Semantic Interoperability of Ambient Intelligent Medical Devices and e-Health Systems

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Safdar Ali

Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)
Fraunhofer Institut für Biomedizinische Technik (FhG-IBMT)
Max-Planck Institut für Informatik (MPII)

Saarbrücken
2010

Dekan der Naturwissenschaftlich-Technischen Fakultät I	Prof. Dr.-Ing. Joachim Weickert
Vorsitzender der Prüfungskommission	Prof. Dr.-Ing. Hans-Peter Lenhof
Wiss. Betreuer	Prof. Dr. rer. nat. Günter R. Fuhr
Wiss. Betreuer	Prof. Dr.-Ing. Gerhard Weikum
Wiss. Begleiter	PD. Dr. rer. nat. Matthias Klusch
Wiss. Begleiter	Dipl.-Inform Stephan Kiefer
Tag des Promotionskolloquiums	16.02.2010

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, den 16.02.2010

____
(Unterschrift)

Dedicated to the land of the pure, Pakistan

Kurzfassung

Hochmoderne mobile medizinische Geräte stellen wichtige therapeutische Funktionen mit wertvollen Informationen über Behandlungsmuster zur patientennahen Versorgung zur Verfügung. Solche Geräte bleiben zumeist unabhängige Informationsinseln, die nicht in der Lage sind, die Daten, die sie sammeln, mit anderen medizinischen Geräten, Krankenhausinformationssystemen oder Laborinformationssystemen in Echtzeit auszutauschen. Organisationen zum Entwickeln von Standards wie IEEE haben in den letzten Jahrzehnten zahlreiche Versuche unternommen, das Interoperabilitätsproblem von medizinischen Geräten mit einzelnen Kommunikationsstandards wie IEEE 1073 zu lösen, die alle mit mäßigem Erfolg endeten.

Während der letzten Jahre haben das Semantic Web und Web-Service-Technologien große Aufmerksamkeit zur Schaffung von Lösungen gewonnen, die semantische Interoperabilität zwischen Informationssystemen und intelligenten Geräten in verschiedenen Domänen bereitstellen, z.B. in der Fertigungsautomatisierung und im Tourismus. Die vorliegende Arbeit nutzt die Vorteile solcher Technologien und präsentiert eine dezentralisierte, semantische Middleware-Infrastruktur für medizinische Geräte, die *Semantic Medical Device Space* (SMDS) genannt wird. Das SMDS-Framework gewährleistet die Interoperabilität sowohl zwischen medizinischen Geräten als auch mit e-Health-Systemen. Es verhilft den aktuellen medizinischen Geräten, sich zur nächsten Generation von *ambienten intelligenten* medizinischen Geräten zu entwickeln, die nicht nur andere medizinische Geräte und Services in einer Gesundheitsversorgungsumgebung erkennen können, sondern auch fähig sind, ihre Messergebnisse durch semantische Web Services in kontextbewussten medizinischen Anwendungen auszutauschen.

Abstract

State-of-the-art mobile medical devices provide important therapeutic functions with valuable information of treatment patterns at the point-of-care. However, such devices mostly remain independent islands of information being unable to share the medical data they gather with other medical devices, hospital information system or laboratory information system on a real-time basis. Standards organizations such as IEEE have made various attempts to resolve the medical devices' interoperability problem using single communication standard, such as IEEE 1073, but ended with moderate success.

During the last years, the Semantic Web and Web Service technologies have gained enormous attention for building solutions to provide semantic interoperability between information systems and smart devices in various domains, such as tourism and factory automation. This thesis takes advantage of these technologies and presents a decentralized semantic middleware infrastructure for medical devices, named *Semantic Medical Devices Space* (SMDS) to provide interoperability among medical devices as well as with e-Health systems. The SMDS framework leverages the current medical devices towards the next generation of *Ambient Intelligent Medical Devices*, which are not only aware of other medical devices and their services present in healthcare environments, but also able to share their measurement results through Semantic Web Services in context-aware healthcare applications.

Zusammenfassung

Aktuelle mobile medizinische Geräte kommunizieren mit Krankenhausinformationssystemen und/oder Laborinformationssystemen bestimmter Anbieter zumeist über proprietäre Protokolle. Die meisten von Ihnen können aber nicht mit anderen medizinischen Geräten kommunizieren, selbst wenn sie vom selben Hersteller entwickelt wurden, obwohl dies in vielen Szenarien benötigt wird. Standardisierungsorganisationen haben auch einige medizinische Geräte kommunikationsstandards entwickelt, wie etwa IEEE 1073 und seine Nachfolger ISO/IEEE 11073, die jedoch weitgehend nicht von der Industrie aufgenommen wurden. Es gibt nur einige wenige Implementierungen dieser Standards. Anderseits eröffnet die Untersuchung fortschrittlicher pervasiver Computing-Szenarien in verschiedenen Lebensbereichen, vor allem in der pervasiven Gesundheitsversorgung. Objekte wie medizinische Geräte, Sensoren, Krankenhausinformationssysteme und Laborinformationssysteme, die in einem intelligenten Umgebungsumfeld agieren, haben unterschiedliche Ziele, Erfahrungen, Fähigkeiten und Kommunikationsschnittstellen, und aufgrund des sehr dynamischen und offenen Charakters der Umgebung, die sie wahllos betreten und verlassen, ist es a *priori* nicht möglich, zu wissen, welche weiteren Objekte zu Einrichtungen in einem bestimmten Zeitraum in der Umgebung agieren.

Diese Doktorarbeit präsentiert eine innovative dezentralisierte semantische Middleware Infrastruktur, *Semantic Medical Device Space* (SMDS) genannt, für medizinische oder mobile Geräte mit eingeschränkten Ressourcen, die weitgehend auf den Grundlagen von Semantic Web und Web Services-Technologien basiert. SMDS wurde als ein Framework entwickelt und designt, das es medizinischen Geräteherstellern erleichtert, ihre *stummen* Geräte in *ambiente intelligente* medizinischen Geräte der nächster Generation umzuwandeln, deren Fähigkeiten und Messungen durch eine Semantic Web Services Schnittstelle bereitgestellt werden, um plattformunabhängigen und direkten Zugang zu diesen medizinischen Geräten zu ermöglichen. Als integrale Bestandteile des SMDS-Frameworks wurde ein leichtgewichtiges semantisches Discovery-Protokoll, genannt *Semantic Medical Device Discovery Protocol* (SMDDP), und ein kleines aber leistungsfähiges System für Wissenabfragen des Fachwissens und Schlussfolgern, nämlich *Micro OWL Description Logic Reasoner* (μOR) entwickelt. SMDDP verhilft ambienten intelligenten medizinischen Geräten in einer pervasiven Gesundheitsumgebung gewünschte andere medizinischen Geräte entsprechend ihrer physikalischen Eigenschaften (z.B. Hersteller, Gruppe/Typ, usw.) und/oder ihrer funktionellen Eigenschaften (z.B. welche Methoden ein Semantischer Web Service anbietet und die semantische Beschreibungen dieser Methoden usw.) semantisch zu erkennen. μOR wurde entwickelt, um die medizinischen oder mobilen Geräte mit der Fähigkeit auszustatten, integriertes Wissen abzufragen, das abgefragt werden kann und daraus durch OWL Beschreibungslogik und Regeln zu schlussfolgern, was zum Vergleichen mit den gesuchten medizinischen oder Mobilen Geräten während des semantischen Discovery-Prozesses verwendet wird.

Summary

State-of-the-art medical devices communicate with hospital information systems and/or laboratory information systems using vendor specific or proprietary protocols only, and most of them can not communicate with other medical devices, even if developed by the same manufacturer, if required in some scenarios. Standards developing organizations have developed few medical device communication standards, e.g. IEEE 1073 and its successor ISO/IEEE 11073, but their adoption by industrial stakeholders is very limited because of the complexity of these protocol stacks and/or missing support for the prevalent technologies like Ethernet or TCP/IP. On the other hand, the study of advanced pervasive computing scenarios in different fields, and particularly in *pervasive healthcare*, has introduced new research challenges, which focus on the provision of healthcare to anyone, at anytime, and anywhere by removing restraints of time and location while increasing both the coverage and the quality of healthcare. The entities, i.e., medical devices, sensors, hospital information systems and laboratory information systems that operate in an ambient environment are expected to have different goals, capabilities, and communication interfaces. Due to the highly dynamic and open nature of the environment where various entities join and leave the environment in an unpredictable way, it is not possible to have a *priori* knowledge about all other entities that are present in the environment at a particular time interval.

This thesis presents an innovative *decentralized* semantic middleware infrastructure for resource-constrained medical devices, or mobile devices in general, namely *Semantic Medical Devices Space* (SMDS), based on Semantic Web and Web Services technologies. SMDS is designed and developed as a framework to facilitate the device manufacturers to turn their "*dumb*" medical devices into the next generation of *Ambient Intelligent* devices, whose capabilities and measurements are exposed through *Semantic Web Service* interface in order to allow platform-independent and direct access to these medical devices. As integral parts of SMDS framework, a small but powerful knowledge querying and reasoning system, namely *Micro OWL Description Logic Querying and Reasoning System* (μOR), and a lightweight semantic discovery protocol, namely *Semantic Medical Device Discovery Protocol* (SMDDP) have been developed.

μOR is developed to enrich the medical devices with the capabilities of integrated Description Logic and Rules based knowledge *querying* and *reasoning* on the local device knowledge base. μOR plays a key role during the semantic discovery of medical devices based on their *physical characteristics* (such as device vendor, device group/-type etc.) and/or *functional characteristics* (such as methods produced by Web Service and their semantic descriptions etc.). We have developed two small ontologies, namely *Medical Device Ontology* (MeDO) and *SMDS Ontology* (SmdsOnto) in order to express the physical and functional characteristics, respectively. Also, we have developed a simple language, namely *Semantic Device Language for N-Triples* (SCENT) to express the

semantic queries over the device knowledge base, as well as the SCENT Resolution Algorithm (SCENTRA) which is used not only to resolve the SCENT queries, but to generate inferences as well.

SMDDP, which is backed by μOR, is developed to support the discovery of desired medical devices in a pervasive healthcare environment, which *match* with the desired SCENT query conditions. It is an HTTP/UDP based protocol, where the *request message*, embedded with the SCENT query, is sent (*broadcast*) over the local network of medical devices, where every device processes this query against its own local knowledge base, and if it is matched, the results are sent back (*unicast*), embedded in the *response message*, to the requesting medical device.

The complete SMDS framework has been evaluated and tested using different types of medical and mobile devices, particularly within the context of FP6 European Commission funded Integrated Project *SmartHEALTH* in which we have developed the next generation of cancer bio-diagnostic devices. Also, SMDS framework has been successfully exhibited in the international fare, Medica[1] Media 2008 (Düsseldorf, Germany) as well as demonstrated in the review[2] of the project in 2009.

[1]`http://www.medica.de/`
[2]`http://www.smarthealthip.com/output.aspx`

Acknowledgement

I am gratified to Prof. Dr. Günter R. Fuhr (Chair for Biotechnology and Medical Engineering, Director of Fraunhofer-Insitut für Biomedizinische Technik) and Prof. Dr. Gerhard Weikum (Chair for Databases and Information Systems, Scientific Director at Max Planck Institute for Computer Science) for accepting me as a Ph.D student at their chairs and providing me the opportunity to carryout this research work at the department of Intelligent Health Systems in Fraunhofer-Institut für Biomedizinische Technik (IBMT), St. Ingbert, Germany.

I am highly obliged to the whole group, especially my advisor Dipl.-Inform. Stephan Kiefer (Head) for providing me not only a family environment in his group, but also the input from the e-Health domain with protracted flourishing discussions in order to finish this research work. Additionally, I am thankful to my co-advisor PD. Dr. Matthias Klusch (Co-Head of the Multiagent Systems group at German Research Center for Artificial Intelligence (DFKI)) for his scientific support during the course of this research work. I am also thankful to Jörg Kruse and Harald Niederländer for their immense network support required to carry out the experiments of this research work.

Finally, I am deeply indebted to my families for their continuous support during my studies. In spite of the distance, they were an incessant source of encouragement. My gratitude for my parents, siblings, and especially my beloved wife is beyond the words.

Contents

Chapter 1

Introduction

Interoperability is defined as "*the ability of two ore more systems or components to exchange the information and to use the information that has been exchanged*" [1]. In order to achieve *true* interoperability, the systems must be able to not only *exchange* the information using shared communication architectures, methods and frameworks, but also *interpret and use* it correctly using shared data types, terminologies/ontologies and coding schemes. Such classification is generally known as *functional interoperability* and *semantic interoperability*, respectively. Most state-of-the-art medical devices use only vendor specific or proprietary protocols in order to have functional interoperability with hospital information systems or laboratory information systems and they are not directly interoperable with other medical devices if required in some scenarios, even provided by the same vendor. However, some vendors do provide customized software or hardware solutions to get the updated measurement values from the medical devices and use them in further medical applications, e.g. Fig. 1.1[1] and Fig. 1.2[2] illustrate browser-based solutions from *Roche Diagnostics* to gather the data from *point-of-care* medical devices and laboratory analyzers respectively, and using them further for meticulous diagnosis.

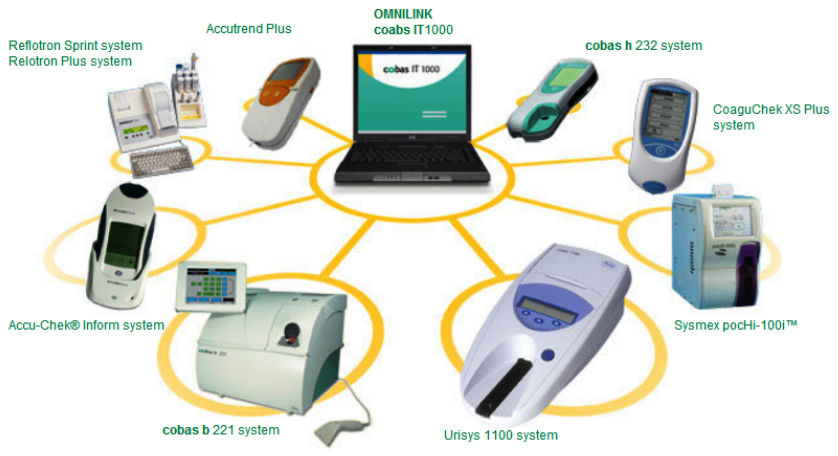


Figure 1.1: Cobas IT 1000 solution for point-of-care medical devices

[1]Cobas IT 1000; `https://www.cobas-roche.co.uk/site/pointofcare.aspx`
[2]Cobas IT 3000; `https://www.cobas-roche.co.uk/site/labsystems.aspx`

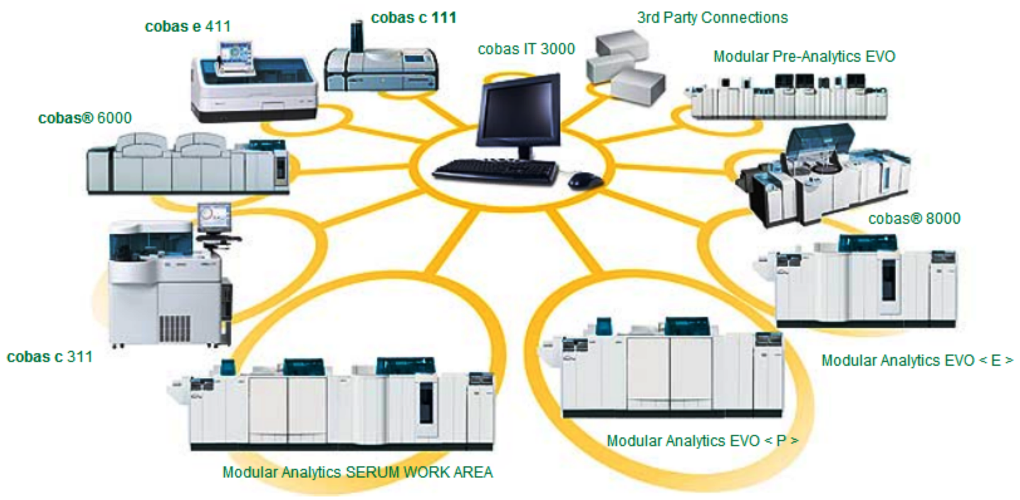

cobas e 411
cobas c 111
cobas IT 3000
3rd Party Connections
Modular Pre-Analytics EVO
cobas® 6000
cobas® 8000
cobas c 311
Modular Analytics EVO < E >
Modular Analytics EVO < P >
Modular Analytics SERUM WORK AREA

Figure 1.2: Cobas IT 3000 solution for laboratory analyzers

Alongside, the standards developing organizations like ISO/IEEE have developed mature communication standards, e.g. ISO/IEEE 11073, the successor of IEEE 1073 [2] for the functional interoperability of point-of-care medical devices, but their adoption by industrial stakeholders is very limited because of the complexity of lower layers of this protocol stack and missing support of the current technologies like Ethernet or TCP/IP [3]. Other initiatives, e.g. MD PnP[3] or IHE PCD[4] have addressed the interoperability issues and highlighted the advantages of having functional interoperability among medical devices, ranging from less development time for data-driven clinical decision support algorithms or medical device safety interlocks to improved patient safety.

Towards enabling *semantic interoperability* among medical devices, few initiatives also have been taken by the standards developing organizations, e.g. NIST[5] has developed an XSchema (*XML Schema*) with ICSGenerator (*Implementation Conformance Statements Generator*) tool [4], based on ISO/IEEE 11073 standard, which facilitates the manufacturers with a systematic approach to unambiguously *define* the specialization of a medical device and *disclose* its features as data sheet. Another example is Continua Health Alliance[6], a non-profit open industry alliance of more than 133 finest healthcare and technology companies, is developing interoperability guidelines and an eco-system of interoperable personal health systems to improve the quality of personal healthcare by using a comprehensive set of industry standards. Continua is targeting different dimensions of healthcare (remote) monitoring, including *Health and Wellness* (e.g. weight, glucose, cholesterol, activity level etc.), *Disease Management* (e.g. post-operation patient monitoring, diabetes, hypertension etc.) and *Aging Independently* (e.g. bed pressure (sleep), bathroom sensor, gas/water sensor, emergency sensor etc.). Fig. 1.3 illustrates the list of first version of *transport independent* device connectivity protocols which Continua health alliance has developed or used so far.

On the other hand, with the advent of Semantic Web and Web Service technologies, the vision of semantic interoperability is becoming reality in various embedded

[3]Medical Device Plug-n-Play Interoperability Program; `http://mdpnp.org`
[4]Integrating Health Enterprise, Patient Care Device Domain; `http://www.ihe.net/pcd/`
[5]National Institute of Standards and Technology, USA; `http://www.nist.gov`
[6]Continua Health Alliance; `http://www.continuaalliance.org/`

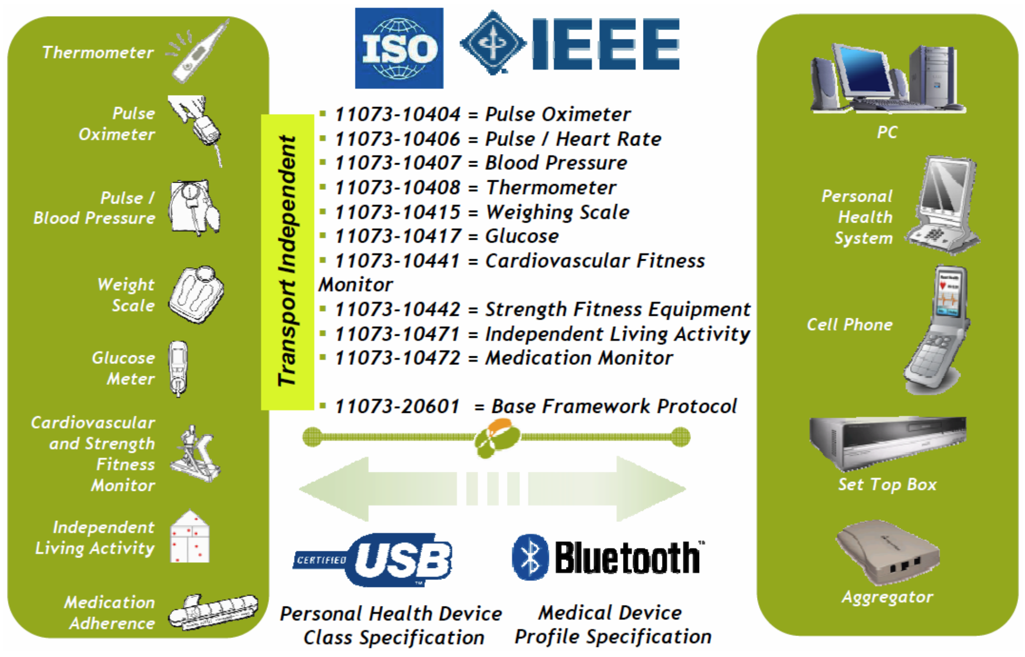

Thermometer
Pulse Oximeter
Pulse / Blood Pressure
Weight Scale
Glucose Meter
Cardiovascular and Strength Fitness Monitor
Independent Living Activity
Medication Adherence
ISO
IEEE
Transport Independent
11073-10404 = Pulse Oximeter
11073-10406 = Pulse / Heart Rate
11073-10407 = Blood Pressure
11073-10408 = Thermometer
11073-10415 = Weighing Scale
11073-10417 = Glucose
11073-10441 = Cardiovascular Fitness Monitor
11073-10442 = Strength Fitness Equipment
11073-10471 = Independent Living Activity
11073-10472 = Medication Monitor
11073-20601 = Base Framework Protocol
CERTIFIED USB
Bluetooth
Personal Health Device Class Specification
Medical Device Profile Specification
PC
Personal Health System
Cell Phone
Set Top Box
Aggregator

Figure 1.3: Continua health alliance - first version of device connectivity standards

fields, for example, *in-vitro* healthcare monitoring, industrial automation, home entertainment etc., in synergy with ubiquitous computing or more recently *ambient intelligent* technologies, where a large number of devices and software components *semantically interoperate* to provide people with services in an unobtrusive fashion. Various research initiatives have addressed these issues, e.g. SAPHIRE [5], SCALLOPS [6] and CAALYX [7] in the field of *healthcare monitoring*; Amigo [8], WSAMI [9] and InHome [10] in the field of *home entertainment*; IMPRONTA [11][12], SOCRADES [13] and SAMIA [14] in the field of *industrial or factory automation.* Most recently, research and industry in above-mentioned domains have shown great interest in an emerging paradigm, namely *Internet of Things and Services (IoTS)* which addresses the issues related to the semantic *identification* and *discovery* of thousands of small physical devices and their semantic interoperability through the Web Services they offer. Not only EU Research projects, e.g. SOCRADES [13], SODA [15], MORE [16], and HYDRA [17] have addressed the goals of IoTS paradigm, but also some national projects, e.g. SemProM [18] are moving in this direction as well.

1.1 Problem Statement

Pervasive healthcare is a conceptual system of providing healthcare to anyone, at anytime, and anywhere by using mobile and wireless technologies, and removing restraints of time and location while increasing both the coverage and the quality of healthcare [19]. Its applications include pervasive health monitoring, intelligent emergency management system, pervasive healthcare data access, and ubiquitous mobile telemedicine. The Healthcare Information and Management Systems Society [20] recently conducted a survey, where the respondents identified the "cross enterprise sharing of patient care medical device data" as one of their highest priorities, which requires establishing different goals, including shortening the decision time, increasing productivity, minimizing transcription errors, and developing ways to correctly define and interpret the medical data being exchanged. In order to meet these goals, *semantic interoperability* among medical devices and health information systems in a *platform independent* fashion is necessary, as medical devices (e.g. infusion pumps) are often the primary source or destination of important patient care information, including therapeutic functions with treatment patterns. However, such devices mostly remain independent islands of information, being *unable* to share the medical data they gather with other medical devices, hospital information system or laboratory information system on a real-time basis due to the software or hardware heterogeneity. On the other hand, as described earlier, the standards developing organizations have made various attempts in the last decades to resolve the medical device interoperability problem using a single communications standard, e.g. IEEE 1073 [2] and more recently its successor ISO/IEEE 11073, but ended with minimal success due to the limited adoption by the medical device manufacturers because of the complexity of protocol stack and missing support for Ethernet and TCP/IP.

In SmartHEALTH Project [21], a new generation of intelligent *lab-on-chip* bio-diagnostic devices for point-of-care applications is being developed that incorporates advanced capabilities for context awareness, data interpretation through soft computing tools, e.g. Neural Networks, ubiquitous communication in pervasive healthcare environments, and the provision of e-Health services. SmartHEALTH devices are intended to operate in all point-of-care driven environments such as hospitals, physicians' offices and ultimately patient self-testing at home or while moving. It is envisioned that SmartHEALTH bio-diagnostic devices at the point-of-care will be able to communicate seamlessly and transparently with the local or remote health information system(s) while respecting the privacy of patients' medical data. Also, most importantly, these devices will operate *autonomously* with respect to semantic contents processing, which means that without using any external proxy system, they are enriched with integrated knowledge processing, querying, and reasoning capabilities, which will facilitate them to adopt according to the requirements of context-aware pervasive healthcare application(s).

In connection with the goals of SmartHEALTH Project and the capabilities envisioned for intelligent SmartHEALTH devices, the overall goal of this thesis is to develop an innovative decentralized middleware infrastructure for the semantic interoperability of resource-constrained medical devices (or mobile devices in general), based on Semantic Web and Web Service technologies, to which the underlying communication layer is transparent. This *service oriented architecture* (SOA) based middleware will be deployable on both new and existing networks of distributed wireless and wired medical or mobile devices, which operate with limited resources in terms of computing power and memory usage. The second objective of this thesis is to provide this middleware as a *framework* or a software development kit, which will facilitate the medical device manufacturers to turn their *plain* medical devices into the next generation of *ambient intelligent* medical devices, whose *physical capabilities* are described using *ontologies*, while *functional capabilities* are exposed in addition through *Semantic Web Services*. This framework will include support for the secure transmission of patient's medical data from device-to-device and device-to-information system in different pervasive healthcare applications.

Fig. 1.4 illustrates one of the pervasive healthcare scenarios of SmartHEALTH Project where we apply our work, namely *post-operation breast cancer home-care monitoring*, where a SmartHEALTH device, besides being a blood cancer markers analyzer, also acts as an active or passive gateway for other medical devices (e.g. blood pressure,

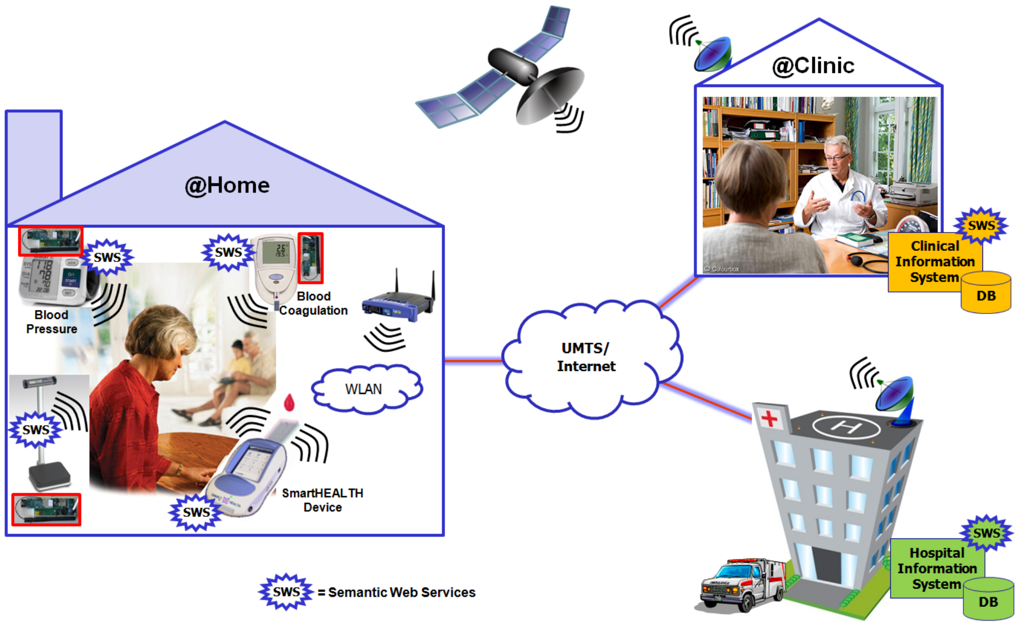


Figure 1.4: Post-operation Home-care Monitoring - A SmartHEALTH Project Scenario

blood coagulation, weight scale etc.) and offers its services as Semantic Web Services. As an *active* gateway device, the SmartHEALTH device semantically discovers other medical devices in the environment, as per the requirements of the healthcare application, collects the measurement values of a patient from these medical devices, makes a collective intelligent interpretation of all the measurement values using soft computing tools, and then forwards these measurement values with their interpretation to the remote hospital information system and/or doctor's clinical information system. As a *passive* gateway device, the SmartHEALTH device only acts as a *data forwarding* device for other medical devices in the environment. Whenever a medical device finishes its measurement, it semantically discovers the SmartHEALTH device as a gateway device, and uses its service to forward the measurement results to the respective information system(s). Finally, the patient's family doctor or health professional in the hospital analyzes the results of blood cancer markers, the measurement values of other medical devices, and their intelligent interpretation in terms of disease status (e.g. cancer's progression or regression) in order to discuss further steps with the patient.

1.2 Main Contributions

With this thesis, we make the following contributions to the area of semantic interoperability of ambient intelligent medical or mobile devices with e-Health systems.

- We present a lightweight but powerful knowledge querying and reasoning system, namely *Micro OWL Description Logic Reasoner* (μOR), which can be integrated easily on resource-constrained medical or mobile devices, and can be used not only for querying the device knowledge base, but also performs OWL description logic and/or *forward chaining* based rules reasoning.

  - We present a simple query language, namely *Semantic Device Language for N-Triples* (SCENT), which is an extension of the N-Triples specification and used to encode semantic queries with desired conditions and patterns.
  - We present a simple unification and resolution algorithm, namely *SCENT Resolution Algorithm* (SCENTRA), which is used not only to resolve the SCENT queries and return the results for the variables of conditions/patterns, but also used by μOR to generate inferences from the device knowledge base. However, one of the main goals of SCENTRA algorithm is to support *matchmaking* process of desired medical or mobile device(s) during the discovery phase.

- We present a lightweight HTTP-based semantic discovery protocol, namely *Semantic Medical Device Discovery Protocol* (SMDDP) to support the semantic discovery among medical or mobile devices, based on their *physical* characteristics (i.e. device vendor, device group/type) and/or *functional* characteristics (i.e. what services does a medical or mobile device offer).

- We have developed small ontologies, namely MeDO (*Medical Device Ontology*) and SmdsOnto (*Semantic Medical Device Space Ontology*) in order to express the physical and/or functional characteristics of a medical or mobile device receptively, as well as to encode the knowledge used in SmartHEALTH pervasive healthcare scenarios. However, it was not desired or attempted to develop these ontologies as *standard ontologies*, but only prototypes for our work.

- Encapsulating all the above-mentioned contributions, we present a SOA based decentralized middleware infrastructure, namely *Semantic Medical Devices Space* (SMDS) to provide a platform for the semantic coordination of medical or mobile devices. SMDS provides a mechanism to expose the functionalities of medical or mobile devices as Semantic Web Services and enables them to semantically *discover* and *match* the desired device(s), and invoke its/their Web Service method(s) as per the requirements a pervasive healthcare application.

1.3 Selected Publications

Various aspects of this thesis have been published in [22], [23], [24], [25], [26], [27], [28], [29], [30]. The most important publications are the following:

Semantic Medical Devices Space (SMDS)

In [22], we have presented the initial results of our SOA based decentralized middleware infrastructure, which encapsulates the SMDDP and μOR as integral components, and provides a platform for the semantic interoperability of medical or mobile devices (cf. Chapter 4).

