tools utilize RDF/XML as the default syntax for serializing ontologies. OWL elements of the example can be presented in RDF/XML syntax as follows:

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<owl:Class rdf:about="http://ee.oulu.fi/o#Person"/>
<owl:Class rdf:about="http://ee.oulu.fi/o#Sensor"/>
<owl:Class rdf:about="http://ee.oulu.fi/o#LocationSensor">
    <rdfs:subClassOf rdf:resource="http://ee.oulu.fi/o#Sensor"/>
</owl:Class>

<owl:ObjectProperty rdf:about="http://ee.oulu.fi/o#own">
    <rdfs:domain rdf:resource="http://ee.oulu.fi/o#Person"/>
    <rdfs:range rdf:resource="http://ee.oulu.fi/o#Sensor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="http://ee.oulu.fi/o#hasOwner">
    <rdfs:domain rdf:resource="http://ee.oulu.fi/o#Sensor"/>
    <rdfs:range rdf:resource="http://ee.oulu.fi/o#Person"/>
    <owl:inverseOf rdf:resource="http://ee.oulu.fi/o#own"/>
</owl:ObjectProperty>
</rdf:RDF>
```

RDF/XML is a widely supported syntax, as it is recommended by W3C. Because OWL supports a bidirectional mapping to RDF triples, it is convenient to be combined in RDF/XML. Other RDF based representations share this benefit, for example N3 and Turtle. However, RDF/XML is a verbose representation for OWL and can hence be difficult to read by human users.

Turtle is more readable than RDF/XML and widely supported. W3C selected Turtle as one of the syntaxes for OWL 2. The following example presents the same OWL elements in Turtle syntax, which is also valid in N3:

```
@prefix xml: <http://www.w3.org/XML/1998/namespace>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl2xml: <http://www.w3.org/2006/12/owl2-xml#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

###  http://ee.oulu.fi/o#Person
<http://ee.oulu.fi/o#Person> rdf:type owl:Class.

###  http://ee.oulu.fi/o#Sensor
<http://ee.oulu.fi/o#Sensor> rdf:type owl:Class.

###  http://ee.oulu.fi/o#LocationSensor
<http://ee.oulu.fi/o#LocationSensor> rdf:type owl:Class;
    rdfs:subClassOf <http://ee.oulu.fi/o#Sensor>.

###  http://ee.oulu.fi/o#hasOwner
<http://ee.oulu.fi/o#hasOwner> rdf:type owl:ObjectProperty;
    rdfs:domain <http://ee.oulu.fi/o#Sensor>;
    rdfs:range <http://ee.oulu.fi/o#Person>.

###  http://ee.oulu.fi/o#own
<http://ee.oulu.fi/o#own> rdf:type owl:ObjectProperty;
```

```
    rdfs:domain <http://ee.oulu.fi/o#Person>;
    rdfs:range <http://ee.oulu.fi/o#Sensor>;
    owl:inverseOf <http://ee.oulu.fi/o#hasOwner>.
```

OWL functional syntax is designed to be easier for OWL 2 specification purposes and to provide a foundation for the implementation of OWL 2 tools. Functional syntax is a simple text based syntax that is used as a bridge between the structural specification and concrete representations. Here, is the example presenting the same OWL elements in functional syntax:

```
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(owl2xml:=<http://www.w3.org/2006/12/owl2-xml#>)

Ontology(
Declaration(Class(<http://ee.oulu.fi/o#LocationSensor>))
Declaration(Class(<http://ee.oulu.fi/o#Person>))
Declaration(Class(<http://ee.oulu.fi/o#Sensor>))
Declaration(ObjectProperty(<http://ee.oulu.fi/o#hasOwner>))
Declaration(ObjectProperty(<http://ee.oulu.fi/o#own>))
SubClassOf(<http://ee.oulu.fi/o#LocationSensor>
    <http://ee.oulu.fi/o#Sensor>)
InverseObjectProperties(<http://ee.oulu.fi/o#hasOwner>
    <http://ee.oulu.fi/o#own>)
ObjectPropertyDomain(<http://ee.oulu.fi/o#hasOwner>
    <http://ee.oulu.fi/o#Sensor>)
ObjectPropertyRange(<http://ee.oulu.fi/o#hasOwner>
    <http://ee.oulu.fi/o#Person>)
ObjectPropertyDomain(<http://ee.oulu.fi/o#own>
    <http://ee.oulu.fi/o#Person>)
ObjectPropertyRange(<http://ee.oulu.fi/o#own>
    <http://ee.oulu.fi/o#Sensor>)
```

The design of RDF/XML makes it difficult to utilize off the shelf XML tools for tasks other than parsing and rendering it. Standard XML tools like XPath or XSLT do not work well with RDF/XML representations of ontologies [78]. Moreover, serializing OWL and OWL 2 requires resources, as OWL needs to be first mapped to RDF, and then RDF needs to be serialized to XML. To overcome these difficulties, OWL/XML was invented as a concrete syntax for a more regular and simpler XML syntax. The syntax is essentially derived directly from the Functional Syntax. The example below shows the example in OWL/XML:

```
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.w3.org/2002/07/owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace">

<Prefix name="owl" URI="http://www.w3.org/2002/07/owl#"/>
<Prefix name="rdf" URI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
<Prefix name="rdfs" URI="http://www.w3.org/2000/01/rdf-schema#"/>
<Prefix name="owl2xml" URI="http://www.w3.org/2006/12/owl2-xml#"/>
<Prefix name="xsd" URI="http://www.w3.org/2001/XMLSchema#"/>

<Declaration>
    <Class URI="http://ee.oulu.fi/o#LocationSensor"/>
</Declaration>
<Declaration>
    <Class URI="http://ee.oulu.fi/o#Person"/>
</Declaration>
<Declaration>
    <Class URI="http://ee.oulu.fi/o#Sensor"/>
</Declaration>
<Declaration>
    <ObjectProperty URI="http://ee.oulu.fi/o#hasOwner"/>
</Declaration>
<Declaration>
    <ObjectProperty URI="http://ee.oulu.fi/o#own"/>
</Declaration>
<SubClassOf>
    <Class URI="http://ee.oulu.fi/o#LocationSensor"/>
    <Class URI="http://ee.oulu.fi/o#Sensor"/>
</SubClassOf>
<InverseObjectProperties>
```

```
    <ObjectProperty URI="http://ee.oulu.fi/o#hasOwner"/>
    <ObjectProperty URI="http://ee.oulu.fi/o#own"/>
</InverseObjectProperties>
<ObjectPropertyDomain>
    <ObjectProperty URI="http://ee.oulu.fi/o#hasOwner"/>
    <Class URI="http://ee.oulu.fi/o#Sensor"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty URI="http://ee.oulu.fi/o#own"/>
    <Class URI="http://ee.oulu.fi/o#Person"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty URI="http://ee.oulu.fi/o#hasOwner"/>
    <Class URI="http://ee.oulu.fi/o#Person"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty URI="http://ee.oulu.fi/o#own"/>
    <Class URI="http://ee.oulu.fi/o#Sensor"/>
</ObjectPropertyRange>
</Ontology>
```

However, OWL/XML suffers from verbose syntax and this often slows down parsing. Manchester syntax [79] is designed for editing and presentation purposes. It provides for OWL ontologies a compact text based representation that is easy to read and write. The primary motivation for the design of the Manchester OWL syntax was to produce a syntax that could be used for editing class expressions. This effort has been extended so that it is possible to represent complete ontologies, and Manchester Syntax is now standardized by W3C. The example in Manchester Syntax is as follows:

```
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: owl2xml: <http://www.w3.org/2006/12/owl2-xml#>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>

Ontology:
Class: <http://ee.oulu.fi/o#Sensor>
Class: <http://ee.oulu.fi/o#Person>
Class: <http://ee.oulu.fi/o#LocationSensor>
```

```
    SubClassOf: <http://ee.oulu.fi/o#Sensor>

ObjectProperty: <http://ee.oulu.fi/o#own>
    Domain: <http://ee.oulu.fi/o#Person>
    Range: <http://ee.oulu.fi/o#Sensor>
    InverseOf: <http://ee.oulu.fi/o#hasOwner>

ObjectProperty: <http://ee.oulu.fi/o#hasOwner>
    Domain: <http://ee.oulu.fi/o#Sensor>
    Range: <http://ee.oulu.fi/o#Person>
    InverseOf: <http://ee.oulu.fi/o#own>
```

However, Manchester Syntax is cumbersome for representing some axioms in OWL, such as general class axioms which have non-atomic class expressions for their left hand side. Attempto Controlled English [80] is a machine-oriented Controlled Natural Language (CNL), that is, a precisely defined subset of the English language, designed for writing unambiguous and precise specification texts for knowledge representation. Attempto Controlled English supports a bidirectional translation into and from OWL 2. Other representations with similar design goals are Sydney OWL syntax [81] and Ordnance Survey's Rabbit [82]. However, we are not aware of its compatibility with OWL 2. Schwitter et al. compared these three representations with examples [83]. Here is the same example in Attempto Controlled English:

```
Every LocationSensor is a Sensor.
Everything that hasOwner something is a Sensor.
Everybody that own something is a Person.
Everybody that is hasOwner by something is a Person.
Everything that is own by somebody is a Sensor.
If X hasOwner Y then Y own X.
If X own Y then Y hasOwner X.
```

However, these representations have complex syntaxes and do not support incremental knowledge transfer. Hence, we do not consider these representations suitable for transferring knowledge in pervasive environments.

2.4 Related work about semantic technologies for pervasive computing

In this section, we present recent work about semantic technologies being implemented in pervasive systems. We introduce some use cases of utilizing semantic data and knowledge representations on mobile devices, reasoning, triple stores and ontologies, and systems that include a large number of devices utilizing semantics.

As of today, numerous practical tools are available for addressing Semantic Web representations in pervasive environments. Le-Phuoc et al. [84] introduced RDF On the Go, which offers an RDF storage and a query processor for mobile devices. About reasoning tools for mobile devices, Crivellaro et al. [85] developed μJena, which is one of the first tools serving to manage ontologies and RDF stored in mobile devices. LOnt [86] is another effort of implementing Jena API for mobile devices. Gu et al. [87] proposed a mobile framework for ontology processing and reasoning. The reasoner contains a forward chaining rule-based inference engine, but it only supports a subset of OWL ontology inference rules. Similarly, μOR [88] reasoner and “MiniOWL and MiniRule” [89] reasoner also reasons over a subset of OWL entailments. Bossam [90] is a Rete-based forward chaining inference engine designed for dynamic and conflicting knowledge space. It promotes the effectiveness of ordinary rule engine’s OWL reasoning capability. Finally, AndroJena [91] is another important effort to enable Android based mobile devices with semantic reasoning capabilities.

RDF stores - also called triple stores - are capable to store, manage, and process large amounts of data in RDF models. The current RDF stores supporting these features include Virtuoso, Apache Jena - TDB, Sesame, Oracle Spatial and Graph, BigData, and AllegroGraph. These databases are able to store semantic data in RDF. Some of these have support for querying and reasoning from stored RDF graphs and are capable to combine inserted RDF data with stored background knowledge. Moreover, NoSQL databases [92] have been experimented for RDF data management as well. However, these databases are not specialized for RDF; thus, querying of RDF graphs, RDF schemas, and expressive ontologies are not directly supported. RDF database storage schemes are based on RDF triples and graphs. Special indexing and query techniques are needed to manage RDF graphs, which makes traditional data management systems perform poorly with graph-based data.

Current RDF database solutions are developed mainly for querying static data on the Web. However, pervasive systems consist of large amounts of devices require good performance for RDF stores as the large amount of data needs to be stored, shared and queried. Frequent updating operations on the RDF graph, which is common for data generated in pervasive environments, may cause poor performance [93].

In open pervasive environments, it is impossible to define a "complete" model without *priori* knowledge of the applications that will utilize it. With the open world assumption, Semantic Web technologies are well suited for modelling pervasive environments. The CONtext ONtology (CONON) [94] is one of the earliest context models that encompasses a common upper ontology for the general concepts in pervasive computing, as well as domain-specific ontologies that apply to different subdomains, such as smart homes. In [95], Lassila and Khushraj consider representing context using DL, where OWL DL is used as a concrete example. Chen et al. [12] utilize OWL to represent context information. Prolog [96] and Java Expert System Shell (Jess) [97] are utilized to realize rule-based reasoning. Wang et al. [98] also modelled context by OWL in semantic space project. They designed a two-layer context model for expressing context information, and reasoning was implemented using Jena. Korpipää et al. [99] also presented a lightweight ontology for context-aware mobile devices.

Moreover, ontologies and reasoning have been widely utilized in mobile recommendation systems. Buriano et al. [100] discuss the role of ontology in a context-aware recommender system, including specifying the input representation, allowing defining intra-context dependencies, sharing information, and reasoning about it. Naudet et al. [101] suggest a set of ontologies and a rule-based matchmaking approach to develop context-aware recommender systems to deliver multimedia content. Ciaramella et al. [102] propose a situation-aware service recommender system, which helps to locate appropriate services proactively. It includes a semantic layer which infers current situations of the user by exploiting domain ontology and semantic rules. A fuzzy inference layer handles the vagueness of some conditions of these rules and outputs an uncertainty degree for each situation determined by the semantic layer. Kritsotakis et al. [103] present a personalized context-aware indoor navigation system enhanced with Semantic Web technologies.

IoT interconnects a vast quantity of devices to the Internet. A large amount of data is provided by diverse heterogeneous sources and the context of data may change frequently. Semantics help to tackle the challenges related to software, services, and algorithms, that is, "to support interoperable machine to machine and 'thing' to 'thing' interaction over a network" [104]. Generally, semantics improves interoperability at the application layer, as devices share the meaning of the communicated data. By adding support to meta-data, semantics provides tools for tackling the challenges related to discovery and search engine technologies as well.

At its best, research and development on IoT can produce a dynamic and universal network where billions of identifiable things communicate with each other whenever and wherever communication is needed. Things become context-aware, configure themselves, exchange information, and show intelligent behavior when exposed to new environments and unforeseen circumstances. Intelligent decision-making algorithms enable rapid responses and revolutionize the ways business value is generated [104]. To make sense of this vast amount of heterogeneous dynamic data and to utilize it meaningfully in real applications, data must be enriched in the way that it can be easily and efficiently shared and interpreted by machines. Semantic technologies provide promising solutions for achieving this.

Wei et al. [105] studied semantic annotations and rule-based reasoning for sensor data by utilizing current Semantic Web concepts and they noted that rule-based reasoning with ontologies is the most promising approach for deriving knowledge from sensor data. They considered reasoning with sensor data to low-level services where raw sensor data is aggregated and preprocessed from various sources. High-level services take derived facts provided by low-level services as input and infer new knowledge from high-level ontologies for context-aware applications. Bikakis et al. [106] studied different reasoning techniques and pointed out several benefits of rule-based reasoning, including simplicity, flexibility, formalism, expressive power, modularity, high-level abstraction, and integration with ontologies. They suggested combining ontologies and rule-based reasoning techniques for contextual reasoning. Contextual reasoning studies reasoning with distributed contexts in dynamic environments [107] and could provide an approach to perform reasoning with variable contexts in IoT environments.

Barnaghi et al. survey approaches of utilizing of semantic technologies in IoT [108], and point out semantic technologies to be important for facilitating data integration and interoperability for IoT. For example, semantic technologies are used in the home automation prototype system for monitoring and controlling heating and air conditioning [109]. A smart farming system is proposed in [110], where Global Sensor Networks (GSN) middleware and SSN ontologies are utilized to automate monitoring and controlling of farming activities. Zhou et al. [111] propose semantic modeling for facilitating demand response optimizations in smart grids with automated real-time load prediction and curtailment. Hristoskova et al. [112] propose ontology-based framework for providing personalized medication for patients, and automated emergency alerting and advanced decisions support system for physicians. Preist et al. [113] demonstrates micro architecture for automated logistics supply chains based on Semantic Web service descriptions.

2.5 Summary

In this chapter, we presented the state of the art of data and knowledge representations. We presented standardized Semantic Web languages and representations designed for resource-constrained devices and pervasive applications with sensor data and a small ontology examples. Moreover, we presented recent work about semantic technologies being implemented in pervasive systems. In Chapter 3 and Chapter 4, we introduce our representations that fulfils our design goals, and in Chapter 5, we compare these representations with most representations we discussed in this chapter with the criteria of expressive power and resource consumption.

3 Entity Notation

This chapter represents, together with the next chapter, the main contribution of this dissertation, Entity Notation (EN). EN is a lightweight data and knowledge representation that can be utilized by resource-constrained devices and environments. Our main goal is to connect networked devices to knowledge-based systems and SemanticWeb. This chapter presents EN for RDF data, and the next chapter presents EN Schema for transferring ontologies.

3.1 Design considerations

One main challenge in connecting networked devices to knowledge-based systems in pervasive environments is that sensors produce raw data in a variety of formats. Hence, each format needs its own solution when sensors are connected to knowledge-based systems. If the networked devices all utilize the same data representation and that representation is compatible with Semantic Web knowledge models, this connection is much easier to realize and tools, such as inference mechanisms, knowledge bases, and semantic query techniques would be available for applications consuming the data. Moreover, such a data representation would allow application developers to easily utilize devices implemented and deployed by others. We tackle this challenge in this chapter.

As RDF and ontology models are based on triple representations, we design a uniform syntax, EN, based on the triple structure for expressing RDF models and OWL 2 ontologies. Moreover, we suggest a compact format for transmission when resources are constrained. The compact format shortens the representation with templates and prefixes.

To meet the objectives formulated in the introduction, we specify the following requirements for EN:

– First, it must be suitable for networked devices with modest computing and communication resources. For example, an unsophisticated embedded sensor should be able to compose EN packets using minimal computation power and deliver packets to knowledge-based systems in an energy efficient manner.
– Second, EN must be expressive enough for representing basic RDF and OWL 2 DL ontology models in a uniform syntax. It should be possible to transform any RDF and

OWL 2 DL description into EN packets, and vice versa. Transforming RDF/XML into EN packets is essential as XML is one of the most widely used serializations for RDF and OWL utilizes XML syntax, as well. Though a number of other serializations exist and have been standardized by W3C recently, many available reasoning tools utilize RDF/XML, such as Jena [40] and Closed World Machine (CWM) [114].

– Third, EN must support distributed knowledge production and incremental knowledge integration. Our goal is that devices and Semantic Web applications can produce small pieces of knowledge, one piece at a time, and these pieces can be discovered, transferred and integrated into larger ontologies. Incremental knowledge integration will be an important feature for the knowledge bases of pervasive systems.
– Fourth, EN must be able to carry information between devices and Semantic Web applications. EN is required to be general and precise in expressing knowledge about any topic, but we target EN for pervasive environments.
– Fifth, the information carried in EN packets needs to be identified in a unique fashion. This is an important feature for an Internet scale system when all the possible usages of the information cannot be specified in advance.

Figure 2 presents the role of EN among Semantic Web technologies. The standard Semantic Web stack is adapted from [115]. Currently, W3C offers standardized solutions for all layers under the Rules layer. RDF and RDF Schema are used as a description language for resources, then there is an ontology layer on top on them. Ontologies describe relationships between types of resources, but they do not indicate how to compute such relationships. As shown in Figure 2, EN and EN Schema are alternative syntaxes for XML and XML Schema. All entities in EN and EN Schema can be transformed into corresponding resources in RDF and ontologies.

RDF provides a good semantic basis for Entity Notation. The RDF model is based on RDF statements with triple (*Subject*, *Predicate*, *Object*) structure. These statements describe resources. The subject and object identify the two resources being described; the predicate identifies a property or characteristic of the subject. These resources may be IRIs, blank nodes, or literals. Besides, in RDF, information is identified in a unique fashion by IRIs.

To fulfil the requirements, we define two EN formats: the complete format and the short format. The complete format has sufficient expressive power, as it follows the triple notation of RDF. Moreover, we adopt almost all terms from RDF Schema and OWL 2 in the complete format. Due to the triple notation, it will be straightforward

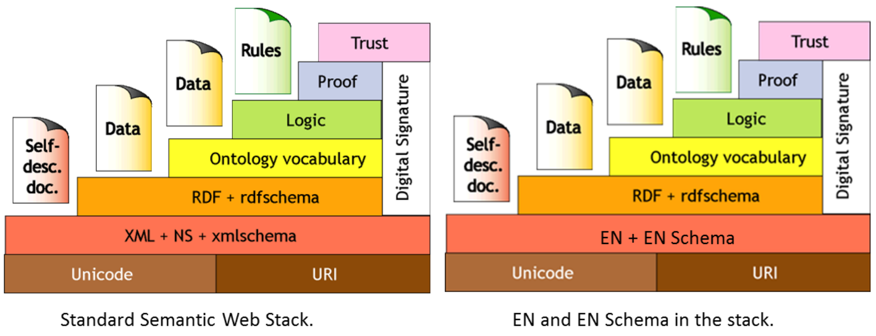


Fig. 2. EN's role in Standard Semantic Web Stack.

to serialize RDF and ontologies into complete EN packets and vice versa. On the other hand, lightweight short packets can support resource-constrained devices and communication links. Support for incremental knowledge definition and communication will be provided as well. We define for RDF and OWL a uniform EN and EN Schema syntax that does not constrain the type of information. Finally, EN uses Universally Unique Identifiers (UUIDs) [116] and IRIs for unique identification.

We design EN to represent RDF in pervasive environments; this means RDF model can be serialized into a set of EN packets that can then be transferred. Then, we describe EN Schema for transferring other elements in ontologies. Design details of EN Schema will be presented in Chapter 4. Generally, EN and EN Schema are based on describing entities and their relationships with values and other entities. An entity is some identifiable whole, concrete or abstract. Concrete entities can be, for example, sensors, objects, and measurements. Abstract entities can be OWL elements, such as concepts and relations between them. This definition of entity is different from that of entity introduced in [19], because we include the option of PropertyType, which can be associated with PropertyValue. This definition will be utilized in the rest of this dissertation.

In an entity description, we identify one entity and then build a number of triples about it. An entity description is of the form:

```
EntityType EntityId
   PropertyName PropertyValue
   ...
   PropertyName PropertyValue
```

EN is designed for transferring data and knowledge. Hence, the main EN syntax has a format that presents entity descriptions as a set of EN packets. EN packet is defined as a sequence of Universal Coded Character Set (UCS) characters containing one entity description.

In general, EntityType and EntityId determine the type and the identifier of an RDF subject or OWL element. They are followed by a number of property pairs (PropertyName, PropertyValue), which specify relationships or characteristics of this entity. PropertyValues can be other EntityIds or literals. Multiple pairs can be defined for the same EntityId simply by listing the (PropertyName, PropertyValue) pairs of these triples one after another. We include type information for EntityId, because it is important to identify the type of EntityId, when EN packets are transferred to RDF statements and EN Schema to ontologies. Similar to other RDF syntaxes, EN can also include optional type information for literal property values. PropertyType follows PropertyValue and is separated with two caret characters, like this:

```
PropertyValue^^PropertyType
```

Type of PropertyValues are XML Schema data types [117]. However, when PropertyValue is not a literal, PropertyType cannot be specified with this approach. When PropertyValue is an entity, its type information can be given with a seperate entity description:

```
    EntityType EntityId
```

EN utilizes plain literals and typed literals. When PropertyValue is a string (i.e. literal), a language tag can be attached to the string. The language tag has to be a well-formed string according to [118]. However, in this thesis, most experiments are developed for pervasive applications, where EN has its main role in M2M communications. Hence, we are not utilizing language tags in the examples.

Table 1 presents the mapping from EN to RDF. When an EN packet is mapped to RDF statements, EntityId maps to the subject of a triple, PropertyNames map to predicates, PropertyValues map to objects, and EntityType maps to the type of the subject. PropertyType maps to the type of an object. After transforming into RDF statements, the identified entities are instances of classes which are defined in the schema.

Table 1. Mapping of EN to RDF.

EN	RDF
EntityType	Type of a subject (rdf:type)
EntityId	Subject of a triple
PropertyName	Predicate of a triple (but cannot be rdf:type)
PropertyValue	Object of a triple
PropertyType (optional)	Type of an object (rdf:type)
Language tag of string literal (optional)	Language Tag of String Literal

Square brackets and angle brackets are utilized to identify the level of knowledge an EN and EN Schema packet should be mapped to. When an entity description is wrapped in square brackets ([and]), the EN packet should be transferred to RDF data. When an entity description is wrapped in angle brackets (< and >), it is an EN Schema packet and should be transferred to ontologies. Designing different brackets for EN and EN Schema benefits a pervasive system with processing efficiency. When a device is receiving both EN data and EN Schema simultaneously, they can be very easily distinguished and delivered to different parsers. For example, knowledge-based systems for IoT applications may process large and highly dynamic EN data with more stable and small EN Schema. Hence, knowledge-based systems could build TBox knowledge with EN schema as quickly as possible for processing a large amount of EN data.

The complete EN format resembles the triple structure of RDF and OWL 2. To shorten packets, we utilize a template that contains descriptions of the constant part of a set of complete EN packets and placeholders for the variable items. Such a template is utilized to build short EN packets that need to contain only a template identifier and the variable items.

3.2 Formal description

EN is a text/plain Multipurpose Internet Mail Extensions (MIME) type format, and encoded in UCS characters. The Extended Backus-Naur Form (EBNF) [119] definition of EN packet and EN Schema packet has the following structure:

```
EN-packet           = complete-EN | short-EN;
EN-Schema-packet    = complete-EN-Schema | short-EN-Schema;
complete-EN         = "[", entity-type, entity-id, {property-pair}, "]";
complete-EN-Schema  = "<", entity-type, entity-id, {property-pair}, ">";
```

```
short-EN             = "[", template-id, {short-entity-id? |
                         {short-propertyvalue}?}, "]";
short-EN-Schema      = "<", template-id, {short-entity-id? |
                         {short-propertyvalue}?},">";
property-pair        = propertyname, propertyvalue;
propertyvalue        = entity-id | literal;
short-propertyvalue = short-entity-id | literal;
literal              = typed-literal | plain-literal;
typed-literal        = literal-string,language-tag?"^^"literal-type;
plain-literal        = literal-string,language-tag?
entity-type          = (* Full IRI determining an entity's type *);
entity-id            = (* Full IRI identifying a single entity *) |
                         blank-node;
template-id          = (* UUID identifying a template *);
propertyname         = (* Full IRI identifying a property *) | blank-node;
propertytype         = (* Full IRI determining a propertyvalue's type *);
short-entity-id      = (* Short IRI identifying an entity *) | blank-node;
literal-string       = (* String for representing a fixed value,
                        enclosed in double quotes *);
language-tag         = @(* String for determining the language of a string
                         *)
literal-type         = (* Full IRI identifying a XML Schema data type *)
blank-node           = (* Identifier for a blank node *)
```

This EBNF definition is utilized as a formal meta syntax notation for EN. This notation can be utilized as a reference, when developing tools for checking the correctness of EN packets. In this EBNF definition, a full IRI is a character string used to identify a name of a Web resource. Full IRIs can be shortened with prefixes, this produces short IRIs. The IRIs of complete packets are full, as this minimizes the processing of EN packets when transforming into RDF and OWL 2 knowledge. Short IRIs in short packets do not cause extra processing for resource-constrained devices, as these devices do not construct full IRIs.

UUID is an identifier standard [116], which enables identifying information without central coordination. Each UUID is unique over the whole system, which is a critical feature when a general solution is required. Utilization of UUIDs guarantees unique template identifiers. Moreover, templates can be created by any device of the system, in a distributed fashion, and without any central registry.

The syntax of EN and EN Schema utilizes only a small number of special characters, thus a device needs to process a small set of characters when composing and decomposing EN and EN Schema packets. Special characters in the EN and EN Schema syntax include backslashes, square brackets, angle brackets, white spaces, double quotes, carets, and commercial at. Square brackets are used to determine the start and end of data packets, while angle brackets are used to determine the start and end of EN packets expressing higher level ontology knowledge. White spaces separate tokens. Double quotes determine the start and end of values (that can contain white spaces). This is because some data types such as base64Binary data type can contain white spaces but not double quotes. Caret chararacter is required when including a typed literal and a commercial at is required when including a language tag. An escape character '\' is needed when special characters occur in the packet content. The escape sequences are listed in Table 2. However, when special characters occur in literals, double quote is the only escape character which requires an escape character '\'.

Table 2. Escape Characters in EN.

Escape Sequence	Meaning
\\	Backslash (\)
\"	Double Quote (")
\[and \]	Square brackets ([and])
\< and \>	Angle brackets (< and >)
\␣	White Space(␣)
\^	Caret (^)
\@	Commercial at(@)

IRIs are used to identify information in the EN packets; in this dissertation, the namespace "http://ee.oulu.fi/o#" is used for these IRIs. Entity descriptions specify claims about the world. EN follows the open world assumption, which means that the truth-value of a statement may be true irrespective of whether or not it is known by any single observer or agent to be true [121]. One entity description makes claims about an entity. Semantics of a set of entity descriptions are arranged as an interpretation which makes the statement true.

3.3 Complete packet format

The format for complete EN packets follows closely that of RDF statements. A basic RDF statement contains a *(subject, predicate, object)* triple. A complete EN packet contains knowledge that can be mapped to a set of RDF statements:

```
[subject predicate object]
[subject predicate object]
 ...
[subject predicate object]
```

In addition to information about subjects, predicates, and objects, an EN packet mandatorily contains type information of subjects, and may contain type information of objects as well. This is a prerequisite for implementing ontology-based inference. The predicates, types, and objects of the subject can also be mapped in a straightforward manner. A complete EN packet that can be transformed into RDF statements has the form:

```
[EntityType EntityId
 PropertyName PropertyValue
 ...
 PropertyName PropertyValue]
```

EntityType and EntityId determine the type and the identifier of the subject. Together, they define the primary entity an EN packet is about. EntityId is a unique identifier for an entity of the type described by EntityType. These items are followed by a number of property pairs (PropertyName, PropertyValue). PropertyName specifies a predicate and PropertyValue specifies an object, which can be an entity or a literal. When PropertyType is included, a complete EN packet has the form:

```
[EntityType EntityId
 PropertyName PropertyValue^^PropertyType
 ...
 PropertyName PropertyValue^^PropertyType]
```

PropertyType can be included in any property pair of an EN packet. EntityType, EntityId, PropertyName, and PropertyType are identified in a unique fashion by IRIs. XML Schema data types are supported for literal types. Literals (including literal IRIs) are enclosed in double quotes.

Some objects can be other entities. In this case, the value of a property is an identifier of the other entity (i.e. EntityId). PropertyType can not be linked to such property values. Here is an example of three EN packets describing a line segment between a mobile sensor and a fixed point:

```
[http://ee.oulu.fi/o#Linesegment http://ee.oulu.fi/o#linesegment25
 http://ee.oulu.fi/o#startsFrom http://ee.oulu.fi/o#locaSensor767
 http://ee.oulu.fi/o#endsWith http://ee.oulu.fi/o#position798
 http://ee.oulu.fi/o#distance "1100"]
[http://ee.oulu.fi/o#LocaSensor http://ee.oulu.fi/o#locaSensor767
 http://ee.oulu.fi/o#owner "JohnSmith"
 http://ee.oulu.fi/o#longitude "25.47"
 http://ee.oulu.fi/o#latitude "65.06"]
[http://ee.oulu.fi/o#Position http://ee.oulu.fi/o#position798
 http://ee.oulu.fi/o#annotation "Good Place for Collecting berries"
 http://ee.oulu.fi/o#longitude "25.4545"
 http://ee.oulu.fi/o#latitude "65.062"]
```

When PropertyType information is included, these EN packets are:

```
[http://ee.oulu.fi/o#Linesegment http://ee.oulu.fi/o#linesegment25
 http://ee.oulu.fi/o#startsFrom http://ee.oulu.fi/o#locaSensor767
 http://ee.oulu.fi/o#endsWith http://ee.oulu.fi/o#position798
 http://ee.oulu.fi/o#distance
 "1100"^^http://www.w3.org/2001/XMLSchema#float]
[http://ee.oulu.fi/o#LocaSensor http://ee.oulu.fi/o#locaSensor767
 http://ee.oulu.fi/o#owner
 "JohnSmith"^^http://www.w3.org/2001/XMLSchema#String
 http://ee.oulu.fi/o#longitude
 "25.47"^^http://www.w3.org/2001/XMLSchema#float
 http://ee.oulu.fi/o#latitude
  "65.06"^^http://www.w3.org/2001/XMLSchema#float]
[http://ee.oulu.fi/o#Position http://ee.oulu.fi/o#position798
 http://ee.oulu.fi/o#annotation
 "Good Place for Collecting berries"
 ^^http://www.w3.org/2001/XMLSchema#String
 http://ee.oulu.fi/o#longitude
  "25.4545"^^http://www.w3.org/2001/XMLSchema#float
 http://ee.oulu.fi/o#latitude
 "65.062"^^http://www.w3.org/2001/XMLSchema#float]
```

One location is measured by a location sensor, while another is a prefixed geospatial point marked as a good place for collecting berries. We could implement some simple rule-based inference based on these packets. For example, the system could calculate the distance between these two points, and recommend a user to collect some berries, when he/she is nearby.

A complete packet can be transferred to a Turtle description in a straightforward fashion because of the natural relationship between EN and RDF. The entity type can be represented as a Class and the entity identifier is the identifier of the resource. Property names and values can be mapped directly. PropertyType is either represented as a Class or a XML Schema data type. To maintain a straightforward mapping between the EN and syntaxes of RDF and to keep syntaxes of RDF simple to process, we do not use nesting. However, when EN is transferred to RDF/XML, we do use the shorthand for resource classes, that is, we replace rdf:description with the resource type and hence do not need to use rdf:type. We also use identifiers, i.e. the rdf:ID attribute instead of rdf:about for making shorter packets.

The three EN packets (without PropertyType) listed above can be transformed into the following Turtle document; the corresponding RDF graph is shown in Figure 3:

```
@prefix e:<http://ee.oulu.fi/o#>.
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.
@prefix xml:<http://www.w3.org/XML/1998/namespace>.
@prefix xsd:<http://www.w3.org/2001/XMLSchema#>.

e:linesegment25 a e:Linesegment;
    e:distance "1100";
    e:endsWith e:position798;
    e:startsFrom e:locaSensor767.
e:locaSensor767 a e:LocaSensor;
    e:latitude "65.06 ";
    e:longitude "25.47";
    e:owner "JohnSmith".
e:position798 a e:Position;
    e:annotation "Good Place for Collecting berries";
    e:latitude "65.062";
    e:longitude "25.4545".
```

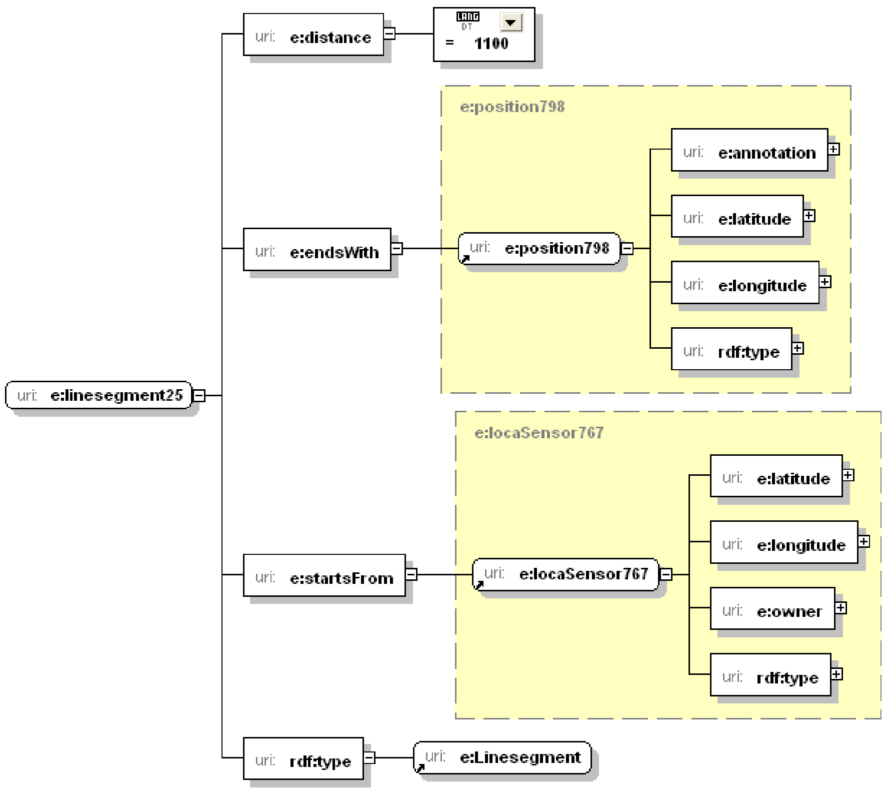


Fig. 3. RDF graph of the line segment example.

The second example presents data from a temperature, acceleration, and magnetic field sensor. This is the same data that is presented in Chapter 2 in EN syntax:

```
[http://ee.oulu.fi/o#TempAccMagSensor
http://ee.oulu.fi/o#tempAccMagSensor01
http://ee.oulu.fi/o#timeStamp "2012-05-18T12:00:00"
http://ee.oulu.fi/o#accX "618"
http://ee.oulu.fi/o#accY "319"
http://ee.oulu.fi/o#accZ "671"
http://ee.oulu.fi/o#magX "123"
http://ee.oulu.fi/o#magY "234"
http://ee.oulu.fi/o#magZ "345"
http://ee.oulu.fi/o#temp "22.5"]
```

The recommended naming convention of the EN follows that of ontologies: entity types (mapping to class names in ontologies) start with an initial capital letter, property names (mapping to property names in ontologies) start with initial lowercase letters. When a property value is an entity, the property name contains a verb and a noun; when the value is a literal, the property name contains only a noun. Identifiers (mapping to individuals in ontologies) are otherwise identical to entity type names but they start with a lowercase letter and end in a number. One exception for naming identifiers is that blank node identifiers start with an underscore and a colon, and end with a number. IRI references are used, i.e. fragment identifiers are used heavily. As usual, xml:base is used in creating full IRIs from the identifiers.

3.4 Short EN format

Complete EN packets can be long because of meaningful type names and IRIs. We suggest a technique based on templates and prefixes to shorten the packets. The basic idea is that a template contains the description of the constant part of a sequence of EN packets and placeholders for the variable items. The short packet sent over the communication links needs to contain only a template identifier and the variable items. It should be noted that short EN packets can be transformed to complete packets without semantic loss and vice versa.

A set of complete EN packets can then be assembled by replacing the template's placeholders with the values contained in the short packet. Prefixes are used to shorten IRI references. However, we do not utilize recursive or iterative data structures.

Which items in the complete packets should be treated as constant parts and which items should be treated as variable items? It is clear that EntityType and PropertyName should be constant items. Then, EntityId and PropertyValue could be variable parts. However, they can also be constants. EntityId can be constant when, for example, a sensor always sends data about the same entity. A short packet without EntityIds has the following format:

```
[UUID PropertyValue PropertyValue ... PropertyValue]
```

When a short packet format includes EntityIds, the format is:

```
[UUID
EntityId PropertyValue PropertyValue ... PropertyValue
    ......
EntityId PropertyValue PropertyValue ... PropertyValue
]
```

UUID in the short packet is used to identify the template. In its canonical form, a UUID is 128 bits long, consisting of 32 hexadecimal digits. For human-readable display, digit groups are separated by hyphens. In URN format, a UUID looks like this:

```
urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

There are alternative lengths for UUIDs, such as 64 bits, 32 bits and, 16 bits. We can choose UUIDs of different length based on the size of the identifier space a pervasive system requires: short UUIDs guarantee unambiguous identification for a small scale system; while 128 bits UUIDs need to be used for an Internet-scale system. Table 3 gives examples for short UUIDs. The hexadecimal numeral length of UUIDs is utilized for parsing and serializing packets. When the packets are delivered, the UUIDs are serialized as a part of a string.

Table 3. Examples of short UUIDs.

UUID length [bit]	Example
16	urn:uuid:59a7
24	urn:uuid:35e6af
32	urn:uuid:e81d4fae
64	urn:uuid:cea700a0c91e6bf6

When a small scale sensing system is integrated into a larger one, alternative 128 bit UUIDs can be generated for the templates and the conversion between short and long UUIDs can be performed using look-up tables. This look-up table can be built on the gateway forwarding data from the small scale system to the larger one, for example. We utilize 24 bit UUIDs later in this chapter and will utilize both 24 bit UUIDs and full 128 bit UUIDs in the prototypes presented in Chapter 5.

Figure 4 shows an example of using different templates to transfer the same location measurement (introduced in the earlier example). This example presents how sensors can have different templates for different application scenarios and corresponding short packets can have different variable items. In the figure, Sensor 1 sends a short packet, including UUID, EntityId and all three PropertyValues to the knowledge-based system. This packet is the longest short packet among the alternatives, as all possible variable

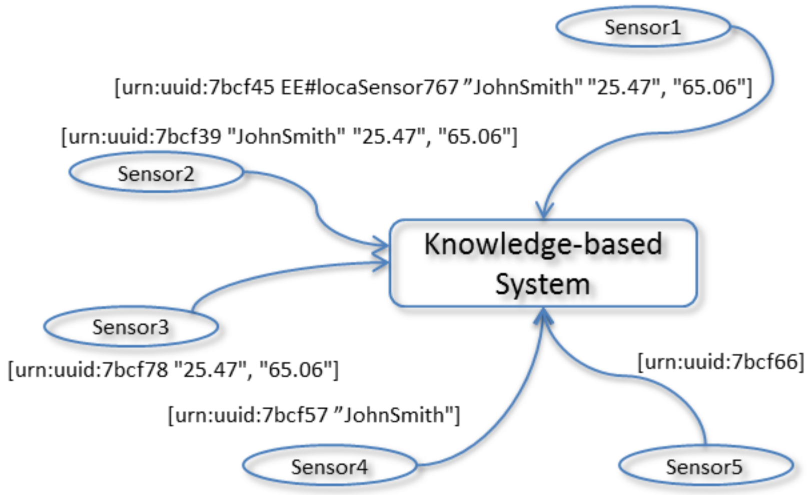

Sensor1
[urn:uuid:7bcf45 EE#locaSensor767 "JohnSmith" "25.47", "65.06"]
[urn:uuid:7bcf39 "JohnSmith" "25.47", "65.06"]
Sensor2
Knowledge-based System
Sensor3
[urn:uuid:7bcf78 "25.47", "65.06"]
[urn:uuid:7bcf66]
[urn:uuid:7bcf57 "JohnSmith"]
Sensor4
Sensor5

Fig. 4. Different short EN formats transferred from sensors to a knowledge-based system.

items are included in this short packet. But in this case, only one template is needed for all location sensors producing this kind of measurements. This short packet format might be used when several sensors produce location data and the amount of memory needed for templates is to be minimized. This template is shown in the first row of Table 4.

Sensor 2 sends a short packet, including UUID and all three PropertyValues. The corresponding template contains the EntityId. As shown in the second row of Table 4, three placeholders in this template reserve places for the values of owner, longitude, and latitude, respectively. Compared with the previous one, this template can only be utilized by a specific location sensor. Then, the number of templates is determined by that of location sensors.

Sensor 3 in Figure 4 sends a shorter packet without an owner item. This template might be used when the sensor is owned by the same person for a long time. Sensor 4 sends a short packet containing only the owner item. This might be the case when the user does not move for a long time (for example, the user is sleeping).

Sensor 5 sends a short packet only including a UUID. Clearly, this is the shortest packet among all alternatives. The whole packet is only seventeen bytes long, and all the actual information is encoded in the template. This might be used in the case that all variables are stable for a long time. The shortest possible packet can be achieved when a 16 bit UUID is utilized and a packet only includes the UUID, then, the packet length is seventeen bytes. The corresponding template is in the last row of Table 4.

Table 4. Templates for the sensors in Figure 3.

Template for Sensor 1:

urn:uuid:7bcf45

Prefix: EE:http://ee.oulu.fi/o

[http://ee.oulu.fi/o#LocaSensor ?1 http://ee.oulu.fi/o#owner ?2 http://ee.oulu.fi/o#longitude ?3 http://ee.oulu.fi/o#latitude ?4]

Template for Sensor 2:

urn:uuid:7bcf39

Prefix: EE:http://ee.oulu.fi/o

[http://ee.oulu.fi/o#LocaSensor http://ee.oulu.fi/o#locaSensor767 http://ee.oulu.fi/o#owner ?1 http://ee.oulu.fi/o#longitude ?2 http://ee.oulu.fi/o#latitude ?3]

Template for Sensor 3:

urn:uuid:7bcf78

Prefix: EE:http://ee.oulu.fi/o

[http://ee.oulu.fi/o#LocaSensor http://ee.oulu.fi/o#locaSensor767 http://ee.oulu.fi/o#owner "JohnSmith" http://ee.oulu.fi/o#longitude ?1 http://ee.oulu.fi/o#latitude ?2]

Template for Sensor 4:

urn:uuid:7bcf57

Prefix: EE:http://ee.oulu.fi/o

[http://ee.oulu.fi/o#LocaSensor http://ee.oulu.fi/o#locaSensor767 http://ee.oulu.fi/o#owner ?1 http://ee.oulu.fi/o#longitude "25.47" http://ee.oulu.fi/o#latitude "65.06"]

Template for Sensor 5:

urn:uuid:7bcf66

Prefix: EE:http://ee.oulu.fi/o

[http://ee.oulu.fi/o#LocaSensor http://ee.oulu.fi/o#locaSensor767 http://ee.oulu.fi/o#owner "JohnSmith" http://ee.oulu.fi/o#longitude "25.47" http://ee.oulu.fi/o#latitude "65.06"]

3.5 Negotiating short packets

For most resource-constrained sensors, sending predefined short packets is the easiest way to minimize resource consumption because no template negotiation is needed. More capable devices can negotiate the usage of short packets during their operation.

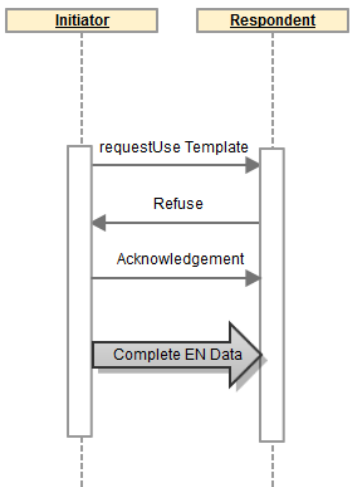


Fig. 5. Sequence diagram of the rejection of utilizing short packets.

An initiator with a template can suggest the short packet format. When an initiator suggests a respondent to use short packets, the respondent can either refuse or agree. If the respondent agrees to use the suggested template, it can request the template from the initiator if it does not have it in the memory. To use the template, the respondent needs to be able to identify the placeholders from the template in order to include the right values in the short packet.

The negotiating process is always done between two peers. Figure 5, 6, and 7 show the negotiating process. Figure 5 presents the message sequence when an initiator requests to utilize a template, but the respondent refuses to utilize this template. The acknowledgment message sent by the initiator confirms this and the initiator sends complete packets. Figure 6 presents the sequence when an initiator requests to utilize a template, and this is agreed by the respondent. The acknowledgment message sent by the initiator confirms this and short packets are then sent in the following data transfer. Figure 7 presents the sequence when an initiator requests to utilize a template, the respondent agrees to utilize this template and requests the template at the same time. The acknowledgment message sent by the initiator confirms this. The initiator sends first the template and then short packets during the data transfer.

Templates can be transferred as EN packets. Because the syntax of the templates does not comply with the complete packet format, we cannot transfer them as EN packets directly. But a template itself is also an entity that can be described with

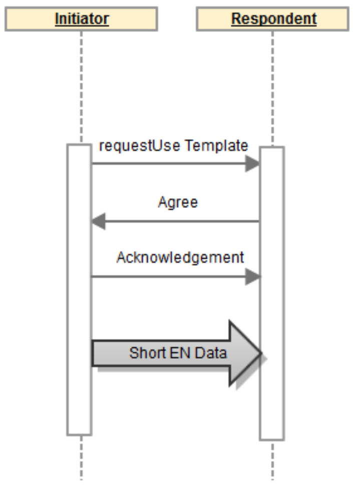


Fig. 6. Sequence diagram of the agreement of utilizing short packets.

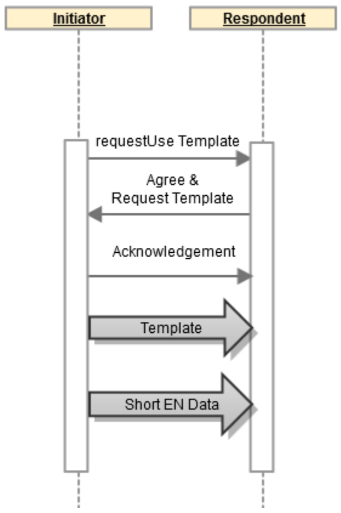


Fig. 7. Sequence diagram of the agreement of utilizing short packets and respondent need a template.

statements about the templateId, statements, and placeholders. Namespace prefixes can also be included in the template. Statements can be described using RDF reification in EN syntax (see [122], RDF Reification subsection). We use complete EN packets for expressing templates and there is no short packet format for them. Here, is the formal EBNF description for template packets:

```
EN-template = template-packet, {statement-packet};
template-packet = "[", template-type, template-Id, UUID-pair,
    {prefix-pair}, {statement-pair}, {placeholder-pair},"]";
UUID-pair = templateId-IRI, UUID-value;
prefix-pair = prefix-IRI, prefix-value;
statement-pair = hasStatement-IRI, statement-id;
placeholder-pair = hasPlaceholder-IRI, placeholder-id;
statement-packet  = "[", statement-type, statement-id, reification, "]";
reification = subject-IRI, entity-id, predicate-IRI, property-name,
object-IRI, value | placeholder-id;
template-type = (* IRI identifying a template class *);
template-Id = (* IRI identifying a single template *);
templateId-IRI = (* IRI identifying a templateID property *);
UUID-value = (* UUID identifying a template, 128-bit value *);
prefix-IRI = (* IRI identifying a prefix property *);
prefix-value = (* IRI identifying a prefix *);
hasStatement-IRI = (* IRI identifying the hasStatement property *);
hasPlaceholder-IRI = (* IRI identifying the hasPlaceholder property *);
entity-id = (* IRI identifying an entity *);
property-name = (* IRI identifying a property *);
statement-type = "http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement";
statement-id = (* IRI identifying a statement *);
placeholder-id = (* IRI identifying a placeholder *);
subject-IRI = "http://www.w3.org/1999/02/22-rdf-syntax-ns#subject";
predicate-IRI = "http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate ";
object-IRI = "http://www.w3.org/1999/02/22-rdf-syntax-ns#object ";
value = (* String for representing a fixed property value *);
```

The following example shows a template with three placeholders. This is the template used by the sensor 2 in Figure 4. A packet defining the template entity is as follows:

```
[http://ee.oulu.fi/o#Template http://ee.oulu.fi/o#template875
 http://ee.oulu.fi/o#templateID
 "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
 http://ee.oulu.fi/o#hasStatement http://ee.oulu.fi/o#statement1
 http://ee.oulu.fi/o#hasStatement http://ee.oulu.fi/o#statement2
 http://ee.oulu.fi/o#hasStatement http://ee.oulu.fi/o#statement3
 http://ee.oulu.fi/o#hasStatement http://ee.oulu.fi/o#statement4
 http://ee.oulu.fi/o#hasPlaceholder http://ee.oulu.fi/o#?1
 http://ee.oulu.fi/o#hasPlaceholder http://ee.oulu.fi/o#?2
 http://ee.oulu.fi/o#hasPlaceholder http://ee.oulu.fi/o#?3]
```

Here are four reification packets in EN format for expressing the statements.

```
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
 http://ee.oulu.fi/o#statement1
 http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
 http://ee.oulu.fi/o#locaSensor767
 http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
 http://ee.oulu.fi/o#owner
 http://www.w3.org/1999/02/22-rdf-syntax-ns#object
 http://ee.oulu.fi/o#?1]

[http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
 http://ee.oulu.fi/o#statement2
 http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
 http://ee.oulu.fi/o#locaSensor767
 http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
 http://ee.oulu.fi/o#longitude
 http://www.w3.org/1999/02/22-rdf-syntax-ns#object
 http://ee.oulu.fi/o#?2]

[http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
 http://ee.oulu.fi/o#statement3
 http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
 http://ee.oulu.fi/o#locaSensor767
 http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
 http://ee.oulu.fi/o#latitude
 http://www.w3.org/1999/02/22-rdf-syntax-ns#object
 http://ee.oulu.fi/o#?3]

[http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
 http://ee.oulu.fi/o#statement4
 http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
 http://ee.oulu.fi/o#locaSensor767
 http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
 http://www.w3.org/1999/02/22-rdf-syntax-ns#type
 http://www.w3.org/1999/02/22-rdf-syntax-ns#object
 http://ee.oulu.fi/o#LocaSensor]
```

These EN packets define a template which is presented in the second row of Table 4. Template EN packets are verbose because of the usage of RDF Reification. However, the same EN template packets can be delivered to multiple devices when they are required. Prefix can also be included in tempalte EN packets when needed.

3.6 Chaining EN packets

Packet chaining operates by chaining packets destined to the same output together [123]. Chaining EN packets can be useful in systems that consist of devices with different capabilities. For example, a gateway could receive packets from sensors, add additional values, and send the extended packets further to knowledge-based systems for reasoning over the data. In this section, we introduce different EN packet chaining solutions.

Figure 8 presents a scenario in which a sensor sends short packets to three gateways. The gateways are receiving the same packet from a location sensor and forwarding different packets to knowledge-based systems. Gateway 1 decodes the short packet partially, changes the template identifier to a new one and adds a timestamp value. The whole packet needs not to be decoded; the user name and location values can be handled in the format received as a character string. The knowledge-based system can then decode the data completely. Gateway 2 has a similar mechanism to Gateway 1, but it only forwards selected values. In this case, it deletes the “owner” value and adds a timestamp.

Gateway 3 works in a simpler way. It does not parse the packet it receives, but simply adds another packet in front of it and forwards them together. The first packet contains template identifier and a timestamp, and they are added by the gateway. The

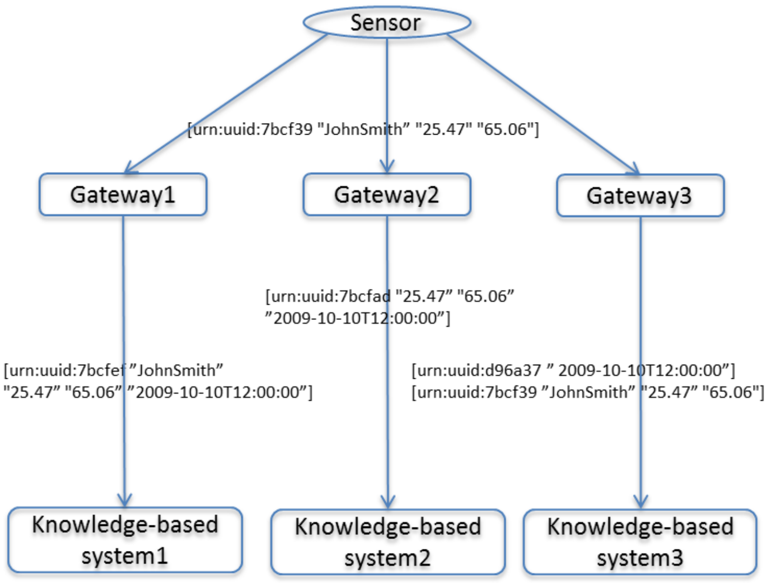

Sensor
[urn:uuid:7bcf39 "JohnSmith" "25.47" "65.06"]
Gateway1
Gateway2
Gateway3
[urn:uuid:7bcfad "25.47" "65.06" "2009-10-10T12:00:00"]
[urn:uuid:7bcfef "JohnSmith" "25.47" "65.06" "2009-10-10T12:00:00"]
[urn:uuid:d96a37 " 2009-10-10T12:00:00"] [urn:uuid:7bcf39 "JohnSmith" "25.47" "65.06"]
Knowledge-based system1
Knowledge-based system2
Knowledge-based system3

Fig. 8. Chaining EN packets.

second packet is the packet received from the sensor as it is. Some other nodes in pervasive systems, e.g. knowledge-based systems, can decompose the packets and use the values. The knowledge-based system is the only node that needs to decode the values encoded in the template identifiers and needs to have the knowledge to interpret the chain as one packet.

This method allows any number of packets to be chained. As long as the receiver knows the individual packets of the packet chain, the packets can be interpreted correctly. In other words, the server can assemble a complete packet from several short packets and the data can be picked from short packets correctly.

More capable gateways can perform a larger variety of operations on EN packets. They can have the capability to discover templates, transform short packets into complete format, pick variables from complete packets, compose new EN packets and transfer them to other system nodes.

3.7 Supported data structures

In this section, we present how different data structures can be represented with EN. These data structures are widely utilized for knowledge representations and pervasive systems. Although some of these structures are deprecated in RDF 1.1, they are supported by EN. In this section, we introduce how to represent RDF blank nodes, RDF Container, RDF Collection, and RDF Reification. Moreover, we also consider representing arrays and enumerations with EN.

EN triples enable the definition of graphs, just as RDF statements do. Blank nodes are used to indicate explicitly the existence of a thing, without using the identifier of that thing. The difference between blank nodes and other entities is in the interpretation; they cannot be accessed outside the interpretation they occur in.

However, the handling of blank nodes needs to be considered carefully as extra blank nodes may need to be introduced when transforming from RDF graphs into EN packets. Blank nodes are used in RDF to describe unnamed resources. They can be used as intermediate nodes for dealing with n-ary relationships. We use the following naming convention for blank node identifiers: an identifier starts with an underscore and a colon and continues with lowercase letters and a number, for example:

```
_:blanknode01
```

This representation follows the blank node naming in RDF/XML and N3. Other entities in the same set of EN packets processed at the same time can refer to blank nodes, but entities outside this set of EN packets cannot refer to this blank node identifier. EN does not support other ways for expressing blank nodes.