- Safdar Ali, S. Kiefer. *Semantic Medical Devices Space: An Infrastructure for the Interoperability of Ambient Intelligent Medical Devices*, In Proc. of International IEEE/EMBS Conference on Information Technology in Biomedicine, Greece, 2006

Micro OWL DL and Rules based Querying and Reasoning System (μOR)

In [23], we have presented the architectural details of the first version of a lightweight but powerful Description Logics based querying and reasoning system, namely μOR for the resource-constrained medical or mobile devices (cf. Chapter 5). μOR makes use of SCENTRA algorithm, which is used to resolve the semantic queries as well as to generate inferences from the existing device knowledge base.

- Safdar Ali, Stephan Kiefer. *μOR - A Micro OWL DL Reasoner for Ambient Intelligent Devices*, In Proc. of 4th International IEEE Conference on Grid and Pervasive Computing, Geneva, Switzerland, LNCS 5529, pp 305-316, 2009

In the final version, in addition to the OWL DL reasoning, μOR was enhanced with *rules based* reasoning with *forward-chaining* mode only.

Semantic Medical Device Discovery Protocol (SMDDP)

In [24], we have presented a lightweight HTTP-based semantic discovery protocol with its application in the field of pervasive healthcare applications (cf. Chapter 6). SMDDP helps the resource-constrained medical or mobile devices to semantically discover the desired devices or even information systems based on their physical and/or functional characteristics in a pervasive healthcare environment.

- Safdar Ali, S. Kiefer. *Semantic Coordination of Ambient Intelligent Medical Devices - A Case Study*, In Proc. of ACM SIGCHI, IEEE, EMB International Conference on Pervasive Computing Technologies for Healthcare, London, U.K, 2009

As one of the best selected papers, an extended version of this paper has been published in [25] .

- Safdar Ali, Stephan Kiefer, *Semantic Coordination of Ambient Intelligent Medical Devices in Future Laboratories*, MASAUM Journal of Basic and Applied Sciences (MJBAS), Volume 1 Issue 2, September 2009

1.4 Thesis Outline

This thesis is organized as follows: Chapter 2 presents the background information about the theoretical concepts, tools and technologies used in this thesis. Chapter 3 gives an overview of the most relevant existing work regarding the use of semantic and SOA technologies in the field of ambient intelligence. Chapters 4 through 6 present the innovative contributions of this thesis. Chapter 4 presents the architectural details of *Semantic Medical Devices Space*, which provides a decentralized SOA based semantic platform for the interoperability of medical or mobile devices. Chapter 5 presents the architectural details of μOR, a lightweight but powerful description logics and forward-chaining rules based querying and reasoning system for the resource-constrained medical or mobile devices. This chapter also outlines the details of a simple query language, named SCENT that we have developed to express the semantic queries, as well as a unification and resolution algorithm, named SCENTRA that we have developed to resolve the SCENT queries and to generate the inferences. Chapter 6 outlines the details of a lightweight HTTP-based semantic discovery protocol, named *Semantic Medical Device Discovery Protocol* for the resource-constrained medical or mobile devices. Chapter 7 describes the implementation details by giving a quick overview of the used programming language(s), APIs and the development tools. Chapter 8 describes the experimental details of the scenarios where we have applied our research results, while Chapter 9 presents the conclusions and an outlook to the future work. At the end, Appendix A describes the UML class diagrams of software implementation, Appendix B describes the SCENT queries used for the performance analysis of μOR, while Appendix C describes the way to define SCENT rules for μOR with an example.

Chapter 2

Background

This chapter introduces some background and state-of-the-art work used for the research and development of this thesis. Section 2.1 gives a detailed overview of Web Services technology, its advantages and functional standards. Section 2.2 gives an overview of the Web Services based specification that is designed for resource-constrained devices, namely *Device Profile for Web Services.* Sections 2.3 and 2.4 explains the emerging Semantic Web and Semantic Web Services standards, their applications and development approaches. Section 2.5 gives introduction to the widely adopted communication protocols for general devices (non-medical). Section 2.6 gives introduction to most important medical healthcare communication standards, while Section 2.7 elaborates how these medical communication standards are used in the ICT infrastructures of hospitals and clinical environments. The last Section 2.8 explains in detail about the e-Health domain and its related issues.

2.1 Web Services

In recent years, distributed programming paradigms have been emerged that allow generic software components to be developed and shared. Although ideal for some enterprise integration and e-Commerce, it has only been with the adoption of XML as a common data syntax that the underlying principles have gained wide scale adoption, through the definition of Web Service standards. According to the definition of W3C, "*A Web Service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts*" [31].

Web Services are well defined, reusable, software components that perform specific, encapsulated tasks via standardized Web-oriented mechanisms. They can be discovered, invoked, and the composition of several services can be choreographed, using well defined workflow modeling frameworks. A Web Service supports direct interactions with other software applications (i.e. agents) using XML based messages exchanged via internet-based protocols[1]. The Web Services concept aims to adopt the principles of the WWW for its vision of seamless Application-to-Application (A2A) integration, regardless of differences in programming languages and platforms, featuring the same level of openness between loosely coupled systems. The key for the broad acceptance of the WWW is that is based on open standards and consequently, Web Services are also founded on the basis of such standards.

[1]W3C Web Services Activity, `http://www.w3.org/2002/ws/`

2.1.1 Key Benefits of Web Services

In this section, we discuss the key benefits of the Web Services to enlighten the reasons of their wide adoption in the industry [32].

Loosely Coupled

Unlike traditional application designs, which depend upon a tight interconnection of all program elements, Web Services are loosely coupled. Loose-coupling means that each service exists independently of the other services that make up the application. This allows individual pieces of the application to be modified without impacting unrelated areas. As a critical requirement for Service Oriented Architecture (SOA) design, loose-coupling is to be provided by Web Services through established standards that define services and how they interoperate.

Enables Service-Oriented Architectures

Web Services represent the convergence between the service-based development of applications and the Web. In the SOA model, the business processes that make up an application are separated into independent, easily distributed components known as services. These services interoperate across processes and machines to create a complete solution for a business problem. This loose-coupling allows for easy changes to the application by inserting new and revised services into the application without having modifying the unrelated services.

Ease of Integration

Unlike other methods of integration, Web Services are becoming widely adopted across the entire software industry. This broad industry adoption helps alleviate companies fears of proprietary technologies that may lock them in for the future. The standards surrounding Web Services are human-readable and publicly available, allowing a developer to view exactly what is happening in the system. Most often, integration between business partners is as easy as agreeing to a standard format for exchange of information defined in XML and WSDL.

Easily Accessible

Finally, Web Services are distributed over the Internet. Web Services make use of existing ubiquitous transport protocols like HTTP, leveraging existing infrastructure and allowing information to be requested and received in real time. Current IT infrastructure for addressing, security and performance can also be applied to Web Services applications.

2.1.2 Roles in the Web Services Architecture

The artifacts in the Web Services model are the objects that are produced and dealt within the context of Web Services. These objects include *Service*, *Service Description* and *Client Application* and the interaction among these object is shown in Fig. 2.1. A short description of these objects is given below:

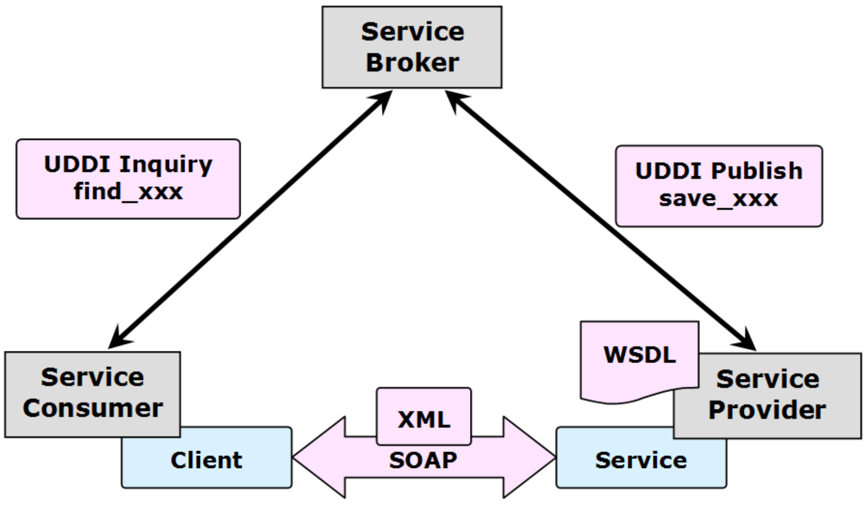


Figure 2.1: The SOA architecture with SOAP, WSDL, and UDDI

Service

The *service* is an implementation of a software module deployed on a network accessible platform provided by the service provider to be invoked by a service requester.

Service Description

The *service description* contains the details of the interface and implementation of the service. The service interface description comprises information about the *operations* and their *signatures* provided by a service; along with the protocol used for communication with the Web Service. The service implementation description contains information about the location where the service is exposed, i.e. the endpoint address of the service. The service provider publishes the complete service description to a *service registry* to make the service accessible to the service requestors. It includes the data types, operations, binding information and network location of the service, as provided by the service interface and implementation descriptions.

Client Application

This is the software application implemented by the *service requestor* to use the functionality of the Web Service by invoking its operations at runtime.

2.1.3 Functional Standards of Web Services

In the most Internet middleware configurations, the three core functional components, transport, description, and discovery in the Web Services architecture are implemented using SOAP, WSDL, and UDDI, respectively. A short description of each of the functional component is given below.

Simple Object Access Protocol (SOAP)

SOAP [33] is a standard that represents a lightweight envelope containing the message payload as it moves between service producers and consumers. It is an XML-based standard that describes the contents of a message and how to process it, and offers a transport binding for exchanging messages. Adjuncts to the envelope and binding framework include a set of encoding rules for expressing instances of application-defined data types and a convention for representing remote procedure calls and responses:

- SOAP Envelope: Describes the contents of a message and how to process it, and contains extra details such as security information or the final destination of the message.
- SOAP Transport Binding Framework: An abstract framework for exchanging SOAP envelops using an underlying protocol, including HTTP or other transports.
- SOAP Serialization Framework: A set of encoding rules for expressing instances of application-defined data types such as numbers and text.
- SOAP RPC Representation: A convention for representing remote procedure calls and responses.

Web Service Description Language (WSDL)

WSDL [34] is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Most often, these messages are bound to the SOAP protocol and the HTTP transport, but these are not the only set of bindings supported. The abstract nature of WSDL for describing services makes it very flexible for describing complex Web Services applications. A WSDL document uses the following elements in the definition of network services:

- Types: A container for data type definitions using some type system (such as XML Schema Definition, XSD).
- Message: An abstract, typed definition of the data being communicated.
- Operation: An abstract description of an action supported by the service.
- Port Type: An abstract set of operations supported by one or more endpoints.
- Binding: A concrete protocol and data format specification for a particular port type.
- Port: A single endpoint defined as a combination of a binding and a network address.
- Service: A collection of related endpoints.

Universal Description, Discovery and Integration (UDDI)

UDDI [35] represents a set of protocols and a public directory for the registration and real-time lookup of Web Services and other business processes. In many ways, UDDI models as White Pages, providing a listing of services available within a network. The primary benefit of a UDDI server is to provide a single point of reference to all available services within an enterprise. The UDDI server allows organizations to:

- Host multiple versions of a service
- Create aliases to services
- Limit access to specific services

While UDDI is a ratified standard, it is still gaining adoption and has been changing frequently. There are many other commercial products, known as *Web Service Management* tools, which perform many of the same tasks as UDDI while providing added benefits such as metering and inspection of the services. UDDI and Web Service Management tools help keeping the loose-coupled connection between services.

2.2 Devices Profile for Web Services

The *Devices Profile for Web Services* (DPWS) [36] was developed to enable secure Web Service capabilities on resource-constraint devices. It allows sending secure messages to and from Web Services, dynamically discovering a Web Service, describing a Web Service, subscribing to, and receiving events from a Web Service. DPWS can be used for *machine-to-machine* communication whereas a specific client uses a specific service hosted on a device. DPWS is not the first SOA that targets device-to-device communication. The technologies such as OSGi[2] (*Open Service Gateway Initiative*), HAVi[3](*Home Audio/Video Interoperability*), JINI[4] (*Java Intelligent Network Infrastructure*) and UPnP[5] (*Universal Plug and Play*) are similar approaches. The big advantage of DPWS compared to all other mentioned SOAs is the reliance on Web Service which implies high acceptance among developers and platform as well as programming language independence. Microsoft has included DPWS in their latest operating system called Windows Vista.

2.2.1 The Underlying Protocols of DPWS

DPWS is partially based on the Web Services architecture and uses further standards and draft specifications from the Web Services protocol family, as shown in Fig. 2.2 [37]. SOAP, WSDL and XML-Schema have already been explained in the previous sections and thus will not be further explained here. A brief description of other protocols is given below, but a detailed description can be found in [36].

WS-Policy

WS-Policy provides a framework for expressing different capabilities, requirements and characteristics for different service implementations. If a Web Service offers policies, service users have to comply with the declarations found in the policy document. The service users have to choose one of the offered policy alternatives which consist of several policy assertions from the policy document. The policies mostly define QoS (*Quality of Service*) characteristics and security considerations which are necessary for service communication.

[2]OSGi; `www.osgi.org/`
[3]HAVi; `http://www.havi.org/`
[4]JINI; `www.jini.org/`
[5]UPnP; `www.upnp.org`

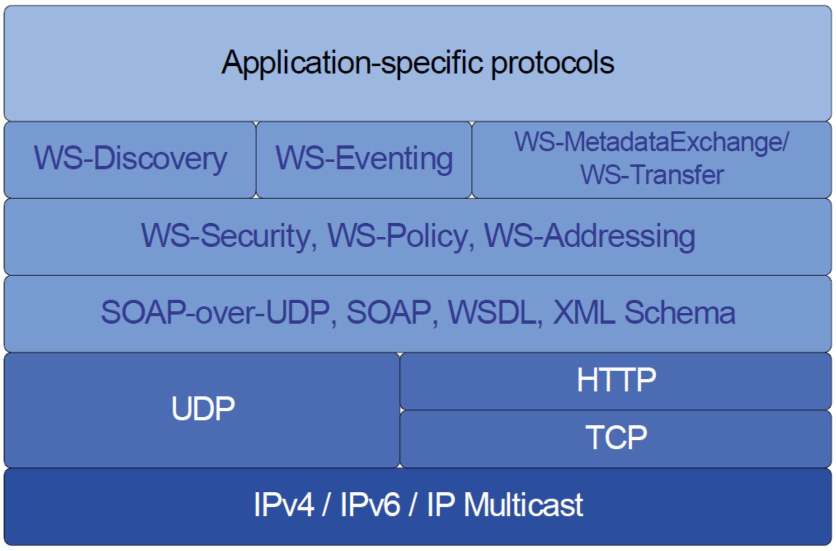

Application-specific protocols
WS-Discovery
WS-Eventing
WS-MetadataExchange/ WS-Transfer
WS-Security, WS-Policy, WS-Addressing
SOAP-over-UDP, SOAP, WSDL, XML Schema
UDP
HTTP
TCP
IPv4 / IPv6 / IP Multicast

Figure 2.2: Devices Profile for Web Services as protocol stack

WS-Addressing

The main objective of *WS-Addressing* is to provide an addressing mechanism for Web Services as well as messages in a *transport-neutral* matter. By introducing both concepts *endpoint references* and *message information headers*, WS-Addressing overcomes the lack of SOAP's independence of underlying transport protocols (in most cases HTTP) and secondly support of asynchronous message exchange. Both limitations are historically caused by the default SOAP to HTTP binding.

SOAP-over-UDP

The main objective of *SOAP-over-UDP* is to decouple SOAP from its underlying binding protocol, and define a binding for SOAP envelopes to UDP. In contrast to TCP the delivery of UDP packets can not be guaranteed. SOAP-over-UDP uses WS-Addressing to support the same message exchange patterns as SOAP-over-HTTP. Unicast as well as multicast transmission is supported. Since UDP's unreliability message retransmission is encouraged, and for efficiency reasons a retransmission algorithm is provided which considers delayed repetitions of the same message.

MTOM

The idea of SOAP *Message Transmission Optimization Mechanism* (MTOM) is to provide an abstract feature for optimizing the transmission of SOAP messages. With MTOM parts of the envelope can be encoded outside of the envelope while still keeping the envelope intact. MTOM defines its feature in an abstract matter which means it has to be adapted to the protocols used beneath. The second part of this specification provides a binding for HTTP based on *XML-binary Optimize Packaging* convention.

WS-Discovery

The *WS-Discovery* is a discovery protocol based on IP multicast for enabling services to be discovered automatically. Discovery introduces three different endpoint types: target service, client and discovery proxy. *Target Services* are Web Services offering themselves to the network, *Clients* may search for target services and discover them dynamically and *Discovery Proxy* is an endpoint enabling discovery in spanned networks since simple discovery is limited to a multicast group and hence to local managed networks only.

WS-MetadataExchange / WS-Transfer

WS-MetadataExchange is a specification that defines data types and operations to retrieve metadata associated with an endpoint. This metadata describes what other endpoints need to know to interact with the described endpoint, and defines the MetadataSection that divides the metadata into separate units of metadata with a dialect specifying its type. *WS-Transfer* is used to retrieve the metadata, which is structured as specified in WS-MetadataExchange. There is a slight functional difference in WS-MetadataExchange and WS-Transfer for retrieval of metadata. WS-MetadataExchange defines operations to retrieve *all or parts* of the metadata of an endpoint. WS-Transfer only can be used to retrieve *all* metadata of an endpoint. WS-Transfer is very similar to HTTP since it defines Create, Get, Put and Delete operations which have almost the same semantics as HTTP request methods.

WS-Eventing

WS-Eventing defines a protocol for managing subscriptions for a Web Services based eventing mechanism. This protocol defines three endpoints: subscriber, event source and subscription manager. *Subscribers* request subscriptions on behalf of event sinks to receive events from event sources. Subscription requests contain an event delivery mode and event filter mechanism to negotiate an *event source* with an event sink. *Subscription Managers* are responsible of holding subscriptions of event sources. Subscriptions must be requested and can expire over time. The subscription manager can be asked to renew or end a subscription, or to get the status of a subscription. There are no limitations or restrictions in supporting other or user-defined event delivery and filter mechanisms.

WS-Security

WS-Security specification provides mechanisms for message integrity and confidentiality to SOAP messaging. These mechanisms are independent of the used technology. The specification defines the structures security header and security token to allow signing full or parts of SOAP messages or encryption of SOAP messages.

While the WS-* specifications define abstract mechanisms, WS profiles define specific constraints and limitations. So the DPWS describes how some specific WS specifications can be used on embedded devices.

2.3 Semantic Web

The Semantic Web [38] is a vision of a Web of meaningful contents and services, which can be interpreted by computer programs. It can also be seen as a vast source of information, which can be modeled with the purpose of sharing and reusing knowledge.

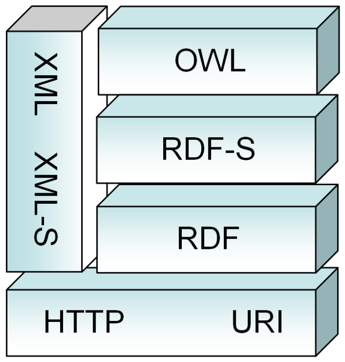


Figure 2.3: General stack of Semantic Web enabling standards

The Semantic Web users will be able to do more accurate searches of the information and the services they need from the tools provided. The Semantic Web provides the necessary infrastructure for publishing and resolving ontological descriptions of terms and concepts. In addition, it provides the necessary techniques for reasoning about these concepts, as well as resolving and mapping between ontologies, thus enabling semantic interoperability of Web Services through the identification and mapping of semantically similar concepts. Fig. 2.3 shows the general stack of Semantic Web enabling standards. *Ontologies* have been developed within the Semantic Web research community in order to facilitate knowledge sharing and reuse. They provide greater expressiveness when modeling domain knowledge and can be used to communicate this knowledge between people and heterogeneous and distributed application systems. As with Web Services, Semantic Web enabling standards fit into a set of layered specifications built on the foundation of URIs and XML Schema. The current components included in the Semantic Web framework are RDF [39], RDF Schema (RDF-S) [40] and the Web Ontology Language (OWL) [41]. These standards build up a rich set of constructs for describing the semantics of online information sources.

RDF is an XML-based standard from W3C for describing resources on the Web. RDF introduces a little semantics to XML data by allowing the representation of objects and their relations through properties. **RDF-Schema** is a simple type system, which provides information (metadata) for the interpretation of the statements given in RDF data. The **OWL** facilitates greater machine interpretability of Web contents than RDF and RDF Schema by providing a much richer set of constructs for specifying classes and relations. OWL has evolved from existing ontologies languages and specifically from DAML+OIL [42]. OWL comes in several variants, that are OWL-Full, OWL-DL and OWLLite, where each variant corresponds to a description logic of different expressivity and complexity. OWL-Lite and OWL-DL are an abstract syntactic form of the description logic $\mathcal{SHIF}(\mathcal{D})$ and $\mathcal{SHOIN}(\mathcal{D})$, respectively, whereas OWL-Full corresponds to the description logic $\mathcal{SHOIQ}(\mathcal{D})*$. For syntax and model-theoretic semantics of these description logics, we refer to [43].

2.4 Semantic Web Services

Whilst promising to revolutionize e-Commerce and enterprise-wide integration, current standard technologies for Web Services, i.e. WSDL provide only *syntactic-level* descriptions of their functionalities, without any formal definition to what the syntactic definitions might mean. In many cases, Web Services offer little more than a formally defined invocation interface, with some human oriented meta-data that describes what the service does, and which organization developed it, i.e. through UDDI [35] descriptions. The software applications may invoke Web Services using a common, extendible communication framework, i.e. SOAP [33]. However, the lack of *machine-readable semantics* necessitates human intervention for automated service discovery and composition within open systems, thus hampering their usage in complex business contexts.

Semantic Web Services (SWS) [44] relax this restriction by augmenting Web Services with rich formal descriptions of their capabilities, thus facilitating automated composition, discovery, dynamic binding, and invocation of services within an open environment. A prerequisite to this, however, is the emergence and evolution of the Semantic Web [38], which provides the infrastructure for the semantic interoperability of Web Services. Web Services will be augmented with rich formal descriptions of their capabilities, such that they can be utilized by software applications or other services without (or less) human assistance or highly constrained agreements on interfaces or protocols. Thus, SWSs have the potential to change the way knowledge and business services are consumed and provided on the current Web.

2.4.1 Why Semantic Web Services?

Semantic descriptions of Web Services are necessary in order to enable their automatic discovery, composition and execution across heterogeneous users and domains. Present technologies for Web Services provide descriptions only at the syntactic level, which makes it difficult for the requesters and providers to interpret or represent nontrivial statements, i.e. the meaning of inputs and outputs or applicable constraints. This limitation may be relaxed by providing a rich set of semantic annotations that augment the service description. A Semantic Web Service is defined through a service ontology, which enables machine interpret-ability of its capabilities as well as integration with domain knowledge. The deployment of Semantic Web Services will rely on the further development and combination of Web Services and Semantic Web enabling technologies. Several initiatives have been started in the industry and academia, e.g. DIP [45], SWSI [46] which are investigating solutions for the main issues regarding the infrastructure for SWS. Fig. 2.4 shows the following three orthogonal dimensions as characterization of Semantic Web Service infrastructure:

- *Usage activities* define the functional requirements, which a framework for Semantic Web Services ought to support.
- *Architecture* of SWSs describes the components needed for accomplishing the activities defined for SWS.
- *Service ontology* aggregates all concept models related to the description of a Semantic Web Service and constitutes the knowledge-level model of the information describing and supporting the usage of the service.

Usage Activities

From the usage activities perspective, SWS are seen as objects within a business application execution scenario. The activities required for running an application using SWS

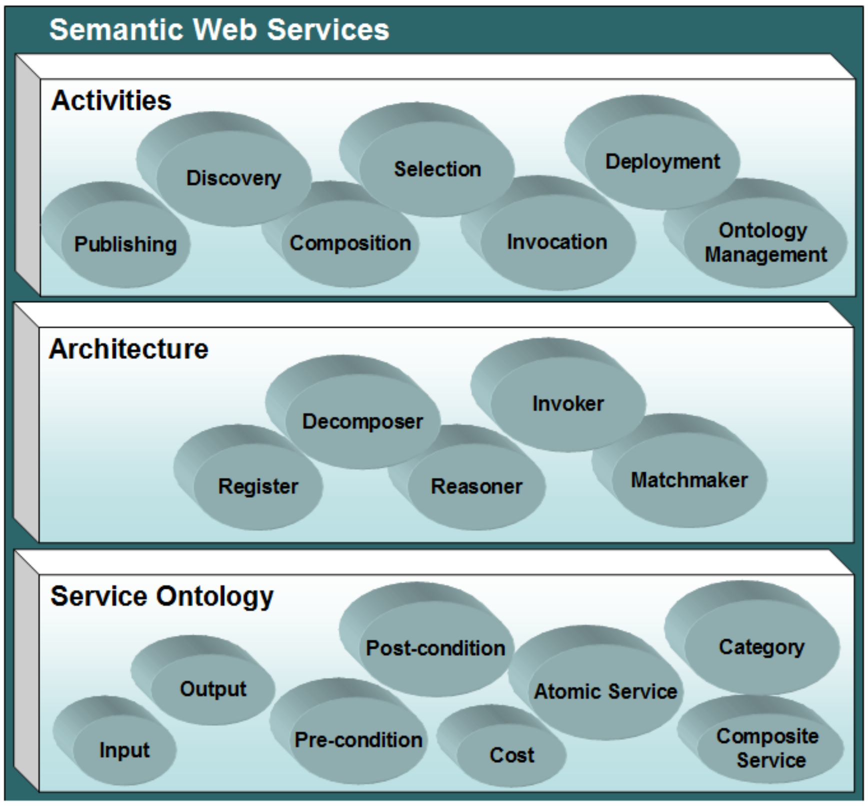


Figure 2.4: Semantic Web Services Infrastructure Dimensions

include: publishing, discovery, selection, composition, invocation, deployment and ontology management, as described next. The *publishing* or *advertisement* of SWS will allow agents or applications to discover services based on its goals and capabilities. A *semantic registry* is used for registering instances of the service ontology for individual services. The *service ontology* distinguishes between information which is used for matching during the discovery and that is used during service invocation. In addition, domain knowledge should also be published or linked to the service ontology. The *discovery* of services consists of a semantic matching between the description of a service request and the description of published service. The queries involving the service name, input, output, preconditions and other attributes can be constructed and used for searching the semantic registry. The *matching* can also be done at the level of tasks or goals to be achieved, followed by a selection of services which solves the task. The degree of matching can be based on some criteria, such as the inheritance relationship of types. For example, an input of type **Infusion Pump** of a provided service can be said to match an input of type **Medical Device** of a requested service.

A *selection* of services is required if there is more than one service matching the request. Non-functional attributes such as cost or quality can be used for choosing one service. In a more specialized or agent-based type of interaction a negotiation process can be started between a requester and a provider, but that requires that the services themselves be knowledge-based. In general, a broker would check that the preconditions of tasks and services are satisfied and prove that the services post-conditions and effects imply goal accomplishment. An explanation of the decision making process should also be provided. The *Composition* or *choreography* allows SWS to be defined in terms of other simpler services. A workflow expressing the composition of atomic services can be defined in the service ontology by using appropriate control constructs. This description would be grounded on a syntactic description such as BEPL4WS [47]. Dynamic composition is also being considered as an approach during service request in which the atomic services required to solve a request are located and composed on the fly. That requires an invoker which matches the outputs of atomic services against the input of the requested service.

The *invocation* of SWS involves a number of steps, once the required inputs have been provided by the service requester. First, the service and domain ontologies associated with the service must be instantiated. Second, the inputs must be validated against the ontology types. Finally the service can be invoked or a workflow executed through the grounding provided. It is also important to monitor the status of the decomposition process and notify the requester in case of exceptions. The *deployment* of a Web Service by a provider is independent of the publishing of its semantic descriptions since the same Web Service can serve multiple purposes. Also, the SWS infrastructure can provide a facility for the instant deployment of code for a given semantic description. The *management* of service ontologies is a cornerstone activity for SWS since it will guarantee that the semantic service descriptions are created, accessed and reused within the Semantic Web.

Architecture

From the *architecture* perspective, as shown in Fig. 2.4, SWSs are defined by a set of components which realize the activities above, with underlying security and trust mechanisms. The components gathered from the discussion above include: a register, a reasoner, a matchmaker, a decomposer and an invoker. The *reasoner* is used during all activities and provides the reasoning support for interpreting the semantic descriptions and queries. The register provides the mechanisms for publishing and locating services in a semantic registry as well as functionalities for creating and editing service descriptions. The *matchmaker* mediates between the requester and the register during the discovery and selection of services. The *decomposer* is the component required for executing the composition model of composed services. The invoker mediates between requester and provider or decomposer and provider when invoking services. These components are illustrative of the required roles in the SWS architecture for the discussion here as they can have different names and a complexity of their own in different approaches.

Service Ontology

The *service ontology* is another dimension under which we can define the SWS, for it represents the *capabilities* of a service itself and the *restrictions* applied to its use. The service ontology essentially integrates at the knowledge-level the information which has been defined by Web Services standards, such as UDDI and WSDL with related domain knowledge. This would include: *functional* capabilities such as inputs, output, pre-conditions and post-conditions; and *non-functional* capabilities such as category, cost and quality of service; *provider related* information, such as company name and address; *task or goal-related* information; and *domain knowledge* defining, for instance, the type of the inputs of the service. This information can, in fact be divided in several ontologies. However, the service ontology used for describing SWS will rely on the expressivity and inference power of the underlying ontology language supported by the Semantic Web.

2.4.2 Semantic Web Services Description Frameworks

Although there exist several approaches for the description of Semantic Web Services, the leading frameworks used for the description of Semantic Web Services (SWS) are the standard SAWSDL [48], OWL-S [49] and WSML [50]. In the following sections, we briefly describe these approaches by taking the text snippets from [43], and refer the reader to this for detailed description.

SAWSDL

The standard language WSDL for Web Services operates at the mere syntactic level as it lacks any declarative semantics needed to meaningfully represent and reason upon them by means of logical inferencing. In a first response to this problem, the W3C Working Group on Semantic Annotations for WSDL and XML Schema (SAWSDL) [48] developed mechanisms with which semantic annotations can be added to WSDL components.

Unlike OWL or WSML, SAWSDL does not specify a language for representing formal ontologies but provides mechanisms by which ontological concepts that are defined outside WSDL service documents can be referenced to semantically annotate WSDL description elements. Based on its predecessor and W3C member submission WSDL-S [51] in 2005, the key design principles for SAWSDL are that (a) the specification enables semantic annotations of Web Services using and building on the existing extensibility framework of WSDL; (b) it is agnostic to semantic (ontology) representation languages; and (c) it enables semantic annotations for Web Services not only for discovering Web Services but also for invoking them (their grounding).

Based on these design principles, SAWSDL defines the following three new extensibility attributes to WSDL 2.0 elements for their semantic annotation:

- An extension attribute, named **modelReference**, to specify the association between a WSDL component and a concept in some semantic (domain) model. This modelReference attribute is used to annotate XML Schema complex type definitions, simple type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults.

- Two extension attributes, named **liftingSchemaMapping** and **loweringSchemaMapping**, that are added to XML Schema element declarations, complex type definitions and simple type definitions for specifying mappings between semantic data in the domain referenced by modelReference and XML. These mappings can be used during service invocation.

One problem with SAWSDL is that it comes as a mere syntactic extension of WSDL, without any formal semantics. In opposite to OWL-S and (in part) WSML, there is no defined formal grounding of neither the XML-based WSDL service components nor the referenced external metadata sources (via modelReference). Another problem with SAWSDL today is its very limited software support. Notable exceptions are the implemented SAWSDL service discovery and composition planning means of the METEOR-S framework [52]. However, the recent announcement of SAWSDL as a W3C recommendation not only supports a standardized evolution of the W3C Web Service framework in principle (rather than a revolutionary technology switch to far more advanced technologies like OWL-S or WSML) but will push software development in support of SAWSDL and reinforce research on refactoring these frameworks with respect to SAWSDL.

OWL-S

OWL-S [49] is an upper ontology used to describe the semantics of services based on the W3C standard ontology OWL and is grounded in WSDL. Its approach originated from an Artificial Intelligence background and has previously been used to describe *agent* functionality within several multi-agent systems as well as with a variety of planners to solve higher level goals. It consists of *three* main upper ontologies: the *Profile*, *Process Model* and *Grounding*.

The **Profile** is used to describe services for the purposes of discovery; service descriptions and queries are constructed from a description of functional properties (i.e. inputs, outputs, preconditions, and effects - IOPEs), and non-functional properties (human oriented properties such as service name, etc, and parameters for defining additional meta data about the service itself, such as concept type or quality of service). In addition, the profile class can be sub-classed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

The **Process** Models describe the composition or orchestration of one or more services in terms of their constituent processes. This is used both for reasoning about possible compositions (such as validating a possible composition, determining if a model is executable given a specific context, etc) and controlling the enactment/invocation of a service. Three process classes have been defined: the composite, simple and atomic process. The *atomic* process is a single, black-box process description with exposed IOPEs. Inputs and Outputs relate to data channels, where data flows between processes. The preconditions specify facts of the world that must be asserted in order for an agent to execute a service. Effects characterize the facts that become asserted given a successful execution of the service, such as the physical side-effects that the execution the service has on the physical world. The **simple** process provides a means of describing service or process abstractions, which means that such elements have no specific binding to a physical service, and thus have to be realized by an atomic process (e.g. through service discovery and dynamic binding at run-time), or expanded into a composite process. The **composite** processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. These process workflows are constructed using a number of different composition constructs, including: $\mathsf{Sequence}$, $\mathsf{Unordered}$, Choice, $\mathsf{If-then-else}$, $\mathsf{Iterate}$, $\mathsf{Repeat-until}$, $\mathsf{Repeat-while}$, Split, and $\mathsf{Split+join}$. The profile and process models provide semantic frameworks whereby services can be discovered and invoked, based upon conceptual descriptions defined within Semantic Web (i.e. OWL) ontologies.

The **Grounding** provides a pragmatic binding between this concept space and the physical data/machine/port space, thus facilitating service execution. The process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types. Additional properties pertaining to the binding of the service are also provided (i.e. the IP address of the machine hosting the service, and the ports used to expose the service).

WSML

The Web Service Modeling Framework (WSMF) [50] provides a conceptual model and a formal language WSML (Web Service Modeling Language) for the semantic markup of Web Services together with a reference implementation WSMX (Web Service Execution Environment). Its main goal is to fully enable e-Commerce by applying Semantic Web technology to Web Services. WSMF is the product of research on modeling of reusable knowledge components. WSMF is centered on two complementary principles: a *strong de-coupling* of the various components that realize an e-commerce application; and a *strong mediation* service enabling Web Services to communicate in a scalable manner. The mediation is applied at several levels: mediation of data structures; mediation of business logics; mediation of message exchange protocols; and mediation of dynamic service invocation. WSMF consists of four main elements: *ontologies* that provide the terminology used by other elements; *goal repositories* that define the problems that should be solved by Web Services; *Web Services descriptions* that define various aspects of a Web Service; and *mediators* which bypass interoperability problems.

WSMF implementation has been assigned to two main projects: *Semantic Web enabled Web Services* (SWWS) [53], and *Web Service Modeling Ontology* (WSMO) [54]. SWWS provides a description framework, a discovery framework and a mediation platform for Web Services, according to a conceptual architecture. WSMO refines the Web Services Modeling Framework and develops a formal service ontology and language for SWS. WSMO Service Ontology includes definitions for goals, mediators and Web Services. The underlying representation language for WSMO is *F-logic* because it is a full *first order logic* language that provides second order syntax while staying in the first order logic semantics, and has a minimal model semantics. The main characterizing feature of the WSMO architecture is that the goal, Web Service and ontology components are linked by four types of mediators as follows:

- **OO** mediators link ontologies to ontologies
- **WW** mediators link Web Services to Web Services
- **WG** mediators link Web Services to goals
- **GG** mediators link goals to goals.

WSML allows to describe a SWS in terms of its functionality (service capability), imported ontologies, and the interface through which it can be accessed for orchestration and choreography. The syntax of WSML is mainly derived from F-Logic extended with more verbose keywords (e.g., "hasValue" for ->, "p memberOf T" for p:T etc.), and has a normative human-readable syntax, as well as an XML and RDF syntax for exchange between machines. WSML comes in five variants with respect to the logical expressions allowed to describe the semantics of service and goal description elements, namely WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full.

Though WSML has a special focus on annotating Semantic Web Services like OWL-S, it tries to cover more representational aspects from *knowledge representation* and *reasoning* under both classical FOL and nonmononotic LP semantics. For example, **WSML-DL** is a decidable variant of F-Logic(FO) with expressivity close to the description logic $\mathcal{SHOIN}(\mathcal{D})$, that is the variant OWL-DL of the standard ontology Web language OWL. **WSML-Flight** is a decidable Datalog variant of F-Logic(LP) (function-free, non-recursive and DL-safe Datalog rules) with (nonmonotonic) default negation under perfect model semantics of locally stratified F-Logic programs with ground entailment. **WSML-Rule** is a fully-fledged logic programming language with function symbols, arbitrary rules with inequality and nonmontonic negation, and meta-modeling elements such as treating concepts as instances, but does not feature existentials, strict (monotonic) negation, and equality reasoning. The semantics of WSML-Rule is defined through a mapping to undecidable (nonmonotonic, recursive) F-Logic(LP) variant with inequality and default negation under well-founded semantics [368]. **WSML-Full** shall unify the DL and LP paradigms as a superset of FOL with non-monotonic extensions to support nonmonotonic negation of WSML-Rule via Default Logic, Circumscription or Autoepistemic Logic. However, neither syntax nor semantics of WSML-Full have been completely defined yet.

2.5 General Device Communication Protocols

This section gives a comparative overview to the protocols or standards that are used for the communication of resource-constrained general (non-medical) devices [55].

2.5.1 Universal Plug and Play

Universal Plug and Play (UPnP) [56] is an industry initiative by Mirosoft to provide simple, robust, peer-to-peer connectivity among devices and PCs. Plug and Play (PnP) made it easier to setup, configure, and add peripherals to a PC, while the purpose of UPnP is to extend this simplicity throughout the network, enabling discovery and control of devices. Its target is *zero configuration*, *invisible networking*, as well as *automatic discovery* for the devices of many vendors. UPnP encompasses many existing, as well as new scenarios, such as home automation, printing, audio/video entertainment, kitchen appliances, etc., and is independent of operating systems, programming languages, or physical media.

The basic components of a UPnP network are: *devices*, *services*, and *control points*. A UPnP device contains services and nested devices. An XML device description document is hosted by each device; this document lists the set of services and properties of the device. A service is the smallest unit of control in UPnP. It is described in the XML document of the device, which also contains a pointer to the service description. A service consists of a state table, a control server, and an event server. The state table controls the state of the service through state variables. The control server updates the state according to action requests. The event server notifies interested subscribers when the service state changes. A controller is capable of discovering and controlling other devices. For true peer-to-peer functionality, devices should incorporate control point functionality. There is no central registry in UPnP, at least not necessarily.

UPnP uses many existing, standard protocols, so that UPnP devices can fit seamlessly and without effort into the existing networks. TCP/IP is the base on which the UPnP protocols are built, as well as many protocols that go with it, such as UDP, ARP, DHCP and DNS. HTTP is a core part of UPnP and its aspects are based on HTTP or its variants. The *Simple Service Discovery Protocol* (SSDP) defines how to find network services. It is both for control points to locate services on the network, as well as for devices to announce their availability. A UPnP control point, when booting up, can send a search request to discover devices and services. UPnP devices, on the other hand, listen on the *multicast* port. If the search criteria match, a *unicast* reply is sent. In the same way, a device, when plugged in, will send out multiple SSDP presence announcements. Apart from the leasing concept that UPnP also shares, SSDP provides a way for a device to notify that it is leaving the network.

The Generic Event Notification Architecture (GENA) defines the concepts of *subscribers* and *publishers* of notifications. The GENA formats are used to create these announcements that are sent using SSDP. SOAPis used in UPnP to execute remote procedure calls, as well as to deliver control messages and return results or error messages to the control points. XML is used in UPnP in device and service descriptions, control messages and eventing. The advertisement message that a device issues contains a URL that directs to an XML file in the network, which describes the capabilities of the device. Each UPnP device must be a DHCP client and search for a DHCP server when connected to a network. In the lack of a DHCP server it must use Auto IP to get an address: the device randomly chooses an address, and then makes an ARP request to see if it is already occupied. This mechanism minimizes administration requirements.

2.5.2 Jini

Jini [57] is a network architecture for distributed systems, developed by Sun Microsystems in Java. The aim of this architecture is to make the network *dynamic* and *self-administered*, where services are *added* and *deleted* in a flexible manner. More precisely, the purpose is to end up with a *monolithic* system where users are able to not only share services and resources in a network, but have easy access to the services, even though the user's location may change. In a way, Jini can be considered as an extension of the Java application environment from a single machine to the whole network. Jini is based on Java environment, which offers built-in security for executing code from another machine and adds functionality on top of that to support the moving of components in a distributed system, as compared to the easy movement of objects in a Java application environment.

Jini makes the assumption that the devices connected to the network have a certain memory capacity and processing power. The limitation on the devices that can be connected directly to the network is a Java legacy; the devices need to have the JVM. Other devices not meeting this criteria need to be presented to the network by means of a proxy, which is a piece of hardware and/or software that meets the above-mentioned requirements. The service proxy has the drawback that in order to have no need for drivers, manufacturers must agree to a common interface. This is hard to achieve for every kind of device, and is exacerbated by the fact that each device tends to encompass multiple and different functionalities. There is benefit from the JVM: it makes Jini platform independent, but the JVM is heavy for hand-held devices and embedded systems. As for Java, Jini depends on the Java application environment, rather than on the Java programming language. Any language, as claimed, producing compliant *bytecode* can be used. However, practically, only the Java language is used.

Services are crucial to the Jini architecture. They can be used by a person, a program, or another service. Some examples of services are: storage, a computation, a hardware device, a user, devices such as printers, software such as applications, information such as databases. A Jini system is made up of services that can be collected in order to complete a particular task. A service can use another service, a client may be a service for another client. A service protocol is used for the communication between services. The one Jini offers is a set of interfaces that define critical service interaction between services. It can be extended further. A lookup service is used to find and resolve services. What it does is a mapping of the interfaces that indicate the functionality of a service to sets of objects implementing the service. A service has to be registered in at least one lookup service. A Jini service provider registers itself with every discovered lookup service. An object can itself consist of other services, this makes room for hierarchical services and lookup. Furthermore, objects may also be the encapsulation of other naming or directory services. References to the Jini lookup system can also be placed in other naming and directory services. In this way, bridging between the Jini lookup system and other forms of lookup service is created.

The discovery, join and lookup protocols are the heart of the Jini architecture. The discovery protocol is used when a device looks for a lookup service to register with. The join protocol is used when a service has located the lookup service and wants to join it. The lookup protocol is used when a client needs to locate and invoke a service. Discovery/join is how a service is added to the system. The service provider multicasts a request for local lookup services to identify themselves (discovery). After this, a service object is loaded into the lookup service (joining). This service object has the Java interface for the service including the methods of the interface to be invoked for using the service. There can also be other descriptive attributes. The lookup process ensures that a copy of the service object is loaded into the client. The client can now use the service. Java *Remote Method Invocation* (RMI) can be used for the communication between services. RMI basically implements the remote procedure call mechanisms in Java. It allows data as well as objects to be passed through the network.

With Jini, administration is still needed, because when a device is introduced to the network, it needs to be assigned an IP. Jini supports distributed events: an object can register its interest in events of another object and be notified whenever these events happen. This adds reliability guarantees to the architecture. Interfaces to devices have to be implemented for the Jini architecture to work. This is a disadvantage for Jini, since other service discovery protocols have them already available for consumer electronics.

2.5.3 SLP

SLP [58] is an IETF standard for service discovery. It enables the discovery and selection of a wide range of services accessible through an IP network. With SLP, a user needs only to search for a particular type of service, and optionally for attributes associated with it. It has three major software entities: *User Agent* (UA), *Service Agent* (SA), and *Directory Agent* (DA).

The SA advertises the location and attributes of one or more services on behalf of them by using broadcasts. It also replies with IP unicast to requests for these services. The services register and deregister with the SA. Each registrations has a lifetime and reregistration is required periodically. It is the same leasing concept already discussed in previous protocols.

The UA receives requests from a client application and forwards multicast requests to SAs. Hence, there is little network overhead for SA discovery. The SA unicasts a response back with the service URL and possible attributes if there is a match with the registered services.

The DA is a central information repository and it is optional. Its main function is to improve the performance of SLP. It can be thought of as a tier between the UAs and the SAs, which communicate with the DA instead of with each other. DAs relieve the network traffic from many multicast requests; the effect is more visible in large network, where multicast traffic can increases sharply because there are many SAs and UAs. A DA keeps the SA advertisements, and responds to UA requests. An SA registers itself with a DA. The registration contains the URL for the services, the lifetime, and descriptive attributes of the service. The registration should be refreshed periodically. A UA sends a request message to the DA, which in turn sends a message containing the URL of the service matched against the UA needs. SLP can work both with and without a DA: (1) with DA, information about the services is kept in the DA, so that the UA sends a query there for services; (2) without a DA, the UA multicasts service requests to the network, and a SA offering the service would reply back. In small networks, there may be no real need for a DA.

SLP scales well in large networks; the reason is the minimal use of multicast messages and the fact that it can have multiple DAs. It has a flexible and scalable architecture, where service browsing and human interaction is possible. SLP offers filtered search for attributes and predicates, such as AND, OR, comparators, and substring matching. SLP shares the concept of leasing with Jini and UPnP. Services can be deployed in small networks without any special configuration or deployment. It works even if there is no DNS, DHCP, SLP DA, or routing. Therefore, home networks would benefit much from the automation of service discovery, because they often lack network administration. However, the architecture with DAs makes the system vulnerable to a single point of failure. SLP is an open source, vendor independent and already implemented. It has the advantage of not depending on any programming language.

2.5.4 Bluetooth SDP

Bluetooth is a transmission technology, meant for short-range (10m) wireless communication between low-power devices. Bluetooth devices form a personal area network, a piconet, with a maximum of 8 members. Groups of piconets communicating with eachother form a scatternet. Every Bluetooth SDP device has to implement a SDP server that provides services. The server has a list of service records, each with a list of attributes that represent different service classes. A Bluetooth SDP client sends a request message with the list of service classes.

Bluetooth is designed for Bluetooth environments, therefore, it offers limited functionality compared to other SDPs. Its functionalities are: search for services by service type; search for services by service attributes; and service browsing without a priori knowledge of the service characteristics. The Bluetooth SDP does not include functionality for accessing services. After services have been discovered with it, the selection, access, and usage can be done with mechanisms out of the scope of SDP, for example by other SDPs such as SLP and Salutation. SDP can coexist with other SDPs, but they are not a necessity.

The strong point of Bluetooth in the home environment is the lack of wires, while the short range it provides is a disadvantage. Also, it has a peer-to-peer connectivity, which does not scale well, because typically the systems lack resources. Another disadvantage is the lack of event notification when services become unavailable. The Bluetooth security mechanisms offer either one-way, two-way, or no authentication. When authentication is used, the Bluetooth devices generate a secure connection with a pairing process that makes use of *PIN codes* entered by users. But for home environment usage, maybe security and privacy have to be addressed also in higher layers. Bluetooth has huge implementation opportunities in home environments. Apart from cell phones and PCs which have already been implemented, there are other potentials. Home automation, for example, could replace doorbells. This would eliminate unnecessary wires around the place. Other goods, such as toys or entertainment devices, could be another area of implementation.

2.6 Medical Device Communication Protocols

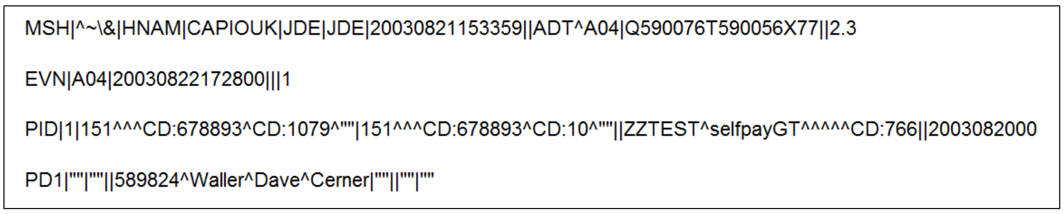
In this section, the most widely used state-of-the-art communication standards for the transmission of medical data among the medical devices and different information systems in the hospitals and laboratories are explained. These communication standards include HL7, ASTM, DICOM and CEN/ISO/IEEE 11073. Additionally, a brief description of other relevant medical communications standards is also given, i.e. EDI, and EDIFACT.

2.6.1 HL7 Standards

HL7 [59] is a standard for the information exchange between medical applications. It is an abbreviation of *Health Level Seven*, 7th OSI layer protocol for the health environment. HL7 is a protocol for data exchange, which defines the format and the content of the messages that applications must use when exchanging data with one another in various circumstances. HL7 specifies a *transport-independent* messaging framework and structure that enables disparate healthcare information systems to exchange data.

The Components of an HL7 Message

This section describes about different components of a typical HL7 message. In Fig. 2.5, a typical HL7 ADTÂ04 message is shown, which is sent when a new patient arrives at the hospital. The patient's demographics are entered into hospital information system and then the information is communicated to all the other systems to avoid multiple entries of the patient's demographic information. HL7 messages are ASCII messages and defined as sequence of segments and/or segment groups. Each segment, group, or



```
MSH|^~\&|HNAM|CAPIOUK|JDE|JDE|20030821153359||ADT^A04|Q590076T590056X77||2.3

EVN|A04|20030822172800|||1

PID|1|151^^^CD:678893^CD:1079^""|151^^^CD:678893^CD:10^""||ZZTEST^selfpayGT^^^^^CD:766||2003082000

PD1|""|""||589824^Waller^Dave^Cerner|""||""|""
```

Figure 2.5: An example of HL7 ADTÂ04 message

message set within a message can be optional and/or repeating. Each message consists of the segments that are delimited by "carriage return" characters ("
r" or 0x0D). That's why we see each segment starting on a different line. Each segment has its own *semantic* purpose and contains information of a specific type. The segments consist of fields that are *composites* and are delimited by "|" (Pipe sign). For example,

- **MSH** segment contains information about the Sender and Receiver of the message, the type of the message, a time stamp, etc.

- **EVN** contains information about the type of message; for example, A04 (Register a patient). Information contained in EVN is duplicated in MSH, so starting from HL7 version 2.3 this segment is excluded from all message definitions.

- **PID** contains demographic information about the patient such as name, id codes, address, and so on.

- **PV1** contains information regarding the patient's stay in the hospital, such as location assigned, referring doctor, etc.

Although there exist a series of dozen HL7 versions, ranging from v.1.0ṽ.3.0, but in the following section, we give a brief introduction to the most important version HL7 v.3.0 only.

HL7 v.3.0

HL v.3 represents a departure from the v.2.x series as it adopts a new methodology for developing messages. Historically, v.2 did not have a rigorous development methodology, and consequently, different parts of the standard were developed in different ways. HL7 v.3, however, has a specific and rigorous methodology that ties together over-arching information and application interaction models with messages and message sets, and ties those in turn to syntax and semantics specifications [59]. The v.3.0 was being built around a single object model, named RIM (*Reference Information Model*), which contained 123 classes, 179 associations and 890 attributes in the first implementation. HL7 v.2.x was mostly focused on the general triggers, structure, and layout for communication, while v.3.0 is more focused on specific contexts, terminology, models, and conceptual definitions and relationships.

After 10 years of development, HL7 v.3.0 was released for the first time in the form of the "Normative Edition 2005". HL7 v.3.0 has been adapted right now to support: large scale integration (state/province, region, country), public health, decision support, and research. HL7 v.3.0 is the standard of choice for countries and their initiatives to create national EHR and EHR data exchange standards as it provides a level of *semantic interoperability*, unavailable with previous versions and other standards.

HL7 is like a language and every language has a grammar. The HL7 RIM specifies the grammar of HL7 messages and specifically the basic building blocks of the language and their permitted relationships. The RIM is neither a model of healthcare, though it is healthcare specific, nor is it a model of any message, though it is used in messages. At first site the RIM is quite simple, as the RIM backbone has just five core classes and a number of permitted relationships between them. In HL7 v.3.0, every event is an *Act*, which is analogous to a *verb* in English. Each Act may have any number of Participations, in Roles, played by *Entities*. These are analogous to *nouns*. Each Act may also be related to other Acts, via *Act-Relationships*. Act, Role and Entity classes also have a number of specializations, for example, Entity has a specialization called Living Subject, which itself has a specialization called Person. Person inherits the attributes of both Entity and Living Subject.

HL7 CDA

HL7 *Clinical Document Architecture* (CDA), which was until recently known as the *Patient Record Architecture*, provides an exchange model for clinical documents, such as discharge summaries and progress notes. CDA was approved as an ANSI standard in November 2000 and it brings the healthcare industry closer to the realization of an electronic medical record. By leveraging the use of XML, the HL7 RIM and coded vocabularies, the CDA makes documents both machine-readable so they are easily parsed and processed electronically, and human-readable, so they can be easily retrieved and used by the people who need them. CDA documents can be displayed using XML-aware web browsers or wireless applications such as cell phones.

2.6.2 DICOM

DICOM (*Digital Imaging and Communication in Medicine*) [60] is a standard which defines a method of communication for the various equipments of digital medical imaging devices and software (*modalities*). It specifies a network protocol utilizing TCP/IP, defines the operation of service classes beyond the simple transfer of data, and creates a mechanism for uniquely identifying *information objects* as they are acted upon across the network. DICOM is also structured as a multi-part document in order to facilitate extension of the standard. Additionally, DICOM defines information objects not only for images but also for patients, studies, reports, and other data groupings. With the enhancements made in DICOM version 3.0, the standard is ready to deliver on its promise not only of permitting the transfer of medical images in a multi-vendor environment, but also facilitating the development and expansion of PACS (*Picture Archiving and Communication Systems*) and interfacing with medical information systems. Fig. 2.6

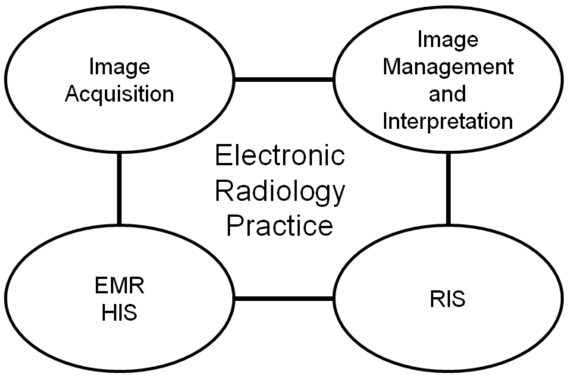


Image Acquisition
Image Management and Interpretation
Electronic Radiology Practice
EMR HIS
RIS

Figure 2.6: Electronic radiology practice and its components

illustrates the usual electronic radiology practice in the hospitals. Image acquisition combined with image management and interpretation is generally considered together as PACS. The communication of the PACS with the other electronic systems in the radiology department, i.e. *Radiology Information System*, and institution, i.e. *Hospital Information System* is essential to fully realize the implementation and benefits afforded by automation. Even though the effect of PACS is far greater outside the radiology department, the function of PACS within the department can be markedly different depending upon the proper design and function of these interfaces.

2.6.3 ASTM

The most relevant standards from ASTM [61] are the standards from the committee E31 on "Healthcare Informatics". This committee is recognized as an accredited organization by ANSI, and the following standards are the most widely used relevant standards in the field of health care informatics [62][63].

- ASTM E1238 standard specification for transferring the clinical observations between independent systems. This standard is developed by the ASTM subcommittee E31.11. This standard is being used by most of the largest commercial laboratory vendors in the United States to transmit laboratory results. It has been incorporated into the *Japanese Image Store and Carry* standard. HL7 has incorporated E1238 as a subset within its laboratory results message format.

- ASTM E1394 standard specification for transferring information between clinical instruments. This standard was developed by ASTM Subcommittee E31.14 and is being used for communication of information from laboratory instruments to the computer systems. This standard has been developed by a consortium consisting of most U.S. manufacturers of clinical laboratory instruments.

- ASTM E1460 standard specification for defining and sharing modular health knowledge bases (*Arden Syntax*). This standard was developed by ASTM Subcommittee E31.15. The Arden Syntax provides a standard format and syntax for representing medical logic and for writing rules and guidelines that can be automatically executed by computer systems. Medical logic modules produced in one site-of-care system can be sent to a different system within another site-of-care and then customized to reflect local usage.
- ASTM E1467 standard specification for transferring digital neuro-physical data between independent computer systems. This standard was developed by ASTM Subcommittee E31.16, and it defines codes and structures needed to transmit electro-physiologic signals and results produced by electro-encephalograms (EEG) and electro-myograms. The standard is similar in structure to ASTM E1238 and HL7, and has been adopted by most of the EEG systems manufacturers.

2.6.4 ANSI/IEEE 1073

ANSI/IEEE Standard 1073 [64], which is also known as *Medical Information Bus* (MIB), is one of the famous standards for medical device communication, which leverages the advantages of standardization to the problem of *patient-connected* device interfacing. In comparison to single-vendor proprietary networks, MIB comprises a truly open system, public domain standard. It provides a robust, cost effective answer to the need for guaranteed electrical and mechanical safety, and data integrity. Moreover, it guarantees *plug-and-play* interoperability between all devices from any manufacturer by specifying a common communication protocol for *all* devices. This provides hospital users the freedom to move equipment around an institution, while eliminating any concerns regarding interfacing compatibility. In addition, MIB allows for great reductions in software development and maintenance costs for device manufacturers, clinical software developers, and hospital end users by specifying a *common data language* applicable to all devices from all manufacturers.

The purpose of the MIB family of standards is to enable the automatic and universal capture of clinical data from any type of bedside device, from any manufacturer, and for this purpose, MIB is highly optimized for *real-time* device communications in an acute care environment. Fig. 2.7 [64] illustrates a generic example of the use of MIB for bedside device interfacing, where MIB bedside subnetwork consists of a configuration with multiple patient-connected devices interfacing to a bedside hub. In MIB terminology, the *hub* is called the bedside communication controller, or BCC. Each patient connected device interfaces by means of a device communication controller, or DCC, which can be either an external converter box, or an embedded implementation. The purpose of an external box is to convert between an RS-232, RS-422, RS-423, RS-485, 4 to 20mA, analog, parallel, or other *legacy* device interface, and an MIB DCC.

2.6.5 CEN/ISO/IEEE 11073

Since 1993 a set of open CEN (European) Standards, for point-of-care device communication has been created to provide the ability to connect devices to each other freely and to exchange data between them. These standards worked to complement those of the IEEE 1073, not to be competitive. In 2000 it was agreed that CEN, ISO and IEEE

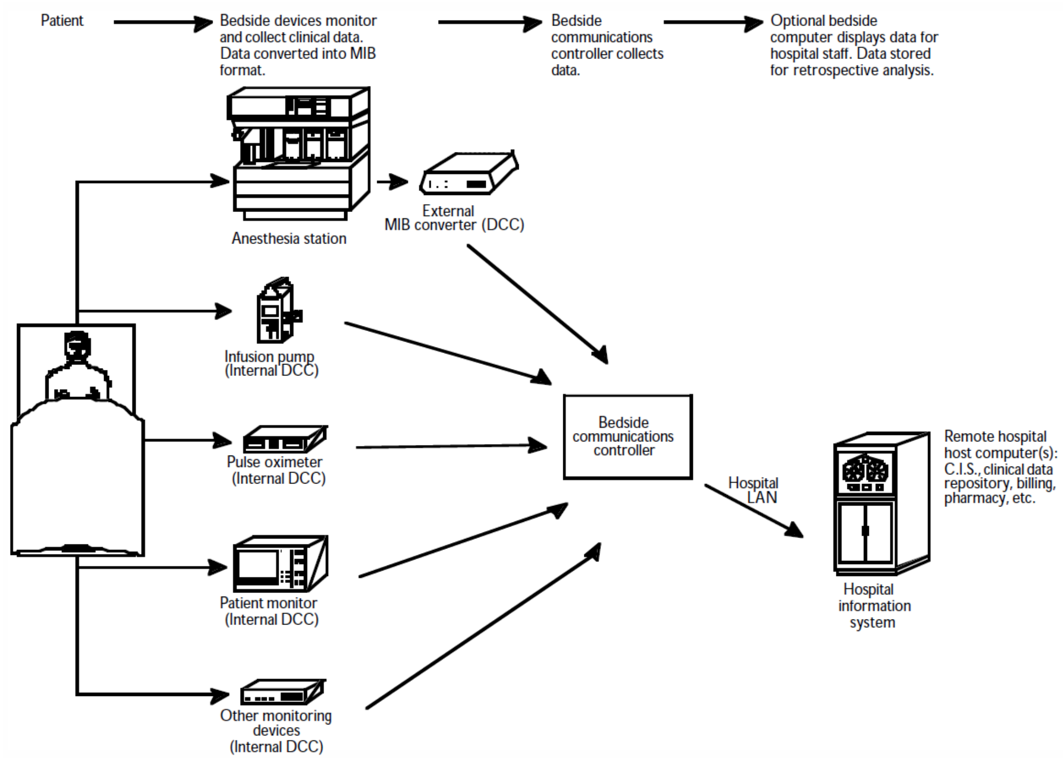

Patient
Bedside devices monitor and collect clinical data. Data converted into MIB format.
Bedside communications controller collects data.
Optional bedside computer displays data for hospital staff. Data stored for retrospective analysis.
Anesthesia station
External MIB converter (DCC)
Infusion pump (Internal DCC)
Pulse oximeter (Internal DCC)
Patient monitor (Internal DCC)
Other monitoring devices (Internal DCC)
Bedside communications controller
Hospital LAN
Remote hospital host computer(s): C.I.S., clinical data repository, billing, pharmacy, etc.
Hospital information system

Figure 2.7: Typical ANSI/IEEE 1073 (MIB) Environment

would jointly migrate and publish this work in a single set of standards, designated as 11073 [65], for point-of-care device communication under the general leadership of IEEE. In 2003 the IEEE approved the core standards, the domain information model, the common nomenclature, the basic communications profiles and two arrangements defining the underlying transport mechanism. In early 2004, approval of these five 11073 standards within CEN and ISO was achieved, with more already in the balloting process. Alongside the core standards, application profiles have been defined for specific types of devices, such as dialysers, ventilators, monitors and infusion pumps.

2.6.6 IHE

Integrating the Healthcare Enterprise (IHE) [66] is an initiative taken by healthcare professionals and industry in 1997 to improve the way computer systems in healthcare share information. IHE promotes the coordinated use of established standards such as DICOM and HL7 to address specific clinical needs in support of optimal patient care. The systems developed in accordance with IHE communicate with one another better, are easier to implement, and enable care providers to use information more effectively. Physicians, medical specialists, nurses, administrators and other care providers envision a day when vital information can be passed seamlessly from system to system within and across departments and made readily available at the point of care. IHE is designed to make their vision a reality by improving the state of systems integration and removing barriers to optimal patient care.

2.6.7 EDI

EDI (*Electronic Data Interchange*) is the computer-to-computer exchange of structured information, by agreed message standards, from one computer application to another by electronic means and with a minimum of human intervention. In common usage, EDI is understood to mean specific interchange methods agreed upon by national or international standards bodies for the transfer of business transaction data, with one typical application being the automated registration of a patient in a hospital. Despite being relatively unheralded, in this era of technologies such as XML Web Services, the Internet and the WWW, EDI is still the data format used by the vast majority of electronic commerce transactions in the present computing world.