RDF provides containers and collections for expressing collections of members. These members can be entities or literals. Containers, including rdf:Bag, rdf:Seq, and rdf:Alt are used to describe an unclosed set of members, while collections are used to express a closed set. EN supports containers and collections. They are like other entities: they consist of EntityType, EntityId, and (PropertyName, PropertyValue) pairs. EntityType defines the type of the data structure, and EntityId identifies the data structure with an IRI. EntityId starts with the type of the container or collection and ends with a number. Then, we have a "http://ee.oulu.fi/o#hasItem" property, which has EntityIds or literals as property values. For example, an rdf:Bag container containing two location sensor descriptions can be expressed as follows:

```
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag
  http://ee.oulu.fi/o#bag208
  http://ee.oulu.fi/o#hasItem http://ee.oulu.fi/o#locaSensor767
  http://ee.oulu.fi/o#hasItem http://ee.oulu.fi/o#locaSensor768]
```

A reification is a statement about another statement. Expressing reifications in EN packets follows that of RDF. An entity statement has three properties: a subject, a predicate and an object for expressing the original statement. RDF reification vocabulary, including rdf:Statement, rdf:subject, rdf:predicate and rdf:object, is used in EN. The following example illustrates reification. First, the complete packet is:

```
[http://ee.oulu.fi/o#LocaSensor
 http://ee.oulu.fi/o#locaSensor767
 http://ee.oulu.fi/o#owner "JohnSmith"]
```

The reification for this packet is:

```
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
 http://ee.oulu.fi/o#statement65
 http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
 http://ee.oulu.fi/o#locaSensor767
 http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
 http://ee.oulu.fi/o#owner
 http://www.w3.org/1999/02/22-rdf-syntax-ns#object "JohnSmith"]
```

The corresponding Turtle document for this reification is:

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

<http://ee.oulu.fi/o#statement65>
  a rdf:Statement;
  rdf:subject <http://ee.oulu.fi/o#locaSensor767>;
  rdf:predicate <http://ee.oulu.fi/o#owner>;
  rdf:object "JohnSmith".
```

Arrays and enumerations are common data structures in programming languages. An array stores a collection of data in one or more dimensions, and an enumeration stores a list of values for variable items of the same type. They can be built from simple data types. An array consists of a collection of members, with each of them identified by one or more integer indices. We use "http://ee.oulu.fi/o#hasItem" property again to express an item in an array. Array indices start from 1. We can store a (longitude, latitude) pair in an array as follows:

```
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Array
 http://ee.oulu.fi/o#array1801
 http://ee.oulu.fi/o#hasItem "25.47"
 http://ee.oulu.fi/o#hasItem "65.06"]
```

The items are listed in the order determined by the indices. A short packet for a one-dimensional array looks like this:

```
 [urn:uuid:89b47e EE#array1801 "25.47" "65.06"]
```

In this packet, the prefix EE is used to shorten a full IRI. The complete representation of arrays is verbose but, on the other hand, can be easily transformed into RDF models. The short packets are not much longer than their representation in programming languages. The corresponding template for this short packet is:

```
urn:uuid:89b47e
Prefix: EE:http://ee.oulu.fi/o
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Array ?1
 http://ee.oulu.fi/o#hasItem ?2
 http://ee.oulu.fi/o#hasItem ?3]
```

The corresponding Turtle document for this array is:

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

<http://ee.oulu.fi/o#array1801>
  a <http://www.w3.org/1999/02/22-rdf-syntax-ns#Array>;
  <http://ee.oulu.fi/o#hasItem> "25.47";
  <http://ee.oulu.fi/o#hasItem> "65.06".
```

Arrays can have more than one dimension. For example, a two-dimensional array is a one-dimensional array having one-dimensional arrays as items. We can store a two-dimensional array as follows:

```
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Array
 http://ee.oulu.fi/o#array1801
 http://ee.oulu.fi/o#hasItem http://ee.oulu.fi/o#array2901
 http://ee.oulu.fi/o#hasItem http://ee.oulu.fi/o#array2902]

[http://www.w3.org/1999/02/22-rdf-syntax-ns#Array
 http://ee.oulu.fi/o#array2901
 http://ee.oulu.fi/o#hasItem "25.47"
 http://ee.oulu.fi/o#hasItem "65.06"]

[http://www.w3.org/1999/02/22-rdf-syntax-ns#Array
 http://ee.oulu.fi/o#array2902
 http://ee.oulu.fi/o#hasItem "27.47"
 http://ee.oulu.fi/o#hasItem "68.06"]
```

With a similar approach, an enumeration can be represented in EN packets. An enumeration contains a set of predefined literals. "http://ee.oulu.fi/o#hasItem" property is utilized to express one item in an enumeration again. Here, is an example of storing compass directions (north, south, east, and west) in an enumeration.

```
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Enum
 http://ee.oulu.fi/o#enum2746
 http://ee.oulu.fi/o#hasItem "North"
 http://ee.oulu.fi/o#hasItem "South"
 http://ee.oulu.fi/o#hasItem "East"
 http://ee.oulu.fi/o#hasItem "West"]
```

The corresponding short EN packet looks like this:

```
 [urn:uuid:ac3e476 EE#enum2746 "North" "South" "East" "West"]
```

The template for this short packet is:

```
urn:uuid:ac3e476
Prefix: EE:http://ee.oulu.fi/o
[http://www.w3.org/1999/02/22-rdf-syntax-ns#Enum ?1
 http://ee.oulu.fi/o#hasItem ?2
 http://ee.oulu.fi/o#hasItem ?3
 http://ee.oulu.fi/o#hasItem ?4
 http://ee.oulu.fi/o#hasItem ?5]
```

The corresponding Turtle document for this array is:

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

<http://ee.oulu.fi/o#enum2746>
  a <http://www.w3.org/1999/02/22-rdf-syntax-ns#Enum>;
  <http://ee.oulu.fi/o#hasItem> "North";
  <http://ee.oulu.fi/o#hasItem> "South";
  <http://ee.oulu.fi/o#hasItem> "East";
  <http://ee.oulu.fi/o#hasItem> "West".
```

3.8 Limitations

EN has sufficient expressive power, because the complete format follows the triple structure of RDF. Moreover, it supports many data types and the basic data structures of programming languages. EN allows building complex data structures from simple data types. However, as EN is deliberately kept simple, these data structures must be built from entities and relations between them. As a result, processing large data structures can require considerable resources.

One limitation of EN is that it does not support nesting. Nesting is widely supported by data representations and it compresses a representation. Here again, we left nesting deliberately out from EN to simplify the structure of individual packets, because we are aiming to minimize the lengths of individual packets, though this may lead to larger size for the complete description. Instead of nesting, IRI references and blank node identifiers are utilized to determine relations. One advantage of a set of small packets over a single large packet is that the small packets can have different routing paths. Moreover, a larger data structure can be sent in smaller pieces once at a time. This can also be done by lower layer protocols. This is possible with EN as every piece of information is identifiable and discoverable.

We restrict the usage of blank nodes in EN packet sets. Other entities in the same set of EN packets can refer to a blank node, but entities outside this set of EN packets cannot refer to that blank node identifier. When including blank nodes, a set of EN packets has to be processed together.

3.9 Summary

In this chapter, we introduced EN in detail, including design considerations, syntax, different packet formats, packet negotiating and chaining, supported data formats, and limitations. We focused on basic EN that can be transferred into RDF statements. We emphasize that EN is a lightweight data representation, which facilitates connecting resource-constrained devices to knowledge-based systems.

4 Entity Notation Schema

The previous chapter described how to transfer data. For a knowledge-based system, this means representing and transferring RDF statements with EN packets. In this chapter, we describe EN Schema for transferring ontologies. This enables EN to achieve the goal of semantic interoperability in pervasive environments. We present design considerations and illustrate with a small sensor ontology example how OWL 2 elements can be presented utilizing EN Schema.

4.1 Design considerations

As today's mobile devices have considerable computing, storage, and communication resources, they can play a big role in pervasive environments. But to do this, they need to access knowledge in a flexible fashion from highly dynamic environments. Knowledge transfer is one of the key issues to enable advanced functionality.

OWL 2 is a standardized Semantic Web language designed for modelling complex knowledge. Ontologies contain fundamental elements including axioms, entities, and expressions. Axioms are the basic statements that an OWL ontology expresses. Entities are elements that refer to real-world things, and expressions combine entities to form complex descriptions from basic ones. Most semantic reasoners are based on OWL 2, and RDF/XML is the only syntax that is mandatory to be supported by OWL 2 tools. However, different syntaxes of OWL 2 have mainly been designed for storing and utilizing knowledge and for editing and presentation purposes. We are not aware of any knowledge representation optimized for communication, although such a representation would enable richer functionality and decrease the communication payload for pervasive environments.

We suggest utilizing EN and EN Schema for transferring ontologies. EN Schema facilitates ontology transfer between mobile devices and knowledge-based systems. Moreover, when ontologies represented with OWL 2 can be transformed into EN and EN Schema, knowledge-based systems complying with Semantic Web technologies can be supported. Similarly to EN for RDF, EN Schema for ontologies has a complete format, which can represent ontology elements in a straightforward fashion. On the other hand, the short format is suitable for communication when resources are constrained. The complete format can easily be transformed into the short format, and vice versa.

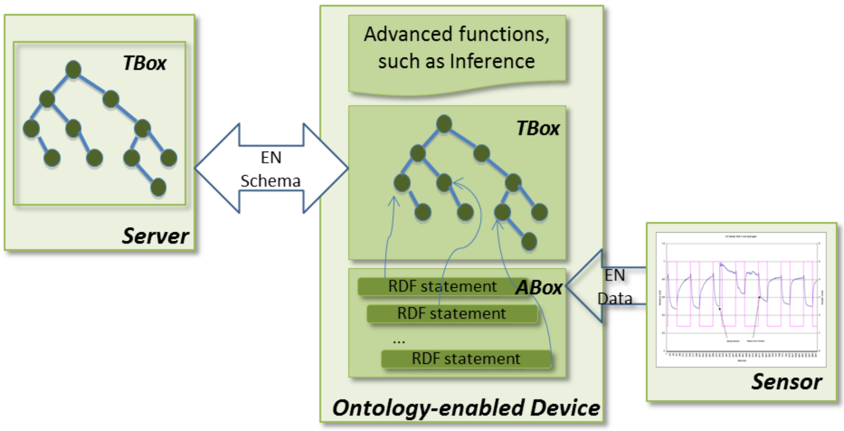


Fig. 9. Ontology-based processing in pervasive environments.

Figure 9 presents how EN facilitates knowledge processing in pervasive environments, consisting of resource-rich servers, resource-constrained sensors, and ontology-enabled devices. By ontology-enabled devices, we refer to mobile devices capable of utilizing ontologies to model the application domain and reason about the properties of the application domain [26]. An ontology-enabled device can access ontologies from a server, and data from resource-constrained devices, such as sensors. It is also possible to access data from a server, which is not shown in this figure. Hence, the device has all the knowledge required for inference and other ontology-based processing. EN and EN Schema play an important role in this scenario, as they enable knowledge transfer to ontology-enabled devices from other devices in pervasive environments.

We set the following requirements for EN Schema:

– First, EN Schema must be expressive enough for representing OWL 2 DL ontology. OWL 2 DL is selected, because it enables reasoners, in principle, to always answer "Yes" or "No". Moreover, up-to-date reasoners can support the entire OWL 2 DL features, but, not for OWL 2 Full. It should be possible to transform any description of OWL 2 DL knowledge into EN Schema, and vice versa. Fulfilling this goal enables serializing OWL 2 DL ontologies into EN Schema. This is essential, as OWL 2 is preferred by W3C and the most widely used syntax, and many reasoning tools are based on OWL 2.

- Second, EN Schema must support distributed ontology definitions. This feature allows decomposing an ontology description into individual packets. This results in a distributed description that can be transferred even one EN Schema packet at a time. Individual packets for one ontology can be distributed to different devices, at different times, and be transferred via different routes. Individual packets produced by different devices can also be transferred incrementally to compose a large shared ontology. This feature enables transferring ontologies in a very flexible fashion in highly dynamic pervasive environments. For example, an ontology-enabled device in Figure 9 can access knowledge incrementally with individual EN packets when it needs.
- Third, EN Schema must have an optimized syntax for communication links with limited bandwidth to transfer EN Schema when resources are constrained. In addition, we require EN Schema to be flexible enough for pervasive systems so that system designers can configure optimized EN and EN Schema for different systems.

We meet these requirements as follows:

- First, to have enough expressive power, EN Schema offers a format aligned with OWL. In the complete format, we adopt almost all terms from OWL and RDFS. It is straightforward to serialize OWL ontology into EN Schema and vice versa.
- Second, EN Schema supports distributed ontology definitions directly, as EN Schema allows fine-grained description of ontologies into EN Schema packets.
- Third, to support simple communication links, EN Schema offers compact short packets, which are easy to serialize and parse. Techniques based on templates and prefixes are utilized to shorten the packets. Each short packet type has a unique identifier and can hence be discovered and identified in an unambiguous fashion even when the packets of an ontology are not delivered as one set. Furthermore, EN Schema is flexible as it offers a large variety of possibilities for the designer to decide how much information to be included in packets and how much to be included in templates.

4.2 Complete packet format

We present our solution for expressing entities of an ontology with the complete EN Schema format in this section. Entities in the EN Schema are abstract concepts and relations between them, for example, classes, properties, individuals, and restrictions.

Complete packet format and short packet format follows the EBNF structure defined in Section 3.2. A complete packet for EN Schema is of the form:

```
  <EntityType EntityName
    EntityRelation RelationName
    ...
    EntityCharacteristics CharacteristicsName>
```

As we discussed above, EN and EN Schema support incremental and distributed definitions for entities. A device might first possess only this description:

```
<EntityType EntityName>
```

Any number of EntityRelation or EntityCharacteristics can be added for this EntityName gradually, when it receives EN Schema packets about the same entity. A complete packet for EN Schema must always contain EntityType and EntityName. In addition, a packet can contain any number of (EntityRelation, RelationName) pairs and any number of (EntityCharacteristics, CharacteristicsName) pairs (including zero for both). Hence, these are valid EN Schema packets:

```
<EntityType EntityName
    EntityRelation RelationName>

<EntityType EntityName
    EntityCharacteristics CharacteristicsName>
```

As we mentioned in Chapter 3, entity descriptions are enclosed within angle brackets ($<$ and $>$) for EN Schema. In this entity description, EntityType describes the primary type of an entity. For example, the type owl:Class indicates the next token as a class name, while the type owl:ObjectProperty indicates the next token as an ObjectProperty. EntityName is interpreted to specify the primary entity this packet is about. Any number of descriptions (i.e. pairs) about this entity can follow this token without repeating EntityName. EntityRelation presents a relationship between this entity and some other entities such as rdfs:subClassOf and rdfs:domain. The value of EntityRelation (i.e. RelationName) determines some other entity. Type information is implicitly defined by EntityRelation. For example, the value of rdfs:subClassOf has the type owl:Class, while the value of owl:onProperty has the value of owl:Property. An entity can have other type information described as EntityCharacteristics. For example, an ObjectProperty can be announced as a TransitiveProperty. In this general form, there is no limitation for the amount or ordering of pairs.

Table 5. Mapping of EN Schema to OWL.

EN Schema	OWL
EntityType	Type of an element (rdf:type)
EntityName	Name of an OWL element
EntityRelation	OWL constructor that relates one OWL element to another
EntityCharacteristics	Characteristics of elements (rdf:type)
RelationName	Name of an OWL element as the target of the relation
CharacteristicsName	Characteristics of OWL elements

A complete packet can be transformed into an OWL description straightforwardly, as presented in Table 5. EntityName can be mapped to an element name directly, while EntityType indicates the type of the element. Relationships between entities, such as rdfs:subClassOf and rdfs:domain, can be mapped to OWL constructors. EntityCharacteristics can be mapped to characteristics of elements the term with utilizing rdf:type, while CharacteristicsName can be mapped directly to characteristics name in OWL.

Table 6 lists the RDF and RDFS vocabulary that can be utilized in EN Schema. That is, EN Schema packets can contain these constructs. Moreover, EN Schema utilizes all OWL 2 constructs. More details of OWL 2 constructs can be found from [124].

Table 6. RDF and RDFS vocabulary for EN and EN Schema.

Construct	Description
rdfs:Class	The class of classes.
rdfs:Literal	The class of literal values.
rdfs:Datatype	The class of RDF datatypes.
rdf:XMLLiteral	The class of XML literal values.
rdf:Property	The class of RDF properties.
rdfs:domain	A domain of a property, a Class.
rdfs:range	A range of a property, a Class.
rdfs:subClassOf	The entity is a sub-class of a class.
rdfs:subPropertyOf	The entity is a sub-property of a property.
rdfs:label	A human-readable name for the entity.
rdfs:comment	A description of the entity.
rdfs:Container	The class of containers.
rdf:Seq	The class of ordered containers.
rdf:Bag	The class of unordered containers.
rdf:Alt	The class of containers of alternatives.
rdfs:ContainerMembershipProperty	The class of container membership properties.
rdf:*n* (digital number)	Enumeration, each of these is a sub-property of rdfs:member.
rdf:List	The class of Lists.
rdf:first	The first item in the list.
rdf:rest	The rest of the list after the first item.
rdf:nil	Indication of the end of the list.
rdf:Statement	The class of statements, used for RDF reification.
rdf:subject	The subject of the statement, used for RDF reification.
rdf:predicate	The predicate of the statement, used for RDF reification.
rdf:object	The object of the statement, used for RDF reification.
rdfs:seeAlso	Further information about the entity.
rdfs:isDefinedBy	The definition of the entity.
rdf:value	Idiomatic property used for structured values.

rdf:type is not utilized in EN. We always specify EntityType before the name of an entity. When transforming a set of EN packets to an RDF graph, the relation of EntityType and EntityId is converted to rdf:type.

Next we present how to represent different OWL 2 DL elements with EN and EN Schema with a small sensor ontology example. We introduce basic constructs for building classes, properties, instances, and their basic modelling features. These are most of the essential features that OWL 2 supports. We follow the ordering of W3C OWL 2 Primer [125]. Base URI xml:base="http://ee.oulu.fi/o/" is utilized in all following examples of complete EN Schema packets.

4.2.1 Classes and instances

In general, classes represent sets of individuals. In modelling, class definitions describe the concepts in ontologies. We define a named class in EN Schema as follows:

```
<owl:Class ClassNameA>
```

For example, Sensor class can be defined as follows:

```
<owl:Class Sensor>
```

This EN Schema packet can be transformed into the following OWL fragment in Turtle:

```
:Sensor a owl:Class.
```

We define an instance as an EN packet as follows:

```
[ClassNameA instancenameA]
```

For example, a location sensor can be defined as an instance of the Sensor class:

```
[http://ee.oulu.fi/o#Sensor http://ee.oulu.fi/o#locaSensor767]
```

This can be transformed into the following fragment in Turtle:

```
http://ee.oulu.fi/o#locaSensor767 a http://ee.oulu.fi/o#Sensor;
```

4.2.2 Class hierarchies

OWL 2 utilizes rdfs:subClassOf to relate a specific class to a more general one. In EN Schema, a subclass relation can be defined as follows:

```
<owl:Class ClassNameB
  rdfs:subClassOf ClassNameC>
```

For example, Sensor and LocationSensor classes can be defined like this:

```
<owl:Class LocationSensor
  rdfs:subClassOf Sensor>
```

This can be transformed into the following OWL fragment in Turtle:

```
:LocationSensor a owl:Class.
:LocationSensor rdfs:subClassOf :Sensor.
```

As we introduce more EN Schema constructs, we add details to these classes. For an ontology defined in a distributed fashion, it is important to relate the distributed parts with each other. OWL 2 provides a mechanism by which classes are considered to be semantically equivalent. Namely, OWL construct equivalentClass is utilized to identify equivalence between classes. EN Schema utilizes this construct straightforwardly:

```
<owl:Class ClassNameD
  owl:equivalentClass ClassNameE>
```

For example, a user might import a new class SensorwithMemoryLimitation, which is an identical concept with MemoryConstrainedSensor. In this case, we can claim that the classes SensorwithMemoryLimitation and MemoryConstrainedSensor are the same class as follows:

```
<owl:Class SensorwithMemoryLimitation
  owl:equivalentClass MemoryConstrainedSensor>
```

This packet can be transferred into OWL fragment in Turtle as follow:

```
:SensorwithMemoryLimitation a owl:Class;
  owl:equivalentClass :MemoryConstrainedSensor.
```

4.2.3 Class disjointness

Class disjointness announces that membership in one class specifically excludes membership in another. This is done as follows in EN Schema:

```
<owl:AllDisjointClasses _:blanknodeA>
  owl:members  CollectionA>
<en:Collection CollectionA
```

```
  hasClassItem ClassNameF
  ...
  hasClassItem ClassNameG>
```

We give each collection an identifier, and utilize the term hasClassItem to list all items for this collection. Below, we utilize one extra new prefix “en” that has the value "http://ee.oulu.fi/o/en#" as a prefix for the collection entity, as there is no OWL 2 standard prefix for this term:

```
<en:Collection CollectionID
  hasClassItem ClassNameH
  ...
  hasClassItem ClassNameI>
```

For example, we can announce that no individual can be an instance of both Sensor class and Person class as follows:

```
<owl:AllDisjointClasses _:blanknode01
  owl:members  collection101>
<en:Collection collection101
  hasClassItem Sensor
  hasClassItem Person>
```

These can be transferred into OWL fragment in Turtle as follow:

```
 []  a   owl:AllDisjointClasses;
     owl:members  (:Sensor :Person).
```

4.2.4 Object properties

Properties present entity relations in a general form, i.e. binary relations for individuals in classes. Two kinds of properties can be distinguished: object properties and datatype properties. Object properties present relations between individuals of two classes. In EN Schema, an object property can be declared as follows:

```
<owl:ObjectProperty ObjectPropertyNameA>
```

We use the example presented in Section 3 as an example. A line segment measured from the location of one location sensor can be defined like this:

```
<owl:ObjectProperty startsFrom>
[http://ee.oulu.fi/o#linesegment25
 http://ee.oulu.fi/o#startsFrom http://ee.oulu.fi/o#locaSensor767]
```

This can be transformed into the following fragment in Turtle:

```
http://ee.oulu.fi/o#linesegment25
 http://ee.oulu.fi/o#startsFrom http://ee.oulu.fi/o#locaSensor767.
```

We can also state that two individuals are not connected by a property with EN Schema as follows:

```
<owl:NegativePropertyAssertion _:blanknode02
  owl:sourceIndividual IndividualA
  owl:assertionProperty DatatypePropertyB
  owl:targetIndividual IndividualB.
>
```

For example, we can define the line segment is not measured from the point of one location sensor. In EN Schema, it can be defined like this:

```
<owl:NegativePropertyAssertion _:blanknode03
  owl:sourceIndividual http://ee.oulu.fi/o#linesegment25
  owl:assertionProperty http://ee.oulu.fi/o#startsFrom
  owl:targetIndividual http://ee.oulu.fi/o#locaSensor767.
>
```

The corresponding fragment in Turtle is as follow:

```
 []  rdf:type               owl:NegativePropertyAssertion;
     owl:sourceIndividual   http://ee.oulu.fi/o#linesegment25;
     owl:assertionProperty  http://ee.oulu.fi/o#startsFrom;
     owl:targetIndividual   http://ee.oulu.fi/o#locaSensor767.
```

4.2.5 Property hierarchies

EN Schema allows to specify a sub-property relation between properties, which behaves similarly to a sub-class relation between classes. In EN Schema, a sub-property relation can be declared as follows:

```
<owl:ObjecttypeProperty ObjectPropertyNameA
  rdfs:SubPropertyOf ObjectPropertyNameB>
```

For example, we can define that hasMother is a sub-property of hasParent:

```
<owl:ObjecttypeProperty hasMother
  rdfs:SubPropertyOf hasParent>
```

This can be transformed into the following OWL fragment in Turtle:

```
:hasMonther rdfs:subPropertyOf :hasParent.
```

Moreover, there is also possibility to define property equivalence, in the same way as defining class equivalence. Equivalent property can be defined as follows:

```
<owl:ObjectProperty PropertyNameB
  owl:equivalentProperty PropertyNameC>
```

4.2.6 Domain and range restrictions

There are several ways to restrict a property, including specifying a domain and a range for a property. Domain and range restrictions carry implicit additional information about individuals. We utilize the RDFS constructs rdfs:domain and rdfs:range for an object property as follows:

```
<owl:ObjectProperty ObjectPropertyNameC
  rdfs:domain ClassNameJ
  rdfs:range ClassNameK>
```

The following object property presents that a sensor is owned by a person.

```
<owl:ObjectProperty hasOwner
  rdfs:domain Sensor
  rdfs:range Person>
```

This packet can be transformed into the following OWL fragments in Turtle:

```
:hasOwner a owl:ObjectProperty;
  rdfs:domain :Sensor;
rdfs:range :Person.
```

4.2.7 Equality and inequality of individuals

Similarly to constructs for identifying equivalence and difference between classes and properties, sameAs, differentFrom, and allDifferent are utilized for describing relations between individuals.

```
<EntityType EntityId
  owl:sameAs individualC>
<EntityType EntityId
```

```
  owl:differentFrom individualD>
<EntityType EntityId
  owl:DifferentIndividuals individualE>
```

For example, the fact that sensor001 is different from sensor002 can be expressed in EN as follows:

```
<owl:NamedIndividual sensor001
   owl:differentFrom sensor002>
```

This packet can be transformed into:

```
:sensor001 a owl:NamedIndividual;
  owl:differentFrom :sensor002.
```

4.2.8 Datatype properties

Datatype properties present relations between an individual and literals or XML Schema data types. Domain and range can also be stated for datatype properties as it is done for object properties. EN Schema supports rdfs:Literal and XML Schema data types that are built-in OWL data types. EN Schema describes a datatype property like this:

```
<owl:DatatypeProperty DatatypePropertyNameA
  rdfs:domain ClassNameL
  rdfs:range xsd:datatype>
```

The following is an example using a datatype property. It describes that sensors have memory capacity expressed as a float.

```
<owl:DatatypeProperty hasMemoryCapability
  rdfs:domain Sensor
  rdfs:range xsd:float>
```

These packets can be transformed into the following OWL fragments in Turtle:

```
:hasMemoryCapability a owl:DatatypeProperty;
  rdfs:domain :Sensor;
  rdfs:range xsd:float;
```

Likewise, EN Schema supports negative datatype property as follow:

```
<owl:NegativePropertyAssertion _:blanknode04
  owl:sourceIndividual IndividualF
  owl:assertionProperty DatatypePropertyC
  owl:targetValue Value1.
>
```

For example, we can define that a sensor's memory capacity is not 32kb. In EN Schema, it can be defined like this:

```
<owl:NegativePropertyAssertion _:blanknode05
  owl:sourceIndividual http://ee.oulu.fi/o#locaSensor767
  owl:assertionProperty http://ee.oulu.fi/o#memoryCap
  owl:targetValue "32768"
>
```

The corresponding fragment in Turtle is as follow:

```
 []  rdf:type               owl:NegativePropertyAssertion ;
     owl:sourceIndividual   http://ee.oulu.fi/o#locaSensor767;
     owl:assertionProperty  http://ee.oulu.fi/o#memoryCap;
     owl:targetIndividual   32768.
```

4.2.9 Advanced class relationships

Advanced class relationships include complex classes, property restrictions, property cardinality restrictions, and enumerations of individuals. These relationships are about how named classes, properties, and individuals can be used as building blocks to define new classes.

Complex classes

New classes can be constructed by using owl:intersectionOf, owl:unionOf, owl:complementOf, and owl:oneOf class operators, as follows:

```
<owl:Class ClassID
  owl:intersectionOf CollectionID>
<owl:Class ClassID
  owl:unionOf CollectionID>
<owl:Class ClassID
  owl:complementOf CollectionID>
```

As introduced in Section 4.2.3, collection is an identified entity and utilizes the term hasClassItem to list all items for this collection. Similarly, it is also possible to use class constructors together with a rdfs:subclassOf constructor to indicate necessary, but not sufficient, conditions for a class:

```
<owl:Class ClassNameM
  rdfs:subClassOf ClassNameN
  owl:intersectionOf CollectionID>
```

If we want to make a new class to express memory constrained location sensor, this can be achieved by applying intersectionOf constructor on LocationSensor class and MemoryConstrainedSensor class. We can have a new class in EN Schema:

```
<owl:Class MemoryConstrainedLocationSensor
  owl:intersectionOf collection201>
<en:Collection collection201
  hasClassItem LocationSensor
  hasClassItem MemoryConstrainedSensor>
```

These two packets can be transformed into OWL representation like this:

```
:MemoryConstrainedLocationSensor a owl:Class;
  owl:intersectionOf (:LocationSensor :MemoryConstrainedSensor).
```

Property restrictions

A property restriction describes an anonymous class. All individuals of this class satisfy the specified restriction. OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints. Term owl:onProperty is always utilized with property restrictions.

OWL typically utilizes nesting to represent property restrictions. Because we do not use nesting in EN and EN Schema, we create restriction entities and give an identifier for each of them. In this way, we split a complex nesting structure into smaller packets. EN Schema describes restrictions like this:

```
<owl:Class ClassName
  owl:equivalentClass  RestrictionID>
<owl:Restriction RestrictionID
  owl:onProperty PropertyName
  owl:allValuesFrom ClassName>
<owl:Restriction RestrictionID
  owl:onProperty PropertyName
  owl:someValuesFrom ClassName>
<owl:Restriction RestrictionID
  owl:onProperty PropertyName
  owl:hasValue Individual>
<owl:Restriction RestrictionID
  owl:onProperty PropertyName
  owl:hasSelf ClassName>
```

For example, the following complete packets specify a Class Sensorowner which links to Sensor class with own property:

```
<owl:Restriction Sensorowner
  owl:onProperty own
  owl:someValuesFrom Sensor>
```

These packets can be mapped to OWL syntax in Turtle as follows:

```
 :SensorOwner  owl:equivalentClass  [
   rdf:type           owl:Restriction ;
   owl:onProperty      :own;
   owl:someValuesFrom  :Sensor].
```

Property cardinality restrictions

Similar to identifications for collections, we give each restriction an identifier. Cardinality restriction is a special property, which enables to specify the number of individuals involved in the restriction. There are three kinds of cardinality restrictions, which specify maximum, minimum, and exact amounts:

```
<owl:Restriction RestrictionID
  owl:onProperty PropertyName
  owl:maxCardinality value>
<owl:Restriction RestrictionID
  owl:onProperty PropertyName
  owl:minCardinality value>
<owl:Restriction RestrictionID
  owl:onProperty PropertyName
  owl:exactCardinality value>
```

For example, the following complete packets specify that sensing nodes having more than one physical sensors are sensorgroup:

```
<owl:Class Sensorgroup
  rdfs:subClassOf  Sensor
  rdfs:subClassOf  restriction001>
<owl:Restriction restriction001
  owl:onProperty hasPhysicalSensor
  owl:minQualifiedCardinality "2">
```

These packets can be mapped to OWL syntax in Turtle as follows:

```
:Sensorgroup a owl:Class;
    rdfs:subClassOf [a owl:Restriction;
            owl:minQualifiedCardinality "2";
            owl:onProperty :hasPhysicalSensor],
        :Sensor.
```

Enumeration of individuals

OWL 2 provides a possibility to describe a class by enumerating the individuals of this class. EN Schema can support this feature utilizing equivalentClass and a collections of individuals:

```
<owl:Class ClassName
  owl:equivalentClass CollectionId>
```

Such a collection includes a set of individuals:

```
<en:Collection CollectionId
  hasItem IndividualId
  ...
  hasItem IndividualId>
```

For example, we can build a class for sensors in TS354 like this:

```
<owl:Class SensorsinTS354
  owl:equivalentClass collection301>
<en:Collection collection301
  hasItem locationSensor203
  hasItem temperatureSensor357>
```

These packets can be mapped to OWL syntax as follows:

```
:SensorsinTS354 a owl:Class;
    owl:equivalentClass [a owl:Class;
    owl:oneOf (:locationSensor203 :temperatureSensor357 ) ].
```

4.2.10 Advanced use of properties

In this section, we now focus on other capabilities with properties, including property characteristics, property chains, and keys.

Property characteristics

Property characteristics specify additional type information for properties. We can utilize EN Schema to describe inverse object property, symmetric object property, asymmetric object property, transitive object property, disjoint object property, reflexive object property, irreflexive object property, functional object property, and inverse functional object property. They can be presented in EN Schema like this:

```
<owl:ObjectProperty PropertyName
  owl:inverseOf PropertyName>
<owl:ObjectProperty PropertyName
  rdf:type SymmetricProperty>
<owl:ObjectProperty PropertyName
  rdf:type AsymmetricProperty>
<owl:ObjectProperty PropertyName
  rdf:type TransitiveProperty>
<owl:ObjectProperty PropertyName
  Owl:propertyDisjointWith PropertyName>
<owl:ObjectProperty PropertyName
  rdf:type ReflexiveProperty>
<owl:ObjectProperty PropertyName
  rdf:type IrreflexiveProperty>
<owl:ObjectProperty PropertyName
  rdf:type FunctionalProperty>
<owl:ObjectProperty PropertyName
  rdf:type InverseFunctionalProperty>
```

For example, the transitive object property locatedIn can be presented as:

```
<owl:ObjectProperty locatedIn
  rdf:type TransitiveProperty>
```

And its corresponding OWL syntax is:

```
:locatedIn a owl:ObjectProperty;
a owl:TransitiveProperty.
```

Among these property characteristics, the expression of inverse object property and disjoint object property characteristics differ from the others as these two have another property name as their value. We can take the property hasOwner as an example. It might have an inverse object property:

```
<owl:ObjectProperty hasOwner
  owl:inverseOf own>
```

The corresponding OWL fragment is:

```
:hasowner a owl:ObjectProperty;
  owl:InverseProperties :own.
```

Property chains

Property chains allow defining a property as the composition of several properties. This is a new feature only available in OWL 2. EN Schema supports this feature by utilizing owl:propertyChainAxiom collection and a collection of properties. EN Schema describes restriction like this:

```
<owl:ObjectProperty ObjectProperyNameD
  owl:propertyChainAxiom collection302>
<en:Collection collection302
  hasItem ObjectProperyNameE
  hasItem ObjectProperyNameF
  hasItem ObjectProperyNameG
  ...
>
```

The following property chain means that when a device made an observation and that observation has a result, the result is the one produced by the device.