The EDI standards were designed from the beginning to be independent of lower level technologies and can be transmitted using Internet protocols as well as private networks. It is important to differentiate between the EDI documents and the methods for transmitting them. There are two major sets of EDI standards. The first one is UN/EDIFACT, which is the only international standard (in fact, a United Nations recommendation) and is predominant in all areas outside of North America. The second is ANSI ASC X12 (X12), which is popular in North America and used worldwide. These standards prescribe the formats, character sets, and data elements used in the exchange of documents and forms, such as purchase orders (called ORDERS in UN/EDIFACT and an 850 in X12) and invoices. These standards say which pieces of information are mandatory for a particular document, which pieces are optional and give the rules for the structure of the document.

Interfacing systems is one of the key challenges faced by IT staff of any healthcare institution. Understanding the differing implementations of standards in various vendor systems and trying to find ways to reconcile them is an expensive, labor-intensive and often painful process. IHE offers a common framework for vendors, IT departments, clinical users and consultants to understand and address clinical integration needs. The IHE technical framework allows flexibility while ensuring that key integration needs are met. IHE promotes integration within and across all units of the healthcare enterprise. The initial successes of IHE were achieved in radiology and the IHE initiative in radiology remains very active. The IHE process has since been adopted in other domains, as well: IT infrastructure, cardiology, laboratory, and medication management. Working in coordination with the others, each of these domains will develop its own technical framework and integration profiles, and implement its own testing and demonstration process. [66].

2.7 ICT Infrastructure in the Hospitals and Clinical Environments

In the existing ICT (*Information and Communication Technology*) infrastructure in the hospitals, laboratories and clinical environments, different types of information systems and medical devices are used, provided by different manufacturers and vendors. Some of the leading manufacturers of hospital information systems (HIS) are Agfa healthcare, Fujifilm medical systems USA, GE medical systems, Philips medical systems, Siemens medical system and SAP etc. Some of the leading point-of-care devices' manufacturers are Roche Diagnostics, BD, Bayer Healthcare, CARESIDE medical etc. Most of the time, the devices manufactured by these companies are compliant to their own proprietary protocols, and are interoperable only to their own developed information systems.

Modern hospitals have at least two data processing systems, one system for finances and general administration, while the other for patient administration and basic patient data (demographics, diagnoses and procedures) [67]. In many cases there are other systems, which are often dedicated departmental systems e.g. system in chemical lab, system(s) in radiology department (RIS and PACS), ICU patient data management system, etc. These data processing systems are dedicated to the processes internal to the departments and in many cases they deal with device data, e.g. control of the analytic equipment in the chemical lab and collection of measurement data, collection and storage of radiological pictures, collection of measurement data of patient monitoring and therapeutic devices in ICU. These systems normally communicate in two different directions, one to devices, and one to hospital information system (HIS). The data collected from devices, images, waveforms, etc. must be readable for machinery and not necessarily be readable by humans. Often this data is understandable by humans only if available in the graphical representation. Nevertheless, these systems communicate with the HIS too in order to receive e.g. demographic data of patients. That can be described by different levels of communication, the HIS or enterprise level, and the departmental or device level.

In fact, communication to the HIS and communication with devices is rather different in nature and indeed there are big differences between e.g. the communication with radiology machinery and to devices in the ICU. The data representation and communication requirements are very different between these domains, e.g., the radiology images to communicate and store are rather huge, while the messages in ICU device communication are rather small compared to image communication. The ICU device communication needs alert messages and real time communication.

In Fig. 2.8, the communication of HIS or enterprise level, as well as the departmental or device level is shown. The *Domain/Enterprise Level* concerns the communication in the whole hospital, communication between different hospitals, and exchange with health care professionals outside the hospital and with health care organizations. Mostly, HL7 standard is used for the communication of patient's medical data on this level, resulting successfully interfacing the HISs, insurance companies and public health organizations. HL7 v.3.0 is the latest version which has been developed and has strongly influenced the future ISO and CEN standards for enterprise level communication in healthcare after establishment of co-operation between CEN/TC251 and HL7 organizations.

The communication on *Device/Departmental Level* is mostly limited to a single department or parts of a single department (ICU) only. As a typical departmental communication standard, as already shown in Fig. 2.8, the DICOM standard is used to exchange images between different modalities like CT (*Computer Tomographs*), MRI (*Magnetic Resonance Tomographs*) etc., to diagnostic workstations, PACS (*Picture Archiving and Communication Systems*) etc. The ASTM-1394 standard for the exchange of data between analytical instruments and laboratory information systems is another example used on the departmental level, usually inside clinical chemistry or lab subsystems.

2.8 e-Health

This section describes briefly about e-Health, healthcare, e-Health systems and their challenges w.r.t interoperability.

In the current communication era, governments and health organizations have realized that the cost-effective application of technologies (including Information Technology, communications, medical devices and telemetry solutions) can assist in improving access to health services and in achieving better quality care and health outcomes. To success-

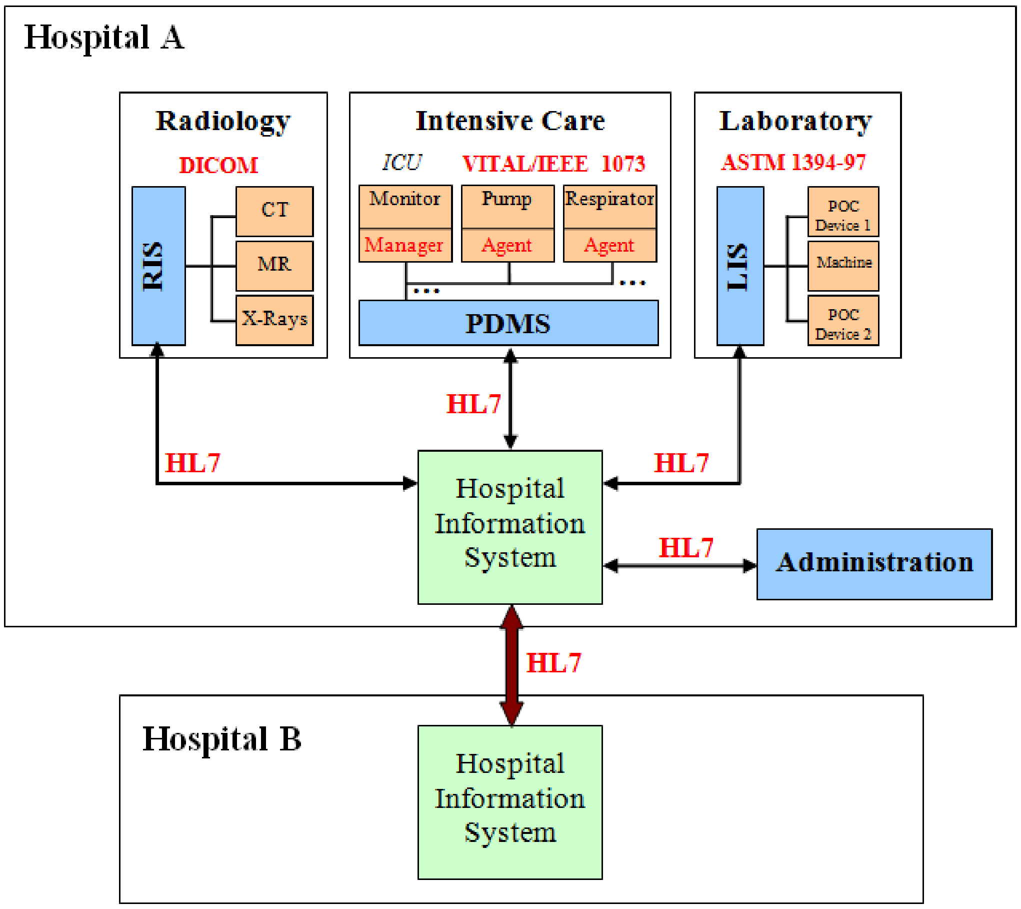


Figure 2.8: The interoperability of medical devices and information systems in hospitals

fully achieve these services, e-Health [68] is the best choice to be opted.

e-Health is an emerging field encompassing the convergence of Telematics, health informatics, healthcare services & business, and refers to healthcare services/information delivered by or enhanced through technologies such as the Internet, UMTS, WLAN, and Body Area Networks. In a broader sense, the term e-Health denotes not only a technical development, but also an organizational mind-set with the goal of improving quality of care by using Information and Communication Technology [69].

e-Health solutions have been used in all forms of electronic health-related service delivery ranging from informational, educational, and commercial products to direct services offered by professionals, non-professionals, businesses and consumers themselves. e-Health services encompass the ”five C’s” [70].

- Content (health-related information e.g. disease management).
- Connectivity (communication channels between care providers, consumers, and organizations).
- Community (support groups).

- Commerce (online marketing and sales of health-related products and/or services).
- Clinical care (clinical diagnostic services, over-the-counter and prescription medications).

e-Health uses the following encompassing technologies [71]:

2.8.1 Use of multiple technologies

e-Health spans over the use of multiple technologies, including:

- Information technology (e.g., computers, software applications, multimedia devices).
- Telecommunications technology (e.g., wire-based and wireless networking devices).
- Telemetry (e.g., medical devices capable of measuring a patientŠs state of health remotely).

2.8.2 Multiple modes of interaction

e-Health supports multiple *modes* of interaction among participating parties, which are:

Real-Time

All parties directly participate in an e-Health session at the same time. For example, at a pre-scheduled date and time, a patient accompanied by a general practitioner in a rural hospital make a videoconferencing connection with the patient's psychiatrist located at the psychiatry care center. Live interactive consultation takes place between the patient, the general practitioner, and the psychiatrist.

Store-and-forward

Involves compilation of data (*store*) and transfer of the data to the receiving party (*forward*). This method of interaction does not require the parties to participate at the same time. For example, in a rural community, a patient x-ray is scanned and captured as an electronic file. This file, along with accompanying medical notes, is sent electronically to the radiologist in the tertiary care center, who reviews the x-rays and notes and confirms and/or makes a diagnosis. The radiological report is then returned to the rural care facility.

Streaming

Involves the delivery of real-time or stored multimedia data such as audio, video, documents, and still-images across networks with a reasonable amount of quality of services. At the receiving site, multimedia data is presented (played, displayed) before the entire content arrives. In other words, data continues to download in the background, while it is presented. For example, physicians located at care centers in different provinces participate in a videoconferencing session to discuss a complex cardiac case. The physician who examined the patient's heart transfers a video file captured during an ultrasound exam to the participants to support his/her recommendation for patient's treatment.

It is important to understand that the term *e-Health* does not necessarily imply the use of the Internet, but also e-Health Web enabled applications can work over private networks using dial-up or dedicated telecommunication links. The ultimate goal of the e-Health strategy is to link a myriad of clinical applications together to provide the right information to the right person at the right place and time. It is critical to state the importance of this in the changing healthcare environment because it means that the care providers will have the intuitive access to the entire depth of clinical information.

2.8.3 Examples of e-Health Applications

In this section, we examine some examples of the e-Health applications, which include:

Electronic Health Record (EHR)

EHR is a secure corpus of accessible patient files with critical information required in order to manage the health of the patient effectively.

Tele-health/e-Homecare

Tele-health provides a means of using telecommunications to deliver more services to persons in outpatient settings. Additionally, this includes the real-time and store-and-forward technologies designed to improve the consultation process. In addition to this, e-Homecare consists of new products, such as clinical devices or even wearable biochemical sensors for tracking any number of physiologic measurements and transmitting them via discrete wireless mechanisms to a constant monitoring system.

Clinical Lab Reports

Applications allowing for the searching and tabulating of all clinical and diagnostic lab information related to the patient.

Radiology/PACS

The systems that are able to acquire, transmit and store digital radiology examinations.

Disease Self Management Programs

The systems by which caregivers can gain access into the home, through the provision of care plans administered in conjunction with video and/or Web based technologies; also accessible by the patient to provide ongoing monitoring of physiological patient parameters.

Internet based e-Learning tools

Publicly assessable information from peer reviewed websites with clinical information, treatment protocols and user groups designed for the education and empowerment of the new clinical consumers.

Personal Digital Assistants (PDA)

A hand-held computer loaded with personal productivity tools such as a calendar, address book, word processing, and spreadsheet functions. Mobile workers use PDAs and work with central databases either with wireless connections using cellular phones or by synchronizing with the host computer with the cradle. Recent interest in the physician e-Health market focused on prescription writing, digital voice dictation or recognition for note taking, and direct access to patient centric database from anywhere at anytime.

e-Pharmacy/e-Prescribe

These are usually wireless Internet based systems that allow physician to transcribe a prescription on a PDA device to a remote pharmacy able to deliver the drugs to where a patient is located. Integrated applications include drug interaction tables, dosage protocols, etc.

In order to reach its full potential, e-Health has to meet with several conditions before it becomes commonly used in the health service delivery system. These conditions include adequate service quality, health system functionality, and interoperability.

2.8.4 What is e-Healthcare

Healthcare is a vital part of the economy and important to every citizen, and yet the healthcare economy has not benefited from the technology revolution that is fundamentally changing whole industries. Leading software companies i.e. Microsoft and IBM believe that the present time is right for the technology to have a dramatic and fundamental impact in improving healthcare delivery, payment and personal health management. In healthcare domain, e-Health initiatives are designed to present proactive, coordinated, and evidence-based healthcare where the clinical, social and technological issues are combined to create a flexible patient-focused healthcare system.

e-Healthcare is defined as a way of delivering and achieving better health outcomes through effective and innovative use of health information. It includes in the health sector *the use of digital data-transmitted, stored and retrieved electronically-for clinical, educational and administrative purposes, both at local site and at a distance* [19].

The aim of e-Healthcare is to provide high quality healthcare to all health consumers; to increase homecare by remotely monitoring the chronically ill patients in their homes; and to reduce the need for hospital care for patients. It is also to develop preventive health education and through the use of information technology reduce errors, waste and costs. This is being achieved through the interchange of collaborative multiple healthcare teams across regional, interstate and international boundaries and by the projection of specialist medical and surgical expertise to rural and remote areas. It is also being developed by the instant access to comprehensive secure, reliable and standardized health records; the integration of hospital, community, insurance industry, pharmacy, government, home and educational health management systems; and the provision of computer based training programs to health professionals [19].

This revolution will be hugely significant and it is relevant to the key problems faced by the industry today, such as:

- Improving patient safety
- Reducing costs
- Decreasing the time to develop and release new drugs and medical devices
- Improving the quality of care and outcomes achieved
- Improving the efficiency of processes for coordinating and paying for care

2.8.5 e-Health Challenges

More than any other recent development, e-Health bridges the barriers between telecommunications and Information technologies. While regulation, standardization and interoperability form the base of the telecommunications industry, the IT industry (the desktop computing industry in particular) has achieved success by encouraging innovation and diversity. While this strategy has resulted in rapid technical advances as well as tremendous cost-effectiveness and efficiencies, it has also suffered from proprietary and evolving standards, and the consequent interoperability problems.

While changing the business culture of these industries is a long-term goal, there are other challenges that the e-Health industry must deal with in short-term [71]. Some of these include:

Limited clinical guidelines

The general-use and well established standards being used in the healthcare sector (e.g. HL7, DICOM) are being complemented by e-Health specific clinical guidelines and protocols. The list of e-Health guidelines includes telephone triage, Teleradiology, homecare, Telesurgery, and health call center applications. Their scope, however, is limited to specific clinical applications and, in some cases, to specific projects.

Limited functional alignment

There is no *official* e-Health standard. The e-Health industry uses high-level healthcare guidelines and technical standards developed for various technology sectors including multimedia conferencing, information technology, data communications, security, and the Internet. These standards focus on *functional* requirements of these sectors and do not necessarily address the needs of the healthcare system.

Limited interoperability

In the current implementations, there is no interoperability standard for e-Health systems and networks in order to provide seamless interoperability. As a consequence, there is no plug-and-play e-Health technology platform.

Limited integration

e-Health systems do not typically offer tools to integrate with other systems to provide a seamless interface and most of the times, the e-Health networks are implemented as *stand-alone* solutions, and are not integrated with any other systems or business processes.

Monolithic and non-expandable architecture

Today, the closed and monolithic architecture of e-Health systems offers limited autonomy and scalability. System components are tightly coupled, so most implementation changes (e.g. addition or removal of components, or changes in their implementation) impact the architecture of the entire solution.

Limited failure recovery

e-Health solutions offer very limited capabilities to recover from failures, or to continue operating in a degraded mode until the failure is repaired. Typically, a failure of a system or a system component affects the entire e-Health solution.

To overcome these challenges, it is necessary to define both the functional model and the architectural framework for e-Health systems. While the functional model must be compliant with the requirements of the healthcare systems, the architectural framework must be based on a plug-and-play computing paradigm, distributed processing principles and open technology standards.

Chapter 3

Related Work

Although, several initiatives have been studied that have experimented the integration of Semantic Web and Web Services technologies into ubiquitous computing architectures in the last years, but this chapter is devoted to have an overlook on most relevant of them, which include HYDRA, SODA, Task Computing, Gaia and SCALLOPS research projects. First of all, it outlines the evaluation criteria, in comparison with the pros and cons of these initiatives with our envisaged architecture for the ambient intelligent medical or mobile devices, and then it identifies the constituent parts of these initiatives, up-to what extent they have applied the Semantic Web and Web Services technologies, and their contributions in the corresponding domains.

3.1 Evaluation Criteria

In this section, we define a set of evaluation criteria by analyzing and evaluating state-of-the-art architectures, and determine how our proposal ranks in comparison with these architectures. Most of the selected criteria can be found, implicitly or explicitly, throughout all the literature concerning ubiquitous and pervasive computing architectures. The examples of these criteria are decentralization, context-awareness, autonomy and/or standards adherence, while other criteria are more specific to our research goals, i.e. having reasoning capability (inferencing) on medical devices, and communication with other medical or mobile devices through Web Services technology. In the following sections, we have organized the criteria into *three* different categories, namely *Architectural*, *Technological* and *Intelligence*, depending on their nature.

- **Architectural**
  - **Decentralization:** At the core of any ubiquitous computing system, decentralization endorses a spontaneous and unanticipated nature, non-critical components in the architecture, ad-hoc reconfiguration according to the situation, and natural deployment of elements and scalability. However, the design and planning of decentralized systems is more difficult, as well as resulting in a higher load of network traffic due to synchronization and coordination messages.
  - **Lightness:** The architectural elements and software components in particular should be designed in such a way that they could be easily embedded

in resource-constrained systems, i.e. medical or mobile devices, as well as promote operational simplicity if possible.

- **Technological**
  - **Standards Adherence:** It represents the degree to which the envisaged system reuses and applies the widely accepted standards, thus taking advantage of previous R&D work and promoting re-usability in the industry and academia. The intention behind our research is to create new original work by assuring the highest possible degree of standards adherence. Please note that we exclude the discussion of adherence to the medical (device) communication standards, as neither the frameworks discussed in this chapter support the most commonly used medical (device) communication standards, nor, to the best of our knowledge, any known SOA framework support them.
  - **Technological Consistency:** It represents the degree of coherence among the technologies used for the development of a system. The complementary technologies must be applied, if possible, in order to obtain synergistic performance and future re-usability.
- **Intelligence**
  - **Reasoning Capability:** It is the ability of the system to acquire and apply knowledge via reasoning (inferencing) process. In order to create the next generation of ambient intelligent medical or mobile devices, artificial intelligence techniques must be applied to a certain degree.
  - **Context-Awareness:** In simple terms, it is the ability of the system to perceive and identify the relevant information from an environment and actively respond according to the defined rules. Context-awareness is inevitably required to meet the ultimate goals of ubiquitous computing and ambient intelligent environments.

The above-mentioned criteria are not completely isolated, rather certain dependencies exist among some of them in such a way that the degree of fulfillment in one concrete criterion can affect the other in a positive or negative manner. A high degree of *decentralization* favors the design of distributed lightweight components, instead of a bulky and heavy central controller. *Lightweight* components make the actual implementation feasible, but they also reduce the degree and quality of the embedded reasoning processes, which normally demand some amount of software complexity. The *Reasoning Capability* affects lightness negatively in the same way, since the more reasoning power the device is provided with, the heavier the component becomes. However, it promotes the context-awareness and autonomy, which can take advantage of the built-in intelligence to react accordingly. The *Context-Awareness* promotes autonomy, since architectural components can self-regulate their behavior depending on the context information provided by the surrounding entities. The *Technological Consistency* can significantly reinforce APIs re-usability during the implementation, thus promoting more lightweight components that would be negatively affected by mixing up non-complementary technologies. At the same time, we consider that technological consistency simplifies the overall design by taking advantage of the existing mechanisms, procedures and interfaces.

Not all these criteria are equally important, but depending on the desired strengths of the resulting architecture, some of them must be promoted. We will focus primarily on maximizing decentralization, reasoning capability, lightness and standards adherence, and, in a lesser extent, technological consistency and context-awareness. Table 3.1 enumerates the criteria with relative weights depending on their importance to our goals.

Table 3.1: Criteria's relative weights of importance

Criterion	**Weight**
Decentralization	4
Reasoning Capability	4
Lightness	3
Technological Consistency	3
Standards Adherence	2
Context Awareness	1

3.2 HYDRA

The HYDRA[1], an FP6 European Commission funded Integrated Project, addresses the problem that is frequently faced by producers of devices and components - the need for (which is actually becoming a trend) networking the products available on the market in order to provide higher value-added solutions for their customers [17]. This requirement is implied by citizen centered demands requiring intelligent solutions, where the complexity is hidden behind user-friendly interfaces to promote inclusion. The vision of the HYDRA project is ambitious: "*To create the most widely deployed middleware for intelligent networked embedded systems that will allow producers to develop cost-effective and innovative embedded applications for new and already existing devices*".

HYDRA develops a middleware based on a Service-Oriented Architecture (SOA), to which the underlying communication layer is transparent. Hydra middleware is designed to run on a variety of stationary and mobile devices, and includes support for distributed as well as centralized architectures, security and trust, reflective properties and model-driven development of applications. It is deployable on both new and existing networks of distributed wireless and wired devices, which operate with limited resources in terms of computing power, energy and memory usage. It allows for secure, trustworthy, and fault tolerant applications through the use of novel distributed security and social trust components and advanced Grid technologies.

The embedded and mobile Service-Oriented Architecture provides interoperable access to data, information and knowledge across heterogeneous platforms, including Web Services, and support true ambient intelligence for ubiquitous networked devices. Furthermore HYDRA develops a Software Development Kit (SDK), which will be used by developers to develop innovative Model-Driven applications using the Hydra middleware, where middleware and connected devices enable the developers to implement applications that depend on and adapt to context information. In particular, the developers stress the acquisition and management of spatial context information that allows for locating devices attached to the system and for the positioning of people and assets. The HYDRA project validates the middleware, the SDK toolkit in real end-user scenarios in three user domains, namely Building Automation, Healthcare, and Agriculture.

[1]`http://www.hydramiddleware.eu`

3.2.1 Software Architecture of HYDRA

The *functional structure model* of HYDRA middleware is divided mainly into two parts: **Application Elements** and **Device Elements**, as shown in Fig. 3.1 [17]. Both elements differ in the following aspects:

- Power of the machine.
- Intended purpose of the components.
- Target developer user.

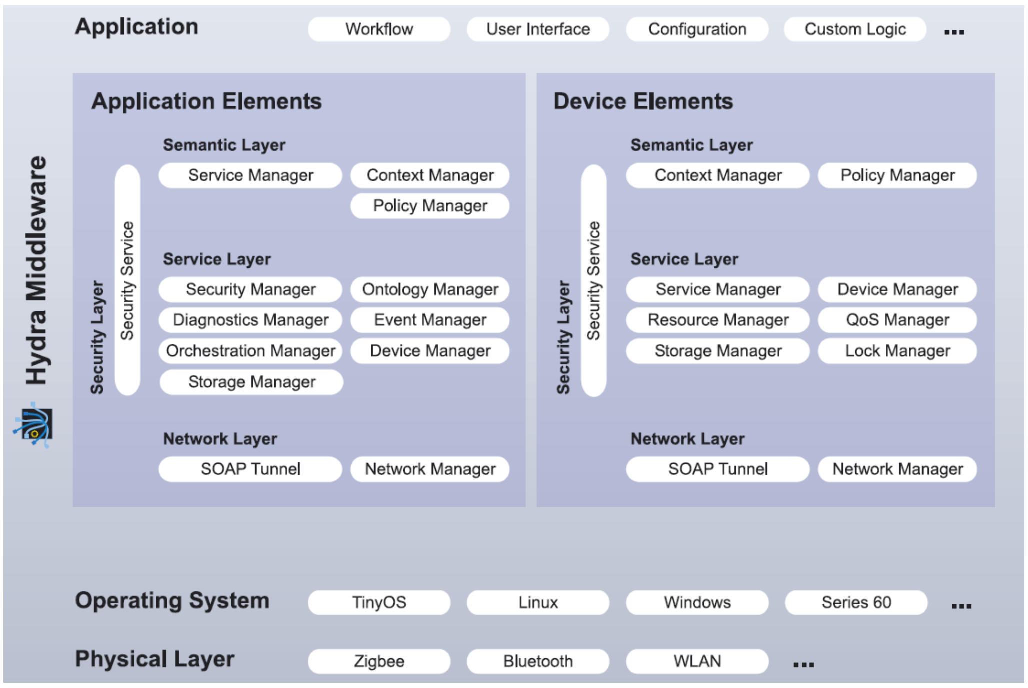


Figure 3.1: Structural Overview of the Hydra Middleware Layers

Application Elements

Application Elements describe components that are usually deployed on hardware which is performance-wise capable of running the application that the solution provider creates. This means these components are meant to be running on powerful machines. They have been put together and configured to work together with other software in order to support a specific application such as building automation by a specific developer, e.g. system integrator.

Device Elements

Device Elements describe components that are usually deployed inside HYDRA-enabled devices, so that they could be deployed in small devices as well which have limited resources in terms of e.g. processing power or battery life. These components have a limited set of functionality but could also be deployed on another machine acting as a proxy for e.g. a mote where it would be highly unlikely that those managers would ever be deployed on such a resource-limited device. They have been put on the device by a device manufacturer to provide certain functions irrespective of which application is using the device.

The HYDRA middleware elements are enclosed by the physical and the application layer shown at the bottom and at the top of the diagram respectively. The *physical layer* realizes several network connections like ZigBee, Bluetooth or WLAN. The *application layer* contains user applications which could contain modules like workflow management, user interface, custom logic and configuration details. These two layers are not part of the HYDRA middleware. The middleware itself consists of three layers - the network, service and semantic layer, where each layer holds elements according to their functionality and purpose. It is to be noted that some device elements have similar and thus likewise named, counterparts among the application elements. Both, device and application elements, have a *Security Manager*, which is an orthogonal service and depicted as vertical to cover all three middleware layers.

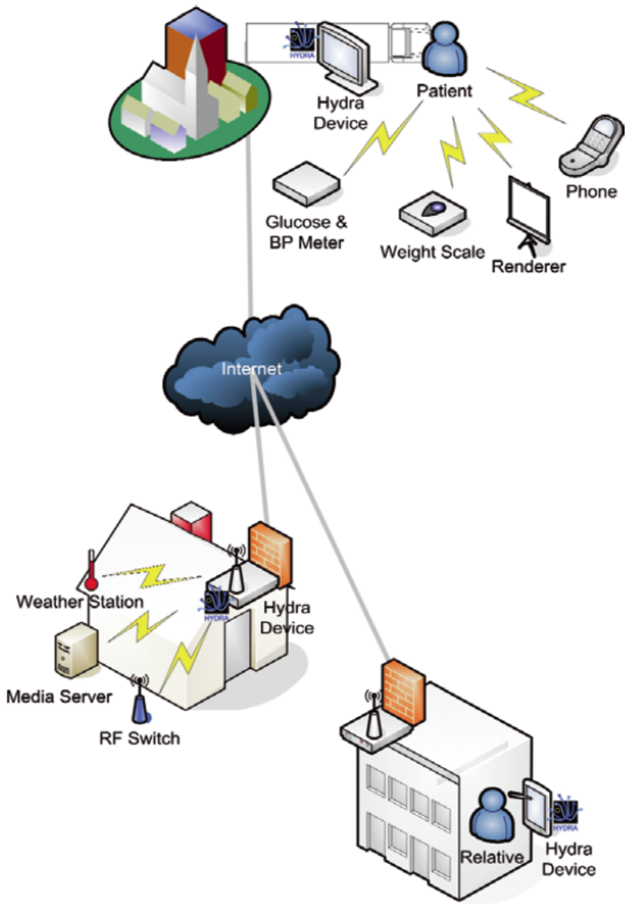


Figure 3.2: Hydra enabled healthcare application using Service-Oriented Architecture

3.2.2 Healthcare Scenario of HYDRA

A common problem for the manufacturers of medical devices and for developers of e-Health systems is the lack of interconnectivity and interoperability of the various proprietary components and subsystems. The Hydra middleware enables devices and subsystems to communicate and allow developers to develop intelligent, secure, multi-parametric healthcare services using a range of medical devices and subsystems. Fig. 3.2 shows an envisaged HYDRA healthcare scenario using Hydra devices, where the patient is being monitored with the attached devices, e.g. glucose meter and blood pressure devices etc., and the medical data is transferred to the home-care services provider, whenever the measurement is performed. In case of emergency, an alert message is sent in parallel to the registered relative of the patient so that s/he could approach the patient immediately besides the emergency services provider.

3.2.3 Conclusion

The HYDRA research project is one of the most relevant research projects related to our research goals. The vision of the HYDRA project is to create the most widely deployed middleware for intelligent networked embedded systems that will allow producers to develop cost-effective and innovative embedded applications for new and already existing heterogeneous physical devices. The embedded and mobile Service-Oriented Architecture will provide interoperable access to data, information and knowledge across heterogeneous platforms, including Web Services, and support true ambient intelligence for ubiquitous networked devices.

However, there are few drawbacks in HYDRA in comparison with our approach. First of all, the HYDRA project does not target the resource-constrained devices themselves, rather it provides an HYDRA device, which is usually a PC/notebook, that acts as a *proxy* for the resource-constrained devices and expose their functionalities as Web Services. It means that the existing resource-constrained devices have to be connected to a PC/notebook, requiring high computing and memory resources. Secondly, the HYDRA project lacks with the use of any *semantic discovery* protocol for the desired physical devices, which means that the simple discovery protocol, i.e. SSDP of UPnP, is enhanced and used to discover the pre-defined physical devices.

The following sections provide analysis of HYDRA against our defined evaluation criteria, and the conclusion is summarized in Table 3.2.

- **Decentralization:** HYDRA is a combination of pure Peer-to-Peer network technologies and traditional Web Service technologies allowing any device to offer, identify, and consume Web Services transparently from the application developer point of view. *Very High.*
- **Lightness:** The Hydra middleware requires a PC/notebook based Hydra device to interconnect other resource-constrained devices. *Low.*
- **Standards Adherence:** The standards and recommendation, i.e. OWL, OWL-S, SAWSDL, UPnP are honored. *High.*
- **Technological Consistency:** HYDRA applies Web Services and Semantic Web technologies in a coherent and synergistic manner. *High.*
- **Reasoning Capability:** Different ontologies, including Security ontology and Device ontology have been developed to support the reasoning process on a HYDRA device. *High.*

- **Context Awareness:** There are few ontologies developed to represent different contexts in the envisaged scenarios. *High.*

Table 3.2: Analysis of HYDRA against the evaluation criteria

Criterion	**Value**
Decentralization	Very High
Reasoning Capability	High
Standards Adherence	High
Lightness	Low
Technological Consistency	High
Context Awareness	High

3.3 SODA

The SODA (Service Oriented Device and Delivery Architecture) project [15], an FP6 European Commission funded Integrated Project, targets to create a service-oriented *ecosystem*[2], built on top of the foundations laid by the groundbreaking SIRENA [72] framework for high-level device communications based on the Service-Oriented Architecture (SOA) paradigm. The SODA project implements a comprehensive, scale-able and easy-to-deploy SOA ecosystem on industry-favorite platforms, supported by wired & wireless communications. The service infrastructure for real-time embedded devices used as a foundation for the SODA project is defined in a platform-, language- and network-neutral way, applicable to a wide variety of networked devices for diverse applications in domains like industrial automation, automotive electronics, home & building automation, telecommunications, medical instrumentation, etc.

The SODA project is intended to:

- Implement a complete ecosystem for designing, building, deploying and running device based applications leveraging the service-oriented breakthrough provided by the SIRENA framework:
  - Providing a complete set of tools for the entire system life-cycle support.
  - Developing and extending the run-time infrastructure in term of performances, serviceability, and security.
  - Insuring seamless integration of device-provided services with high-level business processes.
- Develop elaborated experimental applications in several application domains, i.e. industrial automation, telecommunications, home networking and automotive electronics, to validate the usage of the SOA paradigm at a broad scale and promote its standardization in several vertical application domains.
- Conduct feasibility studies on application of the device-level SOA approach in application areas thus far unexplored.

[2]An **Ecosystem** is a combination of all the living (organisms or actors) and non-living elements (environment) of a complex area organized in a perfect harmony functioning as a whole. In the context of the SODA project, the living elements are similar to the run-time components of a particular SODA system, and the non-living elements are similar to the tools used during design, deployment, etc.

3.3.1 The SODA Ecosystem

A major result of the SODA project is a service-oriented ecosystem for embedded devices, neutral with respect to operating systems and programming languages and independent from physical resources, networks and protocols, as well as from application domains. This ecosystem is built on top of the foundations laid by the SIRENA framework, and features:

- an infrastructure for service-oriented systems based on embedded devices.
- a methodology and guidelines helping users to move from traditional architectures towards service-oriented ones.
- a tooling chain supporting the design, development and operations of service-oriented applications.

This ecosystem supports a substantial improvement of overall software quality by automating design and deployment tasks as much as possible, as well as applies novel service testing and validation methodologies and tools. In order to support the use of SOA concepts in device-based architectures, the ecosystem needs to fulfill the following requirements:

- **Enable services on devices:** publish basic device functionalities as coarse-grained, self-contained and manageable service components.
- **Enable service composition:** build higher-level functionalities through an explicit composition model and dynamic component binding capabilities
- **Provide connectivity and adaptative quality of service:** through built-in infrastructure capabilities, performances, security, management, and reliable messaging.
- **Improve the product development and deployment life cycle:** by leveraging the SOA approach.

Fig. 3.3 [73] highlights the SODA approach to realize its ecosystem. The *right-hand side* depicts the run-time environment, for which the results of SIRENA project, and in particular the Device Profile for Web Services (DPWS) [74] stack provides the high-level SOA infrastructure for device communication. The DPWS defines a minimal set of implementation constraints to enable secure Web Service messaging, discovery, description, and eventing on resource-constrained devices. Its objectives are similar to those of UPnP but, in addition, DPWS is fully aligned with Web Services technology and includes numerous extension points allowing for seamless integration of device-provided services in enterprise-wide application scenarios. The *left-hand side* depicts a comprehensive development environment, which allows to aggregate devices into composite devices, to design and generate applications using those devices, to deploy and manage devices and applications, and to integrate devices into control and business applications.

3.3.2 Conclusion

Like HYDRA, SODA is one of the most relevant research projects related to our research goals. It is the first step towards developing the SOA-based smart devices in different domains, and fulfills many of the requirements for semantic interoperability of SOA-based

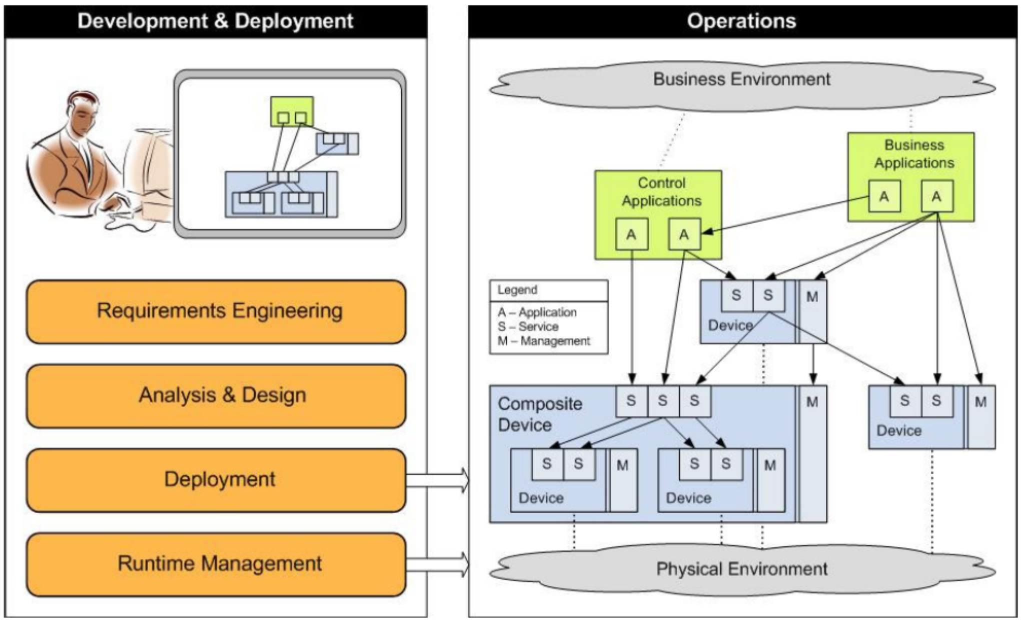


Figure 3.3: The SODA Approach

smart devices, except the semantic discovery and the provision of reasoning capabilities entirely integrated on smart devices. SODA project provides an ontology, named SPMO (SODA Process Modeling Ontology) [75] for semanticizing the descriptions of devices' Web Services, in order to support semantic searching, discovery, invocation, composition and monitoring of their services. There are few drawbacks in SODA in comparison with our approach, i.e. the smart devices are not fully autonomous with respect to composition and invocation of the Web Services that they host, as well as reasoning on the available knowledge base that could help smart devices to make decisions and take the desired actions to fulfill their tasks.

The following sections provide analysis of SODA against our defined evaluation criteria, and the conclusion is summarized in Table 3.3.

- **Decentralization:** SODA ecosystem supports a mixed architecture in which some devices are autonomous (having sufficient memory/processing capabilities to host Web Services), while others (not having sufficient memory/processing capabilities) expose their services through proxies running on a gateway connected to the central network. *High.*
- **Lightness:** DPWS is targeted for the resource-constrained devices. *Very High.*
- **Standards Adherence:** The standards and recommendation, i.e. OWL-S, OWL and DPWS are honored. *High.*
- **Technological Consistency:** SODA applies Web Services and Semantic Web technologies in a coherent and synergistic manner. *High.*
- **Reasoning Capability:** Although, SPMO ontology is developed to semanticize the Web Services' descriptions, but no reasoning capability on smart devices is supported. *Low.*

- **Context Awareness:** The requirements for different domains are defined, but lack with the definition of context awareness for the envisaged scenarios. *Low.*

Table 3.3: Analysis of SODA against the evaluation criteria

Criterion	**Value**
Decentralization	High
Reasoning Capability	Low
Standards Adherence	High
Lightness	Very High
Technological Consistency	High
Context Awareness	Low

3.4 Task Computing

Task Computing[3] (TC) technology is a joint effort by Fujitsu Laboratories of America and the MINDSWAP group, devoted to Semantic Web research, at the University of Maryland Institute for Advanced Computer Studies. The goal of TC is to "fill the gap between the tasks that users want to perform and the services that constitute available actionable functionality" [76]. TC presumes initially that users do not know how to achieve their goals when using computing facilities due to increased complexity at computing environments and tasks, and tries to ease the process by providing the user with an intelligent aid that hides the complexity of coordinating existing devices and services.

TC provides dynamic service discovery, service publishing and management, task creation and execution on the fly [77]. It even assists users in discovering what their goals are by suggesting possible tasks that can be performed with available facilities. All these features try to solve the frustration of users in application-rich environments, where they have to orchestrate a variety of devices and applications. Using Task Computing they can focus on their final goal and accomplish it with a reduced number of simple interactions.

Service composition can be seen as the "*process of creating customized services from existing services by a process of dynamic discovery, integration and execution of those services in a planned order to satisfy a request from a client*" [78]. Some examples of documented scenarios [79] that can be accomplished using TC technology are: exchanging business cards, showing and sharing a presentation, scheduling a future presentation or checking and printing directions to the airport. All of them are accomplished by sharing services on different devices and orchestrating those services to create a workflow in order to carry out the desired task. A prototype of TC has been implemented experimentally for *Smart Conference Rooms* and *Home Multimedia Environments* [80] and the first public results date back to 2003. It consisted in applying Semantic Web technologies to Pervasive Computing scenarios for semi-automatic composition of tasks, based on their previous research [77].

[3]Task Computing - the Technology; `http://taskcomputing.org/`

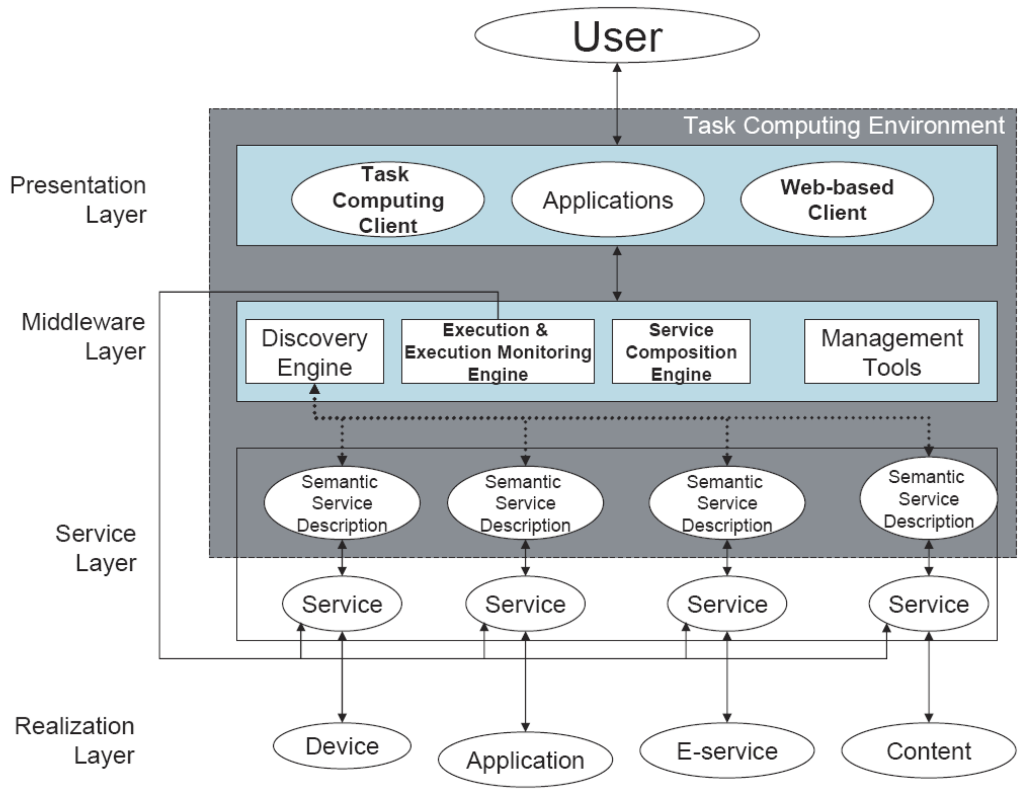


Figure 3.4: General Architecture of Task Computing Environment

3.4.1 Task Computing Architecture

Fig. 3.4 shows the TC architecture [81], which is composed of four different layers, performing complementary activities:

- **Realization Layer:** It is the bottommost layer, directly representing available facilities. There are three different types of entities at this layer: devices, applications and e-services over the Web.
- **Service Layer:** The available facilities from the Realization Layer are embodied into the form of service at this layer and services interfaces are constructed. Semantic Service Descriptions (SSD) comprising knowledge about these services are also created in order to disseminate information.
- **Middleware Layer:** This layer is in charge of service discovery, service composition and execution, and other management activities such as *Service Publishing.* In some way, it glues services created at the Service Layer with available underlaying technologies to support transport and management functions over them.
- **Presentation Layer:** It is considered to be the most important layer in the architecture. It provides the user with an abstraction of available tasks that can be performed in the environment, hiding underneath complexity, and allowing the user

to dynamically assemble components to perform the desired task. A client implementing the Presentation Layer is usually referred to as *Task Computing Client* (TCC) and makes use of well-defined interfaces to the Middleware Layer.

Task Computing is implemented in concrete *Task Computing Environments* (TCEs), which are computational systems able to perform Task Computing functionality and composed of Task Computing Clients, Semantic Service Descriptions, Semantic Service Discovery Mechanisms and Service Controls. The detailed information about all of these components is given in the afore-mentioned literature.

3.4.2 Conclusion

Task Computing is primarily a framework for services orchestration, composition, and execution. All the mechanisms it features are aimed at service publishing and discovery, semantic descriptions, service-ization of resources to make them available to any requester, and so forth. These features can be implemented in multiple environments, not being specially addressed for Ubiquitous Computing scenarios. In fact, Task Computing applies well-known Internet-wide technologies such as UDDI for discovery or traditional Web Services at external servers as endpoints. In order to assume a more context-aware nature, Task Computing has embraced some pervasive computing technologies to complement existing ones, such as UPnP's Simple Service Discovery Protocol (SSDP) for discovery, so services can be found both at a global level via UDDI and at a local level via the *group by subnet* discovery range.

The following sections provide analysis of Task Computing technology against our defined evaluation criteria, and the conclusion is summarized in Table 3.4.

- **Decentralization:** Task Computing is not aimed at creating a network of interconnected devices. The TCEs are intended to run in desktop computers or servers that can be connected to UPnP or Jini networks if device control is required. *Low.*

- **Lightness:** The processes such as semanticization and serviceization as well as the need of the *Task Computing Client* result in complex and heavy software components. The devices need an amount of computing resources (i.e. processing power and screen size) presently not available in every embedded platform. *Low.*

- **Standards Adherence:** In general, Task Computing honors the existing standards and technologies, i.e. OWL-S, WSDL, HTTP, UDDI, and even industry *de facto* standards such as SSDP. *High.*

- **Technological Consistency:** Task Computing applies Web Services and Semantic Web technologies in a coherent and synergistic manner. *High.*

- **Reasoning Capability:** Task Computing uses semantic information to annotate service descriptions and perform service composition, but neither reasoning nor domain ontologies are provided to understand context information. *Low.*

- **Context Awareness:** No form of capturing context or sensing environmental conditions are provided directly by Task Computing, except for service discovery mechanisms which is a technical issue and not related to context-awareness. *Low.*

Table 3.4: Analysis of Task Computing against the evaluation criteria

Criterion	Value
Decentralization	Low
Reasoning Capability	Low
Standards Adherence	High
Lightness	Low
Technological Consistency	High
Context Awareness	Low

3.5 Gaia

Since 2001, the research group at the Department of Computer Science of the University of Illinois at Urbana-Champaign, led by Dr. Roy H. Campbell has been working on the design of an infrastructure to support intelligent Ubiquitous Computing environments. The Gaia project is the result of these efforts, constituting a software infrastructure to support Active Spaces. An active space is a "model that maps the abstract perception of a physical space as a computing system, into a first class software entity" [82].

Thus, the active space acts as a mapping between the real and virtual space, connecting both in such a way that real world actions affect virtual world objects and vice versa. The active space hides the complexity of the real world elements into one unique entity that provides functions for manipulating the space, discovering and locating internal entities, storing and retrieving information from the space and so forth. The name Gaia was adopted from the Gaia theory by James Lovelock that advocated for the earth as a self-regulated super-organism, who in turn borrowed it from the Greek Earth Goddess. The Gaia project tried to replicate the same global awareness and self-regulated behavior for smart environments and their constituent elements.

3.5.1 Gaia Architecture

The *Gaia Operating System* (Gaia OS) is the core element of the whole architecture, which is defined as a meta-operating system, running at the top of others and providing a distributed communication model for coordinating active spaces [83]. Gaia is composed of three main components, as shown in Fig. 3.5.

- **Gaia Kernel:** It provides basic services such as component life-cycle management or remote component execution and management. Gaia relies on CORBA as underlying distributed component model, and extends some CORBA services to provide the so-called Gaia services, such as Event Service, Context Service, Presence Service, Space Repository and Context File System.

- **Application Framework:** It consists of a distributed component-based infrastructure following the MVC (*Model View Controller*) model, including new functionality to manipulate component bindings, a mapping mechanism and policies/rules for application customization.

- **Active Space Applications:** These applications implement the desired functional behavior in the active space, such as the Presentation Manager application that lets the users present slides in multiple displays simultaneously, move slides from one display to another, as well as move the input device functionality.

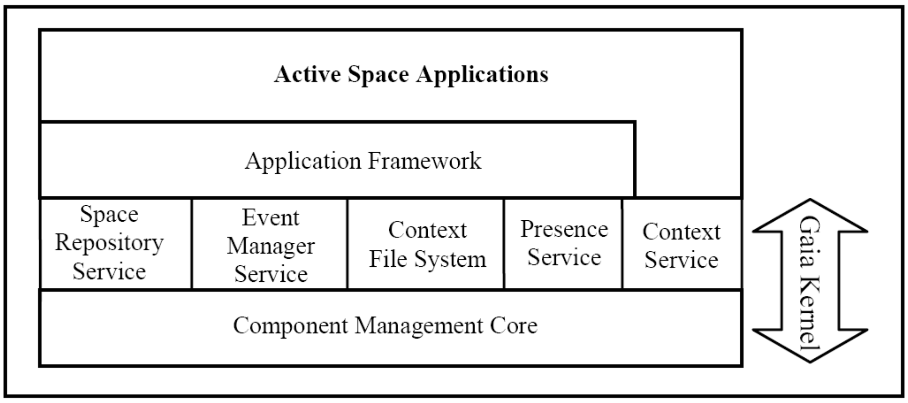


Figure 3.5: General Architecture of Gaia

The mapping mechanism of the Application Framework offers the possibility of describing requirements to find the suitable real device to assign a functional behavior (for example, audio output) so that matching devices can be found within the active space to perform that function during a task. The interesting part is how Gaia represented context in the form of a 4-components structure: Context(<ContextType>, <Subject>, <Relater>, <Object>) that in many ways resembles that of the Semantic Web, for example: Context(locatedIn, CoaguChekS, is, RoomLab123). Later, this model evolved into a predicate-based representation of context information:

- Location(CoaguChekS, isOperating, RoomLab123)
- InternetConnectionStatus(Gateway, is, OffLine)
- User(Safdar, role, Admin)

During 2003, Gaia was extended with a semantic middleware layer for context awareness endorsing existing Semantic Web technologies in order to model and annotate context information, perform reasoning and carry out reactive behavior in response to context changes [84]. DAML+OIL (later OWL) was selected to represent the context information following a predicate model. In order to map the predicates onto the ontology, an ontology class is created for each predicate structure. So, the above-mentioned Location predicate becomes a Location ontology class with three possible relationships to denote the information that was previously enclosed in the predicate parameters.

Representing the context in this way, operations such as search, querying, fusion and so forth, become possible. There are several different entities involved in Gaia's context information infrastructure, as depicted in Fig. 3.6.

- **Context Providers:** These are the sources of context information, probably obtained by sensors.
- **Context Synthesizers:** They retrieve context information from different providers and perform some form of reasoning to infer new information making it available to other agents. Both static rules and machine learning techniques (such as Naive Bayes) can be applied to obtain new information.

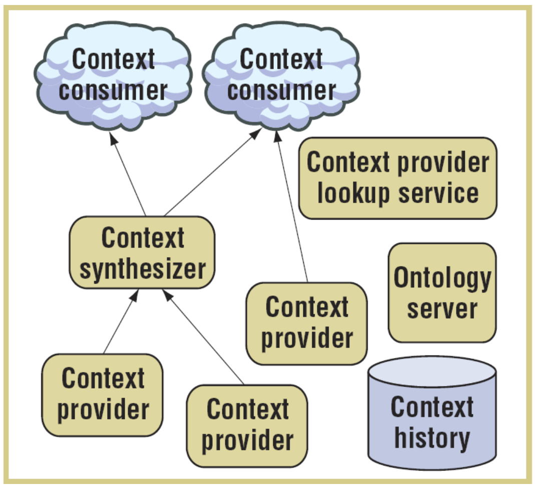


Figure 3.6: The Context Infrastructure in Gaia

- **Context Consumers:** They gather context information from providers and synthesizers, reason about it and perform reactive behavior accordingly.
- **Context Provider Lookup Service:** It is used by the context providers, one per environment, in order to publish the kind of context information they provide in order to be found by context consumers.
- **Context History Service:** It contains database records, one per environment, of the past context information to make them available to the requesting parties.
- **Ontology Server:** It stores ontologies, one per environment, for different types of information.

3.5.2 Conclusion

Gaia fulfills many of the requirements established for a smart Ubiquitous Computing architecture, mainly those related to intelligence support. It makes use of context predicates for representing context information and OWL ontologies for taxonomical purposes. However, Gaia has two important drawbacks and several inconsistencies. The first main drawback is that, as a requisite, three different elements in the architecture must be deployed and properly configured in the environment: the *Context Provider Lookup Service*, the *Context History Service* and the *Ontology Server*. This constraint prevents Gaia from creating spontaneous emergent pervasive computing environments anywhere, since a deployment phase must me performed beforehand, enforcing an undesirable centralization. The second main disadvantage arises from the fact that core elements in the architecture, such as those three mentioned above, seem to be suitable for installation in desktop computers or servers, but not in embedded computers.

The main inconsistencies with Gaia are originated from the initial selection of technologies and the subsequent integration of newer ones, that result in a strange mixture. For instance, representation of context information through predicates was present at the very initial stages, but when OWL ontologies were integrated in Gaia, context predicates remained as knowledge representation mechanisms instead of shifting to RDF, which could have sound more sensible. Another strange mixture is related to the communication model, where Gaia uses CORBA as distributed computing architecture instead of the Web-based model.

The following sections provide analysis of Gaia against our defined evaluation criteria, and the conclusion is summarized in Table 3.5.

- **Decentralization:** The prerequisite of deployment of three main components in the environment leads towards a centralized architecture in Gaia. *Low.*
- **Lightness:** Some elements in the architecture are difficult to migrate to resource-constrained devices, such as the *Context History Service* or the *Ontology Server*. The Gaia architecture does not seem to be deployed on embedded devices, but in desktop computers. *Low.*
- **Standards Adherence:** The standards and recommendation, i.e. CORBA and OWL are honored. *High.*
- **Technological Consistency:** The technological evolution during the Gaia project resulted in lot of changes as new technologies were adopted, which created a strange mixture in the final outcome. The OWL is combined with predicate logics, instead of shifting to RDF. CORBA is applied as distributed computing architecture instead of newer more lightweight web-based communication models that would make the Semantic Web fit better. *Low.*
- **Reasoning Capability:** The application of ontologies, rules, probabilistic logic, fuzzy logic and Bayesian networks in order to perform all sorts of reasoning about context information, promotes a very high degree of intelligence in Gaia. *Very High.*
- **Context Awareness:** The integration of rules and confidence values contribute to situation identification and subsequent adaptation. *High.*

Table 3.5: Analysis of Gaia against the evaluation criteria

Criterion	**Value**
Decentralization	Low
Reasoning Capability	Very High
Standards Adherence	High
Lightness	Low
Technological Consistency	Low
Context Awareness	High

3.6 SCALLOPS

The SCALLOPS[4] (Secure Agent-Based Pervasive Computing) project targets to develop a coherent, methodological framework for the design and implementation of secure agent-based pervasive computing applications in the area of *Personal Health Information* (Health-SCALLOPS). The focus of basic research is on innovative means for flexible agent-coordinated Semantic Web Service discovery and composition, and effective data and service privacy enforcement in open, large scale pervasive computing environments.

[4]SCALLOPS Project; `www.dfki.de/scallops`

3.6.1 Vision of SCALLOPS

Future health information systems will be more and more pervasive within the healthcare system. Daily work in hospitals is dominated already today by mobility of doctors, nurses - and even the patients are on the move. Consequently, there are many medical applications and services for PDAs and other mobile computing devices available and an extensive Web community is making them accessible. Many of these services rely on sensitive data from various medical devices, sensors, and outside laboratories that are usually transfered and processed by sites in different departments, hospitals or doctors' offices. Health-SCALLOPS consists of a potentially vast collection of heterogeneous sites and computing devices that are fixed-line or wireless connected with the Internet. The management of the network infrastructure in Health-SCALLOPS includes appropriate means of resource discovery and all application-specific services are assumed to be hosted at sites of at least one relevant Web Service provider.

3.6.2 Healthcare Scenario of SCALLOPS

This section describes one of the demonstration healthcare scenarios envisaged for SCALLOPS project. Mikka spends his summer vacations in Portugal. Unfortunately, after one week in Lisbon he becomes seriously ill, suffering from a disease unknown to him. In this emergency situation, he activates his personal Health-SCALLOPS agent on his PDA with an integrated phone for help, as shown in Fig. 3.7. The agent quickly finds and calls the nearest local emergency center in Portugal, and Mikka describes the observable symptoms of his disease. At the same time, his Health-SCALLOPS agent transfers general, non-sensitive information about Mikka and his current location to the patient database of the emergency center. The local representative at the Portuguese emergency center quickly recognizes that Mikka's symptoms are very serious and strongly recommends him to immediately visit the nearest hospital. Hospital contact information including geographical map and how to reach the ambulance station are transmitted to Mikka's PDA by the Health-SCALLOPS emergency center agent. Mikka readily arrives at the hospital's ambulance station by taxi, passes his individual *health patient card* (HPC) to the physician, and authorizes him to access his patient record. Mikka may decide whether he wants to get full treatment at the local hospital, or at a hospital of his choice back home in Helsinki. In order to make a reasonable decision, Mikka requests his personal Health-SCALLOPS agent on his PDA for appropriate assistance in this matter.

Given task and requirements, the agent contacts the Health-SCALLOPS agent of Mikka's health insurance (HI) for approval of full coverage of travel and medical expenses in both cases. The HI agent contracts the Health-SCALLOPS agent of an emergency medical assistance (EMA) company in Helsinki to investigate transport options and cost estimations with respect to the regulatory constraints of the insurance fund. For this purpose, the EMA agent plans a composite service to go home and have his treatment in Helsinki. Appropriate services in the network check the availability of regular or charter flights from Lisbon to Helsinki, accommodation and medical escort of Mikka to and from the airport, and the availability of a physician for Mikka's treatment at the hospital in Helsinki at the time of his arrival. Optional plans of patient transport and costs are reported back to the HI agent, which negotiates options for Mikka's treatment with the local hospital agent in Lisbon and pharmacy agents in the network for purchasing the required drugs for his treatment. All agents involved use the information of some individual experience while handling similar emergency cases in the past. The results of both kinds of negotiations enable the HI agent to make its decision through individually

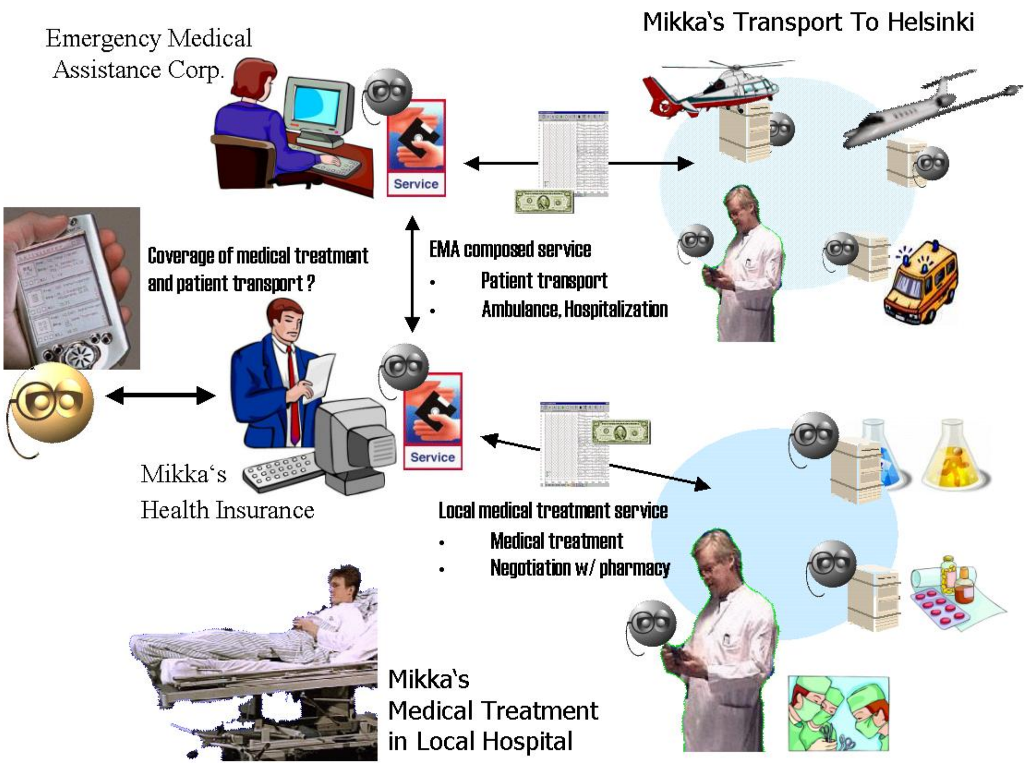

Emergency Medical Assistance Corp.
Service
Mikka's Transport To Helsinki
Coverage of medical treatment and patient transport ?
EMA composed service
• Patient transport
• Ambulance, Hospitalization
Service
Mikka's Health Insurance
Local medical treatment service
• Medical treatment
• Negotiation w/ pharmacy
Mikka's Medical Treatment in Local Hospital

Figure 3.7: SCALLOPS Scenario - Coverage of Emergency Medical Care and Transport

composed health care expense approval service of Mikka's insurance fund according to his membership status. In this scenario, the HI agent confirms full coverage of expenses for local treatment and recreation in Stockholm, as well as return flight, but no emergency transport with escort back home.

Finally, after two weeks treatment in the hospital and one week of full recovery in a recreation area in the outer bound of Stockholm selected by his insurance fund, Mikka returns to Helsinki and continues his summer vacation with his family.

3.6.3 Conclusion

SCALLOPS provides means of Semantic Web Service discovery via middle agents such as brokers and matchmakers, which adapt to the non-deterministically changing environment, where agents and services may enter or leave at any time. Therefore, SCALLOPS agents are able to interleave service composition planning with service discovery to find alternative services provided by other agents. SCALLOPS fulfills many of the requirements established for the agents-based secure communication of a smart ubiquitous computing architecture, mainly those related to intelligence support. It makes use of context predicates for representing context information and OWL ontologies for taxonomical purposes.

The main disadvantage of SCALLOPS is that its architecture is not targeted to be used or hosted *solely* on mobile devices, hence the core components developed in this project, i.e. Semantic Web Service composition planner and matchmaker can not be deployed on resource-constrained medical devices. The following sections provide analysis of SCALLOPS against our defined evaluation criteria, and the conclusion is summarized in Table 3.6.

- **Decentralization:** SCALLOPS architecture uses JXTA technology, which enables to create decentralized peer-to-peer applications for the connected devices on the network, ranging from cell phones and wireless PDAs to PCs and servers, to communicate and collaborate in a peer-to-peer manner. *Very High.*
- **Lightness:** Some core components of the SCALLOPS architecture can not be migrated to resource-constrained medical or mobile devices, such as the Semantic Web Service *composition planner. Low.*
- **Standards Adherence:** The standards and recommendations, i.e. OWL-S, OWL are honored. *High.*
- **Technological Consistency:** SCALLOPS applies multi-agents, Web Services, Semantic Web technologies in a coherent and synergistic manner. *High.*
- **Reasoning Capability:** The application of ontologies (OWL-DL), rules and fuzzy logic in order to perform reasoning about context information, promotes a high degree of intelligence in SCALLOPS. *High.*
- **Context Awareness:** The integration of rules and confidence values contribute to situation identification and subsequent adaptation. *Very High.*

Table 3.6: Analysis of SCALLOPS against the evaluation criteria

Criterion	**Value**
Decentralization	Very High
Reasoning Capability	High
Standards Adherence	High
Lightness	Low
Technological Consistency	High
Context Awareness	Very High

3.7 Comparative Analysis

Table 3.7 shows the comparative analysis of the research projects discussed in the light of evaluation criteria, and ranks the SCALLOPS and HYDRA projects *best* among all others. The final weighted value for each criteria is obtained by multiplying the *criterion value* with the *criterion weight*, as mentioned in Table 3.1, while the criterion value is assigned as: None = 0, Low = 1, Medium = 2, High = 3, Very High = 4. In general, all the architectures feature a correlation among *intelligence* and *centralization*, which implies that higher the level of intelligence in the system, the more centralized it is.