```
<owl:ObjectProperty hasResult
  owl:propertyChainAxiom collection303>
<en:Collection collection303
  hasItem madeObservation
  hasItem ObservationResult
>
```

The corresponding OWL fragment is:

```
:hasResult  owl:propertyChainAxiom ( :madeObservation :ObservationResult).
```

Keys

Keys allow each named instance of a class expression to be uniquely identified by the set of values which these properties attain in relation to the instance. Keys can be expressed in EN Schema as follows:

```
<owl:Class ClassNameO
  owl:haskey collection401>
<en:Collection collection401
  hasItem ObjectProperyNameH
  hasItem ObjectProperyNameI
  ...
>
```

For example, each instance of Class device is unique identified by an UUID:

```
<owl:Class Device
  owl:hasKey collection402>
<en:Collection collection402
  hasItem hasUUID
  ...
>
```

And the corresponding OWL fragment is:

```
:Device owl:hasKey (:hasUUID ...).
```

4.2.11 Advanced use of datatypes

This section presents more advanced features of datatype usage, which is introduce in OWL 2. Datatypes can be restricted via facets, borrowed from XML Schema Datatypes. For example, we define a new datatype for a mobile device's memory size by constraining the datatype integer to values between (inclusively) 0 and 17179869184.

```
<rdfs:Datatype mobiledevicemomorysize
   owl:onDatatype  xsd:integer
   owl:withRestrictions collection502>
<en:Collection collection502
  hasItem anonymous1
  hasItem anonymous2
>
```

```
<owl:Class anonymous1
xsd:minInclusive "0"^^xsd:integer
>
<owl:Class anonymous2
xsd:maxInclusive "17179869184"^^xsd:integer
>
```

And the corresponding OWL fragment is:

```
mobiledevicemomorysize
    owl:withRestrictions
  ( [ xsd:minInclusive "0"^^xsd:integer  ]
[ xsd:maxInclusive "17179869184"^^xsd:integer ] ) ;
    owl:onDatatype xsd:integer.
```

With a similar approach, datatypes can be combined just like classes by complement, intersection and union. Moreover, a new datatype can be generated by enumerating the data values it contains.

4.2.12 Document information and annotations

Similar to other syntax, EN Schema also supports features for providing information about the ontology itself, including annotation assertion, defining names for an ontology, and importing other ontologies.

We could add information to one of the classes of the ontology, giving a natural language description of its meaning. EN Schema utilizes rdfs:comment like this:

```
<owl:Class Sensor
  rdfs:comment "A device which detects or measures a physical property
  and records, indicates, or otherwise responds to it."^^xsd:string>
```

And the corresponding OWL fragment is:

```
:Sensor rdfs:comment
"A device which detects or measures a physical property
  and records, indicates, or otherwise responds to it."^^xsd:string.
```

We can also provide a name for an ontology, which is generally the place where the ontology document is located.

```
<owl:Ontology http://ee.oulu.fi/o/IoTOntology>
```

And the corresponding OWL fragment is:

```
<http://ee.oulu.fi/o/IoTOntology> a owl:Ontology.
```

We utilize Turtle syntax for defining prefixes in EN Schema. For example, the standard expansion for *owl* is defined as follows:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

OWL allows the import of the contents of entire ontologies in other ontologies, using import statements. With EN Schema, we define the import like this:

```
<owl:Ontology http://ee.oulu.fi/o/IoTOntology
 owl:imports http://ee.oulu.fi/o/SensorOntology.owl>
```

And the corresponding OWL fragment is:

```
<http://ee.oulu.fi/o/IoTOntology> owl:imports
<http://ee.oulu.fi/o/SensorOntology.owl>.
```

4.2.13 Sensor ontology example

Part of the EN Schema packets we introduced in previous sections can be assembled to form the sensor ontology example presented in Figure 10. This ontology includes several concepts about sensors, some characteristics, and restrictions for these sensors, and relations among them. Though this small ontology has only 4360 bytes, it offers a formal description to support rule-based reasoning. For example, when an individual location sensor sends packets about its owner and capability to a computing node that has this ontology, the received information can be registered and cataloged. After that, location measurements from this sensor can potentially be utilized to infer activities of this user.

4.3 Short packet format

The complete format offers a solution for representing and transferring ontologies. Complete packets can be quite long because of meaningful IRIs and verbose OWL constructors. Similar to the idea for RDF statements, we utilize templates and prefixes to shorten the packets. A template contains a description of the constant part of a set of EN Schema packets and placeholders (expressed by a question mark followed with an integer) for the variable items. The packet sent over the communication links needs to

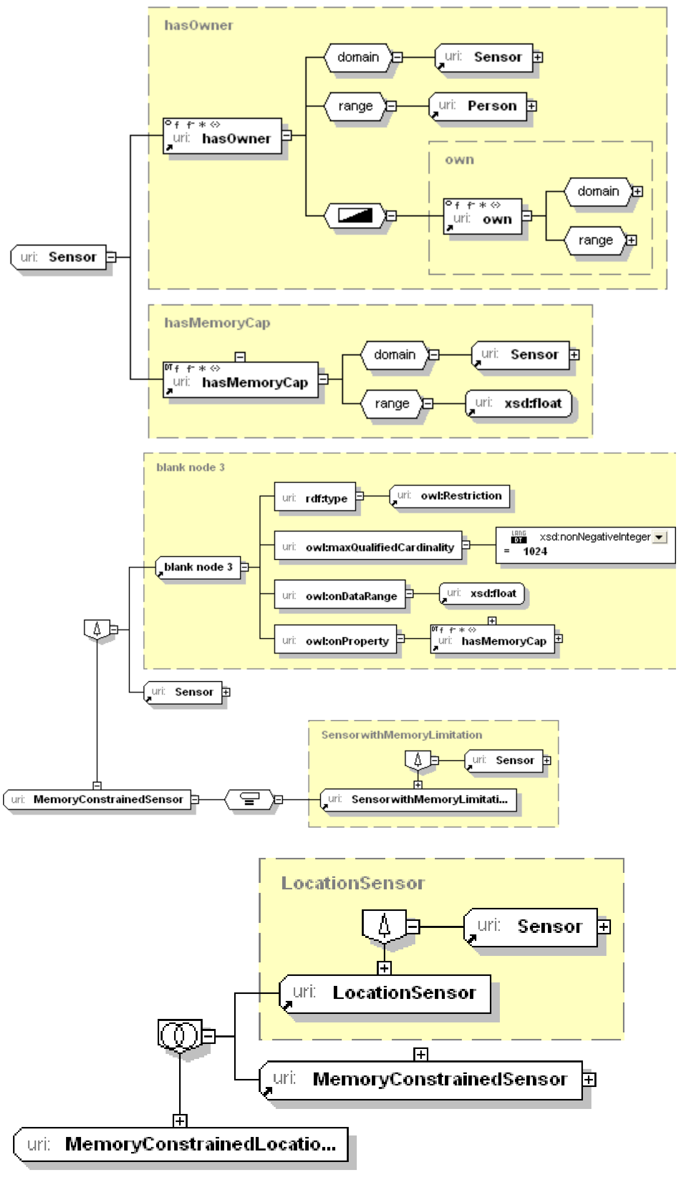


Fig. 10. Sensor ontology example.

contain only a template identifier and the variable items. A set of complete packets can then be assembled by replacing the template's placeholders with the values contained in a short packet. Prefixes are used to shorten IRI references.

Generally, the EntityType, EntityRelation, and EntityCharacteristicsName should be constant parts for the short packets, while the EntityName, RelationName, and CharacteristicsName could be variable items. Different from complete packets, short packets do not have the restriction that one packet can only describe one entity. Any number of entities and their description can be described in one short packet. A short packet describing one entity has the following format:

```
<UUID EntityName Value  ... Value>
```

Here, we use "Value" for expressing names for the relations and characteristics. When a short packet format includes EntityNames for several entities, the format is:

```
<UUID
EntityName Value ... Value
  ......
EntityName Value ... Value
>
```

Moreover, a short packet can include names for some entities, but exclude some others. For example, the EntityName for the last few values is excluded like this:

```
<UUID
EntityName Value ... Value
  ......
           Value ... Value
>
```

The short packet format is quite flexible; hence, it leaves space for designers to optimize their pervasive systems. A good system design minimizes the usage of resources, like bandwidth, memory, and CPU cycles, by optimizing information allocation between templates and short packets.

An UUID in a short packet is used to identify the template. Figure 11 presents an example of using different templates to transfer the same ontology fragment about the sensor class and its relations. It shows two mobile devices with different templates, and the corresponding short packets they receive. A mobile device at the left side is accessing a short packet starting with a 32 bit UUID and following with all values about sensor class within one short packet. The mobile device has the following template:

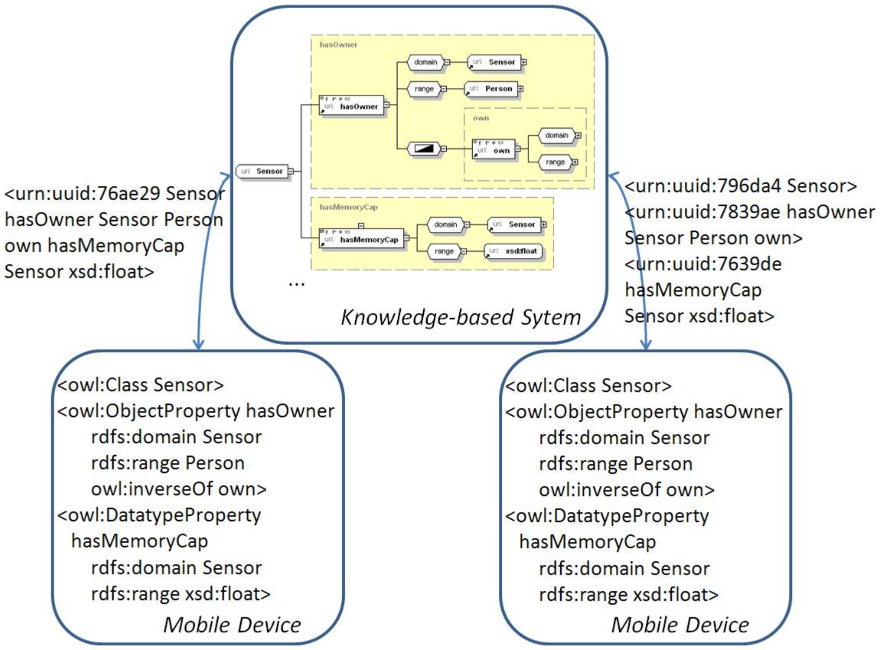


Fig. 11. The example ontology in use.

```
<owl:Class ?1>
<owl:ObjectProperty ?2
    rdfs:domain ?3
    rdfs:range ?4
    owl:inverseOf ?5>
<owl:DatatypeProperty ?6
    rdfs:domain ?7
    rdfs:range ?8>
```

In this template, there are eight placeholders in total for class name, object property name, its domain, its range, its inverse property name, datatype property name, domain, and range of this property. Placeholders from multiple complete packets can reserve places for values in one short packet. In other words, one short packet can be transformed into more than one complete packet.

A mobile device at the right side hosts templates for three complete packets:

```
<owl:Class ?1>
<owl:ObjectProperty ?1
    rdfs:domain ?2
    rdfs:range ?3
    owl:inverseOf ?4>
<owl:DatatypeProperty ?1
       rdfs:domain ?2
       rdfs:range ?3>
```

These complete packets can then be assembled and transferred separately. Compared with the short packet at the left side, the short packets at right side are 39.5% longer. On the other hand, multiple templates introduce flexibility. Each of these templates only includes essential constructors for one kind of element in an ontology. For example, when another datatype property needs to be transferred, the third template (related to the datatype property) can be adopted to reuse.

By using the template above, only an UUID and a set of values need to be included in short packets when ontology descriptions are transferred between devices and knowledge-based systems. In this way ontology can be transferred without information loss.

4.4 Summary

In this chapter, we presented EN Schema for representing OWL ontologies. We introduced design requirements, complete and short EN for representing OWL 2 knowledge, utilization of OWL constructs in EN Schema, and a small sensor ontology example. We emphasize that EN Schema enables semantic interoperability in pervasive environments.

5 Evaluation

In this chapter, we evaluate EN and EN Schema. We first present small evaluations and prototypes of a few system components. Then we present larger prototypes of complete systems that utilize a wider set of EN features. First, we compare the expressive power of EN against other alternatives. Then, we evaluate the lengths of EN messages in a simulated system and with well-known data sets and ontologies for both EN and EN Schema. The simulator also verifies that EN can be utilized to control devices, such as robots. In the first small prototype, EN is utilized in a wearable sensor for detecting and reporting the well-being of elderly people. With the second prototype, we analyze the resource usage of EN and alternative representations. The first larger prototype adopts EN to build a knowledge-based system for ambient social interactions. EN is utilized to transfer Global Positioning System (GPS) data and social data produced by human users of mobile phones. In the second larger prototype, we study how EN can be utilized in a general framework for two layer inference.

The author of this thesis performed all the research in the Section 5.1, Section 5.2, and Section 5.3. Section 5.4 presents experiments of transferring ontologies. The author of this thesis developed the first version of the parser for OWL and Pingjiang Li developed the second version of the parser for OWL 2 based on the latest EN Schema design. Section 5.5 presents two sensor prototypes. The author of this thesis designed the EN descriptions, developed the PC side applications of these prototypes, was involved in the resource consumption evaluation of the sensors, and analyzed the results. Embedded programming was done by Dr. Janne Haverinen and Mr. Markus Koskimies. Section 5.6 presents an ambient social interaction application. The author of this thesis designed the general framework, EN for ambient social interactions, and analyzed EN in the prototype. Together with Ekaterina Gilman, Paweł Kwiatkowski, Tomasz Latkowski, Alma Pröbstl, Bartłomiej Wójtowicz, and Jukka Riekki, the author performed the design, implementation, and testing of the prototype. Section 5.7 presents the two-layer inference framework. The author of this thesis was responsible for EN related work, was involved in the prototyping, and performed the analysis. Davide Fucci was involved in the brainstorming and discussions.

5.1 Expressive power

We first consider the expressive power of EN. In Table 7, we compare the semantic expressive power of EN against RDF/XML [126], N3 [44], SenML [46], and JSON-LD [10]. Among these alternatives, RDF/XML, N3, JSON-LD, and EN can be mapped to graphs [127]. Hence, they support ontologies straightforwardly. RDF/XML and N3 have a triple centric structure as the base representation. Other alternatives, including SenML, JSON-LD, and EN, follow the entity centric approach. SenML has a more free form data structure, which cannot be mapped to graphs in a similar fashion. Hence, SenML data cannot be utilized by knowledge-based systems as easily as the other alternatives. On the other hand, SenML may be easy to produce by devices, because it can be mapped easily to the basic data structures of programming languages. JSON-LD shares this benefit with SenML. Meanwhile, short EN is also easy to produce as it has such a simple structure. Describing types of entities is important for all data formats, because it enables linking to higher level knowledge. All alternatives can express entity types, but complete EN packets require this as a mandatory element. RDF, N3, and JSON-LD support rich XML Schema data types, while SenML supports only four basic data types, that is, floating point, integer, Boolean value, and string. All these data formats support external semantic information. RDF and N3 support mechanisms to import additional knowledge. EN does not have a similar mechanism, but its packet structure enables a natural way of knowledge integration. SenML and JSON-LD support additional semantics via linking to other Web resources.

Table 7. Comparison of expressive power.

	RDF/XML	N3	SenML	JSON-LD	EN
Mapping to Graphs	Y	Y	N	Y	Y
Triple Centric Structure	Y	Y	N	N	N
Entity Centric Structure	N	N	Y	Y	Y
Device Type	Y	Y	Y	Y	Y
Data Types	XSD	XSD	4 types	XSD	XSD
External Semantics	Y	Y	Y	Y	Y

5.2 Resource usage

We discuss the resource usage of EN and introduce a way to optimize a system at design time in this section. From the examples presented earlier, it can be seen that when a device sends data to other system nodes as short EN packets, a packet needs to contain only the template identifier and the variable items. A receiver can assemble the complete packet without any information loss as long as it has the template. EN leaves space for the designers for optimizing their pervasive systems by tuning packet lengths and the number of templates to produce an optimal combination of bandwidth usage for sending short packets and Central Processing Unit (CPU) and memory usage for processing the packets.

Next, we sketch how resource usage can be controlled at design time by considering the number of templates. We focus mainly on resource-constrained devices. We assume that:

n = number of variable items in a complete packet.

The items are EntityIds, PropertyNames, and PropertyValues in EN packets. Entity-Types are excluded, because they do not appear in Short EN packets. The number of items in a corresponding short packet varies depending on templates used. The shortest packet only includes a UUID, while the longest one includes all variable items. Hence, we assume:

p = number of variable items in a short packet, $p \in [0, n]$.

We consider the minimal and maximal number of templates. The minimal amount of templates is one, it is achieved when all networked sensors send identical messages and all variable items are encoded in short packets. For example, if there are five location sensors sending their owners' locations, the template presented for Sensor 1 in Table 4 can be utilized for all these five sensors. The maximum amount of templates is achieved when only UUIDs are included in each short packet, similar to Sensor 5 in Figure 3.

Assuming the number of different values of the item $\boldsymbol{i}$ as x_i, we can calculate the maximum number of templates as:

$$\prod_{i=1}^{n} x_i.$$

For example, there might be five temperature sensors and 50 different temperature values. In this case, the number of items is two, i.e. a packet contains one EntityId and one PropertyValue. 250 (5 times 50) templates are required if only UUID is included in short packets. 250 different UUIDs are required as well.

For a system not having templates for every different combination of values, we need to consider both the amount of variable items in the short packets and values of these items. We assume that, when p>0, items from x_{n-p+1} to x_n are included in the short packet (indices start from 1). When p equals n, exactly one template is needed, as all values are in short packets. When p is smaller than n,

$$\prod_{i=1}^{n-p} x_i$$

templates are needed to communicate all possible values. Every short packet has p values. The values for the rest of the variables are encoded in the templates.

As the resource-constrained sensors do not store templates, but just construct short packets, the amount of memory required by templates does not need to be considered. Instead, we estimate how placing one variable item from a short packet to a template affects resource usage. Briefly, such an operation increases the number of templates by a factor of x_i. Instead of writing the value to the short packet, the resource-constrained sensor needs to select the correct template identifier for the short message. This produces more code and hence more memory is needed to store the program and more CPU cycles to execute it. On the other hand, bandwidth usage decreases as one less variable item is included in the short packet.

We estimate memory usage with $f_m(p)$, CPU usage with $f_c(p)$, and bandwidth usage $f_b(p)$. We normalize them to $F_b(p)$, $F_m(p)$, and $F_c(p)$. Resource consumption can be optimized by selecting the optimal number of items for short packets. The costs of different resources can be considered. If all resources have the same cost, the designer can minimize:

$$F_b(p) + F_m(p) + F_c(p).$$

When the costs are different, the designer can use weights to present differences and minimize:

$$w_b * F_b(p) + w_m * F_m(p) + w_c * F_c(p).$$

As all these functions strongly depend on the underlying technology and the implementation of the software, these functions will not be discussed in more detail here.

It should be noted that the sensors do not need to have any information about the templates, they only insert template identifiers in the messages they send. Usage of short packets can be decided by the engineer at design time. The second option is that the sensors negotiate the usage during communication.

5.3 Simulator and RDF/XML data sets

In this section, we evaluate EN by using a device simulator and some standard RDF/XML datasets. To evaluate EN, we develop Java functions and applications to encode and decode EN packets. An EN encoder realizes the functionality of encoding RDF/XML documents into a set of EN packets. A decoder, in its turn, performs this process in the opposite direction. Figure 12 and Figure 13 present the main components of an encoder and a decoder, respectively. It should be noted that when EN is processed in embedded devices, the encoding process is often easier than this, because an embedded device builds EN packets directly from its internal data values, not from RDF/XML.

The encoding process from RDF/XML to EN is as follows: First, the encoder reads an RDF/XML file and preprocesses it. In the next step, RDF statements are split and grouped based on their EntityIds. That is, statements about the same entity are grouped together. Then, the complete packet encoder serializes statements with Jena and builds complete EN packets. If the communicating peers utilize short packets, complete packets are matched with the templates used by the peers, and short packets including only UUIDs and values are built. Finally, packets are sent to the recipients.

The decoder works in the opposite direction: when a recipient receives a packet, the decoder checks whether the packet is a short packet or a complete packet. Short packets can be recognized based on the UUIDs that are always the first items. For short packets, templates and prefixes are utilized to build complete packets. After that, complete packets are transformed into RDF/XML.

The simulator contains four types of devices: cleaning robots, temperature sensors, location sensors, and wireless devices designed for reporting a patient's well-being to nurse [128]. The simulator consists of a server and user interface for each device.

Users can simulate the devices by writing messages on the user interface, assembling short packets or complete packets, and sending them to a server. A message contains

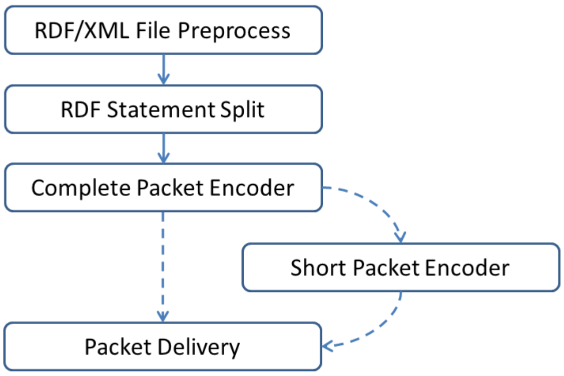


Fig. 12. EN encoder.

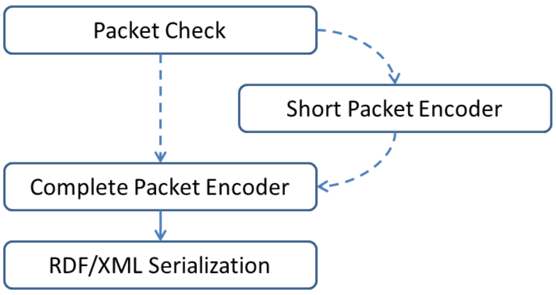
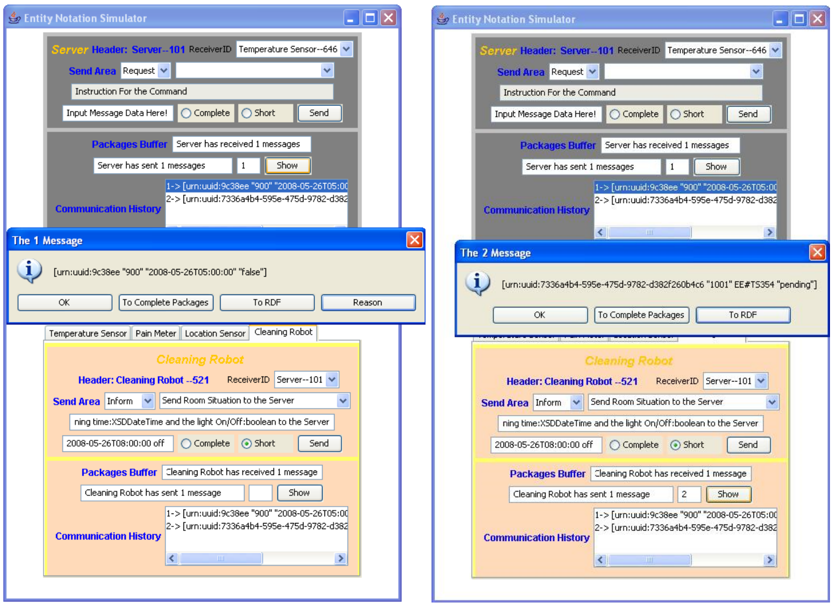


Fig. 13. EN decoder.

one or more packets. When a server receives a message, it transforms the packets into RDF/XML, adds them into an ontology model, and reasons on data in the model for deducing a result based on pre-defined rules. The deduced result is delivered to the specific device or all devices according to the entity of the deduced packet. For example, when the deduced packet contains an entity identification of a cleaning robot, a message is sent to this robot. We include one packet in front of other packets in each message to describe the message itself, including the type, the sender, receiver, and a sequence number. This packet is also included in the corresponding template; but the sequence number is the only item that is included in short packets. This added packet is useful in the simulator and some real implementations. In those implementations that same information is carried by a lower layer protocol, such an packet is not needed.


Entity Notation Simulator

Server Header: Server--101 ReceiverID Temperature Sensor--646
Send Area Request
Instruction For the Command
Input Message Data Here! ○ Complete ○ Short Send
Packages Buffer Server has received 1 messages
Server has sent 1 messages 1 Show
Communication History
1-> [urn:uuid:9c38ee "900" "2008-05-26T05:0
2-> [urn:uuid:7336a4b4-595e-475d-9782-d382

The 1 Message
[urn:uuid:9c38ee "900" "2008-05-26T05:00:00" "false"]
OK | To Complete Packages | To RDF | Reason

Temperature Sensor | Pain Meter | Location Sensor | Cleaning Robot
Cleaning Robot
Header: Cleaning Robot --521 ReceiverID Server--101
Send Area Inform Send Room Situation to the Server
ning time:XSDDateTime and the light On/Off:boolean to the Server
2008-05-26T08:00:00 off ○ Complete ◉ Short Send
Packages Buffer Cleaning Robot has received 1 message
Cleaning Robot has sent 1 message Show
Communication History
1-> [urn:uuid:9c38ee "900" "2008-05-26T05:0
2-> [urn:uuid:7336a4b4-595e-475d-9782-d382

Entity Notation Simulator

Server Header: Server--101 ReceiverID Temperature Sensor--646
Send Area Request
Instruction For the Command
Input Message Data Here! ○ Complete ○ Short Send
Packages Buffer Server has received 1 messages
Server has sent 1 messages 1 Show
Communication History
1-> [urn:uuid:9c38ee "900" "2008-05-26T05:0
2-> [urn:uuid:7336a4b4-595e-475d-9782-d382

The 2 Message
[urn:uuid:7336a4b4-595e-475d-9782-d382f260b4c6 "1001" EE#TS354 "pending"]
OK | To Complete Packages | To RDF

Cleaning Robot
Header: Cleaning Robot --521 ReceiverID Server--101
Send Area Inform Send Room Situation to the Server
ning time:XSDDateTime and the light On/Off:boolean to the Server
2008-05-26T08:00:00 off ○ Complete ◉ Short Send
Packages Buffer Cleaning Robot has received 1 message
Cleaning Robot has sent 1 message 2 Show
Communication History
1-> [urn:uuid:9c38ee "900" "2008-05-26T05:0
2-> [urn:uuid:7336a4b4-595e-475d-9782-d382

Fig. 14. The simulator is sending EN messages with cleaning robot.

Figure 14 shows a situation in which the cleaning robot has sent a short packet to the server and has just received a command as a response. The cleaning robot can assemble both complete and short messages, and receives messages in the corresponding format from the server. The server responds to the robots' messages automatically, by sending the deduced packet. The left screenshot in Figure 14 shows the following short packet sent from the robot to the server:

```
[urn:uuid:9c38ee "900" "2008-05-26T05:00:00" "false"]
```

In this message, the identifier urn:uuid:9c38ee determines the template, 900 is the sequence number, and the last two values are the previous cleaning time of the room and the Boolean value of "light on" context. The server reasons with the following rule (slightly modified from a real Jena rule):

```
(EE#TS354,lightIntensity,off) ∧ (EE#TS354,previousCleaningTime,?PT)
∧ greaterThan(currentTime-PT,24hours)
-> (cleaningRobot521,clean,roomTS354)
```

This rule states that room TS354 needs to be cleaned when over 24 hours has elapsed from the previous cleaning and the lights are off. It is assumed that a room is empty when lights are off. When this rule is fired, the server commands the cleaning robot to clean the room by the following short packet:

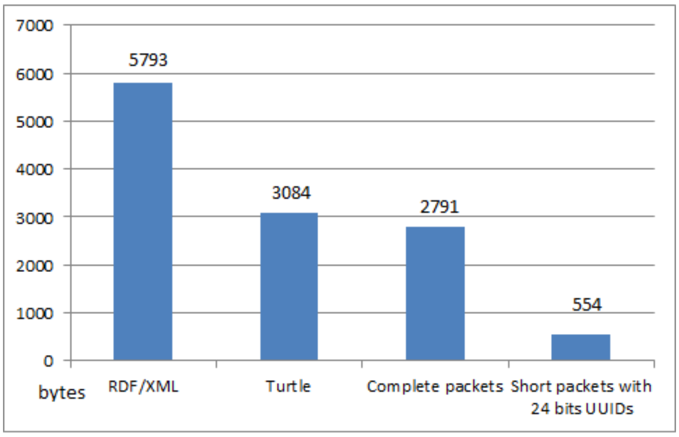


Fig. 15. The lengths of RDF/XML, Turtle, complete packets and short packets with 24 bit UUIDs in the device simulator.