As mentioned earlier, an Hydra device enjoys the full memory & computing resources of a PC/Notebook in order to control other attached devices, and to expose their functionalities as Web Services, which puts a question mark on its *lightness* and prevents using the Hydra middleware entirely on resource-constrained devices. Same is the case with SCALLOPS architecture. Although, the Web Services' description is enhanced using Semantic Web technologies, i.e. SAWSDL, OWL-S and ontologies, but they still lack with the capability of a semantic discovery protocol for mobile devices, which is inevitably required to provide the serendipitous collaboration among smart devices in different AmI scenarios. All this leads us to develop a lightweight SOA framework for the resource-constrained embedded systems with integrated miniaturized components, i.e. knowledge querying and reasoning system, together with a lightweight semantic discovery protocol, besides making maximum use of the existing experiences and technologies as much as possible.

Table 3.7: Comparative analysis of related work against the evaluation criteria

Criterion	**SCALLOPS**	**HYDRA**	**SODA**	**Gaia**	**Task Computing**
Decentralization	*Very High (16)*	*Very High (16)*	*High (12)*	*Low (4)*	*Low (4)*
Reasoning Capability	*High (12)*	*High (12)*	*Low (4)*	*Very High (16)*	*Low (4)*
Standards Adherence	*High (6)*	*High (6)*	*High (6)*	*High (6)*	*High (6)*
Lightness	*Low (3)*	*Low (3)*	*Very High (12)*	*Low (3)*	*Low (3)*
Technological Consistency	*High (9)*	*High (9)*	*High (9)*	*Low (3)*	*High (9)*
Context Awareness	*Very High (4)*	*High (3)*	*Low (1)*	*High (3)*	*Low (1)*
Total	**50**	**49**	**44**	**35**	**27**

Chapter 4

Semantic Medical Devices Space

This chapter provides the details about the semantic middleware infrastructure, named *Semantic Medical Devices Space* (SMDS), that has been developed to provide a pervasive computing platform for the semantic interoperability of *Ambient Intelligent* (AmI) medical or mobile devices by exploiting the *Semantic Web* and *Semantic Web Services* (SWSs) technologies. SMDS is designed and developed as a framework[1] which could be used or integrated easily in the existing *novel* communication software solutions for the resource-constrained medical or mobile devices to enhance them with the capabilities of *ambient intelligence* and *semantic coordination.* SMDS provides a platform for all types of *real-time* and *non-real-time* medical devices and is particularly suited for the emerging pervasive computing healthcare environments consisting of many interconnected medical or mobile devices. On the other hand, SMDS can also be used on Hospital Information Systems (HISs) or Laboratory Information Systems (LISs) sides to expose their functionalities regarding medical data *reception* or *provision* as SWSs, in order to allow the AmI medical devices to send their measurement results or request a particular patient's medical information.

4.1 Philosophy of SMDS

SMDS is built on the philosophy of *ad-hoc networking* of autonomous medical or mobile devices, which includes the *semantic discovery/matching* of the desired devices and then *coordinating* with the matched devices based on the SWSs offered by them. Before any of two autonomous medical or mobile devices can interact with each another, they need to know what *interfaces* each of them supports and what *protocols or commands* they understand. In ambient intelligent healthcare environments, this cannot be known in advance, where new devices or sensors may enter in the environment at any time and need to interact with the existing medical devices and the HISs or LISs. This interaction must be based on unambiguous and commonly agreed terms and concepts, which are well-defined in the application/domain ontologies.

[1]In a Java library form, named smds.jar

4.2 Overall Architecture of SMDS

This section describes the detailed information about the complete architecture of SMDS. Fig. 4.1 shows the *multi-tier* architecture of SMDS, showing the inter and intra-layer interaction of all the components. However, the detailed UML design diagrams of SMDS software framework are given in Appendix A.

4.2.1 Storage Layer

The *Storage Layer* is responsible for storing all types of information locally on each medical device. These types could include storing the measurement results of a medical device in a small database, the RDF triples, the application/domain ontologies and the context-aware application rules in a knowledge base. The following sections describe each type of information in detail that is stored locally.

Device Database

The medical devices performing measurements and producing measurement results use the *device database* to store these results permanently. Every measurement is consisted of various *parameters* and other important *properties* including its date/time and (optionally) the patient's ID. For example, when a urine analyzer performs urine analysis, it measures not only different urinary parameters, i.e. pH, LEU, NIT etc. but stores the date/time and the patient's ID as well. Fig. 4.2 shows generalized schema of a medical device database. The Measurement, Measurement_Values and Parameter are the most important tables which keep record of a complete measurement, including the medical device identifier, so that when a medical device sends its measurement to a HIS or LIS, the system could keep record of which medical device(s) performed which measurement(s). In Appendix A, Fig. A.7 shows the UML class diagram of device database.

Application/Domain Ontologies

This unit is used to store the application/domain ontologies in OWL/RDF(S) form. These ontologies are used to express the *physical* and/or *functional* characteristics of a medical or mobile device, the *context* in which a medical or mobile device is being used, and the *rules* which a context-aware application must follow in order to fulfill a task. Fig. 4.3 shows the snapshot of partial medical device ontology that is used to express the *physical characteristics* of a medical or mobile device, while Fig. 4.4 shows the partial snapshot of measurement capability ontology that is used to express one of its *functional characteristics.*

Device Knowledge Base

The *device knowledge base* is a file based storage unit, which is used to store the *physical* and/or *functional* characteristics of a medical or mobile device in the form of RDF triples, described using the application/domain ontology(ies), and associated with the instance of that particular device. As shown in Fig. 4.1, the device knowledge base is used by the *Knowledge Querying and Reasoning Engine* during the medical device discovery process. The following snippet is a part of the knowledge base of a medical device, which shows its physical property (*device vendor*).

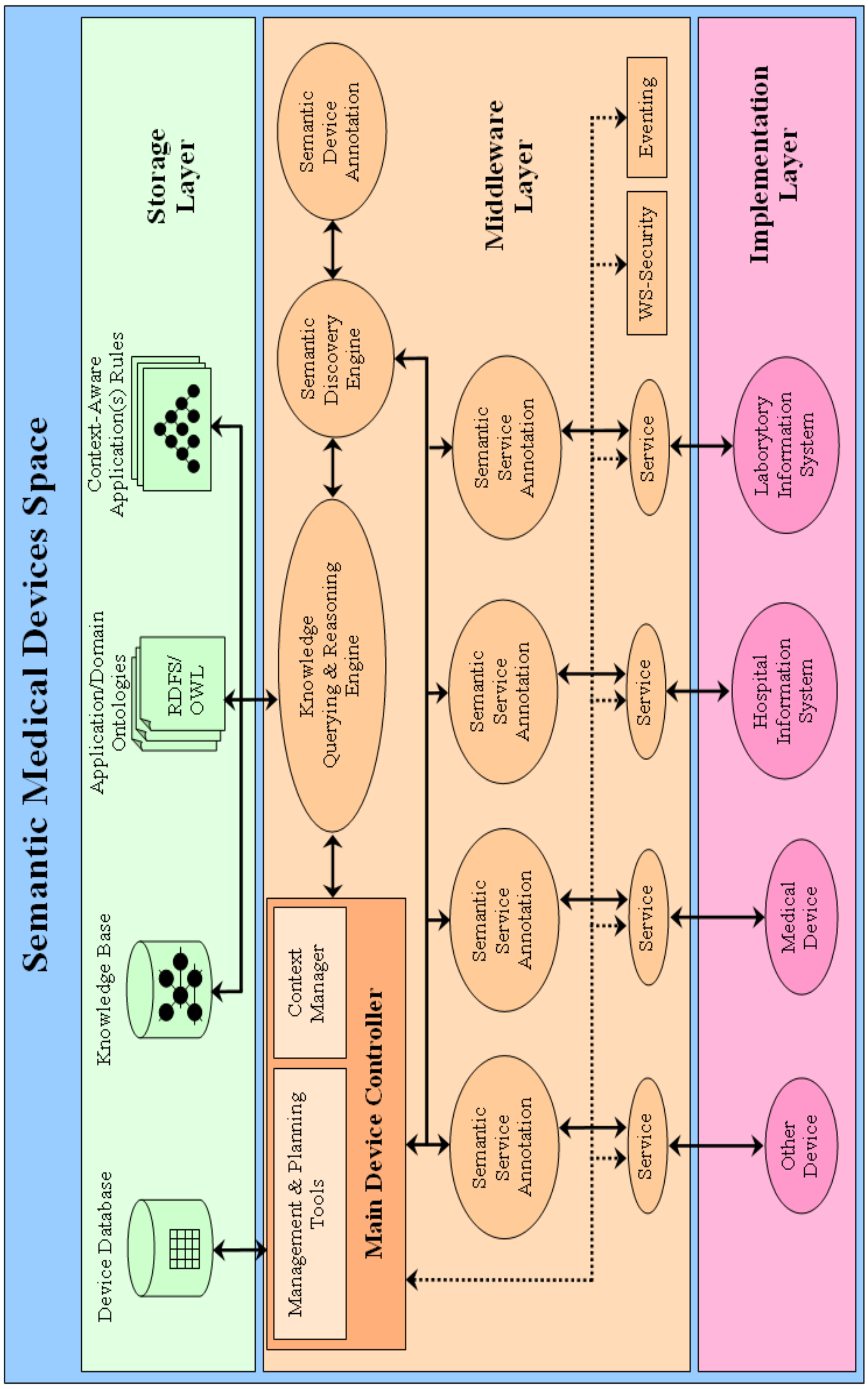

Semantic Medical Devices Space
Storage Layer
Device Database
Knowledge Base
Application/Domain Ontologies
RDFS/ OWL
Context-Aware Application(s) Rules
Middleware Layer
Management & Planning Tools
Context Manager
Main Device Controller
Knowledge Querying & Reasoning Engine
Semantic Discovery Engine
Semantic Device Annotation
Semantic Service Annotation
Semantic Service Annotation
Semantic Service Annotation
Semantic Service Annotation
Service
Service
Service
Service
WS-Security
Eventing
Implementation Layer
Other Device
Medical Device
Hospital Information System
Laborytory Information System

Figure 4.1: Overall Architecture of Semantic Medical Devices Space

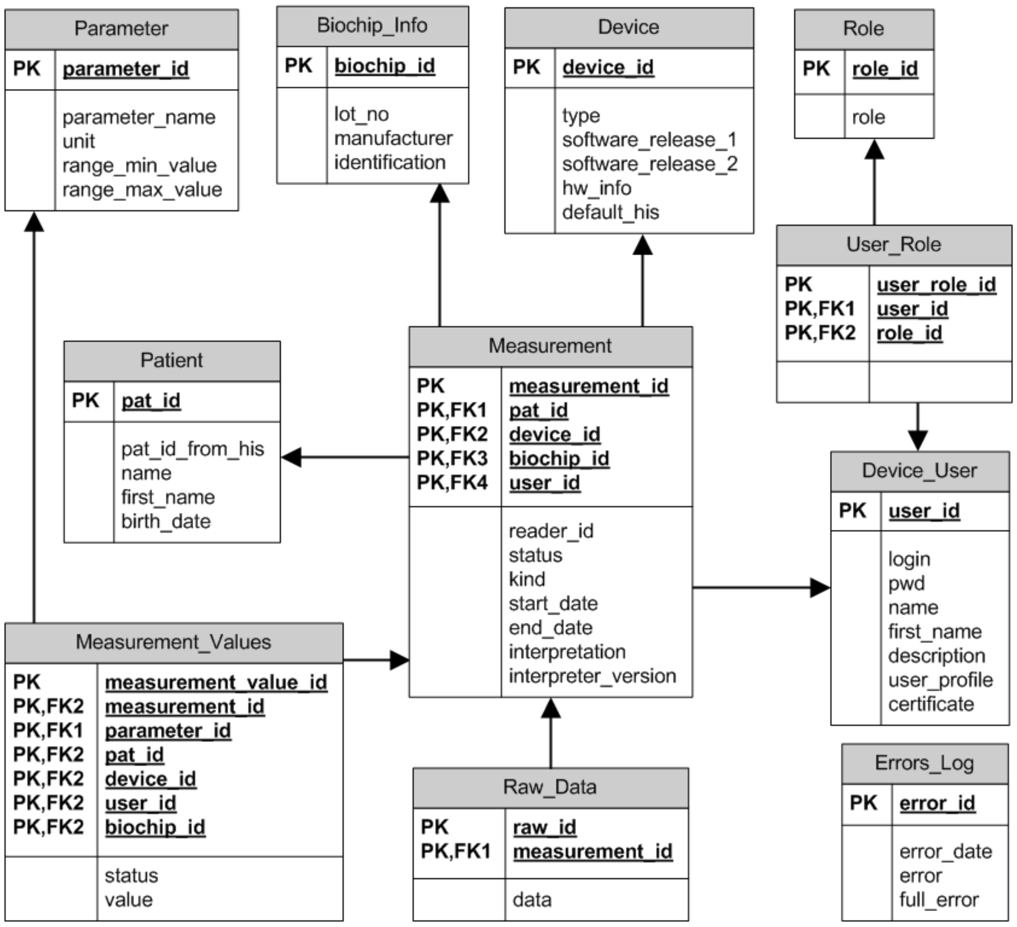


Figure 4.2: Generalized Schema of a Medical Device Database

```
<mdonto:DeviceVendor rdf:about="http://vendors.com/Roche">
   <mdonto:vName>Roche Diagnostics</mdonto:vName>
   <mdonto:vUrl>http://www.roche.de</mdonto:vUrl>
</mdonto:DeviceVendor>
```

Context-Aware Application Rules

It is a file(s) based storage unit, which is used to store the *behavioral rules* for the context-aware application(s) of medical devices. A *rule* is a combination of two or more *pre-conditions* and a *post-condition*, and in order to execute the post-condition, all of the pre-conditions must be fulfilled. These behavioral rules, or *profile(s)* are either stored locally on the medical devices or obtained from the external sources, e.g. a central repository. Within the context of this thesis, this unit is not fully implemented, as we restricted the behavior of medical devices only for the envisaged healthcare scenarios, where only a *single* medical application is executed.

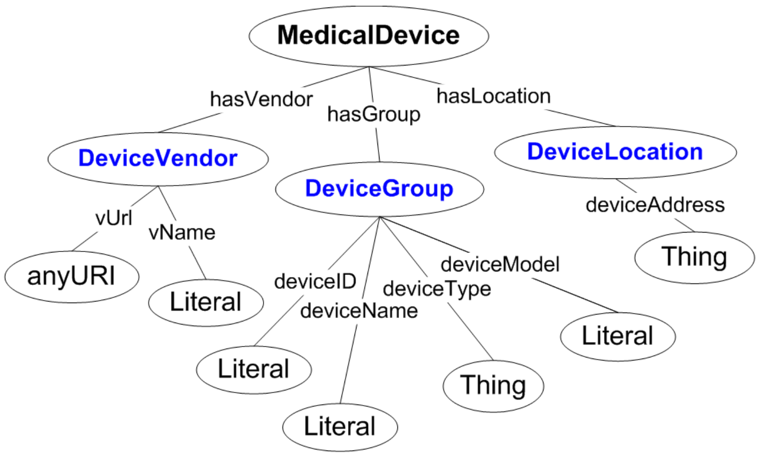


Figure 4.3: Medical Device Ontology - A Partial Snapshot

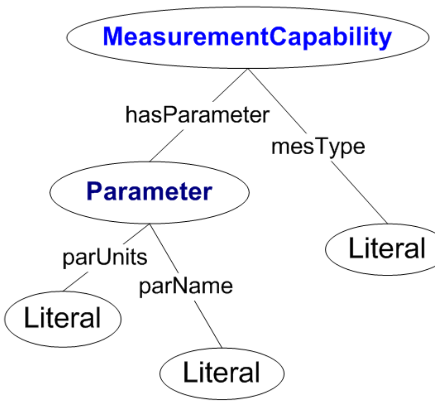


Figure 4.4: Measurement Capability Ontology - A Partial Snapshot

4.2.2 Middleware Layer

Middleware Layer is the core-layer of the whole SMDS infrastructure, as it provides various components for a range of tasks. The following sections describe each type of component and its characteristics in detail.

Service

The *Service* component is actually the Web Service that is hosted by a medical or mobile device. As Fig. 4.1 shows, there could be more than one Web Services offered by a medical or mobile device, catering different type of its client, i.e. a medical device, a simple mobile device, an HIS or LIS. This Web Service functionality can be developed using any programming language, i.e. Java, and can be hosted using any compact Web Service server. The implementation details for service component of SMDS framework are given in Chapter 7.

Semantic Service Annotation

The *Semantic Service Annotation* component is used to semantically annotate and enhance the Web Services description, which is usually described in a WSDL (Web Service Description Language) file. In SMDS infrastructure, the WSDL description of Web Services offered by a medical or mobile device is enhanced/annotated using the W3C standard, called SAWSDL (Semantic Annotation for WSDL and XML Schema) [48], which requires the application/domain ontologies to annotate the plain WSDL document with semantics. Such semantic annotation of Web Service helps during the device discovery process, when a medical or mobile device is searched based on its *functional characteristics. Please note that in the whole thesis, when we talk about the execution or invocation of Semantic Web Service of a medical or mobile device, we actually mean the execution of its* **grounding***, which has binding with its SAWSDL description.* Further implementation details and UML class diagram of this component are given in Chapter 7 and Appendix A, respectively.
When a medical or mobile device is matched with the requirements of the client medical or mobile device, the SAWSDL file of the matched device is *downloaded* on the client device side and *parsed* to extract the Web Service method name, which is then invoked automatically. While parsing the SAWSDL file, the input parameter *types* and their *sequence* as well as the output parameter type are matched with the client's requirements (input parameter types, output parameter type). This process or task is similar to what a service broker performs, which is to match the available Web Services with the requirements given by a Web Service client.

Semantic Device Annotation

The *Semantic Device Annotation* component is used to semantically annotate the physical characteristics of a medical device. In SMDS infrastructure, these physical characteristics are annotated using application/domain ontologies, as described under Section 4.2.1. Such semantic annotation helps during the semantic medical device discovery process, when a medical device is searched based on its *physical characteristics.*

WS-Security

One of the most important aspects of any software systems is its security, which means how it protects itself from different foreign attacks and how it allows the desired users to access its functionalities. In SMDS infrastructure, the security mechanism is developed using the guidelines laid down in *WS-Security* [86] specification. It is made possible to configure which security functionalities are required, i.e. if it is desired to have encryption and decryption functionalities, both of these options are set to yes or true in the configuration file. Also, if it is desired to have signing and verification functionalities too, both of these can also be set to yes or true in the configuration file. The Web Service SOAP messages, which are encrypted and/or signed, takes more time to reach on other node(s) of the network as compared to the plain SOAP messages.

Fig. 4.5 shows the complete sequence diagram of the security handling between two medical devices in SMDS infrastructure, where Device A acts as a *Client* device, while Device B acts as a *Server* device. The important steps for the medical devices in this sequence diagram are downloading the digital certificate(s) of other medical device(s) and then storing them in their local *truststores*, if it does not exist already. The URI to download the digital certificate is provided in the response messages, as shown in <cert>

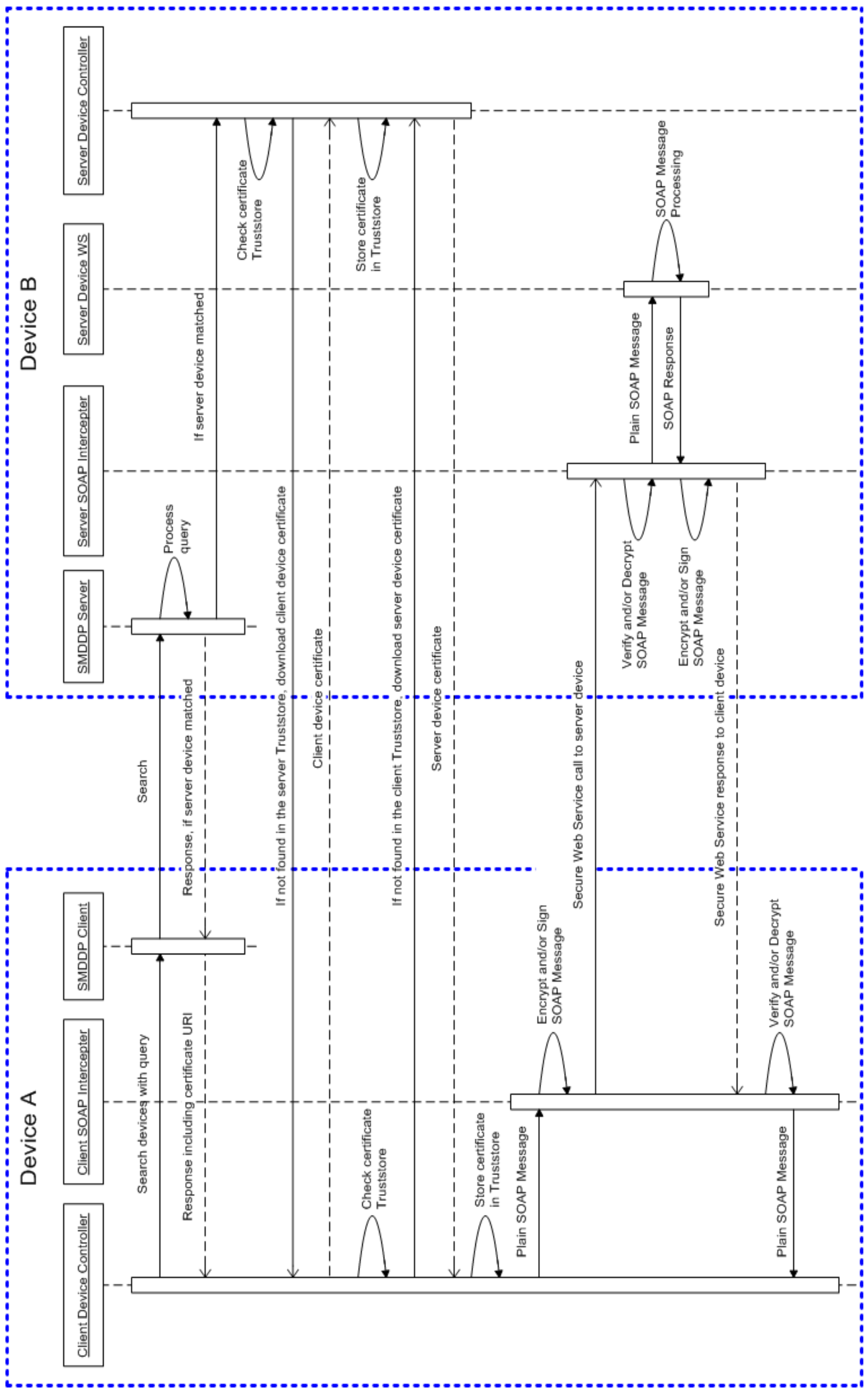

Device A
Client Device Controller
Client SOAP Intercepter
SMDDP Client
Device B
SMDDP Server
Server SOAP Intercepter
Server Device WS
Server Device Controller
Search devices with query
Search
Process query
If server device matched
Response, if server device matched
Response including certificate URI
Check certificate Truststore
If not found in the server Truststore, download client device certificate
Client device certificate
Store certificate in Truststore
Check certificate Truststore
If not found in the client Truststore, download server device certificate
Server device certificate
Store certificate in Truststore
Plain SOAP Message
Encrypt and/or Sign SOAP Message
Secure Web Service call to server device
Verify and/or Decrypt SOAP Message
Plain SOAP Message
SOAP Message Processing
SOAP Response
Encrypt and/or Sign SOAP Message
Secure Web Service response to client device
Verify and/or Decrypt SOAP Message
Plain SOAP Message

Figure 4.5: Security Mechanism of Semantic Medical Devices Space

tag of Listing 6.4. In Appendix A, Fig. A.6 shows the UML class diagram of security framework implementation.

Eventing

The *Eventing* mechanism is used in communication protocols, where a client device registers itself on a server device in order to receive a notification about a change in the state of its variable(s). In SMDS infrastructure, this mechanism is partially implemented in order to facilitate the medical devices to get notifications from other medical devices, whenever they are finished with performing their measurement(s). When a client medical devices receives such notification, it can directly call the Web Service method of that medical device to retrieve its measurement results.

Knowledge Querying and Reasoning Engine

The *Knowledge Querying and Reasoning Engine* (KQRE) is one of the research outcomes of this thesis, and it has been described in detail in Chapter 5. The KQRE facilitates the medical or mobile device to perform querying on the local device knowledge base, as well as to perform Description Logics and/or Rules based reasoning to generate inferences (*new facts*) from this knowledge base, based on the available application/domain ontologies.

Semantic Discovery Engine

The *Semantic Discovery Engine* (SDE) is also one of the research outcomes of this thesis, and it has been described in detail in Chapter 6. The SDE, which internally uses the KQRE, facilitates a medical or mobile device to semantically search for the desired medical or mobile devices based on their physical and/or functional characteristics.

Main Device Controller

The *Main Device Controller* (MDC) is the central component of SMDS infrastructure, which acts as a manager and provides coordination among all the components. The MDC controls the sequence of launching and dis-launching the relevant processes, whenever it is desired during the semantic discovery of medical devices or during the Web Service communication for the retrieval of measurement results.

4.2.3 Implementation Layer

The *Implementation Layer* shows the entities, i.e. medical device, other (mobile) device, hospital information system, and/or laboratory information system, which can make use of SMDS infrastructure, although the SMDS is primarily targeted for the resource-constrained medical or mobile devices. A brief illustration of each type of entity is given below.

Medical Device

The primary target entity of SMDS infrastructure is the *medical device* class, as the title of the thesis suggests. The SMDS is realized and tested on *non-real-time* medical devices, i.e. urine analyzer, blood coagulation meter, weight scale, as well as on *real-time* medical devices, i.e. ECG. The experimental evaluation details about all of these medical devices is given under Chapter 8.

Other Device

The secondary target entity of SMDS infrastructure is other device, which includes all other classes of resource-constrained devices, except medical devices. Such classes include mobile devices, robots with sensors and actuators etc. Within the context of this thesis, only the mobile devices have been tested along with other medical devices for the envisioned healthcare scenarios.

Hospital Information System & Laboratory Information System

Although, SMDS is not targeted for the information systems side, i.e. hospital information system, and/or laboratory information system, but it can be used to expose the stored medical information of patients as Semantic Web Services, which will enable the HIS and/or LIS to be involved in the semantic discovery process. In advanced healthcare scenarios, where the medical devices send/receive the patients' medical information to the information systems, such functionalities are inevitably required. Within the context of this thesis, only the HIS entity is tested and realized regarding sending the measurement information from medical device(s) to HIS through Semantic Web Services. The experimental evaluation details are covered under Chapter 8 regarding the semantic coordination of medical devices and hospital information system, and/or laboratory information system.

Chapter 5

Micro OWL Querying and Reasoning System

This chapter describes the design and implementation details of a powerful micro OWL Description Logic and Rules based Reasoning & Querying system, named μ**OR**, which is developed to support semantic *querying* and *reasoning* capabilities *entirely* integrated on the resource-constrained medical or mobile devices. μOR is an integral part of *Semantic Medical Device Space* (SMDS) middleware framework, as described in Chapter 4, which is designed and developed for the semantic interoperability of next generation Ambient Intelligent (AmI) medical devices. Section 5.1 presents details about the syntax specification and semantic formalism of SCENT language that is designed to express the semantic queries on the device knowledge base, while Section 5.2 describes the SCENTRA algorithm, which is a *resolution* and *patterns matching* algorithm designed for SCENT queries. Sections 5.3.1 and 5.4 outline the expressivity requirements and the architectural details of μOR, while makes use of SCENT and SCENTRA to query the knowledge base and make inferences on it. Section 5.5 gives a comparative analysis of μOR with some other reasoning systems, while the last Section 5.6 discusses the scalability issues of μOR.

5.1 SCENT - Semantic Device Language for N-Triples

The Resource Description Framework (RDF)[39] is a data model for representing information about World Wide Web resources. In order to query RDF data, several query languages have been designed and developed (see [87] for a detailed comparison of these languages). SPARQL [88] is a W3C standard and probably the best positioned candidate, as its syntax provides rich levels of expressiveness for constructing *conditions* about resources in a RDF graph, and provides its efficient query engine [89] for query processing. Normally, the resource-constrained medical or mobile device do not have sufficient memory/computing power to host such a query engine and to support such a query language, when required. One solution is to use N-Triples [90] instead, which is the basic standardized RDF notation and a subset of Notation 3 [91]. N-Triples features a line-based, absolute URIs based, plain-text format and a simple grammar for encoding RDF graphs, and allows typed literals and blank nodes. However, an important drawback in N-Triples notation is that it can only express *concrete* RDF triples, neither *patterns* nor *conditions* which are required to express a semantic query.

To cope with this problem, a simple alternative solution must be developed by modifying the EBNF (*Extended Backus-Naur Form*) grammar of the original N-Triples specification, as described in [92]. We have developed a simple RDF query language, named SCENT (Semantic Device Language for N-Triples), which represents a subset of SPARQL expressiveness much in the similar way as N-Triples represents a subset of Notation 3 expressiveness, and its simplicity makes it possible to be processed by resource-constrained medical or mobile devices. In the following sections, we describe the EBNF grammar, algebraic syntax and the formal semantics of SCENT query language on the similar lines as described for SPARQL query language in [93].

5.1.1 EBNF Syntax of SCENT Query Language

Table 5.1 shows a glimpse of comparison between original N-Triples syntax and the SCENT syntax specifications. The original subject and object productions are modified with the inclusion of variable term (starting with ? sign) which now allows us to use *variables* at the places of subject or object of a triple to build a semantic query.

Table 5.1: Comparison between original N-Triples and SCENT syntax specifications

Original N-Triples specification	**SCENT syntax specification**
subject ::= uriref | nodeID object ::= uriref | nodeID | literal	subject ::= uriref | nodeID | **variable** object ::= uriref | nodeID | literal | **variable** **variable ::= ’?’ name**

Listing 5.1 shows the complete EBNF grammar of SCENT query language, which uses the above modification made in the *subject* and *object* productions, and encapsulates some of the productions used in EBNF of N-Triples [90], i.e. ws, absoluteURI, CRLF and datatypeString. A SCENT query (line 1) comprises one or more SCENT patterns/conditions where each pattern is an RDF like triple and composed of *subject*, *predicate*, optional *operator*, *object* (line 2) and a *dot* at the end, followed by a *carriage return* and *line feed* characters (line 6). Other productions (lines 3-11) are simple and self-explanatory to define subject, predicate, operator and object of each condition triple. The optional operator production, as the name suggests, is used to define *comparative* binary operator for the object of a pattern, and its default value is Equals. Other possible values are NotEquals, GreaterThan, GreaterOrEqualsThan, LessThan and LessOrEqualsThan.

Listing 5.1: EBNF Grammar of SCENT Query Language

```
1   scent-query ::= [scent-condition]+
2   scent-condition ::= subject ws predicate ws [operator] ws object
3   subject ::= uriref | nodeID | variable
4   predicate ::= uriref
5   operator ::= uriref
6   object ::= [uriref | nodeID | literal | variable] ws ’.’ CR LF
7   uriref ::= ’<’ absoluteURI ’>’
8   nodeID ::= ’_:’ name
9   literal ::= ’_:’ datatypeString
10  variable ::= ’?’ name
11  name ::= [A-Za-z][A-Za-z0-9]*
```

5.1.2 Semantics of SCENT Query Language

In this section, we first present the algebraic formalization of the core fragment of SCENT, which is then used to define formal semantics of SCENT, as well as to study fundamental properties of this language. We start by introducing the necessary notions about RDF (for details on the formalization of RDF see [39][94]).

Definition 5.1 (**RDF Terms, Triples, and Variables**)**.** Assume that there are pairwise disjoint sets I, B, and L (URIs, Blank nodes, and Literals, respectively). A tuple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple.* In this tuple, s is the *subject*, p the *predicate* and o the *object*. We denote the union $I \cup B \cup L$ by T (*RDF Terms*). Assume additionally the existence of a set V of *variables* disjoint from the above sets.

Definition 5.2 (**RDF Graph** [39])**.** An RDF graph is a set of RDF triples, where the *nodes* is the set of *subjects* and *objects*, and the *edges* is the set of *predicates* (a.k.a *properties* or *directed arcs*) of RDF triples in the graph. The directed arc shows relationship between subject and object, and always points towards the object. If G is an RDF graph, then $\text{term}(G)$ is the set of elements of T appearing in the triples of G, and $\text{blank}(G)$ is the set of blank nodes appearing in G, i.e. $\text{blank}(G) = \text{term}(G) \cap B$.

Definition 5.3 (**Mapping**)**.** A *mapping* μ from V to T is a partial function $\mu : V \to T$. The domain of μ, denoted by $\text{dom}(\mu)$, is the subset of V where μ is defined. The empty mapping $\mu_\emptyset$ is a mapping such that $\text{dom}(\mu_\emptyset) = \emptyset$.

To define the algebraic structure of SCENT, we need to introduce the notions of *triple pattern* and *scent graph pattern.* A triple pattern is a tuple $t \in (I \cup V) \times I \times (I \cup L \cup V)$ and a scent graph pattern is a finite set of triple patterns. Notice that a triple pattern is essentially an RDF triple with the *subject* and *object* positions replaced by variables. Also notice that in our definitions of triple pattern and scent graph pattern, we are not considering *blank nodes*[1]. We make this simplification here to focus on the *pattern matching* part of the language.

Definition 5.4. Let P be a SCENT graph pattern, and $\text{var}(P)$ is used to denote the set of variables occurring in P, then the SCENT query is a tuple (W,P) where $W = \text{var}(P)$.

In order to define the semantics of SCENT graph pattern expressions, we use the algebraic representation of SCENT introduced above, and start by introducing some formal terminologies in the light of SPARQL semantics [93].

Definition 5.5 (**Triple and SCENT Graph Pattern**)**.** A tuple $t \in (I \cup V) \times I \times (I \cup L \cup V)$ is a *triple* pattern. A *SCENT Graph Pattern* is a finite set of triple patterns. Given a triple pattern t, $\text{var}(t)$ is the set of variables occurring in t. Similarly, given a SCENT graph pattern P, $\text{var}(P) = \bigcup_{t \in P} \text{var}(t)$, i.e. $\text{var}(P)$ is the set of variables occurring in P.

Definition 5.6 (**SCENT Graph Pattern and Mappings**)**.** Given a triple pattern t and a mapping μ such that $\text{var}(t) \subseteq \text{dom}(\mu)$, $\mu(t)$ is the triple obtained by replacing the variables in t according to μ. Similarly, given a SCENT graph pattern P and a mapping μ such that $\text{var}(P) \subseteq \text{dom}(\mu)$, we have that $\mu(P) = \bigcup_{t \in P} \{\mu(t)\}$, i.e. $\mu(P)$ is the set of triples obtained by replacing the variables in the triples of P according to μ.

[1]A *blank node* (or anonymous resource or bnode) is a node in an RDF graph which is not a *URI reference* or a *literal*. In graph patterns, blank nodes are essentially defined as variables whose values cannot be retrieved by a query

Definition 5.7 (**RDF Graph Evaluation**). Let G be an RDF graph over T, and P a SCENT graph pattern. The *evaluation* of P over G, denoted by $[\![P]\!]_G$, is defined as the set of mappings:

$$[\![P]\!]_G = \{\mu : V \rightarrow T \mid \mathrm{dom}(\mu) = \mathrm{var}(P) \text{ and } \mu(P) \subseteq G\} \qquad \square$$

If $\mu \in [\![P]\!]_G$, we say that μ is a solution for P in G.

Note: For every RDF graph G, $[\![\emptyset]\!]_G = \{\mu_\emptyset\}$, i.e. the evaluation of an empty SCENT graph pattern against any graph always results in the set containing only the empty mapping. For every SCENT graph pattern $P \neq \emptyset$, $[\![P]\!]_\emptyset = \emptyset$

Definition 5.8 (**SCENT Query Result**). Given a SCENT query (W, P), where P is a SCENT graph pattern and $W = \mathrm{var}(P)$, then the evaluation/answer of (W, P) in a triples graph G is the *evaluation* of P over G, as defined above.

$$[\![(W, P)]\!]_G = [\![P]\!]_G \qquad \square$$

5.1.3 Comparison between SCENT and SPARQL Expressiveness

In this section, we compare the expressiveness of SCENT and SPARQL query languages, as well as show how a SCENT query can be easily converted to a SPARQL query, when required. The query model of SCENT is **query-by-example** style, which means that the query specifies the known literals and leaves the unknowns as variables. Furthermore, all SCENT patterns represent *conjunctive* conditions, hence the variables that occur in multiple patterns imply *joins*. It means that the SCENT language supports only SELECT like query, which produces constructions of bounded variables with values. On the other hand, SPARQL queries are much more powerful than SCENT queries, as they can express conditions and filters with logical connectives and a broad range of operators, i.e. FILTER and OPTIONAL. Besides SELECT queries, SPARQL also supports CONSTRUCT and DESCRIBE queries which return a *graph*, and ASK query which returns a *boolean* value. However, all these additional constructions and keywords are neither required nor useful during the discovery process of medical or mobile devices.

Listing 5.2 shows an example of a SCENT query[2] which is composed of three *patterns or conditions*, where each pattern contains *at least* one variable and is terminated with a 'dot'. This semantic query is about searching all the *urine analyzers* in a pervasive healthcare environment, i.e. a clinical laboratory or a hospital. SCENT queries can be easily formulated using any *domain/application* ontologies to exhibit the *physical* and/or *functional* characteristics of the medical or mobile devices and their services. However, this example SCENT query is formulated using our own developed MeDO ontology, as described under Section 4.2.1, which is used to encode or express *only* the physical characteristics of a medical device. The condition in line 1 says that we are searching all the devices (?d) having *type* of MedicalDevice, the condition in line 2 says that the devices (?d) belong to particular *group* (?g), and the condition in line 3 says that this group (?g) of devices has *category/type* of UrineAnalysis.

[2]Apparent line breaks are due to the space (width) limitation

Listing 5.2: Example of a SCENT Query

```
?d <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://www.ibmt.fhg.de/onto/2008/04/md#MedicalDevice> .
?d <http://www.ibmt.fhg.de/onto/2008/04/md#hasGroup> ?g .
?g <http://www.ibmt.fhg.de/onto/2008/04/md#deviceType>
    <http://www.ibmt.fhg.de/onto/2008/04/md#UrineAnalysis> .
```

On the other hand, Listing 5.3 shows how the above SCENT query can be easily translated into a SPARQL query. As we can see, the SPARQL language supports the use of PREFIX for the predicates of RDF triples, but SCENT does not support it because it is not supported by N-Triples. From this translated SPARQL query, it is reflected that if the future medical or mobile devices get required computing resources to process SPARQL queries and to integrate/host the SPARQL query engine, the SCENT queries can be easily transformed to SPARQL queries without affecting rest of the system.

Listing 5.3: Example of a SCENT query translated into a SPARQL query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX md: <http://www.ibmt.fhg.de/onto/2008/04/md#>
SELECT ?d ?g
WHERE { ?d rdf:type md:MedicalDevice .
        ?d md:hasGroup ?g .
        ?g md:deviceType md:UrineAnalysis .
}
```

5.2 SCENTRA - The SCENT Resolution Algorithm

To resolve a SCENT query, as shown in Listing 5.2, we have developed an algorithm, named SCENTRA (SCENT Resolution Algorithm), which is based on the formal definitions presented in the previous section, particularly Definitions 5.6 and 5.7. It works as a *variables' unification* and *patterns matching* algorithm, so that it could be used by both *query processor* and the *inference engine*, to find the matching triples from KB against the provided SCENT patterns and generate inferences, respectively. It is structured in the following steps.

1. Extract all the variables to a set V from the SCENT patterns/conditions graph P.

2. Select the triples from the knowledge base graph G that *match* every SCENT pattern/condition, and annotate the combination of valid values of the triple (* for any value) for every variable.

3. Substitute the asterisk (*) with the available values of the used variable, which exist in combinations in other SCENT patterns/conditions.

4. Search for the combinations of values that exist in all the SCENT patterns/conditions. If no combination is found, the query is irresolvable.

5.2.1 Example

In this section, we describe in detail how a SCENT query will be resolved by following the afore-mentioned steps. For the sake of simplicity, we use `rdf` and `md` for `http://www.w3.org/1999/02/22-rdf-syntax-ns` and `http://www.ibmt.fhg.de/onto-/2008/04/md` respectively. Let's assume that a set P is a *patterns* graph containing the SCENT patterns given in Listing 5.2, and a set G is a *triples* graph containing the following knowledge base triples of a medical device (e.g. urine analyzer).

```
<urn:uuid:urisys> <rdf#type> <md#MedicalDevice> .
<urn:uuid:coaguchek> <rdf#type> <md#MedicalDevice> .
<urn:uuid:weightscale> <rdf#type> <md#Device> .
<urn:uuid:urisys> <md#hasGroup> <md#UrineAnalyzer> .
<urn:uuid:coaguchek> <md#hasGroup> <md#BloodMeter> .
<urn:uuid:weightscale> <md#hasGroup> <md#WeightMeter> .
<md:UrineAnalyzer> <md#deviceType> <md#UrineAnalysis> .
<md:BloodMeter> <md#deviceType> <md#BloodAnalysis> .
<md:WeightMeter> <md#deviceType> <md#WeightAnalysis> .
```

1. Extract all the variables from graph P to a set V.

?d	?g

2. Select the triples from G, that *match* with every element of P, and annotate the combination of valid values of the triple (* for any value) for every variable.

- Results of pattern/condition 1:

```
<urn:uuid:urisys> <rdf#type> <md#MedicalDevice> .
<urn:uuid:coaguchek> <rdf#type> <md#MedicalDevice> .
```

?d	?g
`<urn:uuid:urisys>`	*
`<urn:uuid:coaguchek>`	*

- Results of pattern/condition 2:

```
<urn:uuid:urisys> <md#hasGroup> <md#UrineAnalyzer> .
<urn:uuid:coaguchek> <md#hasGroup> <md#BloodMeter> .
<urn:uuid:weightscale> <md#hasGroup> <md#WeightMeter> .
```

?d	?g
`<urn:uuid:urisys>`	`<md#UrineAnalyzer>`
`<urn:uuid:coaguchek>`	`<md#BloodMeter>`
`<urn:uuid:weightscale>`	`<md#WeightMeter>`

- Results of pattern/condition 3:

```
<md:UrineAnalyzer> <md#deviceType> <md#UrineAnalysis> .
```

?d	?g
*	<md:UrineAnalyzer>

3. Substitute the asterisk (*) with the available values of the used variable, which exist in combinations in other SCENT patterns/conditions.

Pattern/Condition 1:

?d	?g
<urn:uuid:urisys>	<md#UrineAnalyzer>
<urn:uuid:urisys>	<md#BloodMeter>
<urn:uuid:urisys>	<md#WeightMeter>
<urn:uuid:coaguchek>	<md#UrineAnalyzer>
<urn:uuid:coaguchek>	<md#BloodMeter>
<urn:uuid:coaguchek>	<md#WeightMeter>

Pattern/Condition 2:

?d	?g
<urn:uuid:urisys>	<md#UrineAnalyzer>
<urn:uuid:coaguchek>	<md#BloodMeter>
<urn:uuid:weightscale>	<md#WeightMeter>

Pattern/Condition 3:

?d	?g
<urn:uuid:urisys>	<md:UrineAnalyzer>
<urn:uuid:coaguchek>	<md:UrineAnalyzer>
<urn:uuid:weightscale>	<md:UrineAnalyzer>

4. Search for the combinations of values that exist in all the SCENT patterns/conditions. If no combination is found, the query is irresolvable.

?d	?g
<urn:uuid:urisys>	<md:UrineAnalyzer>

As we can see above that there is only one combination that appears in every SCENT pattern/condition, hence it is the solution to the query of *Search all the Urine Analyzers*. The values of ?d are the network addresses of the Urine Analyzer(s). We have studied other alternative algorithms, i.e. Rete [95] and its improved forms [96] which normally require more computing time and memory to create the temporal structures for efficient matching, while our algorithm is easy and straightforward to be implemented for the resource-constrained medical or mobile devices and provides the required compatibility and functionality for their semantic discovery and matching.

Complexity Analysis

The overall complexity of SCENTRA algorithm, or more precisely the query *Evaluation* (without reasoning process), is $O(n^2)$, where $n = |G|$. Please note that $|P|$ is constant relative to $|G|$, hence the complexity of processing SCENT conditions graph P is ignored.

5.3 μOR - A Micro OWL Querying and Reasoning System

This section describes details about μOR, a small querying and reasoning system which works on OWL description logics based rules, which are either created *implicitly* from the given set of domain/application ontologies or *explicitly* defined in a text-based rules file using SCENT language, as explained in Section 5.1. μOR uses SCENTRA algorithm, as explained in Section 5.2, for two purposes; *first* to process the SCENT queries and match the knowledge base triples graph with the SCENT pattern graph, *secondly*, to generate inference graph on the provided knowledge base triples graph.

5.3.1 Semantics of μOR Expressiveness

OWL is the W3C recommendation for creating and sharing ontologies on the Web and its theoretical background is based on the Description Logic (DL) knowledge representation formalism, a subset of predicate logic. As explained in Section 2.3, the OWL recommendation actually consists of three languages of increasing expressive power, namely OWL Lite, OWL DL and OWL Full. OWL-Lite and OWL-DL are an abstract syntactic form of the description logic $\mathcal{SHIF}(\mathcal{D})$ and $\mathcal{SHOIN}(\mathcal{D})$, respectively, whereas OWL-Full corresponds to the description logic $\mathcal{SHOIQ}(\mathcal{D})*$.

Since μOR is targeted for the resource-constrained medical or mobile devices, the set of *expressivity* requirements to be fulfilled would be rather concise and easier to be implemented. Beside that, the supported degree of OWL-DL expressivity of μOR must be high enough to cater the semantically annotated/harmonized application(s) knowledge of our pervasive healthcare scenarios, as described in detail in Chapter 8, as well as most of the other ubiquitous computing scenarios in general. Thus, we provide support of a subset of OWL Lite[3] [98] axioms for μOR, as listed in Table 5.2, because it has even a lower formal complexity than OWL Lite and it attempts to capture many of the commonly used features of OWL Lite, as well as to provide more descriptive language than RDF(S) for Semantic Web applications. The implementation of OWL-Lite$^-$ axioms by μOR gives us the desired degree of expressiveness (near to $\mathcal{SHIF}$[4]) and the following advantages:

- They do not produce or lead towards *conflicts* in term of *decidability*, when individually used to represent the knowledge base triples.
- They avoid the complexities of *non-determinism.*

However, it should be noted that μOR is *not* intended to be used for *knowledge consistency* checking, as supported by other DL reasoning systems, i.e. Pellet [99], rather it is used only for *querying* and *reasoning* over consistent knowledge bases. We use the definitions of Section 5.1 and define some new formal definitions for μOR.

[3]Hereafter referred to as OWL-Lite$^-$

[4]The axioms which are not supported are given in Table 5.3

Table 5.2: OWL-Lite$^-$ axioms currently supported by μOR

Name	**DL Syntax**	**FOL Semantics**	**Abbr.**
Thing, Nothing	$\top, \bot$	$\Delta^{\mathcal{I}}, \emptyset$	
Concept (Class)	A	$A^{\mathcal{I}}$	
Role (Property)	R	$R^{\mathcal{I}}$	S
Concept Inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	
Concept Equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$	
Trans. Role Closure	$R^+ \sqsubseteq R$	$(R^{\mathcal{I}})^+$	
Role Inclusion	$R \sqsubseteq P$	$R^{\mathcal{I}} \subseteq P^{\mathcal{I}}$	H
Inverse Role	$\exists R^-.C$	$\{x \in \Delta^{\mathcal{I}} : \exists y \in \Delta^{\mathcal{I}}.(y,x) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	I
Functional Role	$(\leq 1R)$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}}\}\rvert \leq 1\}$	F

Definition 5.9. Let $\mathcal{V}_{\mathcal{O}}$ be an OWL-Lite$^-$ vocabulary where $\mathcal{V}_{\mathcal{O}} = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_D, \mathcal{V}_{lit})$ is a 5-tuple where $\mathcal{V}_{cls}$ is the set of URIs denoting class names, $\mathcal{V}_{op}$ is the set of URIs denoting object properties, $\mathcal{V}_{dp}$ is the set of URIs denoting datatype properties, $\mathcal{V}_D$ is the set of URIs denoting datatype names, and $\mathcal{V}_{lit}$ is the set of well-formed RDF literals. In OWL-Lite$^-$, $\mathcal{V}_{uri}$ is the union of $\mathcal{V}_{cls}$, $\mathcal{V}_{op}$, $\mathcal{V}_{dp}$, and $\mathcal{V}_D$ and does not include any of builtin URIs from RDF, RDF-S, or OWL namespace.

Definition 5.10. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a tuple where $\Delta^{\mathcal{I}}$, the domain of discourse, is a union of two disjoint sets $\Delta^{\mathcal{I}}_O$ (the object domain) $\Delta^{\mathcal{I}}_D$ (the data domain); and $\mathcal{I}$ is the interpretation function that gives meaning to the entities defined in the ontology. $\mathcal{I}$ maps each OWL class $C \in \mathcal{V}_{cls}$ to a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_O$, each object property $p \in \mathcal{V}_{op}$ to a binary relation $p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_O \times \Delta^{\mathcal{I}}_O$, each datatype property $t \in \mathcal{V}_{dp}$ to a binary relation $t^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_O \times \Delta^{\mathcal{I}}_D$, each datatype $D \in \mathcal{V}_D$ to a subset $D^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_D$, and each literal $l \in \mathcal{V}_{lit}$ to an element $l^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_D$.

Definition 5.11. Let $\mathcal{O}$ be an OWL-Lite$^-$ ontology, $\mathcal{V}_{\mathcal{O}} = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_D, \mathcal{V}_{lit})$ the vocabulary for $\mathcal{O}$, and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation for $\mathcal{O}$, we say that a query atom is compatible with $\mathcal{V}_{\mathcal{O}}$ if all the URIs used in query atoms are typed correctly, e.g. for $\mathsf{Type}(l, C)$ we have $l \in \mathcal{V}_{lit}$ and $C \in \mathcal{V}_{cls}$ and so on.

Definition 5.12. We define an evaluation $\sigma : \mathcal{V}_{var} \cup \mathcal{V}_{lit} \rightarrow \Delta^{\mathcal{I}}$ to be a mapping from the variable names and literals used in the query to the elements of interpretation domain $\Delta^{\mathcal{I}}$ with the requirement that $\sigma(v) = v^{\mathcal{I}}$ if $v \in \mathcal{V}_{var}$ or $v \in \mathcal{V}_{lit}$. The interpretation $\mathcal{I}$ satisfies a query atom q w.r.t. σ (denoted as $\mathcal{I} \models_\sigma q$) if q is compatible with $\mathcal{V}_{\mathcal{O}}$ and the corresponding condition listed in Table 5.2 is met.

Definition 5.13. The interpretation I satisfies a query $Q = q_1 \wedge ... \wedge q_n$ w.r.t. an evaluation σ (written $\mathcal{I} \models_\sigma q$) iff $\mathcal{I} \models_\sigma q_i$ for every $i = 1, ..., n$. Note that we are only interested in the existence of an evaluation and we simply say that $\mathcal{I}$ satisfies a query Q (written $\mathcal{I} \models Q$) if there exists an evaluation σ such that $\mathcal{I} \models_\sigma Q$. Finally we say that Q is a logical consequence of the ontology $\mathcal{O}$ (written $\mathcal{O} \models Q$) if the query is satisfied by every model of $\mathcal{O}$, i.e. $\mathcal{I} \models \mathcal{O}$ implies $\mathcal{I} \models Q$.

Definition 5.14. The solution to a SCENT query $Q = q_1 \wedge ... \wedge q_n$ w.r.t. an OWL-Lite$^-$ ontology $\mathcal{O}$ is a *variables mapping* $\mu : \mathcal{V}_{var} \rightarrow \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$ such that when all the variables in Q are substituted with the corresponding value from μ, we get a query $\mu(Q)$ compatible with $\mathcal{V}_{\mathcal{O}}$ and $\mathcal{O} \models \mu(Q)$. The solution set $S(Q)$ for a query Q is the set of all such solutions.

5.3.2 Description of the OWL-Lite^{-} Axioms

This section gives a brief description about the OWL-Lite^{-} axioms/constructs which are currently supported by μOR.

OWL Lite^{-} RDF Schema Features

OWL can be viewed as an extension of a restricted view of the RDF language, which implies that every OWL document is an RDF document, but not all RDF documents are OWL documents. All terms are in the OWL namespace unless explicitly stated otherwise. Thus, the term Class is more precisely stated as owl : Class and rdfs : subPropertyOf shows that subProperty is from the rdfs namespace. Also, the term *individual* will refer to the objects that belong to classes as well as to objects that are datatypes.

- ***Class:*** A class defines a group of individuals that belong together because they share some properties. There is a built-in most general class named Thing that is the class of all individuals and the superclass of all OWL classes. We may choose to make a new subclass of the class Thing named Device, and we may also create a new class named MedicalDevice that is a *subclass* of Device. From this, μOR can deduce that any instance of MedicalDevice is also an instance of Device.

- ***rdfs:subClassOf:*** The class hierarchies can be created by making one or more statements that a class is a subclass of another class. For example, UrineAnalyzer could be stated as a subclass of MedicalDevice. From this, μOR can deduce that if an individual belong to UrineAnalyzer, then it is also a MedicalDevice.

- ***rdfs:Property:*** The properties can be used to state relationships between individuals or from individuals to data values. The examples of properties are: *hasGroup*, *hasVendor*, *hasLocation*, *hasConnection* etc.

- ***rdfs:subPropertyOf:*** The property hierarchies can be created by making one or more statements that a property is a subproperty of one or more other properties. For example, *hasBluetoothConnection* may be stated as a subproperty of *hasConnection*. From this, μOR can deduce that if an individual is related to another by the *hasBluetoothConnection* property, then it is also related to the other by the *hasConnection* property.

- ***rdfs:domain:*** A domain of a property limits the individuals to which the property can be applied. If a property relates an individual to another individual, and the property has a class as one of its domains, then the individual must belong to the class. For example, the property *hasMeasurement* may be stated to have the domain of MedicalDevice. So, if UrineAnalyzer *hasMeasurement* X, then from this μOR can deduce that the UrineAnalyzer is a MedicalDevice.

- ***rdfs:range:*** The range of a property limits the individuals that the property may have as its value. If a property relates an individual to another individual, and the property has a class as its range, then the other individual must belong to the range class. For example, the property *hasGroup* may be stated to have the range of DeviceGroups. From this, μOR can deduce that if *RocheUriSys* is related to UrineAnalyzer by the *hasGroup* property, then *RocheUriSys* is a MedicalDevice.

OWL Lite$^-$ Equality and InEquality

The following features related to equality or inequality are included:

- ***sameAs:*** Two individuals may be stated to be the same. These constructs may be used to create a number of different names that refer to the same individual. For example, the individual *RocheUriSys* may be stated to be the same individual as *UriSys1100*.

- ***differentFrom:*** An individual may be stated to be different from other individuals. For example, the individual *Soehnle* may be stated to be different from the individuals *UriSys1100* and *CoaguChekS*. Thus, if the individuals *Soehnle* and *UriSys1100* are both values for a property that is stated to be *functional* (the property that has at most one value), then there is a contradiction.

OWL Lite$^-$ Property Characteristics

There are special identifiers in OWL Lite$^-$ that are used to provide information concerning the properties and their values.

- ***ObjectProperty:*** It shows the relations (*binary*) between instances of two classes (Individuals-to-Individuals).

- ***DatatypeProperty:*** It shows the relations (*binary*) between instances of classes, RDF literals and XML Schema datatypes (Individuals-to-Datatypes).

- ***inverseOf:*** One property may be stated to be the inverse of another property. If the property P1 is stated to be the inverse of the property P2, then if X is related to Y by the P2 property, then Y is related to X by the P1 property. For example, if *hasVendor* is the inverse of *isVendorOf* and UriSys1100 *hasVendor* Roche, then μOR can deduce that Roche *isVendorOf* UriSys1100.

- ***TransitiveProperty:*** The properties may be stated to be transitive. If a property is transitive, then if the pair (x,y) is an instance of the transitive property P, and the pair (y,z) is an instance of P, then the pair (x,z) is also an instance of P. For example, if isLocatedIn is stated to be transitive, and if the operation theatre *OT123* is located on *Floor123*, and *Floor123* is located in *Building123*, then μOR can deduce that *OT123* is located in *Building123*.

- ***SymmetricProperty:*** The properties may be stated to be symmetric. If a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then the pair (y,x) is also an instance of P.

- ***FunctionalProperty:*** The properties may be stated to have a unique value. If a property is a FunctionalProperty, then it has no more than one value for each individual (or it may have no values for an individual). For example, *hasDeviceModel* may be stated to be a FunctionalProperty. From this, μOR can deduce that no individual can have more than one *device models*.

- ***InverseFunctionalProperty:*** The properties may be stated to be inverse functional. If a property is inverse functional then the inverse of the property is functional, which implies that the inverse of the property has at most one value for each individual. This characteristic is also referred as an *unambiguous property.*

OWL Lite$^-$ Datatypes

OWL Lite$^-$ uses most of the built-in XML Schema datatypes, where references to these datatypes are described by the URIs defined in `http://www.w3.org/2001/XMLSchema`.

5.4 Architectural Details of μOR

Fig. 5.1 shows the overall architecture of a small but powerful querying and reasoning system that is developed for resource-constrained AmI medical or mobile devices. It consists of a Query Processor, Inference Engine and SCENTRA algorithm.

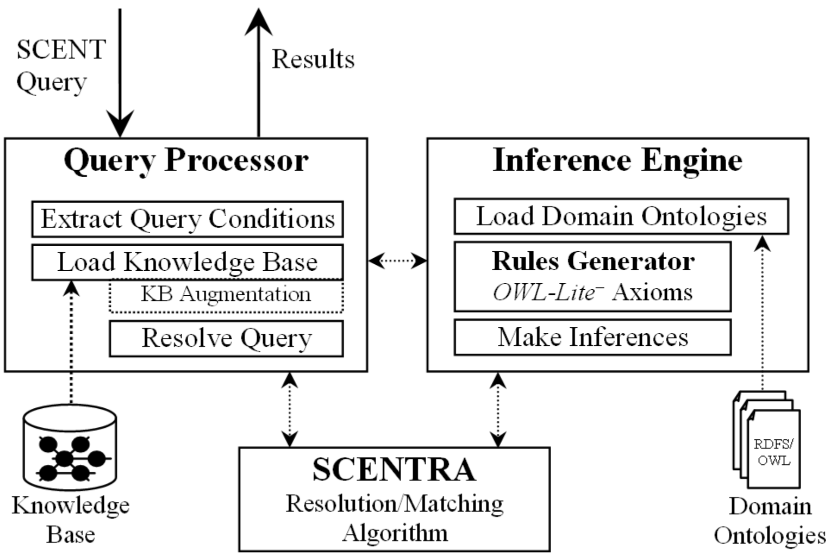


Figure 5.1: Architecture of Micro Querying and Reasoning System - μOR

5.4.1 The Query Processor

The *Query Processor* (QP) is responsible for processing the SCENT queries, as shown in Listing 5.2. In a pervasive computing environment, such query (embedded in the device discovery *request* message) is broad-casted from one device to all the other devices in the network, where the query processor running on each device processes this query; and if matches are found, it returns back the results (embedded in the device discovery *response* message) to the requesting device.

First of all, the QP *extracts* all the conditions/patterns from the SCENT query, and adds them into a set P. Secondly, it loads the available knowledge base triples stored on the device in a set G and requests the *inference engine* to make inferences on it, based on the OWL-Lite$^-$ axioms/constructs. When the QP gets back the inference results as set I, it augments the set G with these inferences using the set *union* operation, and then uses the SCENTRA algorithm to find the final results. When it gets back the final results, it formulates the results *only* for the requested variable, e.g. in case of SCENT query presented in Listing 5.2, only the result for variable ?d is returned.

Algorithm 1: *MakeInferences*: Find a set N of new facts (*inferences*) by creating a set $\mathbb{R}$ of *rules* from a Set $\mathbb{O}$ of *ontologies* and applying these rules on a set G of *knowledge base* triples.

```
   Input: A set $G$ of knowledge base triples
   Output: A set $N$ of (inferences)
1  begin
2      $C_{pre} \leftarrow \emptyset$   /* A set of preconditions */
3      $C_{post} \leftarrow \emptyset$   /* A set of postconditions */
4      $\mathbb{R} \leftarrow \emptyset$  /* A set of rules as triples*/
5      $M \leftarrow \emptyset$  /* A temporary set of matched triples */
6      $N \leftarrow \emptyset$  /* A set of new facts (inferences) */
7      $\mathbb{O} \longleftarrow$ Load the triples of ontologies
8      $\mathbb{R} \longleftarrow$ CreateRules($\mathbb{O}$)  /* Algorithm 2 is used */
9      /* Here $\mathbb{R} = \{r_1, r_2, ..., r_x\}$ */
10     for $r \in \mathbb{R}$ do
11         $C_{pre} \longleftarrow$ preconditions of $r_x$
12         $C_{post} \longleftarrow$ postconditions of $r_x$
13         /* Here $C_{post} = \{c_1, c_2, ..., c_y\}$ */
14         $M \longleftarrow$ MatchTriples($C_{pre}, G$) /* using Definition 5.7 */
15         /* Here $M = \{m_1, m_2, ..., m_z\}$ */
16         for $m \in M$ do
17             for $c \in C_{post}$ do
18                 /* Substitute the variable(s) in $c_y$ with $m_z$ using
                   Definition 5.6 to create new fact and store in set $N$ */
19                 $N \longleftarrow_{+}$ SubstituteVariables($c_y$, $m_z$)
20 end
```

5.4.2 The Inference Engine

The Inference Engine (IE) is responsible for generating new facts (*inferences*) from a provided set G of knowledge base triples graph and a set $\mathbb{O}$ of triples created from domain/application ontologies and/or user-defined rules (see Appendix C). The IE uses the formal definitions, as explained under Section 5.3.1, to produce these inferences. First of all, it loads all the locally stored domain/application ontologies as RDF triples, and if required, generates the *implicit rules* from these triples based on the OWL-Lite^{-} axioms, as mentioned in Table 5.2. Additionally or alternatively, *explicit rules* can also be defined in a separate rules file, as shown in Appendix C. Every implicit or explicit rule contains two sets, one for *preconditions* and one for *postconditions.* The preconditions set C_{pre} of each rule is matched with the knowledge base triples graph G using SCENTRA algorithm, and if the matches are found, it creates new facts as triples by *substituting* the variables of the *postconditions* set C_{post} with these matches. The resultant set $\mathbb{N}$ of new facts (*inferences*) is then returned back to the query processor. This set of new facts is

Algorithm 2: *CreateRules*: Creates a Set $\mathbb{R}$ of *rules* by *matching* a Set $\mathbb{O}$ of *domain ontologies* with *OWL-Lite*$^-$ axioms, Part 1.