```
[urn:uuid:bd61dee5-e9b8-422e-948d-e7799caae04b "1001" EE#TS354 "pending"]
```

In this message, 1001 is sequence number. The following values tell that the room to be cleaned is TS354 and the action status is pending. The status "pending" is generated with another rule of the reasoner for every new task. Other statuses are "interrupted" and "completed" for tasks of cleaning robots.

In the simulator, the server and the robots can send seventeen different packets in total: the server can send requests to the robots, while the robots can reply to the server's requests, and send asynchronous information to the server. The lengths of these seventeen packets in different formats are shown in Figure 15. It can be seen that the short EN packets utilizing 24-bit UUIDs contain on average 20% of the characters of complete packets and 10% of those of the corresponding RDF document.

In the second experiment, we selected RDF files from the W3C standard test sets [129] for evaluating the EN syntax. 35 RDF data files were selected, containing most of the data structures and data types. Some of the RDF files were modified slightly. Modifications included deleting annotations and adding EntityType when necessary. EntityTypes were added to avoid blank nodes as EntityTypes and makes RDF/XML files more readable. XML annotations were deleted as well.

Figure 16 presents the lengths of different representations for the RDF data (from left to right): RDF/XML, N3, EN complete packet and EN short packets with different

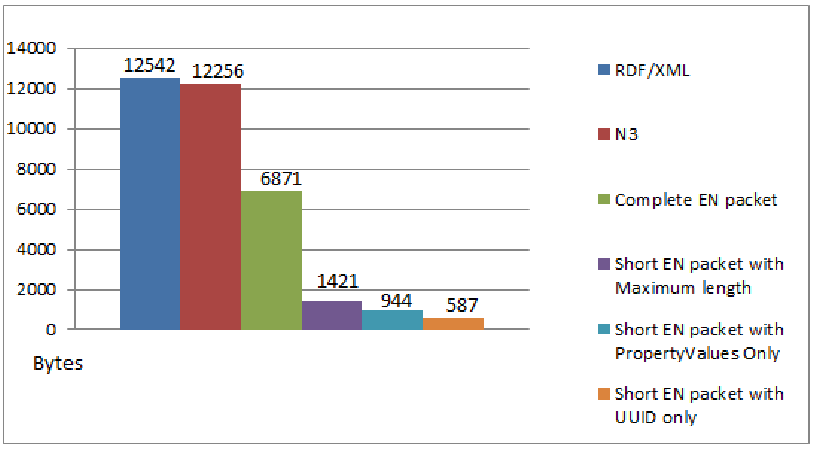


Fig. 16. RDF data file lengths for different representations.

templates. N3 is generated with an online converting service. [2] When a template with all possible placeholders is utilized, we have the longest short packets with the EntityId and all property values (third from the right in Figure 16). Short packets with all property values, but not EntityIds are somewhat shorter, as expected (second from the right in Figure 16). As an extreme case (at the right in Figure 16), packets can contain only UUIDs. The average length of the longest short packets is around 11% of that of the corresponding RDF/XML files. The average length of short packets with property values is around 8% of RDF/XML, while in the shortest case, the packets only with UUIDs is just 5% of that of the corresponding RDF files. This is the case when 16 bit UUIDs are utilized. The ratio of the short packet size to that of the RDF file can be even smaller for larger data structures; specifically, when the structures contain large constant parts.

XML annotations were removed. Complete EN format is more compact than RDF/XML in most cases, like expressing triple structure. But RDF/XML can be more compact for expressing collections, containers and reifications because of nesting. Our experiment with simulator and RDF data set verified that EN is a compact representation, hence, delivering EN packets can be done in a resource-efficient manner.

5.4 Transforming ontologies

In this section, we evaluate transforming ontologies to EN Schema and vice versa. We have developed a Java based implementation to verify transforming EN Schema

[2]http://www.easyrdf.org/

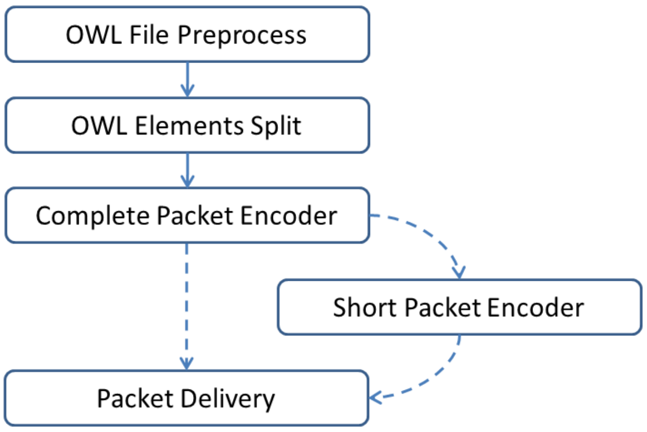


Fig. 17. EN Schema encoder for Ontologies.

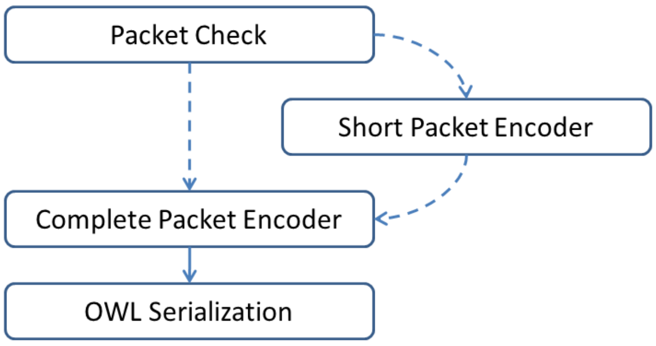


Fig. 18. EN Schema decoder for Ontologies.

packets from and to OWL ontologies. Similar to the encoder and decoder for EN, this software has also EN encoder for OWL and EN decoder for OWL. The encoder realizes the functions of transforming OWL ontology into complete packets, and then to short packets if necessary. The decoder performs this process to the opposite direction. Figure 17 and Figure 18 present the main components of the encoder and decoder, respectively.

The encoding process is as follows: First, the encoder reads an OWL ontology file and preprocesses it. In this step, the component checks the content of this ontology and removes the information that is not needed by mobile devices, like human readable annotations and comments. In the next step, OWL elements are grouped into groups, including classes, datatype properties, object properties, individuals, restrictions, and collections. Among these groups, individuals are transferred to EN packets, while others are transferred to EN Schema packets. Then, the complete packet encoder reads entities from these groups, connects PropertyNames and PropertyValues to them, and places them within brackets. When two communicating peers use short packets, complete packets are matched with available templates, and short packets including only UUIDs and values are built. Finally, the packets are sent to recipients.

The decoding component works in the opposite way: when a recipient receives a packet, it checks whether the packet is a short packet or a complete packet by checking UUIDs. For short packets, templates and prefixes are utilized to build complete packets. After that, complete packets are transformed into OWL syntax according to the mapping rules introduced in the previous chapter, and added to an OWL ontology.

Table 8. EN Schema packets and corresponding OWL fragments in RDF/XML.

EN Schema:	<owl:Class Environment rdfs:subClassOf Context>
OWL:	<owl:Class rdf:ID="Environment">
	<rdfs:subClassOf rdf:resource="#Context"/>
	</owl:Class>
EN Schema:	<owl:Class Researcher rdfs:subClassOf Person>
RDF/XML:	<owl:Class rdf:ID="Researcher">
	<rdfs:subClassOf>
	<owl:Class rdf:resource="#Person"/>
	</rdfs:subClassOf>
	</owl:Class>
EN Schema:	<owl:Class Light rdfs:subClassOf Enviroment>
RDF/XML:	<owl:Class rdf:ID="Light">
	<rdfs:subClassOf rdf:resource="#Enviroment"/>
	</owl:Class>
EN Schema:	<owl:Class Robot rdfs:subClassOf Device>
RDF/XML:	<owl:Class rdf:ID="Robot">
	<rdfs:subClassOf rdf:resource="#Device"/>
	</owl:Class>
EN Schema:	<owl:ObjectProperty personLocatedin rdfs:domain Person
	rdfs:range Location owl:inverseOf hasPersoninside>
RDF/XML:	<owl:ObjectProperty rdf:about="#personLocatedin">
	<rdfs:domain rdf:resource="#Person"/>
	<rdfs:range rdf:resource="#Location"/>
	<owl:inverseOf rdf:resource="#hasPersoninside"/>
	</owl:ObjectProperty>
EN Schema:	<owl:DatatypeProperty lightIntensity rdfs:domain Light
	rdfs:range xsd:boolean>
RDF/XML:	<owl:DatatypeProperty rdf:ID="lightIntensity">
	<rdfs:domain rdf:resource="#Light"/>
	<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
	</owl:DatatypeProperty>

Table 8 presents selected elements in one ontology and their corresponding EN Schema packets. We utilized the ontology we presented in Chapter 4. This is a very lightweight ontology, but still includes all basic elements, including 22 classes, eleven datatype properties, seven object properties, and some individuals. With the encoder and decoder we introduced in this section, we successfully transform this ontology into EN and EN Schema packets, and vice versa. After transforming into EN and EN Schema and then back to OWL, we compare this OWL file with original OWL file and confirm that they are semantically equal with manual comparison. The order of fragments of elements can be changed in this process, but all elements and their properties and relations are the same as in the original file.

In addition to this test, we evaluate Complete EN Schema packet with a set of eight ontologies, which are well known and widely utilized in pervasive, IoT, and other domains. These ontologies include COBRA-ONT [130], Semantically Interconnected Online Communities (SIOC) ontology [131], Semantic Sensor Network Ontology [14], IoT-Lite ontology [132], Organization ontology [133], and the sample ontology of OWL 2 primer [125]. COBRA-ONT includes a collection of ontologies for describing vocabularies in an intelligent meeting room use case. We are testing with COBRA-ONT ontologies version 0.6 which contains Ebiquity-geo, Ebiquity-meetings, and Ebiquity-actions ontologies. The SIOC ontology is designed for describing online communities such as forums, blogs, mailing lists, and wikis. The SSN ontology describe the sensors and observations, and related concepts in pervasive environment. The IoT-Lite ontology is a lightweight ontology to represent IoT resources, entities, and services. It is also a meta ontology that can be extended in different domains. The Organization Ontology is designed to enable publication of information on organizations and organizational structures including governmental organizations. It is designed as a generic, reusable core ontology that can be extended or specialized for use in particular situations. Finally, the sample ontology of OWL 2 primer is utilized for verifying OWL 2 features.

Figure 19 presents the comparison of the lengths of different syntaxes, including Functional Syntax, EN and EN Schema, Turtle, Manchester Syntax, RDF/XML, JSON-LD, N-Triple, and OWL/XML (from left to right). Complete EN Schema and EN packets are tested in this evaluation. We noticed that EN and EN Schema are among most lightweight syntaxes. Among alternatives, they are most compact representations for five ontologies, including COBRA-ONT: ebiquity-meeting, SIOC, SSN, the Organization Ontology, and OWL 2 primer sample ontology. For other ontologies, EN and EN Schema are second, third, or forth compact representations among eight alternatives.

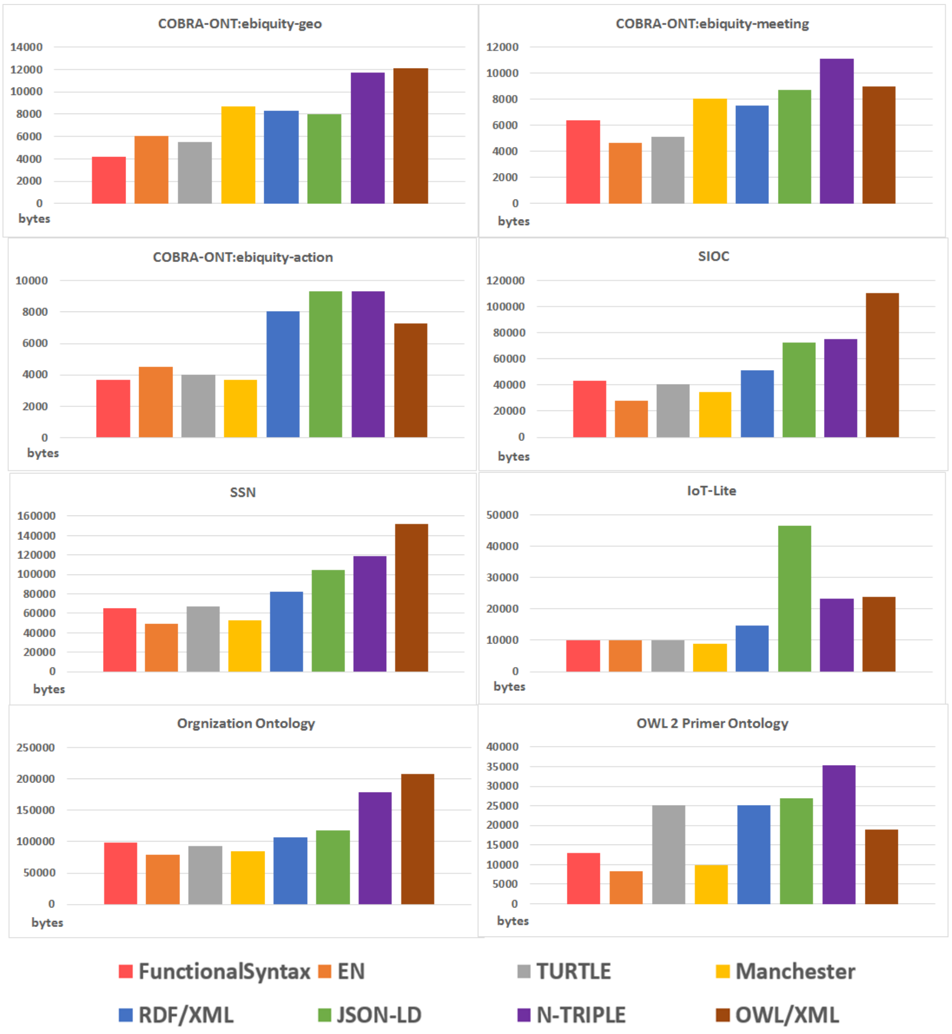


Fig. 19. Comparison of the lengths of Functional Syntax, EN and EN Schema, Turtel, Manchester Syntax, RDF/XML, JSON-LD, N-Triple, and OWL/XML with different ontologies.

Taking SSN ontology as an example, the length of EN Schema and EN is about 36.6% of N-Triple and about 52.8% of RDF/XML.

Moreover, we have verified that major constructs of OWL 2 ontologies can be transformed to EN Schema. We tested these transformation with ontologies mentioned before. Especially, example ontology of OWL 2 primer include new constructs of OWL 2. All OWL 2 constructs presented in this sample ontology are transformed successfully. OWL constructs related to classes, instances, class hierarchies, class disjointness, object properties (excluding NegativePropertyAssertion), property hierarchies, domain and range, equality and inequality of individuals, datatype properties(excluding Negative-PropertyAssertion), owl:intersectionOf, owl:unionOf, owl:complementOf, property restrictions, property cardinality restrictions (excluding ExactCardinality), property characteristics (excluding ReflexiveProperty, IrreflexiveProperty, and FunctionalProperty), owl:propertyChainAxiom, and owl:hasKey have been verified. Our tests also verify that EN Schema is often a more compact representation than other alternatives for representing OWL 2 ontologies.

5.5 Sensor prototypes

In this section, we present two prototypes. This first one is a well-being reporter, and the second one extends this system to a general sensor system consisting two sensors and a knowledge processing component on a PC.

5.5.1 Well-being reporter

To evaluate EN in small resource-constrained devices, we built a pervasive system prototype reporting the wellbeing of an elderly person. This system contains a wearable sensor device and a PC application. The sensor detects possible falling down of the person carrying it and irregular temperatures as well. Figure 20 presents the main components and the data flow of the system. In this system, a sensor composes EN packets and sends them to the PC application. This application decodes the packets and transforms them into RDF statements. The template is stored in the EN decomposer. Therefore, template cost is not considered. The PC application integrates the received values into a small ontology and triggers an inference engine. The engine detects abnormal situations and stores the reasoned facts into the ontology. The system reports these situations to persons whose contact information has been configured to the system.

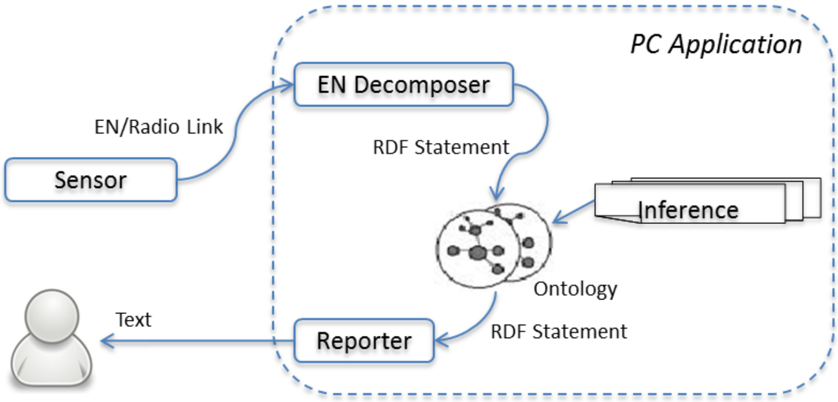


Fig. 20. Main components and data flow of the well-being reporter.

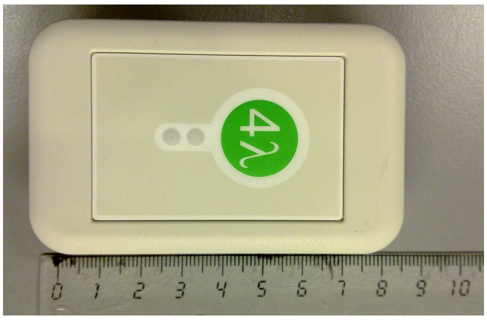


Fig. 21. A wearable sensor for detecting well-being.

The sensor node used in the experiments is shown in Figure 21. The sensor node consists of an Atmel's 8-bit ATmega32 MCU with 32kB flash and 2kB Static random access memory (SRAM), a short range radio link and sensors; a 3-axis magnetometer, a 3-axis accelerometer, and a thermometer. In addition, the sensor node includes a push button and a real time clock; thus, it can send a time stamp together with the measurements. The software of the sensor node has been implemented as a standalone application and no operating system has been used.

An EN packet generated by the sensor node contains a timestamp, acceleration and magnetic field data and the binary state of the push button. All data items are represented by 16-bit signed integers except the timestamp for which 32-bit unsigned integer representation has been used. Floating point data type was avoided in order to decrease the memory footprint of the C standard library. The EN packet, including 128 bit UUID and the sensor readings, is generated simply by using sprintf function, as shown below. The string (i.e. EN packet) produced by the sprintf function is sent to the PC application by calling write function which is associated with the low-power radio interface of the sensor. The C code for generating the EN packet is the following:

```
sprintf(b,
    "[urn:uuid:311b4e80-d9fd-11de-8a39-0800200c9a66
     \"%lu\" \"%d\" \"%d\" \"%d\" \"%d\" \"%d\" \"%d\"
     \"%d\" \"%d\"]",
     timestamp, acceleration_x, acceleration_y,
     acceleration_z, magnetic_x, magnetic_y,
     magnetic_z, temperature, buttonstate);
write((const char*) b, strlen(b));
```

From this code, it can be seen that sensors can generate EN packets in a simple and straightforward fashion. The UUID value is used as a constant and concatenated with the produced values.

The PC application detects possible falling down and abnormal temperature from sensory data. Generally, individual values can easily be extracted from short EN packets when the types and the order of the values are known. This enables fast preprocessing of sensor data with lightweight algorithms - which in its turn reduces the burden of the inference engine. For example, for detecting abnormal temperature, temperature values can be extracted from the message strings and cached for three minutes. Then, the mean value can be calculated from the cached values and sent to the reasoner. Moreover, falling down can be detected by first extracting accelerometer readings from EN packets (from the character strings) and then processing them with a fall detection algorithm. When resources are constrained, a simple algorithm can be utilized. The algorithm can first detect total acceleration exceeding a predefined threshold (indicating a physical impact). After such acceleration, the algorithm can analyze if the predefined threshold value for the vertical inclination angle is exceeded for a certain period of time, which suggest a non-standing posture. A non-standing posture following a physical impact is categorized as a fall.

EN packets can be transformed into RDF statements, as introduced in Chapter 3. New RDF statements trigger the reasoner when they are added into the ontology. The reasoner matches RDF statements with the antecedents of rules in its rule set. The reasoning produces new RDF statements and adds them to the ontology.

The resource usage of this sensor, including CPU cycles, power consumption and transmission time is shown in Figures 22 - 25. We compare EN with N3 and RDF/XML

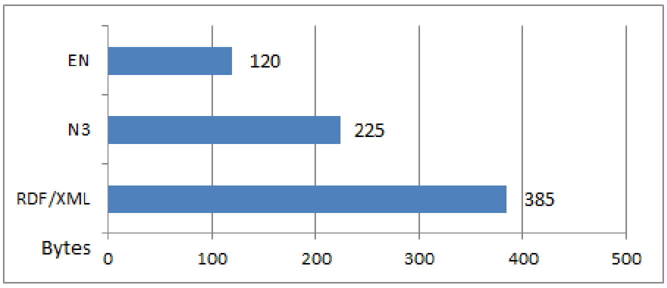


Fig. 22. Packet lengths of different representations.

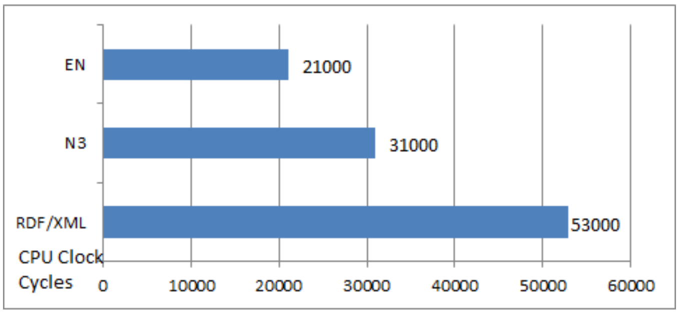


Fig. 23. CPU clock cycles of generating different representations.

because of their similar semantic expressive power. EN, N3, and RDF/XML were generated and sent using sprintf and write functions as discussed above.

Figure 22 compares the lengths of EN, N3, and RDF/XML packets sent by the same sensor. EN has nearly the longest possible short packet, as 128 bit UUID is utilized, and all values are included in the short packet, but EntityId is not included. The length of an EN packet is around 53% of that of the corresponding N3 packet, and around 31% of that of the corresponding RDF/XML packet. The bandwidth required for transferring a packet is roughly proportional to its length. Memory usage is roughly proportional to the lengths of the packets as well.

We measured that the code to generate an EN packet takes about 21,000 clock cycles to execute on ATmega32 MCU. This is around 68% of generating the corresponding N3 packet and around 40% of generating the corresponding RDF/XML packet. The exact number of clock cycles depends on the C compiler and the code optimization settings.

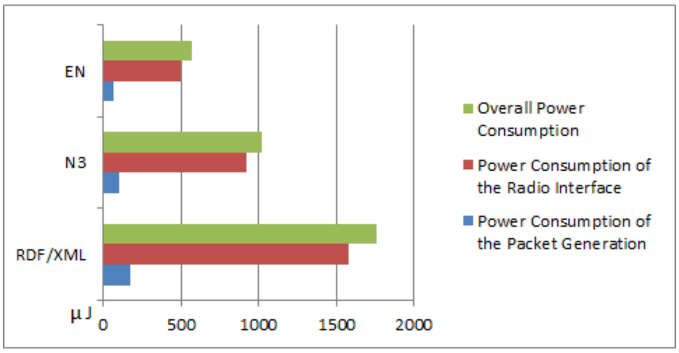


Fig. 24. Power consumption of generating different representations.

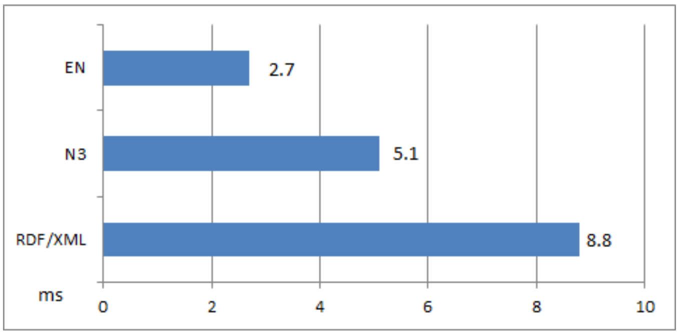


Fig. 25. Transmission time of different representations.

We focus on energy consumption of two operations, i.e. encoding/decoding data with the MCU and sending/receiving data packets with the transmission chip. The overall energy consumption thus consists of computing energy and communication energy:

$$E_{Overall} = E_{Comp} + E_{Comm}.$$

We do not consider energy consumption of sensing electronics, RAMs, and LEDs, because this consumption is independent of data formats. Computing energy is strongly dependent on MCU cycles and communication energy on message lengths:

$$E_{Overall} = f(Cycle) + f(Length).$$

The power consumption of Atmel's 8-bit ATmega32 MCU is 1.1mA at 1Mhz with 3V operating voltage. We assume that each instruction takes one clock cycle to execute, so an average of:

$$0.0011\text{A} \times 3\text{V} \times 0.000001\text{s} = 3.3\text{nJ}$$

energy is required for each executed instruction. Hence, the code for generating each EN packet consumes the energy:

$$21000 \times 3.3\text{nJ} = 69.3\mu\text{J}.$$

The radio interface of the sensor node is implemented using Bluegiga's WT12 Bluetooth module. The maximum power consumption of this Bluetooth module is 60 mA at 3V operating voltage. The data rate of WT12 Bluetooth module is 350 kbps when the Radio Frequency Communication (RFCOMM) protocol stack is used. Transmission time in Figure 25 stands for the time required to send the packet using the 350 kbps data rate. The transmission time for sending each EN packet is around 53% of that of the corresponding N3 packet, and around 31% of that of the corresponding RDF/XML packet.

One EN packet can be send in approximately 2.7 ms. This consumes energy approximately:

$$0.06\text{A} \times 3\text{V} \times 0.0027\text{s} = 500\mu\text{J}.$$

The energy consumption scales linearly with the payload size. The total energy consumption of each EN packet is approximately:

$$69.3\mu\text{J} + 500\mu\text{J} = 569.3\mu\text{J},$$

while the radio interface consumes 88% of the total energy consumption. When compared with N3 and RDF/XML, generating and sending an EN packet consumes energy around 56% of that of the corresponding N3 packet, and around 32% of that of the corresponding RDF/XML packet. There are low-power radio solutions which are superior to Bluetooth in energy consumption, but changing the radio does not affect these ratios. This experiment also verifies that EN packets can be successfully transformed into RDF and can be directly utilized by Jena reasoning engine.

5.5.2 General sensor system

We extended the elderly people reporter system to a general sensor system, consisting of two sensor nodes and a knowledge processing component on a PC. We measured the resource usage of encoding and decoding different data formats of the same data in this

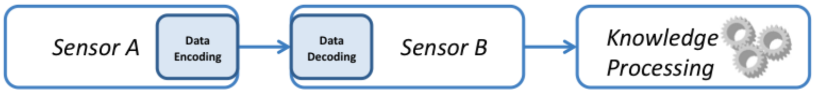


Fig. 26. Architecture of the general sensor system in our experiment.

sensor system. As shown in Figure 26, Sensor A encodes the different formats and sends them to Sensor B. Sensor B decodes the received data formats to formats that are easy to use by the knowledge-processing component. For instance, EN packets are converted to RDF/XML documents and EXI documents are converted to XML documents. RDF/XML, N3, SenML in XML and JSON, and JSON-LD are considered as easy-to-use formats, so Sensor B simply forwards them. All these packets are sent from Sensor B to a knowledge processing component on a PC and integrated into the OntoSensor [13] ontology. As a result, the data generated by sensor nodes is compatible with the knowledge system, which can reason additional knowledge and actions based on this data.

We evaluate EN against RDF/XML, N3, different formats of SenML and JSON-LD, using resource usage as criteria. In our experiment, we assume that encoded messages are delivered under similar circumstances, i.e. using the same protocol, along the same route. Comparing the selections of formats with previous section, SenML and JSON-LD are added, because they are emerging standards and have potential for large scale IoT systems.

Sensors in the system are similar to the one which we used in the well-being reporter prototype. Sensor A measures temperature, as well as acceleration and magnetic field in three dimensions. The MCU core is the same, but it has more Flash (256kB) and SRAM (128kB) memory. The Flash memory is needed for storing the program code, and SRAM is used to store program data, such as variables.

As we were interested in the payload only, we did not use any specific protocol, but simply created a message and sent it with Bluetooth. In this prototype, we did not use sprintf function to generate packets. Instead, we fill the data values in a function by handling a character array. SenML/EXI messages were encoded using the "Embeddable EXI implementation in C" software (EXIP). [3] The available EXIP software was utilized. However, there is still a possibility to optimize memory footprint with EXIP by leaving out the functionality not needed in this experiment. All messages were sent by calling the write function, which is associated with the low-power radio interface of the sensor

[3]http://exip.sourceforge.net/.

node. Examples of these data packets in RDF/XML, N3, SenML, and JSON-LD are presented in Section 2.2, and examples of EN packet are presented in Section 3.3. Here is a template for the short EN packet shown in the example of Section 3.3:

```
[http://ee.oulu.fi/o#TempAccMagSensor
 http://ee.oulu.fi/o#tempAccMagSensor01
 http://ee.oulu.fi/o#timeStamp ?1
 http://ee.oulu.fi/o#accX ?2
 http://ee.oulu.fi/o#accY ?3
 http://ee.oulu.fi/o#accZ ?4
 http://ee.oulu.fi/o#magX ?5
 http://ee.oulu.fi/o#magY ?6
 http://ee.oulu.fi/o#magZ ?7
 http://ee.oulu.fi/o#temp ?8]
```

The corresponding short EN packet is:

```
[urn:uuid:311b4e80-d9fd-11de-8a39-0800200c9a66 "2012-05-18T12:00:00"
"618" "319" "671" "123" "234" "345" "22.5"]
```

Figure 27 presents the packet lengths of different formats communicated between Sensor A and Sensor B. These packets contain the same data and semantic information. Short EN format is the most compact format, while SenML/EXI is the second shortest packet. Compact JSON-LD is the longest packet format, while the other formats produce somewhat shorter packets. The shortest format (Short EN) is about 28% of the longest format (Compact JSON-LD).

Figure 28 presents the amount of CPU cycles needed to generate the messages by Sensor A. It is clear that generating EXI messages requires much more computation. All other messages are produced by filling measurement values in a string.

As shown in Figure 29, generating SenML/EXI messages requires more energy (MCU Energy in the figure) than other alternatives, but transmission energy consumption for SenML/EXI is among the lowest ones. Energy consumption is calculated as presented on pages 117-118. When comparing overall energy consumption on Sensor A, the short EN format requires the least energy and N3 requires the second least. Generating short EN messages only consumes about 24% of generating SenML/EXI messages, which consumes the largest amount of energy.

On Sensor B, messages with RDF/XML, N3, SenML in XML and JSON, and JSON-LD data format are forwarded to the knowledge processing component without

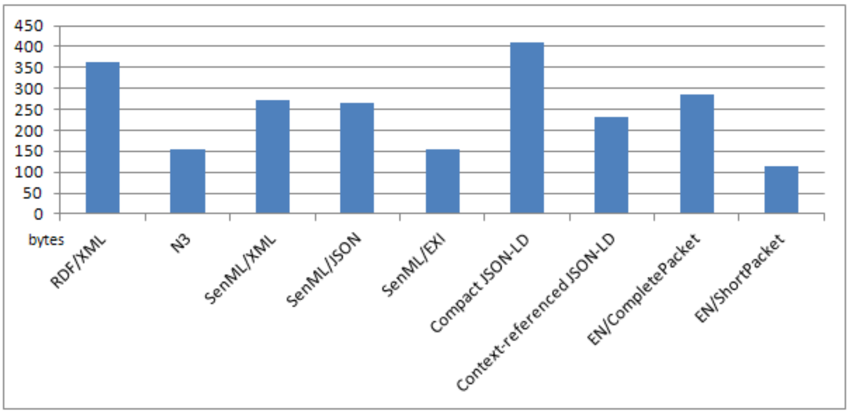


Fig. 27. Packet lengths of different data formats communicated between Sensor A and Sensor B.

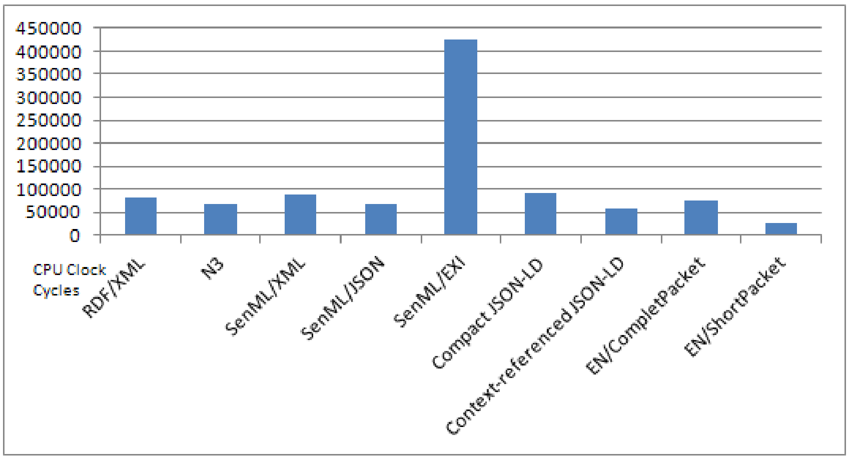


Fig. 28. CPU cycles consumed by Sensor A.

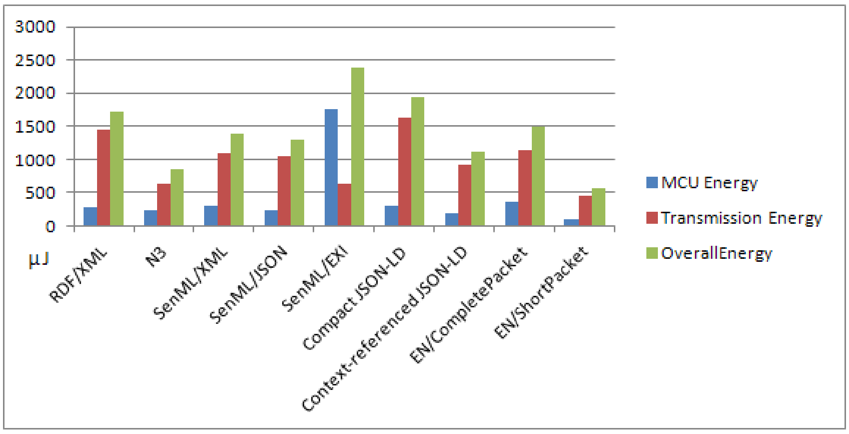


Fig. 29. Energy consumption on Sensor A.

any decoding. Three data formats, i.e. SenML/EXI, EN complete packet, and EN short packet, require decoding. EN packets are converted to RDF/XML and EXI packets are converted to XML documents. To test how small energy consumption can be achieved, we implemented a function on Sensor B that transforms short EN packets directly to RDF. With this function, the energy consumption of a short packet on Sensor B is only about 38% of the corresponding complete packet. This case is referred in the following text and figures as "EN/ShortPacket".

EN packets are transformed to RDF/XML, which can directly be utilized by the knowledge processing component. SenML/EXI packets are transformed to XML. Figure 30 presents how much transmission energy is needed for reception and MCU energy needed for decoding operation on Sensor B. Transmission energy consumption equals to overall energy consumption on Sensor B for those data formats that only need forwarding. Figure 31 shows the overall energy consumption for both sensors, including data sending, receiving, encoding, and decoding operations. SenML/EXI processing generates XML and requires additional processing for producing RDF/XML. We can also conclude that the short EN packets require the least energy to produce data in RDF/XML.

The overall MCU energy consumption for encoding and decoding data depends on the corresponding algorithms, and EXI is clearly the most complex one. The transmission energy consumption of different formats scales linearly with the payload size. Short EN format requires the smallest amount of transmission energy, while SenML/EXI requires the second least amount of transmission energy. However, SenML/EXI requires quite a

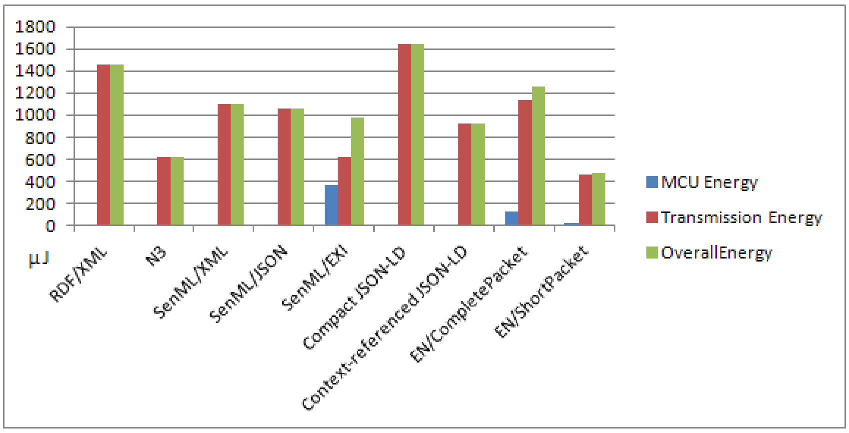


Fig. 30. Energy consumption for the decoding operation on Sensor B.

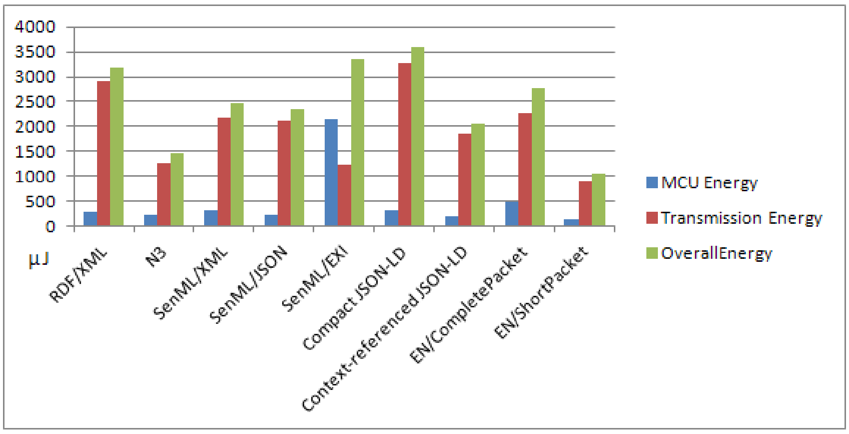


Fig. 31. Overall energy consumption of the sensor system.

lot computation so the total energy consumption is the second largest. SenML/XML and SenML/JSON require a similar amount of energy in our experiment. Compact JSON-LD format consumes the largest amount of overall energy because of its longest messages. RDF/XML messages are second longest. N3 requires less energy than others but more than Short EN. Short EN packets consume the least energy among these formats, 88% of the second best alternative (N3), and 29% of the worst one (Compact JSON-LD).

As a conclusion, short EN is the best data format, when energy consumption has to be minimized, N3 being the second best. For those systems with limited communication resources but not limited with computing resources, Short EN is the best and SenML/EXI the second best. However, for systems with limited computing resources, SenML/EXI is not an alternative. If only Sensor A is required to minimize energy consumption, short EN and N3 are the best alternatives. If only Sensor B is required to minimize energy consumption, for example, it is a very simple gateway without any processing capabilities, then short EN would be the best option. This experiment verifies that EN packets can be successfully transformed into RDF and can be directly utilized by Jena reasoning engine.

5.6 Ambient social interactions

Ambient social interactions can be defined as human like interactions in AmI environments [134]. Such interactions exist not only between AmI systems and humans, but also among humans. To enable ambient social interactions, pervasive systems need to aggregate information from the social information of users, available services, and the physical environment. This aggregation can improve user experience and enable novel functionality for ambient social applications. Building knowledge-based systems is a convenient way to enable ambient social interactions. Heterogeneous information, like user profiles, shared social information (e.g. common goals) and real time sensor measurements (e.g. location) can be integrated into knowledge-based systems using Semantic Web technologies.

We identify ambient social interactions as an important direction of research in pervasive computing. However, the challenges presented earlier, such as minimizing the resource usage, are valid for ambient social interactions as well. EN can be adopted to tackle the challenges of utilizing Semantic Web technologies for implementing intelligent applications for ambient social interactions. Hence, we implement a larger prototype to verify EN in this application area. We suggest a knowledge-based framework for ambient social interactions. EN is utilized in this framework for connecting all information together, including sensor data, social content, and Semantic Web-based inference. In order to verify our framework, we implement an application that reasons over social content produced by users and real-time location data to enable interactions among multiple users. We demonstrate a location-based reminder service, which presents reminders and maps to mobile users based on their needs.

5.6.1 General framework

Figure 32 presents a general framework to build knowledge-based systems for ambient social interactions. Information from heterogeneous sources, such as social information, physical environment, and available services, is aggregated to knowledge-based systems. Social information is not only for individuals, but also for communities. Communities are groups of people with common roles, interests or tasks. Social information can be extracted from profiles, subscriptions, and social networks. The physical environment offers information about the physical surroundings of users, such as location, time, noise level, and temperature. Available services are other important sources for knowledgebased systems. Services provide more complex and processed data, for instance from networked sensors.

Knowledge-based systems are key enablers for ambient social intelligence. As described earlier, they include knowledge bases and reasoners. Knowledge bases store formal models of knowledge, and reasoners infer new logical relations from the knowledge bases. Moreover, we utilize Semantic Web technologies and ontology-based methods. As shown in Figure 32, users and applications are considered as actors in our framework. They consume deduced results produced by knowledge-based systems. However, users can also act as sources of social information by entering information with their mobile devices. Moreover, observations can be performed by sensors to monitor users' activities. Knowledge-based systems enable the aggregation of user specific social information, sensor produced data, and all other information. That is, knowledge-based systems establish the common meaning of entities and their interactions. Interacting entities are able to share not only data, but also semantics for ambient social intelligence. Knowledge-based systems provide mechanisms to identify implicit logical connections and enable semantic interoperability straightforwardly and unambiguously.

However, knowledge-based systems require a lot of computing resources from the hosting devices. A knowledge base can consume a large amount of memory, which is not

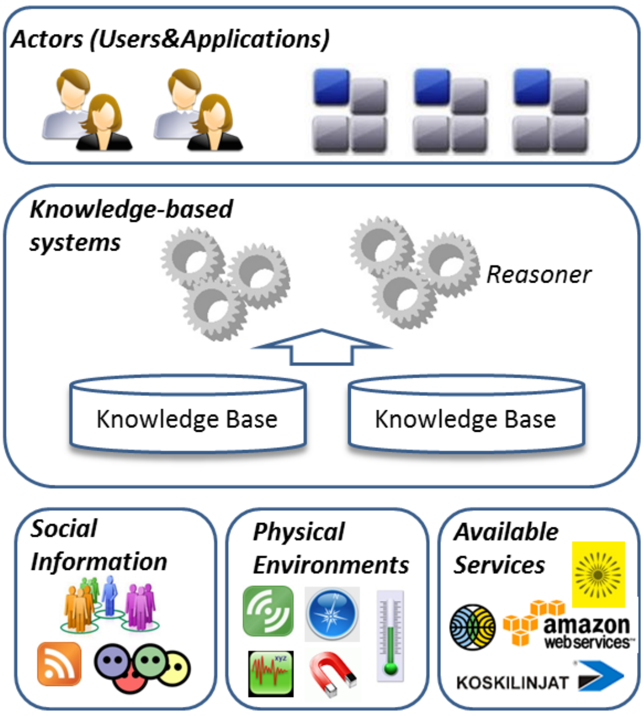


Fig. 32. Framework for knowledge-based systems for ambient social interactions.

available at mobile devices. Also reasoning is a resource consuming process and only a limited set of mobile devices currently available can perform this operation continuously. Hence, it is a common practice to build big knowledge bases and complex reasoning processes on the server side. However, this introduces communication overload for the ambient system. A lightweight communication mechanism is needed to decrease communication overload.

5.6.2 EN for ambient social interactions

Aggregating heterogeneous information from different sources requires a general, powerful, and flexible representation. EN fulfills these requirements and is suggested to interconnect all information sources and consumers. As EN is compatible with Semantic Web models, such as RDF and OWL 2, it can be utilized together with popular social information, like Friend of A Friend (FOAF) ontology and RDF Site Summary

(RSS) social syndication. Moreover, EN is lightweight and can hence decrease the communication load.

We demonstrate two scenarios. In the first scenario, two persons share a task of buying pizza at noon, and the one who is closer to the pizza restaurant (that is the Point of Interests (PoI)) will get an alert with a map and a suggestion to visit the restaurant. In the second scenario, a wife specifies a shared task with her husband to pick up their kids from school (that is the PoI). The couple has decided to pick up their kids as early as possible. Thus, at the predefined time, the one who is closer to the school gets a reminding alert with a map on his or her mobile phone and a suggestion to pick up their kids.

Here are two examples of utilizing EN packets in our scenarios. In the first packet, Alice is sharing a task with Bob, which is to pick someone up at the University of Oulu. Hence, this is social information:

```
[http://ee.oulu.fi/o#SharedTask
 http://ee.oulu.fi/o#pickupSharedTask101
 http://ee.oulu.fi/o#ownerID  "Alice"
 http://ee.oulu.fi/o#peerID  "Bob"
 http://ee.oulu.fi/o#interestingPlace
 http://ee.oulu.fi/o#universityofOulu]
```

The following packet represents the location sensor measurement of Alice. This is physical environment information:

```
[http://ee.oulu.fi/o#LocationSensor
 http://ee.oulu.fi/o#locaSensor767
 http://ee.oulu.fi/o#ownerID  "Alice"
 http://ee.oulu.fi/o#longitude  "25.468"
 http://ee.oulu.fi/o#latitude "65.058"]
```

The first packet about pickupSharedTask101 can be transformed to the upper RDF graph of Figure 33. The graph includes three resources: the task owner, the peer, and the place. The second packet about the location sensor can be transformed to the lower RDF graph of Figure 33.

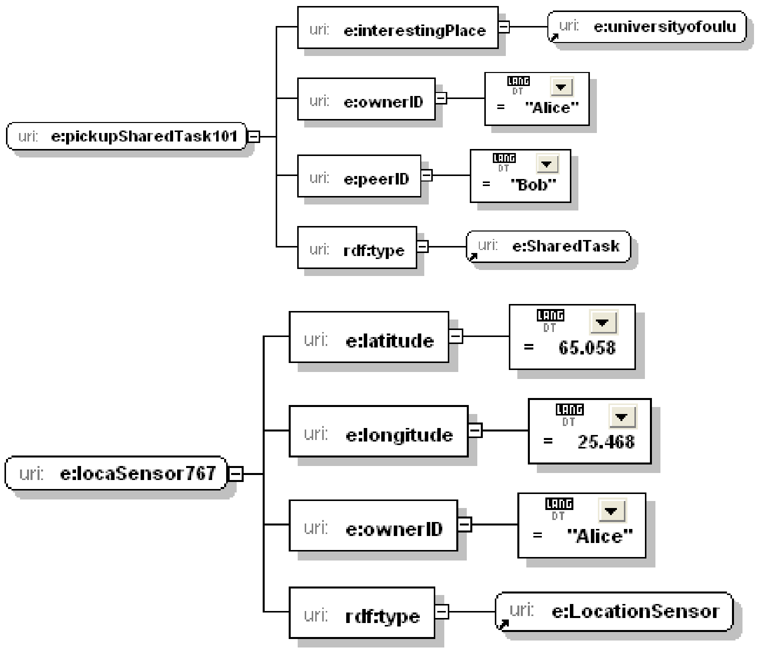


Fig. 33. **RDF graphs for task and location entities.**

With templates and prefixes, we can shorten these two examples as follows:

```
[urn:uuid:76eac2 "Alice" "Bob" EE#universityofOulu]

[urn:uuid:539ea2 "Alice" "25.468" "65.058"]
```

EN is also employed to represent other data communicated between the server and the clients, including generating new tasks, deleting tasks, and browsing information of tasks, users and positions. These packets can be transformed into RDF statements, and added among the facts in the ontology. Figure 35 and Figure 36 present part of this ontology.

5.6.3 Event map

To verify the framework and EN, we have implemented an ambient social application, Event Map, which provides reminders for mobile users. The application renders maps to mobile devices' displays. PoIs, and the locations of the users are shown on the map.

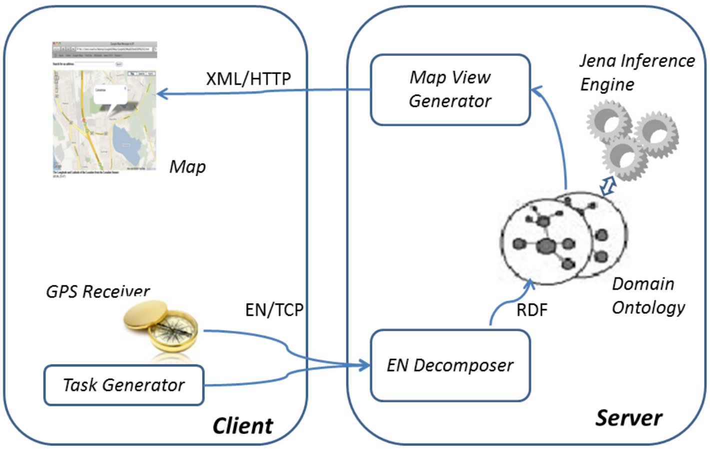


Fig. 34. Event Map Application.

The shared tasks are presented as well. The application is normally executed in the background and maps with reminders come to the foreground when there is important information to be shown.

Figure 34 presents the event map application, which consists of two main parts: a client and a server. The client consists of user interface showing the map, a task generator, and a GPS receiver. With the task generator, users can specify their preferences and settings: which reminder to show, when to show them, and with whom to share tasks. Both location data and specified tasks are sent as short EN packets to the server. Examples of these packets are presented in Section 5.6.2. The server consists of an EN decomposer, a domain ontology, a reasoning engine, and a map view generator. This experiment verifies that the EN decomposer transforms EN packets to RDF statements. These RDF triples are added into the domain ontology and trigger the inference engine.

The domain ontology is a small knowledge base. We define an ontology in OWL, which includes predefined PoIs (such as restaurants and schools), user profiles, shared tasks, and other facts relevant to the system. The inference engine is a rule-based system which utilizes the ontology and pre-designed rules to constantly check if any task defined by any user can be executed and triggers corresponding rules. When a user is to be reminded about a task, the map view generator renders a map according to the task settings. This map is sent to a specific person at a specified time. Figures 35 and 36 show the structure of knowledge about shared tasks and users in the domain ontology we developed. Figure 35 presents knowledge about SharedTasks, including the target places of tasks, the persons a task being shared with, and the distance from current places to target places and shared members. Figure 36 presents knowledge about users, including the ID of users, the location of users, the shared tasks of users, and whether the user is a member of a certain group of sharing tasks. The RDF statements, as shown in the RDF graph of Figure 33 (top), are added as instances of the shared task. The RDF statements, as shown in the RDF graph of Figure 33 (bottom), are added as instances of the user-related knowledge. Table 8 shows some rules defined for our application scenarios. The left column describes the rule in natural language and the right column presents the Jena rules, applied to the ontology.

The prototype fully implements the framework presented in Figure 32. We utilize user locations and shared tasks as suppliers to the knowledge-based system. As can be seen from Figure 34, the knowledge-based system in our implementation consists of domain ontology and inference engine components. The logic flow of the application is the following: After a user starts the application and logs in, he or she can create his or her own tasks. These task descriptions are then sent to the server. After tasks have been created, the application starts to run in the background, allowing users to perform their daily routine. The application appears at the foreground only when the execution time of a task is reached. Task descriptions are represented as RDF statements, and GPS sensors continuously send location data as EN packets. EN data trigger reasoner to reason with predefined knowledge and rules. New knowledge about reminders and map information is generated by the reasoner, stored in the ontology, and sent to the user when the execution time of a task is approaching.

We implement Event Map client application on Nokia N95 mobile phones as a Java ME application. The server is implemented with Java and the inference engine is implemented with Jena [40]. TBox and ABox knowledge is stored as RDF/XML files. TCP is utilized for connecting client and server, and EN packets are payload of TCP.

We tested the system in an open access Wi-Fi network. Two groups of users specified both tasks of buying pizza and picking up kids, and then shared them with each other. Then, they were asked to drive in the central area of Oulu. While driving, they received maps and reminders. The concepts of the tasks, points of interests and the logic of the application were clear to the users and they could easily manage tasks by themselves. The field test also demonstrated that users are eager to share their location and task information “as long as I [Test user] am in control what, when and whom I share it with”. The overall application was considered as a nice system which actually could be used as “one part of a calendar application”. In the real-world test, all predefined tasks

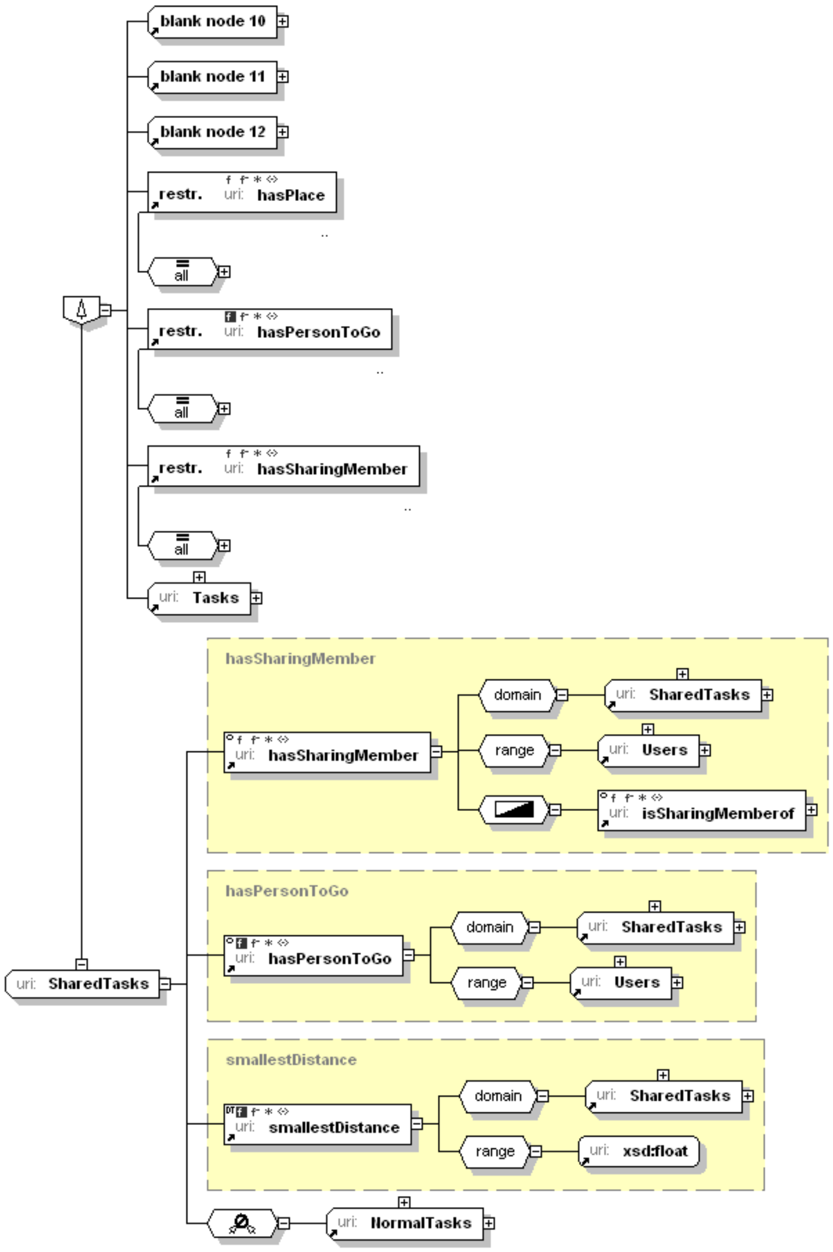


Fig. 35. Knowledge about Shared Tasks.

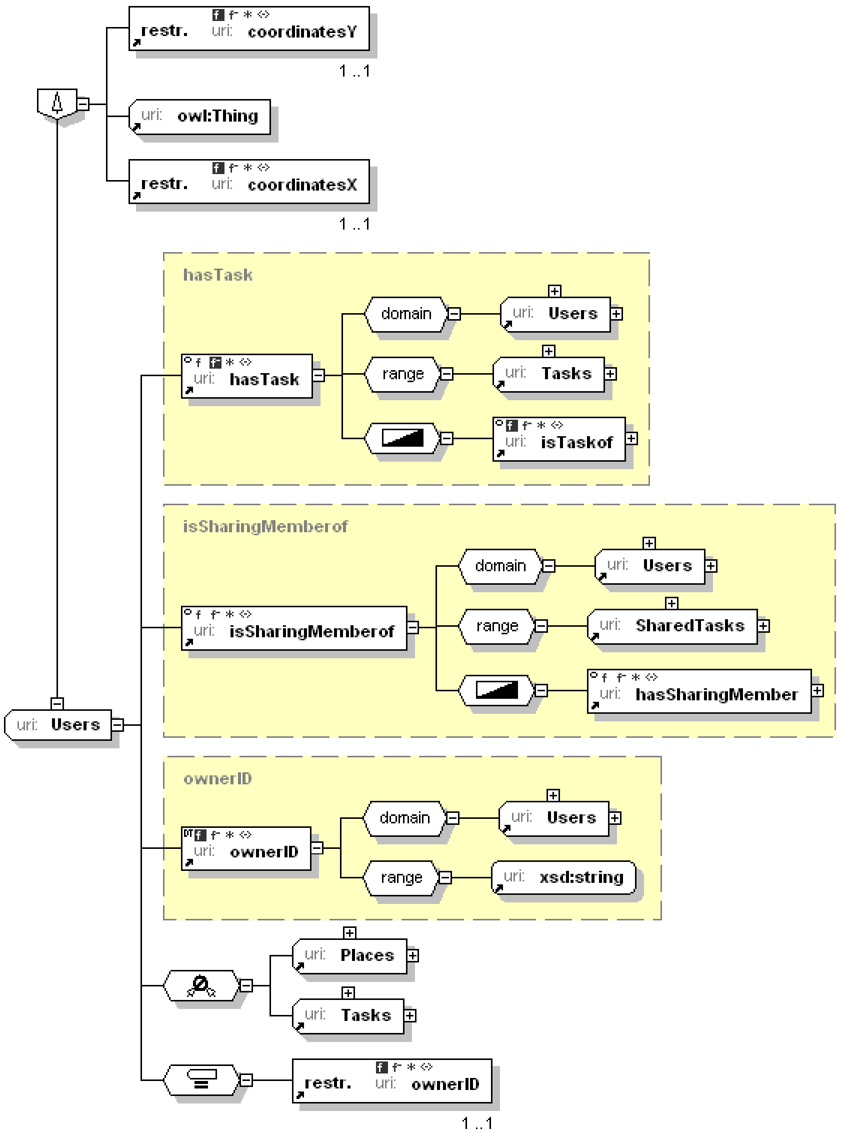


Fig. 36. Knowledge about Users.

were inferred correctly, but only around 80% of the predefined tasks were shown in time and correctly due to Internet connection problems.

In all tested situations, the system behaved as expected; hence, we conclude that our rule set is adequate for the given task. The left picture in Figure 37 is a screen shot from an emulator, which demonstrates the picking up kids scenario. A red mark at the upper left corner specifies the target location and a blue mark at the lower right corner specifies the current location of the user. Also a text reminder of this task is shown to the user. The picture at the right captures the real user interaction with the application during the field test.

Table 9. Social interaction rules for event map.

Rule description	Reasoning rule
This rule checks for the shared task (?sharedTask) whether Bob (?user) is closer to the point of interest (?place).	[rule1_check_closest_user: (?sharedTask rdf:type sw:SharedTasks) (?sharedTask sw:hasPersonToGo ?person) (?sharedTask sw:smallestDistance ?smallestDistance) (?sharedTask sw:hasSharingMember ?user) (?user sw:X ?userX)(?user sw:Y ?userY) (?sharedTask sw:hasPlace ?place) (?place sw:X ?placeX)(?place sw:Y ?placeY) countDistance(?userX, ?userY, ?placeX, ?placeY, ?distance) lessThan(?distance, ?smallestDistance) noValue(?smallestDistance sw:firedFor2 ?sharedTask) —> remove(1,2) (?sharedTask sw:smallestDistance ?distance) (?sharedTask sw:hasPersonToGo ?user) (?smallestDistance sw:firedFor2 ?sharedTask) hide(sw:firedFor2)]

Rule description	Reasoning rule
This rule checks if the current time (?currentTime) is within the shared task validity time (?validity), if it is then the task is ready to be executed, hence its transmission property becomes "transmission_succeded". Also, the new time for task execution is set according to task repetition property.	[rule2_set_transmission_for_shared_task: (?sharedTask rdf:type sw:SharedTasks) (?sharedTask sw:time ?time) (?sharedTask sw:transmission 'no_transmission') now(?currentTime) validityTime(?time, ?validity) le(?time, ?currentTime) ge(?validity, ?currentTime) (?sharedTask sw:repetition ?repetition) setTime(?repetition, ?time, ?newtime) (?sharedTask sw:hasPlace ?place) (?place sw:X ?placeX)(?place sw:Y ?placeY) (?sharedTask sw:hasPersonToGo ?user) (?user sw:X ?userX)(?user sw:Y ?userY) noValue(?user sw:firedFor3 ?task) —> remove(1,2) (?sharedTask sw:time ?newtime) (?sharedTask sw:transmission 'transmission_succeded') (?user sw:firedFor3 ?sharedTask) hide(sw:firedFor3)]
This rule gathers all necessary information for the shared task (?sharedTask) which is ready to be executed (transmission property equals "transmission_succeded"). This information is obtained to generate the map with the reminder.	[rule3_send_map_for_shared_task: (?sharedTask rdf:type sw:SharedTasks) (?sharedTask sw:time ?time) (?sharedTask sw:transmission 'transmission_succeded') (?sharedTask sw:repetition ?repetition) (?sharedTask sw:hasPlace ?place) (?place sw:X ?placeX)(?place sw:Y ?placeY) (?sharedTask sw:hasPersonToGo ?user) (?user sw:X ?userX)(?user sw:Y ?userY) (?sharedTask sw:comment ?comment) —> mapInfo(?sharedTask,?comment,?user,?userX, ?userY,?place,?placeX,?placeY,?repetition,?time)]

Next, we analyze the performance of EN in our implementation. There are six different kinds of short EN packets in total in this implementation. GPS data packets are sent to a server with a frequency of one packet per second, and other packets are sent on request. Different short packets have slightly different compression ratios to corresponding

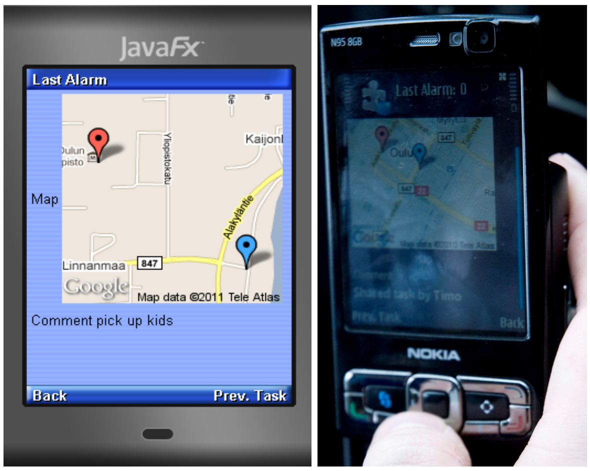


Fig. 37. Application screenshots.

N3 and RDF/XML packets. During this test, two users shared ten tasks during one hour. The short EN packet format contains about 49% of the characters of N3 packets and about 23% of those of the corresponding RDF/XML. The bandwidth and energy required for transferring a packet are roughly proportional to its length. Compared with other knowledge serializations, EN packets can be very compact, hence, the communication overhead over server and mobile client communication links can be significantly reduced. This enables building knowledge-based systems on the server side with small communication overhead.

5.6.4 Summary

In this section, we presented our work towards building knowledge-based systems to enable ambient social interactions. We discussed a general framework to enable knowledge-based systems in ambient social interactions and utilization of EN in this framework. EN enables semantics for social and sensory data. The implemented scenario demonstrates the usefulness of our framework. The implementation also verifies that EN packets can be transformed to RDF statements and can be added into a knowledge base to trigger the inference engine. EN and Semantic Web technologies offer promising solutions for heterogeneous information integration.

By their nature, ambient social applications must be available for users anytime and anywhere. What is more, they are expected to be lightweight, allow normal mobile phone routine, and respond fast. Knowledge-based systems facilitate different social interactions, even though it is a challenge to utilize the servers for mobile applications due to the latency the communication introduces. EN is a lightweight representation to decrease the amount of transferred data.

Building knowledge-based systems for social applications brings a clear separation of the functionality between system components. That is, an advantage of this approach is that we can isolate the application logic from other components of the system. The application logic is mainly performed by rules which can be easily modified and extended on the fly, without a recompilation of the whole system. Hence, the functionality of the system can be easily changed.

5.7 Two-Layer inference framework

In this section, we apply EN to a two-layer inference framework to enable Semantic Web technology-based intelligent functionality for pervasive environments. In addition to connecting data sources to knowledge-based systems, this framework supports distributed inference. When compared with the framework of ambient social interactions, EN is not utilized only to decrease the amount of communication, but also to communicate schema knowledge between devices, and this enables distributed reasoning.

This section presents the design of this framework and illustrates its usage by a use case. The framework enables two-layer inference according to the capabilities of different devices. In this framework, resource-constrained devices deliver measurement data to a server and mobile devices. Ontology-enabled mobile devices (see Chapter 4) perform simple inference tasks (low-level inference) based on the data from sensors. The server supports full capabilities of semantic technologies, such as advanced inference, context extraction and reuse, and coordinating the devices. Low level inference engines can be deployed on mobile devices to perform RDF inference and limited OWL-based inference. When inference is performed on ontology-enabled mobile devices, there is no need to transfer sensory data and user profiles to the server side. That is, the framework facilitates minimizing the amount of communication.

5.7.1 Design

Figure 38 shows a concept diagram of the two-layer inference, which includes an ontology and a rule set. The low-level inference makes use of ABox level and small

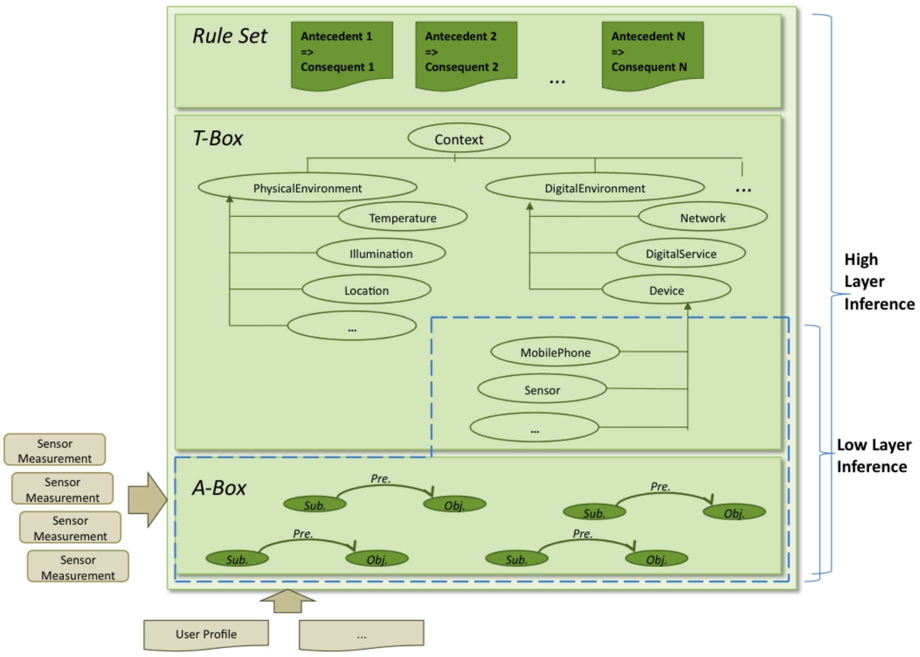


Fig. 38. Two-layer inference framework.

parts from TBox level. These parts include knowledge directly relating to the possible RDF statements in ontology-enabled mobile devices. It's possible to access additional knowledge from ontology when needed. The usage of TBox level knowledge results in more capable low-level inference than utilizing only RDF, but the low level does not support any rule-based inference. The high level inference engine can support the full capabilities of ontological inference and rule-based inference. Complex intelligent functionality, like multiple user interactions and device interoperability, can be realized at the high level.

Figure 39 illustrates how two-layer inference can be deployed at devices in pervasive environments. A server, ontology-enabled mobile devices, and resource-constrained sensor nodes form the two-layer inference system. A knowledge-based system manages a domain ontology at the server, which hosts all advanced knowledge-based applications. All Semantic Web functionalities, like reasoning, publishing semantic data and semantic matching, can be implemented at the server side. Mobile devices share parts of the domain ontology from the server and perform low-level inference based on shared ontology and RDF statements. Sensors have limited resources and can only send data

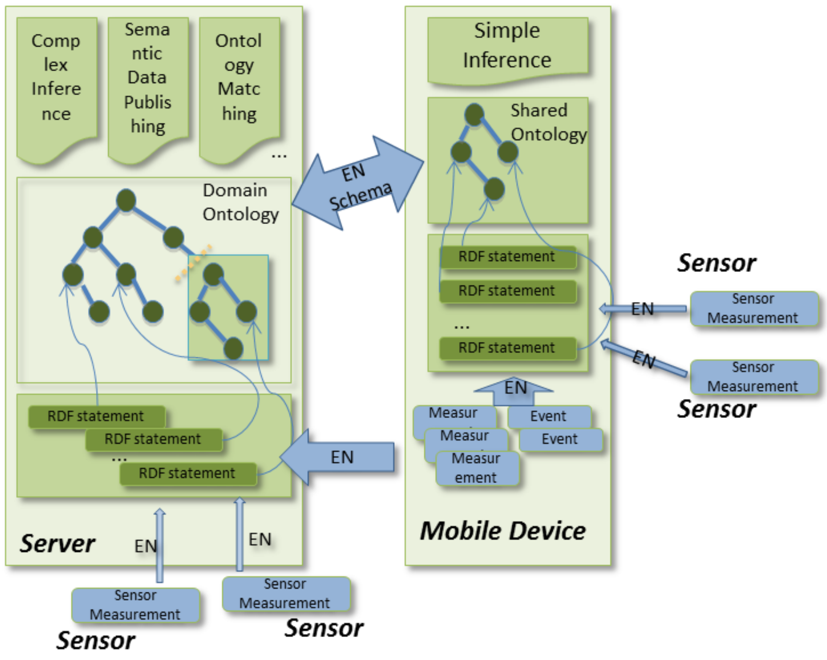


Fig. 39. Framework of two-layer inference.

to the mobile devices. The data can also be forwarded to the server, when necessary. Servers and mobile devices can send commands to some sensors as well.

EN and EN Schema play an important role in the communications. The knowledge-based system on the server can compose EN Schema packets to transfer parts of the ontology to the ontology-enabled mobile devices. Then, these devices decompose EN Schema packets and get the knowledge they need in their applications, depending on applications and scenarios. The ontology-enabled mobile devices can also compose EN Schema packets and transfer them to the knowledge-based system on the server. The resource-constrained sensors cannot handle any ontology or reasoner, but still, they can compose EN packets based on their measurements and transfer them to the ontology-enabled mobile devices or the server. The knowledge-based system can reason on the transferred data and offer deduced results for these resource-constrained devices.

5.7.2 Use case

In this section, we extend the children pickup scenario (presented in section 5.6.2) as a use case to illustrate the usability of this framework. The extended scenario is the following:

A couple is driving home from different working places after working, and one of them needs to pick up their children from the kindergarten. The couple has decided to pick up their children as early as possible. When a parent has left the workplace, she/he is reminded to pick up the children if she/he is expected to arrive kindergarten with less time. At the same time, another reminder will be shown to advise the spouse to drive home directly.

This scenario includes Radio-frequency Identification (RFID) technology: when a parent gets off work, he/she touches with his phone an RFID tag nearby the exit of the building to record the time he leaves. Devices for this scenario include: mobile phones with GPS receivers for children, a server and ontology-enabled mobile devices with RFID readers, GPS receivers and map applications for parents. The following example demonstrates how data is passed among the main components, and how simple inference and complex inference is achieved.

An EN packet like this records the time a parent leaves the work place:

```
[urn:uuid:739ae5 "ExitB" "JohnSmith" "2010-03-25T16:21:40"]
```

Here, is the corresponding RDF/XML statement of this EN packet.

```
<e:RfidTag rdf:ID="rfid21">
  <e:tagType>ExitPlace</e:tagType>
  <e:personTouching>JohnSmith</e:personTouching>
  <e:timeTouched>2010-03-25T16:21:40</e:timeTouched>
</e:RfidTag>
```

Inference can be performed in the mobile device based on this RDF statement and an ontology. In the following SWRL rule, a person who leaves the building after 15:00 (3 p.m.) is assumed to go home:

```
Implies(Antecedent timeTouched(?person, ?time1)
                   swrlb:greaterThan(?time1, 15:00:00)
Consequent(gohome(?person) )
        )
```

The embedded reasoner on the ontology-enabled device deduces that John Smith is going home, and starts to send GPS data to the server. The GPS packet with the user's name, longitude and latitude data in EN syntax is the following:

```
[urn:uuid:7bcf39 "JohnSmith" "25.47" "65.06"]
```

This packet can also be transformed into an RDF statement. In this scenario, high level inference can be applied for interaction among multiple devices. For example, mobile map applications can estimate the driving distances of the parents. Moreover, high level inference can decide who should visit the kindergarten by using the following SWRL rule:

```
Implies(Antecedent drivingdistance(?person1, ?dis1)
                   drivingdistance(?person2, ?dis2)
                   swrlb:greaterThan(?dis1, ?dis2)
Consequent(showReminder(?person1 "Please visit kindergarten
to pick up your kids")
           showReminder(?person2 "Please go home directly")) )
```

The above example shows the utilization of the inference engine at two levels. More complex reasoning can be performed based on the data from more sensors and users. Three kinds of devices with different capabilities are considered: resource-constrained sensors, ontology-enabled mobile terminals and servers. The two-layer inference framework can be deployed on ontology-enabled terminals and servers. Ontology-enabled terminals support fast and local reasoning react the changing environment, while servers afford complex inference and other resource-consuming semantic functionalities.

5.8 Summary

In this chapter, we verified with simulation, data sets and prototypes that EN and EN Schema are expressive ontology-based representations amenable to distributed systems. The semantic expressive power is comparable with most standardized knowledge representations. Meanwhile, it is so lightweight that small resource-constrained device can afford it; this is a unique feature which makes EN an ideal solution for building pervasive systems. Our sensor system results show that EN can have shorter packets and requires less processing resources and energy for encoding and decoding packets.

We developed larger prototypes that utilized most features of EN and EN Schema. These features include modelling basic RDF and OWL 1.1. EN features which are not verified in these experiments include: arrays and enumeration. These prototypes verified that EN is an ideal candidate as an underlying representation for different information sources. It is expressive for representing information from social and physical contexts. Moreover, the two layer inference framework presents transferring TBox knowledge with EN schema to ontology-enabled devices and harnessing the computing capabilities of the pervasive system.

6 Discussion

In this dissertation, we addressed the topic of data and knowledge exchange, which is one key issue when knowledge-based systems are built for pervasive environments. This chapter summarizes the dissertation by collecting results from the previous chapters and analyzing how research objectives were achieved. We also summarize the contribution of this dissertation, discuss open issues, and suggest future work.

6.1 General analysis

One of the key issues of data and knowledge exchange in pervasive systems is a syntax for encoding knowledge that fulfills the requirements of pervasive environments. EN and EN Schema are lightweight, powerful, and human-readable representations for data and knowledge exchange. They have been designed especially for pervasive environments and applications, but can be utilized in the wide application area of distributed systems. EN is an affordable syntax for resource-constrained sensors and supports Semantic Web models at the same time. Devices can compose EN packets with small computing resources, and still the data can be transferred into knowledge models. EN can be the payload of various underlying protocols, such as HTTP, TCP, and sensor network protocols; hence, information from different sources can be expressed and transferred. These features make EN a flexible representation for pervasive environments. Moreover, EN is designed as a lightweight representation for well-known knowledge models, which enables distribution of knowledge. Ontology knowledge fragments produced by any person and any device in everyday environments can be integrated into a knowledge base unambiguously. Moreover, EN Schema facilitates moving knowledge based processing from servers to mobile devices. Finally, EN and EN Schema are text-based representations that are completely architecture and language independent.

Among general representation categories used for representing context in pervasive environments, such as key-value pairs, ontology-based representations, and case based representations [28], EN and EN Schema can be classified as ontology-based representations, because of their compatibility with well-known knowledge models. EN and EN Schema are unique in their combination of expressiveness and compactness.

EN is mainly targeted to resource-constrained devices and our main emphasis is to transfer simple data structures efficiently. We deliberatively kept EN simple to minimize

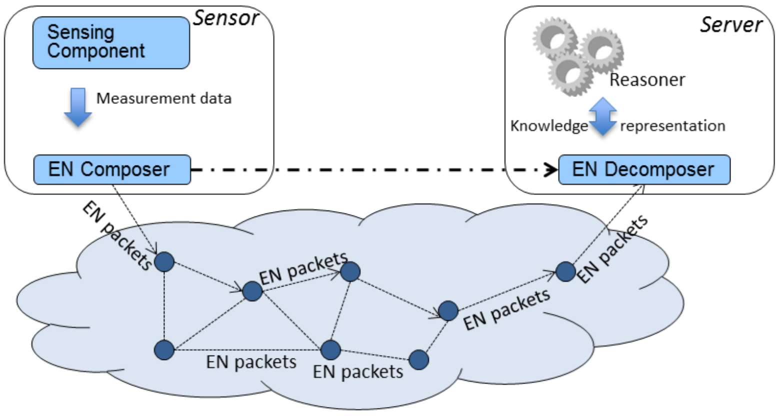


Fig. 40. The role of EN in distributed systems.

the amount of resources required from devices for processing data. However, in spite of its simplicity, EN can be utilized to send different types of information between devices. Complex data structures can be built from simple ones.

The role of EN in distributed systems is illustrated in Figure 40, which presents communication between a sensor and a server. Sensors and servers need to have an EN composer and a decomposer for utilizing EN. A composer produces EN packets from sensor measurements. A decomposer transforms EN packets into a knowledge representation that can be utilized by the knowledge processing component, for example, a reasoner at the server side.

EN has the potential to connect resource-constrained sensors to Semantic Web. EN plays a role in the “Knowledge hierarchy” suggested by Barnaghi, et al. [108]. Resource-constrained sensors could form a whole new periphery for Semantic Web providing a significant amount of real-time measurement data. Figure 41 presents the role of EN in bridging the gap between Semantic Web and networked sensors. Networked sensors can compose either complete or short EN packets depending on their capabilities. All these packets can be transformed into Semantic Web representations at the gateways. Knowledge bases and Semantic Web applications can access sensor data and knowledge straightforwardly. Moreover, some devices in sensor networks can perform the commands that the Semantic Web applications produce.

EN has short packets that can be composed by simply concatenating UUID and values. There is no need to handle any complex algorithms, nor predefined shorthands when composing short packets. The packet format is compact, and it has a simple

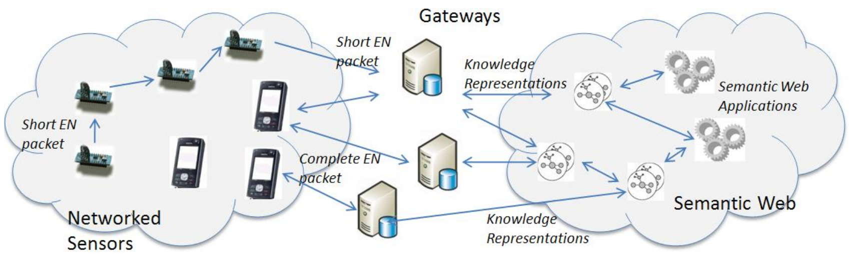


Fig. 41. EN bridges the gap between Semantic Web and networked sensors.

grammar. On the other hand, EN has a complete format that can be transformed into other knowledge representations in an unambiguous fashion. This enables Semantic Web-based intelligent functionality, such as inference over context data and collaboration among devices.

Utilization of unique identifiers (IRIs and UUIDs) makes EN a scalable solution. IRIs and UUID-based approach is well suited to distributed systems. IRIs describe entities in pervasive environments unambiguously. UUIDs with different lengths can be selected based on the size of sensing system: most pervasive systems only need 16 bits or 24 bits UUIDs, while 128 bits UUIDs can identify short messages and templates for an Internet-scale system. Meanwhile, short UUIDs can be mapped to full UUIDs using look-up tables. This facilities the integration of small scale systems to larger ones.

We offer flexibility for designers to optimize their systems. EN supports UUIDs with different lengths, short packets with different variable items, alternatives for negotiating packets, and different methods for packet chaining on gateways. Short packets can be used over simple and unreliable communication links and offer a large variety of possibilities on how much information is included in packets and how much in templates. The calculations we presented in Section 5.2 can be utilized to minimize the cost of a sensing system. That is, bandwidth, memory, and CPU requirements of a system can be balanced by tuning the amount of variable items in short packets.

6.2 Revisiting the research objectives

In this section, we return to the research objectives introduced in Section 1.2 and consider how this research meets these objectives.

The first objective was lightweightness, that is, to design a representation that can be utilized by any resource-constrained device. For achieving this objective, EN was designed to produce short packets which in their simplest form are only seventeen bytes long, and still have good expressive power. Our evaluations show that EN is much shorter than corresponding RDF/XML and N3 representations and requires lesser resources for encoding and decoding. With this representation, any simple sensor in pervasive environments should be able to connect with knowledge-based systems using minimal computation power and energy, and to deliver packets to other system components in a resource-efficient manner.

The second design objective was interoperability, that is, to design a representation that facilitates transferring different levels of knowledge. For achieving this objective, we designed EN and EN Schema for transferring knowledge at different levels. Both data and knowledge can be transferred utilizing EN and EN Schema. This makes EN and EN Schema practical representations when intelligent functions are based on OWL 2.

The third design objective was generality, that is, to design a representation that is feasible for general distributed systems and can be utilized by different applications and systems. For achieving this objective, we utilized unique identifiers as the information should always be identifiable. This is important for a general scalable system when all the possible usages of the information cannot be specified in advance. Moreover, EN and EN Schema can be utilized to transfer different types of data and knowledge, they supports a good selection of data structures, and they have sufficient expressive power. We verified EN an EN Schema with several prototypes; and transferred different types of information with EN as payload. Hence, we can state that EN is a general representation.

All in all, we achieved the objectives by developing a lightweight and general data and knowledge representation that advances interoperability.

6.3 Contributions

Our contribution is threefold. ***First, we design Entity Notation (EN), a lightweight representation that can be handled by resource-constrained sensors, transferred over low-power communication links with limited bandwidth, and transformed into knowledge representations in a straightforward fashion.***

Several alternatives were presented in Chapter 2 and most of them were evaluated in Chapter 5. When compared with EN, other Semantic Web syntaxes require more computing and communication resources, and other formats and approaches do not offer similar transformations to Semantic Web knowledge models.

Second, we present EN Schema for knowledge exchange. The constructs of EN Schema extend EN into a uniform representation language for different Semantic Web languages for pervasive environments.

When considering the other knowledge representations presented in Chapter 2, they do not provide a similar combination of short messages and expressive power.

Third, we verify that EN is a practical solution. The verification is performed by comparing the expressive power of EN and that of other representations, evaluating EN utilizing resource consumption as the criteria, and implementing EN and EN Schema for several pervasive systems and evaluating its benefits and shortcomings in these systems.

With these comparisons and evaluations, we verify that EN and EN Schema have sufficient semantic expressive power, require modest resources for pervasive systems, and are compatible with Semantic Web representations.

As a conclusion, the main contribution of this dissertation are EN and EN Schema, publicly available representations for exchanging data and knowledge models of Semantic Web. Such a syntax bridges the gap between Semantic Web and devices, such as sensors and actuators. If all devices could communicate using the same compact format that can be easily transformed to common knowledge representations, knowledge-based systems in pervasive environments would be much easier to build and maintain. This would facilitate implementing the general vision of pervasive computing described by Mark Weiser.

6.4 Open issues and future work

There are still several issues that need more research, including coordination of semantics in pervasive environments, protocol and resource usage, and privacy and security. We discuss these issues in this section and consider future work as well.

It's a challenge to coordinate semantics in heterogeneous and unpredictable pervasive environments. Though a first step in this direction has already been undertaken in Section 5.7, more research is required. The focus is to coordinate heterogeneous components communicating with EN and EN Schema and utilize the data and knowledge for reasoning. Coordination of semantics requires a distributed architecture, in order to allow components to publish and retrieve information. Moreover, it's important to support asynchronous interaction among components. Interaction should be uncoupled in space and time in order to allow components to publish and retrieve information in a flexible manner. Finally, scalability as a central issue because of the dynamic and unpredictable nature.

In this dissertation, we focused on representations and utilized the available protocols and radio interfaces in the prototypes. Optimized protocols, like Constrained Application Protocol(CoAP) [11], are more suitable for resource constrained networks for decreasing communication load. CoAP is complementary to HTTP as it is targeted for resource constrained networks instead of traditional IP networks. Power, memory and computation constraints were taken into account when CoAP was designed.

EN does not provide any security mechanisms at the packet level, as the possible solutions are quite heavyweight operations for resource-constrained sensors. But cryptographic protocols, like Secure Sockets Layer (SSL) [135], could be utilized by the protocol transporting EN packets. For example, EN packets could be transferred using HTTP over SSL. In addition, the aggregation of information sources poses challenges for privacy. To solve this, data sent from mobile devices and applications could be encrypted with the classical K-anonymity [136] criterion. Data accessed from social networks are guarded by the corresponding privacy policies of each of those networks.

EN and EN Schema can be developed further. First, short packets could be utilized for fast reasoning on mobile devices. In such a case, a reasoner would not need to have templates nor transform EN and EN Schema into well-known knowledge representations. Such reasoning would offer a lightweight mechanism for inference over network data. Second, EN can be further developed for data streams. This idea is inspired by emerging stream reasoning technologies [137], which offer a time based data model where data items can be annotated with time stamps, either with the occurrence time or validity time period. Third, EN and EN Schema could be utilized also to represent logic rules, then all representations in a pervasive system could be transferred using one uniform format. Fourth, the preliminary study of optimizing a system at design time could be extended for more complex pervasive systems. Fifth, EN Schema could be evaluated with larger ontologies. Other potential developments include a mechanism to discover EN packets and data sources.

6.5 Concluding remarks

This dissertation addressed the topic of data and knowledge representations for pervasive environments. EN and EN Schema are expressive ontology-based representations amenable to distributed systems. EN is so lightweight that resource-constrained devices can afford it. This is a unique feature which offers benefits when building knowledge-based systems in pervasive environments. Moreover, EN is a candidate as a representation for wide application areas of distributed systems.

Semantic technologies enable machine-interpretable representation formalism for describing entities, sharing and integrating information, and inferring new knowledge. In pervasive environments, semantics helps creating machine-interpretable and self-descriptive data. EN combines expressiveness and compactness.

This dissertation presents the design details of EN and EN Schema and their implementations on a simulator, embedded sensors, and smart phones. This thesis is one step towards building knowledge-based systems for pervasive environments.

References

1. Weiser M (1991) The computer for the 21st century. Scientific American 265(9): 66-75.
2. Smith RG (1985) Knowledge-based systems concepts, techniques, examples. URI: http://www.reidgsmith.com/Knowledge-Based_Systems_-_Concepts_Techniques\_Examples_08-May-1985.pdf. Cited 2015/06/15.
3. Aarts E & Riuter BD (2009) New research perspectives on ambient intelligence. Journal of Ambient Intelligence and Smart Environments 1(1): 5-14.
4. Encyclopedia (2015) Data. URI: http://www.encyclopedia.com/. Cited 2015/06/15.
5. Schreiber G & Raimond Y (2014) RDF 1.1 primer. URI: http://www.w3.org/TR/rdf11-primer/. Cited 2015/06/15.
6. Hitzler P, Krötzsch M, Parsia B, Patel-Schneider PF & Rudolph S (2012) OWL 2 Web ontology language primer (Second Edition). URI: http://www.w3.org/TR/owl2-primer/. Cited 2015/06/15.
7. Beckett D, Berners-Lee T, Prud'hommeaux E & Carothers G (2014) RDF 1.1 Turtle Terse RDF triple language. URI: http://www.w3.org/TR/turtle/. Cited 2015/06/15.
8. Generic Sensor Format (GSF). URI: https://www.leidos.com/maritime/gsf. Cited 2016/06/29.
9. Crockford D (2006) The application/json media type for Java-Script object notation (JSON). URI: http://tools.ietf.org/html/rfc4627. Cited 2015/06/15.
10. Sporny M, Longley D, Kellogg G, Lanthaler M & Lindström N (2014) JSON-LD 1.0 A JSON-based serialization for Linked Data. URI: http://www.w3.org/TR/json-ld/. Cited 2015/06/15.
11. Shelby Z, Hartke K & Bormann C (2014) Constrained application protocol (CoAP). URI: http://tools.ietf.org/html/rfc7252. Cited 2015/06/15.
12. Chen H, Finin T & Joshi A (2005) The SOUPA ontology for pervasive computing. In: Tamma V, Cranefield S, Finin TW & Willmott S(ed) Ontologies for Agents: Theory and Experiences. Birkhäuser Basel: 233-258.
13. Russomanno DJ, Kothari CR & Thomas OA (2005) Building a sensor ontology: a practical approach leveraging ISO and OGC Models. Proceedings of the 2005 International Conference on Artificial Intelligence. Las Vegas, NV: 637-643.
14. W3C Semantic Sensor Network Incubator Group (2011) Semantic sensor network ontology. URI: http://purl.oclc.org/NET/ssnx/ssn. Cited 2015/06/15.
15. Goddard W & Melville S (2004) Research methodology: an introduction. 2nd edition. Juta Academic.
16. Riekki J, Su X & Haverinen J (2008) Connecting resource-constrained robots to knowledge-based systems. Proceedings of the International Conference on Modelling, Identification and Control. Innsbruck, Austria, ACTA Press.
17. Su X, Riekki J & Haverinen J (2009) Semantic support for resource-constrained robot swarm. Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics. Milan, Italy: 271-277.
18. Su X, Riekki J & Tarkoma S (2009) An approach to achieve context-aware maps: combining semantic web technology with sensor data. Proceedings of the 5th International Conference on Intelligent Environments. Barcelona, Spain, IOS Press: 193-203.

19. Su X, Riekki J & Haverinen J (2012) Entity Notation - enabling knowledge representations for resource-constrained sensors. Personal and Ubiquitous Computing 16(7): 819-834.
20. Su X, Gilman E, Kwiatkowski P, Latkowski T, Pröbstl A, Wójtowicz B & Riekki J (2011) Knowledge-based systems for ambient social interactions. Proceedings of the second International Joint Conference on Ambient Intelligence. Amsterdam, Netherlands, Springer: 61-71.
21. Su X, Riekki J, Nurminen, JK, Nieminen J & Koskimies M (2015) Adding semantics to internet of things. Concurrency and Computation: Practice and Experience 27(8): 1844-1860.
22. Maarala AI, Su X & Jukka R (2014) Semantic data provisioning and reasoning for the internet of things. Proceedings of the 4th International Conference on the Internet of Things. Cambridge, MA, IEEE: 13-18.
23. Maarala AI, Su X & Jukka R (In press) Semantic reasoning for advanced Internet of Things applications. IEEE Internet of Things Journal.
24. Su X, Gilman E & Riekki J (2014) Building knowledge-based systems to enable ambient social interactions. Journal of Ambient Intelligence and Smart Environments 6(2): 121-135.
25. Su X & Riekki J (2010) Transferring ontologies between mobile devices and knowledge-based systems. Proceedings of the 8th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. Hongkong China, IEEE: 127-135.
26. Su X, Fucci D & Riekki J (2010) A framework to enable two-layer inference for ambient intelligence. In: Augusto JC, Corchado JM, Novais P & Analide C (ed) Ambient Intelligence and Future Trends - International Symposium on Ambient Intelligence. Springer: 29-36.
27. Su X & Riekki J (2010) Bridging the gap between semantic web and networked sensors: a position paper. Proceeding of the 3nd International Workshop on Semantic Sensor Networks, with 2010 International Semantic Web Conference. Shanghai, China, CEUR-WS.
28. Perttunen M, Riekki J & Lassila O (2009) Context representation and reasoning in pervasive computing: a review. International Journal of Multimedia and Ubiquitous Engineering. 4(4): 1-28.
29. Strang T & Linnhoff-Popien C (2004) A context modeling survey. Proceeding of the Workshop on Advanced Context Modelling, Reasoning and Management, with the Sixth International Conference on Ubiquitous Computing. Nottingham, England.
30. Gruber TR (1993) A translation approach to portable ontology specifications. Knowledge Acquisition 5(2): 199-221.
31. Studer R, Benjamins VR & Fensel D (1998) Knowledge engineering: principles and methods. Data and Knowledge Engineering 25 (1-2): 161-197.
32. Baader F, Horrocks I & Sattler U (2007) Chapter 3: Description logics. In: Harmelen F, Lifschitz V & Porter B (ed) Handbook of Knowledge Representation. Elsevier: 135-179.
33. Hitzler P, Krötzsch M & Rudolph S (2009) Foundations of Semantic Web technologies. Chapman & Hall/CRC.
34. Soylu A, Causmaecker PD, Preuveneers D, Berbers Y & Desmet P (2011) Formal modelling, knowledge representation and reasoning for design and development of user-centric pervasive software: a meta-review. International Journal of Metadata, Semantics and Ontologies 6 (2): 96-125.
35. Brickley D & Guha RV (2014) RDF Schema 1.1. URI: http://www.w3.org/TR/rdf-schema/. Cited 2016/06/29.

36. Horrocks I (2002) Reasoning with expressive description logics: theory and practice. Proceeding of the 19th International Conference on Automated Deduction. Copenhagen, Denmark, Springer:1-15.
37. Motik B, Grau BC, Horrocks I, Wu Z, Fokoue A & Lutz C (2012) OWL 2 Web ontology language profiles (second edition). URI: http://www.w3.org/TR/owl2-profiles/. Cited 2016/06/29.
38. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B & Dean M (2004) SWRL: A Semantic Web Rule Language Combining OWL and RuleML. URI: https://www.w3.org/Submission/SWRL/. Cited 2016/06/29.
39. Kifer M & Boley H (2013) RIF Overview (Second Edition). URI: https://www.w3.org/TR/rif-overview/. Cited 2016/06/29.
40. Apache Jena (2016) Reasoners and rule engines: Jena inference support. URI: http://incubator.apache.org/jena/documentation/inference/. Cited 2016/06/15.
41. Unisens - a universal data format. URI: http://www.unisens.org/index.php. Cited 2016/06/29.
42. Kwon TM (2004) Unified Transportation Sensor Data Format (UTSDF): Introduction. Transportation Data Research Laboratory, Doc #2004021.
43. Berners-Lee T, Hendler J & Lassila O (2001) The Semantic Web. Scientific American. May 2001: 29-37.
44. Berners-Lee T & Connolly D (2011) Notation3 (N3): A readable RDF syntax. URI: https://www.w3.org/TeamSubmission/n3/. Cited 2016/06/29.
45. W3C RDF Core WG (2015) N-Triples W3C RDF core WG internal working draft. URI: http://www.w3.org/2001/sw/RDFCore/ntriples/. Cited 2016/06/29.
46. Jennings C, Shelby Z & Arkko J (2012) Media types for sensor markup language (SENML). URI: https://tools.ietf.org/html/draft-jennings-senml-10. Cited 2016/06/29.
47. Schneider J, Kamiya T, Peintner D & Kamiya T (2014) Efficient XML Interchange (EXI) format. URI: http://www.w3.org/TR/exi/. Cited 2016/06/29.
48. Shelby Z. (2012) Constrained RESTful Environments (CoRE) link format. URI: http://tools.ietf.org/html/rfc6690. Cited 2016/06/29.
49. Davic I, Steiner T & J Le Hors A (2013) RDF 1.1 JSON alternate serialization (RDF/JSON). URI: https://dvcs.w3.org/hg/rdf/raw-file/default/rdf-json/index.html. Cited 2016/06/29.
50. Nathan (eds) (2013) JSN3 unofficial draft. URI: http://webr3.org/apps/specs/jsn3/. Cited 2016/06/15.
51. W3C. (2010) JTriples. URI: http://www.w3.org/wiki/JTriples. Cited 2016/06/29.
52. Birbeck M (2009) RDFj: Semantic objects in JSON. URI: http://markbirbeck.com/blog/2009/04/20/rdfj-semantic-objects-in-json/. Cited 2016/06/29.
53. Duerst M & Suignard M (2005) Internationalized Resource Identifiers (IRIs). URI: https://www.ietf.org/rfc/rfc3987.txt. Cited 2016/08/17.
54. W3C (2011). JSON serialization examples. URI: http://www.w3.org/2011/rdf-wg/wiki/JSON-Serialization-Examples. Cited 2016/06/29.
55. Sheth A, Henson C & Sahoo SS (2008) Semantic sensor web. IEEE Internet Computing 12(4): 78-83.
56. W3C (215) RDFa 1.1 primer - third edition rich structured data markup for web documents. URI: http://www.w3.org/TR/xhtml-rdfa-primer/. Cited 2016/06/29.
57. Henson CA, Pschorr JK, Sheth AP & Thirunarayan K (2009) SemSOS: semantic sensor observation service. Proceeding of the 2009 International Symmposium on Collaborative Technologies and Systems. Baltimore, MD, IEEE: 44-53.

58. Toma I, Simperl E & Hench G, (2009) A joint roadmap for semantic technologies and the internet of things. Proceeding of the 3rd STI Roadmapping Workshop Charting the next Generation of Semantic Technology. Heraklion Greece.
59. Martin B & Jano B (1999) WAP binary XML content format. URI: http://www.w3.org/TR/wbxml/. Cited 2016/06/29.
60. Noemax (2015) FastInfoset.net. URI: http://www.noemax.com/products/fastinfoset/index.html. Cited 2016/06/29.
61. Kangasharju J, Tarkoma S & Lindholm T (2005) Xebu: A binary format with schema-based optimizations for XML data. Proceeding of 6th International Conference on Web Information Systems Engineering. New York, NY, Springer Berlin Heidelberg: 528-535.
62. Bournez C (2009) Efficient XML interchange Evaluation. URI: http://www.w3.org/TR/exi-evaluation/. Cited 2016/06/29.
63. Cheney J (2001) Compressing XML with multiplexed hierarchical PPM models. Proceeding of the Data Compression Conference. Snowbird, UT, IEEE: 163-172.
64. XML Solutions (1999) XMLZip. URI: http://www.xmls.com/. Cited 2015/06/15.
65. Buneman P, Grohe M & Koch C (2003) Path queries on compressed XML. Proceeding of the 29th International Conference on Very Large Data Bases. Berlin, Germany, VLDB Endowment: 141-152.
66. Min JK, Park MJ & Chung CW (2003) XPRESS: A queriable compression for XML data. Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. San Diego, CA, ACM: 122-133.
67. Ng W, Yeung LW & Cheng J (2006) Comparative analysis of XML compression technologies. World Wide Web 9(1): 5-33.
68. Fernández JD, Martínez-Prieto MA, Gutierrez C & Polleres (2011) Binary RDF representation for publication and exchange (HDT). W3C Member Submission. URI: http://www.w3.org/Submission/2011/SUBM-HDT-20110330/. Cited 2016/06/29.
69. Fernández JD, Martínez-Prieto MA & Gutierrez C (2010) Compact representation of large RDF data sets for publishing and exchange. Proceeding of the 9th international semantic web conference. Shanghai, China, Springer Berlin Heidelberg: 193-208.
70. Hasemann H, Kröller A & Pagel M (2012) RDF provisioning for the internet of things. Proceeding of the 3rd International Conference on the Internet of Things. Wuxi, China, IEEE: 143-150.
71. Woods WA & Schmolze JG (1992) The KL-ONE family. Computers & Mathematics with Applications 23 (2-5): 133-177.
72. Kifer M & Lausen G (1989) F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. Proceedings of the 1989 ACM SIGMOD international conference on Management of data. Portland, Oregon, ACM: 134-146.
73. Heflin J, Hendler J & Luke S (1999) SHOE: A Knowledge Representation Language for Internet Applications. Technical report CS-TR-4078, Department of Computer Science, University of Maryland.
74. McGuinness DL, Fikes R, Stein LA & Hendler J (2002) DAML-ONT: an ontology language for the Semantic Web. In: Fensel F, Hendler J, Lieberman H & Wahlster W (ed) Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press.
75. Fensel D, van Harmelen F, Horrocks I, McGuinness DL & Patel-Schneider PF (2001) OIL: an ontology infrastructure for the Semantic Web. IEEE Intelligent Systems 16(2): 38-45.

76. McGuinness DL, Fikes R, Hendler J & Stein LA (2002) DAML+OIL: an ontology language for the Semantic Web. IEEE Intelligent Systems 17(5): 72-80.
77. Carroll J, Herman I & Patel-Schneider PF. OWL 2 web ontology language RDF-based semantics (Second Edition). URI: http://www.w3.org/TR/owl2-rdf-based-semantics/. Cited 2016/06/29.
78. Yu LY (2014) A deveoper's guide to the Semantic Web. Springer-Verlag Berlin Heidelberg.
79. Horridge M & Patel-Schneider PF (2012) OWL 2 Web Ontology Language Manchester Syntax (Second Edition). URI: https://www.w3.org/TR/owl2-manchester-syntax/. Cited 2016/06/29.
80. Fuchs NE, Kaljurand K & Kuhn T (2008) Attempto controlled English for knowledge representation. In: Baroglio C, Bonatti PA, Małuszyński J, Marchiori M, Polleres A & Schaffert S (ed) Reasoning Web. Springer Berlin Heidelberg: 104-124.
81. Cregan A, Schwitter R & Meyer T (2007) Sydney OWL syntax - towards a controlled natural language syntax for OWL 1.1. Proceedings of the OWLED 2007 Workshop on OWL: Experience and Directions. Innsbruck, Austria, CEUR-WS.
82. Hart G, Johnson M & Dolbear C (2008) Rabbit: developing a control natural language for authoring ontologies. Proceedings of the 5th European Semantic Web Conference. Tenerife, Canary Islands, Spain, Springer Berlin Heidelberg: 348-360.
83. Schwitter R, Kaljurand K, Cregan A, Dolbear C & Hart G (2008) A comparison of three controlled natural languages for OWL 1.1. Proceedings of the 4th OWL Experiences and Directions Workshop. Washington, USA.
84. Le-Phuoc D, Pareira JX, Reynolds V & Hauswirth M (2010) RDF on the go: an RDF storage and query processor for mobile devices. Proceedings of the International Semantic Web Conference 2010 Posters & Demonstrations Track. Shanghai, China, CEUR-WS.
85. Crivellaro F (2007) μJena: Gestione di ontologie sui dispositivi mobili. MSC Thesis of Politécnico di Milano.
86. Koziuk M, Domaszewicz J, Schoeneich RO, Jablonowski M & Boetzel P (2008) Mobile context-addressable messaging with dl-lite domain model. In: Roggen D, Lombriser C, Tröster G, Kortuem G & Havinga P (ed) Smart sensing and context. Zürich, Switzerland, Springer Berlin Heidelberg: 168-181.
87. Gu T, Kwok Z, Koh KK & Pung HK (2007) A mobile framework supporting ontology processing and reasoning. Proceedings of the 2nd workshop on requirements and solutions for pervasive software infrastructures, in conjunction with the 9th international conference on ubiquitous computing. Innsbruck, Austria: 16-19.
88. Ali S & Kiefer S (2009) μOR - Micro a micro owl dl reasoner for ambient intelligent devices. In: Abdennadher N & Petcu D (eds) Advances in grid and pervasive computing. Geneva, Switzerland, Springer Berlin Heidelberg: 305-316.
89. Vazquez Gomez JI (2007) A reactive behavioural model for context-aware semantic devices. Doctoral Dissertation of Universidad de Deusto.
90. Jang M & Sohn JC (2004) Bossam: an extended rule engine for OWL inferencing. In: Antoniou G & Boley H (ed) Rules and rule markup languages for the semantic web. Hiroshima, Japan, Springer Berlin Heidelberg: 128-138.
91. Androjena (2015) Porting of Jena to Android URI: https://github.com/lencinhaus/androjena. Cited 2016/06/29.

92. Hecht R & Jablonski S (2011) Nosql evaluation: a use case oriented survey. Proceedings of the 2011 International Conference on Cloud and Service Computing. Hong Kong, China, IEEE: 336-341.
93. Abadi DJ, Marcus A, Madden SR & Hollenbach K (2007) Scalable Semantic Web data management using vertical partitioning. Proceedings of the 33rd international conference on Very large data bases. Vienna, Austria, VLDB Endowment: 411-422.
94. Wang XH, Zhang DQ, Gu T & Pung HK (2004) Ontology based context modeling and reasoning using OWL. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. Lugano, Switzerland: 18-22.
95. Lassila O & Khushraj D (2005) Contextualizing applications via semantic middleware. The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services. San Diego, USA, IEEE: 183-189.
96. Clocksin, WF & Mellish CS (2003). Programming in Prolog: Using the ISO Standard. Springer Science & Business Media.
97. Sandia National Laboratories (2013) Jess, the rule engine for the JavaTM platform. URI: http://herzberg.ca.sandia.gov/. Cited 2016/06/29.
98. Wang X, Dong JS, Chin, CY, Hettiarachchi, SR & Zhang, D (2004) Semantic Space: An Infrastructure for Smart Spaces, IEEE Pervasive Computing 3(3): 32-39.
99. Korpipää P & Mäntyjärvi J (2003) An ontology for mobile device sensor-based context awareness. Proceedings of the 4th international and interdisciplinary conference on Modeling and using context. Stanford, CA, USA, Springer-Verlag: 451-458.
100. Buriano L, Marchetti M, Carmagnola F, Cena F, Gena C & Torre I (2006) The role of ontologies in context-aware recommender systems. Proceedings of the 7th International Conference on Mobile Data Management. Nara, Japan, IEEE:80.
101. Naudet Y, Mignon S, Lecaque L, Hazotte C & Groues V (2008) Ontology-based matchmaking approach for context-aware recommendations. Proceedings of the International Conference on Automated Solutions for Cross Media Content and Multi-channel Distribution: Florence, Italy, IEEE:218-223.
102. Ciaramella A, Mario GCAC, Lazzerini B & Marcelloni F (2009) Situation-aware mobile service recommendation with fuzzy logic and semantic Web. Proceedings of the 9th International Conference on Intelligent Systems Design and Applications. Pisa, Italy, IEEE: 1037-1042.
103. Kritsotakis M, Michou M, Nikoloudakis E, Bikakis A, Patkos T, Antoniou G & Plexousakis D (2009) Design and implementation of a semantics-based contextual navigation guide for indoor environments. Journal of Ambient Intelligence and Smart Environments 1(3): 261-285.
104. Sundmaeker H, Guillemen P, Friess P & Woelfflé S (2010) Vision and challenges for realising the internet of things. CERP-IoT Cluster of European Research Projects on the Internet of Things. Brussels, Belgium.
105. Wei W & Barnaghi P (2009) Semantic annotation and reasoning for sensor data. In: Barnaghi P, Moessner K, Presser M & Meissner S (ed) Smart Sensing and Context. Guildford, UK, Springer Berlin Heidelberg: 66-76.
106. Bikakis A & Antoniou G (2010) Rule-based contextual reasoning in ambient intelligence. Proceedings of the 4th International Web Rule Symposium. Washington, DC, Springer Berlin Heidelberg: 74-88.
107. Bikakis A, Patkos T, Antoniou G & Plexousakis D (2008) A survey of semantics-based approaches for context reasoning in ambient intelligence. In: Mühlhäuser M, Ferscha A &

Aitenbichler E (ed) Constructing Ambient Intelligence. Darmstadt, Germany, Springer Berlin Heidelberg: 14-23.

108. Barnaghi P, Wang W, Henson C & Taylor K (2012) Semantics for the internet of things: early progress and back to the future. International Journal on Semantic Web & Information Systems 8(1): 1-21.
109. Kao YW & Yuan SM (2012) User-configurable semantic home automation. Computer Standards & Interfaces 34(1): 171-188.
110. Taylor K, Griffith C, Lefort L, Gaire R, Compton M, Wark T, Lamb D, Falzon G & Trotter M (2013) Farming the web of things. IEEE Intelligent Systems 28(6): 12-19.
111. Zhou Q, Natarajan S, Simmhan Y & Prasanna V (2012) Semantic information modeling for emerging applications in smart grid. Proceedings of the 9th International Conference on Information Technology: New Generations. Las Vegas, NV, IEEE: 775-782.
112. Hristoskova A, Sakkalis V, Zacharioudakis G, Tsiknakis M & Turck FD (2014) Ontology-driven monitoring of patients vital signs enabling personalized medical detection and alert. Sensors 14 (1): 1598-1628.
113. Preist C, Esplugas-Cuadrado J, Battle SA, Grimm S & Williams SK (2005) Automated business-to-business integration of a logistics supply chain using semantic web services technology. Proceedings of the 4th International Semantic Web Conference. Galway, Ireland, Springer Berlin Heidelberg: 987-1001.
114. W3C (2009) Cwm. URI: http://www.w3.org/2000/10/swap/doc/cwm.html. Cited 2016/06/29.
115. Berners-Lee T, Semantic Web -XML 2000, URI: https://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html. Cited 2016/06/26.
116. Leach P, Mealling M & Salz R, (2005) A universally unique IDentifier (UUID) URN namespace. URI: http://www.ietf.org/rfc/rfc4122.txt. Cited 2016/06/29.
117. Biron PV, Permanente K & Malhotra A (2004) XML Schema Part 2: Datatypes Second Edition. URI: http://www.w3.org/TR/xmlschema-2/. Cited 2016/06/15.
118. Phillips A & Davis M (2009) Tags for Identifying Languages, URI: https://tools.ietf.org/html/rfc5646. Cited 2016/06/09.
119. Pattis RE EBNF: A Notation to Describe Syntax. URI: https://www.ics.uci.edu/ pattis/ICS-33/lectures/ebnf.pdf. Cited 2016/06/29.
120. Russell S & Norvig P (2010) Artificial intelligence: A modern approach (3rd edition). Prentice Hall.
121. Domingue J, Fensel D & Hendler, JA (2011) Handbook of Semantic Web Technologies, Springer.
122. Manola F & Miller E (2004) RDF Primer. URI: https://www.w3.org/TR/2004/REC-rdf-primer-20040210/. Cited 2016/06/29.
123. Michelogiannakis G, Jiang N, Becker D & Dally W (2011) Packet Chaining: Efficient Single-Cycle Allocation for On-Chip Networks. IEEE Computer Architecture Letters 10(2):33-36.
124. Bao J, Kendall EF, McGiommess DL & Patel-Schneider PF (2012) OWL 2 Web Ontology Language Quick Reference Guide (Second Edition). URI: https://www.w3.org/TR/owl2-quick-reference/. Cited 2016/06/09.
125. Hitzler P, Krötzsch M, Parsia B, Patel-Schneider PF, & Rudolph S (2012) OWL 2 Web Ontology Language Primer (Second Edition). URI: https://www.w3.org/TR/owl2-primer/. Cited 2016/06/09.

126. Gandon F & Schreiber G (2014) RDF 1.1 XML Syntax.URI: https://www.w3.org/TR/rdf-syntax-grammar/. Cited 2016/06/09.
127. Shadbolt N, Beerners-Lee T & Hall W (2006) The semantic web revisited. IEEE Intelligent Systems 21(3): 96-101.
128. Riekki J, Alakärppä I, Koukkula R, Angeria J, Brockman M & Saloranta T (2007) Wireless pain monitoring. Proceedings of the 2nd International Symposium on Medical Information and Communication Technology. Oulu, Finland: 1-7.
129. Grant J & Beckeet D (2004) RDF test cases, W3C Recommendation. URI: http://www.w3.org/TR/rdf-testcases/. Cited 2015/06/15.
130. Chen H, Finin T & Joshi A (2003) An ontology for context-aware pervasive computing environments. The Knowledge Engineering Review 18(3):197-207.
131. Berrueta D, Brickley D, Decker S, Fernández S, Görn C, Harth A, Heath T, Idehen K, Kjernsmo K, Miles A, Passant A, Polleres A, Polo L & Sintek M (2007) SIOC Core Ontology Specification. URI: https://www.w3.org/Submission/sioc-spec/. Cited 2016/08/12.
132. Bermudez-Edo M, Barnaghi P & Elsaleh T (2015) iot-lite Ontology. URI: http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite. Cited 2016/08/12.
133. Reynolds D (2014) The Organization Ontology. URI: https://www.w3.org/TR/vocab-org/. Cited 2016/08/12.
134. Ruyter BD (2010) Social interactions in ambient intelligent environments. Doctoral Dissertation of Eindhoven University of Technology.
135. Freier AO, Karlton P & Kocher PC (1996) The SSL protocol. version 3.0. URI: http://home.mit.bme.hu/ hornak/adatbiz/ssl3/ssl-toc.html. Cited 2016/06/29.
136. Sweeney L (2002) k-Anonymity: A model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10(5): 557-570.
137. Su X, Gilman E, Wetz P, Riekki J, Zuo Y & Leppänen T (2016), Stream reasoning for the internet of things: Challenges and gap analysis. Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics. Nîmes, France, ACM:1-10.

ISBN 978-952-62-1334-7 (Paperback)
ISBN 978-952-62-1335-4 (PDF)
ISSN 0355-3213 (Print)
ISSN 1796-2226 (Online)