```
Input: A Set $\mathbb{O}$ of triples of ontologies where $\mathbb{O} = \{t_1, t_2, ..., t_u\}$
Output: A Set $\mathbb{R}$ of rules
begin
    $\mathbb{R} \leftarrow \emptyset$ where $\mathbb{R} = \{r_1, r_2, ..., r_x\}$ /* A Set of Rules */
    $\mathbb{C} \leftarrow \emptyset$ where $\mathbb{C} = \{C_{pre}, C_{post}\}$
    /* $C_{pre}, C_{post}$ are sets of triples for (pre/post)conditions*/
    /* $t_s, t_p, t_o$ are subject, predicate, object of a triple $t$, resp. */
    for $t \in \mathbb{O}$ do
        if $t_p$ = rdfs:subClassOf or $t_p$ = rdfs:subPropertyOf then
            $C_{pre} \longleftarrow_+ <?s$ rdf:type $t_s>$
            $C_{post} \longleftarrow_+ <?s$ rdf:type $t_o>$
            $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        else if $t_p$ = rdfs:domain then
            $C_{pre} \longleftarrow_+ <?s\ t_p\ ?o>$
            $C_{post} \longleftarrow_+ <?s$ rdf:type $t_o>$
            $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        else if $t_p$ = rdfs:range then
            $C_{pre} \longleftarrow_+ <?s\ t_p\ ?o>$
            $C_{post} \longleftarrow_+ <?o$ rdf:type $t_o>$
            $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        else if $t_p$ = owl:inverseOf then
            $C_{pre} \longleftarrow_+ <?s\ t_s\ ?o>$
            $C_{post} \longleftarrow_+ <?o\ t_o\ ?s>$
            $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
            $C_{pre} \longleftarrow_+ <?s\ t_o\ ?o>$
            $C_{post} \longleftarrow_+ <?o\ t_s\ ?s>$
            $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        else if $t_o$ = owl:TransitiveProperty then
            $C_{pre} \longleftarrow_+ <?s\ t_s\ ?o_1>$
            $C_{pre} \longleftarrow_+ <?o_1\ t_s\ ?o_2>$
            $C_{post} \longleftarrow_+ <?s\ t_s\ ?o_2>$
            $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        else if $t_o$ = owl:SymmetricProperty then
            $C_{pre} \longleftarrow_+ <?s\ t_s\ ?o>$
            $C_{post} \longleftarrow_+ <?o\ t_s\ ?s>$
            $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        else
            /* See the remaining part in Algorithm 3 */
end
```

stored locally on each device in order to avoid repetetion of the inference process, unless the knowledge base is updated, or domain ontology(ies) are modified/changed or more explicit rules are defined.

Algorithm 3: *CreateRules*: Creates a Set $\mathbb{R}$ of *rules* by *matching* a Set $\mathbb{O}$ of *domain ontologies* with *OWL-Lite*$^-$ axioms, Part 2.

```
begin
    if $t_p = owl{:}sameAs$ then
        $C_{pre} \longleftarrow_+ < t_s\ ?p\ ?o >$
        $C_{post} \longleftarrow_+ < t_o\ ?p\ ?o >$
        $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        $C_{pre} \longleftarrow_+ < t_o\ ?p\ ?o >$
        $C_{post} \longleftarrow_+ < t_s\ ?p\ ?o >$
        $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        $C_{pre} \longleftarrow_+ <?s\ ?p\ t_s >$
        $C_{post} \longleftarrow_+ <?s\ ?p\ t_o >$
        $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
        $C_{pre} \longleftarrow_+ <?s\ ?p\ t_o >$
        $C_{post} \longleftarrow_+ <?s\ ?p\ t_s >$
        $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
    else if $t_o = owl{:}FunctionalProperty$ then
        $C_{pre} \longleftarrow_+ <?s\ t_p\ ?o_1 >$
        $C_{pre} \longleftarrow_+ <?s\ t_p\ ?o_2 >$
        $C_{post} \longleftarrow_+ <?o_1\ owl{:}sameAs\ ?o_2 >$
        $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
    else if $t_o = owl{:}InverseFunctionalProperty$ then
        $C_{pre} \longleftarrow_+ <?s_1\ t_p\ ?o >$
        $C_{pre} \longleftarrow_+ <?s_2\ t_p\ ?o >$
        $C_{post} \longleftarrow_+ <?s_1\ owl{:}sameAs\ ?s_2 >$
        $\mathbb{R} \longleftarrow_+$ CreateRule($\mathbb{C}$)
    ....
end
```

Complexity Analysis

If $\mathbb{R} = \{r_1, r_2, ..., r_x\}$ is a set of rules, and $n = |G|$, then x is constant w.r.t n, hence the complexity of overall inferencing process would be $O(n^2)$,which is the complexity of SCENTRA algorithm. Please note that $|M|$ is also constant w.r.t n, where M is a set of matched knowledge base triples.

Finally, the total complexity of both query processing and reasoning would be $O(n^2)$:

$$O(n^2) + O(n^2) \Longrightarrow O(n^2) \tag{5.1}$$

5.5 Comparative Analysis of μOR

This section provides the details about the comparative analysis of μOR, in terms of its *memory usage* and *performance* in comparison with a couple of other small reasoning systems, namely Pocket KRHyper [100] and Bossam [101]. Although there exist various implementations of OWL DL reasoners, e.g. Pellet [99], FaCT++ [102] and RacerPro [103], but their resource requirements (memory/processing) are quite high, restricting them to be used only on desktop systems or servers, and not on resource-constrained medical or mobile devices. Such reasoners mostly implement *tableaux* algorithms that are developed for the expressive DL knowledge representation with high complexity.

The reasons to choose Bossam and Pocket KRHyper for comparative analysis are as follows: Bossam is a DL reasoner and a *forward chaining* production rule engine, which works on RETE [95] algorithm, and takes comparatively less resources (750Kb of runtime-memory). Bossam requires *namespace prefixing* to express knowledge base triples and queries which is not supported by N-Triples, and hence not supported by our SCENT. Pocket KRHyper, to the best of our knowledge, is the only reasoner which is targeted for mobile devices, thus most relevant to our work. It is a First Order Logic theorem prover and model generator based on the *hyper tableau calculus* [104]. Its disadvantage is that it works on a set of *clauses* and does not support *direct* DL reasoning, rather it adds an additional layer for transforming all the DL expressions into first order *clausal logic* and the inference results back to DL expressions, which is clearly an overhead.

A Windows XP system with Pentium® 4, Intel® 2.40 GHz processor, and 1.0GB RAM was used to perform tests with a set of ten different SCENT queries. Ideally, we should have used any benchmark system, e.g. LUBM [105] in order to analyse and compare the performance of μOR with other reasoning systems, but since the test cases (queries, knowledge base, ontology) provided by it have compliance with OWL-Full or OWL-DL specification, it can not be tested on μOR. Therefore, we have designed our own test cases, which consist of a set of ten different queries, having compliance with OWL-Lite^{-} entailments (see Appendix B). Fig. 5.2 shows the overall runtime memory size (**24 Kb**) of complete μOR, which is far less than the run-time memory sizes of Bossam (**750 Kb**) and Pocket KRHyper (**245 Kb**). Fig. 5.3, Fig. 5.4 and Fig. 5.5 show the comparison of loading times of varying sized knowledge bases, loading times of different application/domain ontologies and the overall reasoning times respectively.

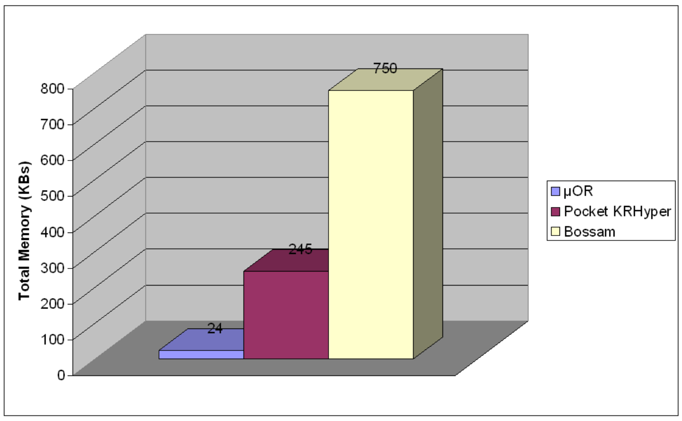


Figure 5.2: Comparison of Runtime Memory Usage

Based on the afore-mentioned comparative study of μOR, Bossam and Pocket KRHyper with respect to the run-time memory usage and RDF triples processing performance, it is pretty evident that μOR performed much better than the others in all aspects. The results show that μOR can be easily integrated *solely* on the (mobile/medical) devices and can help in realizing the vision of having *autonomous* (mobile/medical) devices which are enriched with semantic contents processing capabilities. Secondly, because μOR is

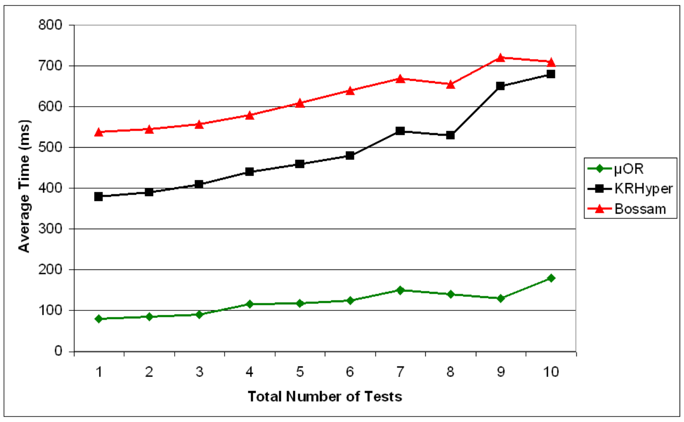


Figure 5.3: Comparison of Knowledge Base Loading Time

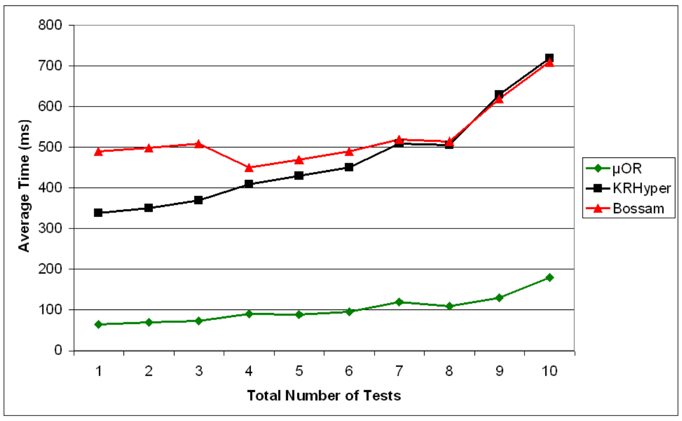


Figure 5.4: Comparison of Ontologies Loading/Conversion Time

developed in Java language, it inherently provides *platform independence* support, in terms of integrating and using it within the existing software architectures.

5.6 Scalability Issues of μOR

This section discusses the scalability of μOR in terms of *expressivity* and the *performance efficiency*. First of all, as described in Section 5.3.1, the expressivity of μOR is currently based on OWL-Lite^{-}, a subset of OWL-Lite axioms, which is sufficient to cater the semantically annotated/harmonized knowledge of our envisaged pervasive healthcare scenarios and application(s). However, if the axiomatic support, as presented in Table 5.2 is not sufficient to support the requirements of future context-aware applications, the expressivity can be extended easily by implementing the desired axioms, provided that the medical or mobile device has sufficient computing and memory powers to cater them all. Table 5.3 shows the OWL Lite axioms that are currently *not* supported by μOR, because these axioms are currently not needed for our envisioned healthcare scenarios.

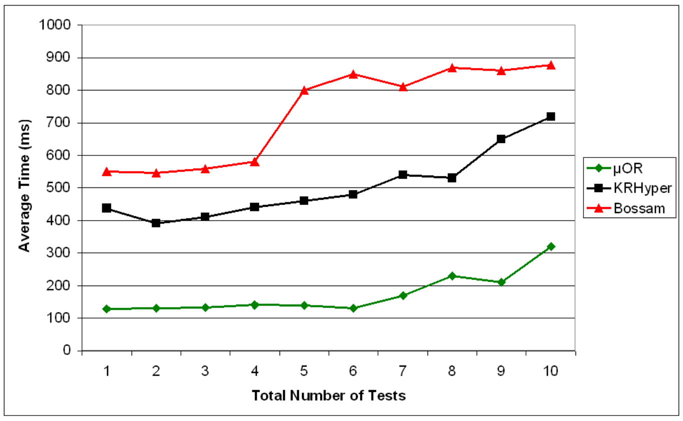


Figure 5.5: Comparison of Overall Reasoning Time

So, μOR does not currenlty support querying and reasoning on a knowledge base which is built using these axioms, hence the SCENT queries which use these axioms will not be answered and a NULL value is returned.

Table 5.3: OWL-Lite axioms currently **not** supported by μOR

Name	DL Syntax	FOL Semantics	Abbr.
Concept Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	
Concept Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	
Concept Complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	S
Universal Role-Value Restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} : \forall y \in \Delta^{\mathcal{I}}.(x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$	
Existential Role-Value Restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} : \exists y \in \Delta^{\mathcal{I}}.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	
Non-Qualified Role (Cardinality) Restr.	$(\leq nR)$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}}\}\rvert \leq n\}$	
	$(\geq nR)$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}}\}\rvert \geq n\}$	N
	$(= nR)$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}}\}\rvert = n\}$	
Qualified Role (Cardinality) Restr.	$(\leq nR.C)$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\rvert \leq n\}$	
	$(\geq nR.C)$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\rvert \geq n\}$	Q
	$(= nR.C)$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\rvert = n\}$	
Nominals	$\{o_1, ..., o_n\}$	$\{o_1^{\mathcal{I}}, ..., o_n^{\mathcal{I}}\}$	O
	$\exists R.\{o\}$	$\{x \in \Delta^{\mathcal{I}} : \exists y \in \Delta^{\mathcal{I}}.(x,y) \in R^{\mathcal{I}} \wedge y^{\mathcal{I}} \in \{o\}\}$	
Concept Membership	$a : C$	$a \in C$	
Role Membership	$(a,b) : R$	$(a,b) \in R^{\mathcal{I}}$	

Secondly, although the performance efficiency of μOR has been tested with small-scale knowledge bases (100 $\sim$ 300 triples), as shown in the above Figures (5.3, 5.4, 5.5), the efficiency of μOR would not be much affected if the sizes of knowledge base and/or ontologies' triples are increased, because the SCENTRA algorithm is based on *set-theory*, which inherently avoids the duplication of KB triples and the time required for set operations (*Union, Intersection*) theoretically remains same.

Chapter 6

Semantic Medical Device Discovery Protocol

Discovery is one of the most important activities in distributed computing paradigm, since recent pervasive computing and ad-hoc networking have identified the device(s) and service(s) discovery as one of the major *design* components of an architecture [106][107]. Although various network protocols and architectures have been developed over the last years, as described in Chapter 2, mainly for the discovery of device(s) and network service(s), but still there is no consensus on using a unified discovery protocol. This chapter describes the design and implementation details of *Semantic Medical Device Discovery Protocol* (SMDDP), a lightweight HTTP based protocol that is designed and developed, both as an integral part of *Semantic Medical Device Space* (SMDS) middleware framework, as described in Chapter 4, and as a broader goal itself for the *semantic* discovery of resource-constrained medical or mobile devices and their services. Section 6.1 articulates the (*minimum*) requirements for the semantic discovery protocol which could fulfill the *objectives* of our envisioned pervasive healthcare scenarios, while Section 6.2 describes the overall workflow of SMDDP. Section 6.3 and Section 6.4 describe the *request* and *response* messages' format respectively, while Section 6.5.1 gives the performance evaluation and compares SMDDP with the existing network discovery protocols.

6.1 Requirements for Semantic Discovery Protocol

The required semantic discovery protocol must support both the device and service discoveries based on semantics, since the device and service characteristics are harmonized and processed using ontologies. In order to achieve our goals, the semantic discovery protocol must be designed with the following *minimum* requirements:

- To support context-awareness on medical or mobile devices regarding the spontaneous presence of other devices, a pure (un)structured Peer-to-Peer network would suffice, thus getting rid of a central server and providing a decentralized solution.
- An expressive language for semantic search queries must be supported.
- The *response* messages from the *matched* medical device(s) must include
  - the URI for the Web Service *methods* invocation.

  - the URIs for the Web Service *description files* (WSDL, SAWSDL).
  - the URI for the *digital certificate* of medical device.

- It must be based on TCP/IP stack, integrating HTTP and taking advantage of its security mechanism, i.e. HTTP Basic or Digest Authentication, or even HTTPS.

The resulting design is SMDDP, an HTTP/UDP based semantic discovery protocol for ambient intelligent medical devices which fulfills the afore-mentioned requirements. SMDDP provides mechanism to search the peer-to-peer network and identify the desired medical device(s) matching with particular *conditions*. The following queries, which can also be thought as combination of each other making the notion more complex, are the few examples which depict what sort of discovery capabilities are expected from the SMDDP protocol. The *italicized* terms refer to the *concepts* of domain/application ontologies, which are developed within the context of this thesis.

- Find all the *medical devices* in the network.
- Find all the *medical devices* in the *operation theater XYZ*.
- Find all the *medical devices* of type *urine analyzer* (or blood pressure, blood coagulation, blood glucose, ECG etc.) in the network.
- Find a *medical device* of type *urine analyzer* which has performed the *measurement* of a patient having ID *"3KD2008"*
- Find a device in the network which has *Internet connectivity* and offers a Web Service to send *measurement* results to *XYZ Information System*.

6.1.1 Advantages of using URIs

In the SMDDP response messages, as described earlier, sending *only* the URIs is more efficient than directly sending the whole volume of data, and offers the following advantages:

- Less traffic is generated as the packets are significantly smaller than conveying the whole volume of data.
- The decision about downloading the Web Service description files, the digital device certificate, or even invoking the Web Service methods is totally left up to the client, if it is needed to perform all these tasks.
- The URIs can be reused by the *client* medical devices without re-searching the network, if it is required to periodically poll the data about *server* medical devices.

The URI for the Web Service methods invocation can be used not only by the medical devices, but other (health) information systems on the network as well, in order to retrieve their measurement values/results. Normally, the medical devices and/or health information systems will be pre-installed with the Web Service *stubs/skeletons*, but the (health) information systems may also dynamically generate the Web Service *stubs/skeletons* by downloading the WSDL document from the given WSDL URI using various tools available for desktop systems, i.e. WSDL2Java[1].

[1]`http://ws.apache.org/axis/java/user-guide.html`

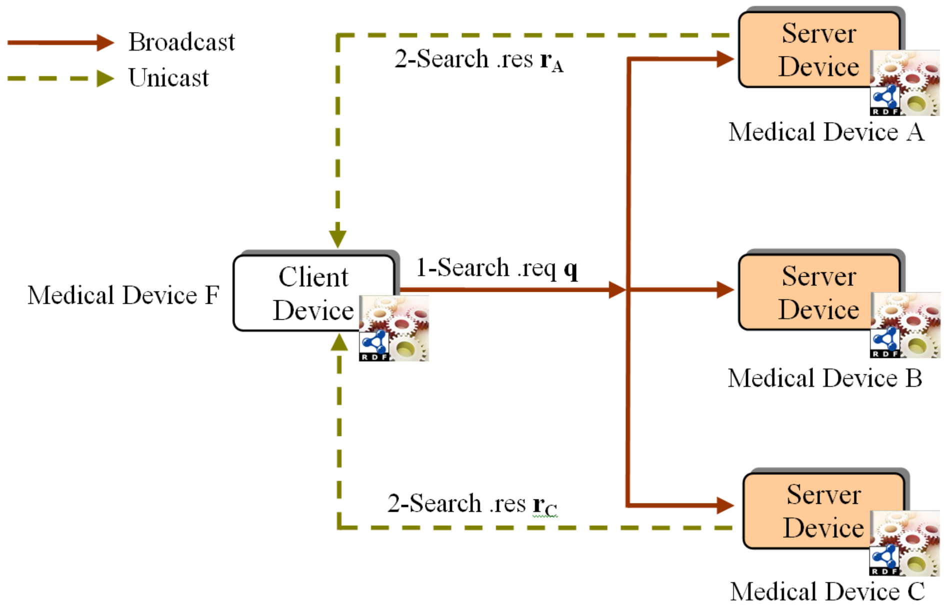


Figure 6.1: Overall Workflow of SMDDP

On the other hand, the medical devices cannot use such tools because of the unavailability of compilers for resource-constrained (mobile) devices. However, the URI for SAWSDL file can be used both by medical devices and (health) information system(s) in order to download and process it locally using the SAWSDL[2] API, and extract the name(s) of the desired Web Service method(s) to invoke. The last URI is for the digital device certificate, which the client medical device uses to download it from the server medical device. Before initiating the communication, every client/server medical device(s) *must* have the digital certificate of each other, which is used later for *encryption/decryption* and *signing/verification* functionalities.

6.2 Overall Workflow of SMDDP Protocol

The workflow of SMDDP is quite simple, consisting of two *agents* like threads, smddp client and smddp server, both co-exist on a medical device. Whenever a medical device needs to find a particular medical device in a pure (un)structured peer-to-peer network of medical devices,

- The *smddp client* on the client medical device reads the (*configurable*) query conditions defined as searching criteria for a particular healthcare application and embed them in a complete request message containing the HTTP Callback-URI. Finally, it broadcasts this request message over the network and handles the response(s) sent back from the *matched* medical device(s).

- On the other hand, the *smddp server* on the server medical device(s) receives the request messages from the client medical devices, processes them against the available knowledge base of the medical device using μOR, as explained in Chapter 5,

[2]`http://lsdis.cs.uga.edu/projects/meteor-s/SAWSDL/`

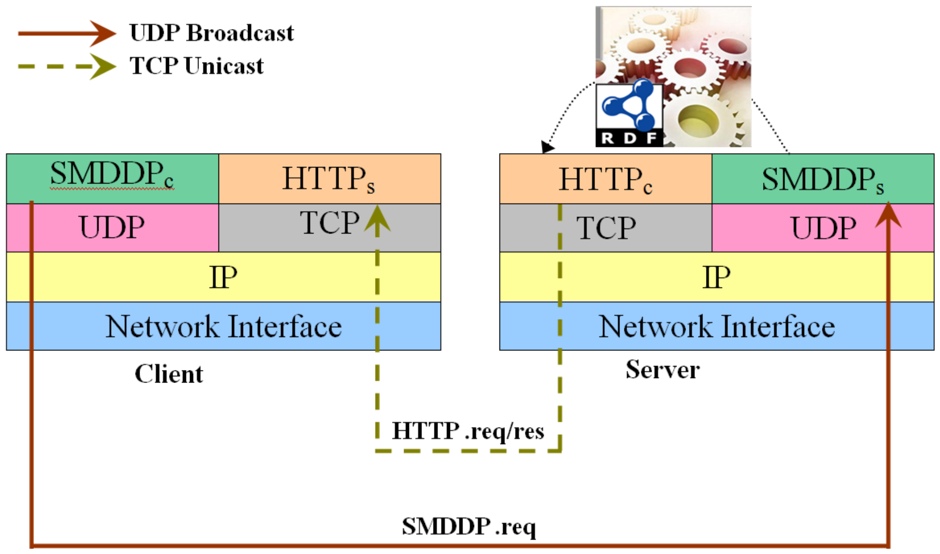


Figure 6.2: SMDDP over the TCP/IP Stack

and if matched, forms the results into a response message and sends it back to the respective client medical device(s) on the provided Callback-URI.

Fig. 6.1 shows the interaction of a peer-to-peer network of four medical devices, namely Medical Device A, Medical Device B, Medical Device C and Medical Device F, each of them running both smddp client and smddp server threads. In order to search for the desired medical device(s), the *smddp client* of Medical Device F *broadcasts* a query q to the network using UDP (*Universal Datagram Protocol*) as the *transport protocol*, as shown in Fig. 6.2. On the other hand, every medical device processes this query and two of them, Medical Device A and Medical Device C respond with the responses r_A and r_C respectively as the *matched* medical devices. Since the responses are always *unicast* messages, so they can be sent back using the traditional HTTP protocol on the provided Callback-URI in the request message.

6.3 SMDDP Request Message Format

This section describes the details about the SMDDP *request* messages. SMDDP request messages, containing a SCENT query are *broad-casted* over UDP to the (configurable) address 228.5.6.7 and UDP port 6790. The MIME type of SMDDP request message is application/de.fhg.ibmt.scent and it consists of *multi-line* syntax, being extensible through new headers to provide additional semantics information. Most Internet protocols use ASCII characters of 13 and 10 (CRLF) to separate lines, which constitute the major divisions in the message format. Listing 6.1 shows the Augmented Backus-Naur Form (ABNF) of SMDDP request message, which encapsulates some of the widely used productions in Internet protocols, i.e. SP, OCTET, DIGIT [108], and token, absoluteURI [109].

The request − message is composed of a start − line, zero or more headers and a mandatory body section (line 1). The main element of the start − line is the SEARCH command, followed by the main variable to be resolved, and a protocol version number (lines 2-5).

Listing 6.1: Augmented Backus-Naur Form of SMDDP Request Messages

```
1   request-message = start-line * (msg-header CRLF) CRLF msg-body
2   start-line = c-search SP version CRLF
3   c-search = "SEARCH" SP variable
4   variable = "?" token
5   version = "smddp/" number "." number
6   msg-header = SeqN / Content-Type / Content-Length / Callback-Uri
7   SeqN = "SeqN" ":" SP number
8   Content-Type = "Content-Type" ":" SP cont-type
9   Content-Length = "Content-Length" ":" SP number
10  Callback-Uri = "Callback-Uri" ":" SP i-Callback
11  i-Callback = absoluteURI
12  msg-body = 1 * OCTET
13  number = 1 * DIGIT
14  cont-type = type "/" subtype
15  type = token
16  subtype = token
```

A brief description of each of the four headers of SMDDP request message (line 6) is given below, which follows the same format as traditional headers in the standardized protocols, i.e. HTTP.

- SeqN : The request *sequence number* (line 7) for detecting duplicate request messages from the same client medical device, and matching the requests and responses for it.
- Content − Type : The *MIME type* (line 8) of the message body section. Currently, SMDDP supports only one MIME type, which is application/de.fhg.ibmt.scent. In future, if the medical devices get enough computing and memory resources, the MIME type for SPARQL (or other languages) could also be included.
- Content − Length : The total *length* (line 9) of the message body section in *bytes*.
- Callback − Uri : The *HTTP endpoint* (line 10) where the server medical device sends back the response to the client medical device.

Listing 6.2: An Example of Complete SMDDP Request Message

```
1  SEARCH ?d smddp/1.0
2  SeqN: 20
3  Content-Type: application/de.fhg.ibmt.scent
4  Content-Length: 400
5  Callback-Uri: http://192.168.2.11/smddpcallback
6
7  ?d <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://www.ibmt.fhg.de/onto/2008/04/md#MedicalDevice> .
8  ?d <http://www.ibmt.fhg.de/onto/2008/04/md#hasGroup> ?g .
9  ?g <http://www.ibmt.fhg.de/onto/2008/04/md#deviceType>
      <http://www.ibmt.fhg.de/onto/2008/04/md#UrineAnalysis> .
```

Listing 6.2 shows an example of SMDDP request message, encapsulating the SCENT query conditions (lines 7-9), as mentioned earlier in Listing 5.2. Due to the unreliable nature of UDP protocol, some UDP packets *could be* lost during the broadcasting phase, so the SMDDP request messages must be sent *three* times, as per the UPnP-SSDP specification [56] recommendation, in order to increase the probability of SMDDP request message to reach the possible destinations without generating much network traffic.

6.4 SMDDP Response Message Format

This section describes the details about the SMDDP *response* messages. When a server medical device receives the SMDDP request message, it extracts the SCENT query conditions from it and resolves all of its variables. After resolving the SCENT query, the smddp server constructs the reply message, which is an HTTP response back to the Callback − Uri provided by the client medical device in the request message, and convey the required information in a suitable *format*.

Listing 6.3: Augmented Backus-Naur Form of SMDDP Response Messages

```
1   response−message = start−line ∗ (msg−header CRLF) CRLF msg−body
2   start−line = "POST" SP Callback−Uri SP version CRLF
3   Callback−Uri = "/" subtype
4   version = "HTTP/" number "." number
5   msg−header = Host / SeqN / Content−Type / Content−Length
6   Host = "Host" ":" SP IP4−Address
7   SeqN = "SeqN" ":" SP number
8   Content−Type = "Content−Type" ":" SP cont−type
9   Content−Length = "Content−Length" ":" SP number
10  msg−body = 1 ∗ OCTET
11  number = 1 ∗ DIGIT
12  cont−type = type "/" subtype
13  type = token
14  subtype = token
```

Listing 6.3 shows the Augmented Backus-Naur Form (ABNF) of SMDDP response messages which mostly uses the productions that are already used in ABNF of SMDDP request messages. The response − message is composed of a start − line, zero or more headers and a mandatory body section (line 1). The main element of the start − line is the POST command, followed by the Callback − Uri where the response is going to be sent, and a protocol version number (lines 2-4). There are four types of headers (Lines 5,6) where, besides the already defined three headers, a new header Host is introduced, which describes the IP address of the client medical device using the production of IP4 − Address [110].

Listing 6.4: An Example of Complete SMDDP Response Message

```
POST /smddpcallback HTTP/1.0
Host: 192.168.2.11
SeqN: 20
Content−Type: application/de.fhg.ibmt.smiddel
Content−Length: 561

<?xml version="1.0" encoding="UTF−8"?>
<smiddel xmlns="http://www.ibmt.fhg.de/smds/2008/04/smiddel"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
  xsi:schemaLocation="http://www.ibmt.fhg.de/smds/2008/04/smiddel
  http://www.ibmt.fhg.de/smds/2008/04/smiddel.xsd">

  <medevice uri="urn:uuid:urisys">
    <wsurl>http://192.168.2.25:8086/csoap/UrisysService</wsurl>
    <wsdl>http://192.168.2.25:8086/csoap/wsdl/urisys.wsdl</wsdl>
    <sawsdl>http://192.168.2.25:8086/csoap/wsdl/urisys.sawsdl</sawsdl>
    <cert>http://192.168.2.25:8086/csoap/cert/urisys.cert</cert>
  </medevice>
</smiddel>
```

Listing 6.4 shows an example of the simple SMDDP response message that is sent back from a *urine analyzer* to the client medical device against the SMDDP request message shown in Listing 6.2. The most important part (lines 13-17) of this response message is the *address* of the responding medical device (line 13), the URI for the Web Service *methods invocation* (line 14), the URI for the *simple* Web Service description file (line 15), the URI for the *semantically annotated* Web Service description file (line 16), and the URI for downloading the *digital device certificate* file (line 17). All these URIs are used by the client medical device and/or (health) information system in the network as per the requirements of an application, e.g. if it is required to invoke just the Web Service methods of a *server* medical device, the *client* medical device will use only the URI of wsurl.

6.4.1 SMIDDEL: A Schema for SMDDP Response Messages

The SMDDP response messages are very simple and straightforward to be processed by the client medical devices, as they contain just the URIs of the matched medical devices. In order to annotate the SMDDP response message and markup its body section, a simple schema has been designed, called SMIDDEL (**S**emantic **M**ed**i**cal **D**evice **D**escription **E**ndpoints **L**anguage), which is used by the client medical devices to process the SMDDP response messages.

Listing 6.5: SMIDDEL XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!-- SMIDDEL XML Schema -->
<!-- Author: Safdar Ali -->
<!-- Version: 2008045.1 -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:smiddel="http://www.ibmt.fhg.de/smds/2008/04/smiddel"
targetNamespace="http://www.ibmt.fhg.de/smds/2008/04/smiddel"
elementFormDefault="qualified">

<xs:element name="wsurl" type="xs:anyURI" />
<xs:element name="wsdl" type="xs:anyURI" />
<xs:element name="sawsdl" type="xs:anyURI" />
<xs:element name="cert" type="xs:anyURI" />
<xs:element name="medevice">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="smiddel:wsurl" minOccurs="0" maxOccurs="1" />
   <xs:element ref="smiddel:wsdl" minOccurs="0" maxOccurs="1" />
   <xs:element ref="smiddel:sawsdl" minOccurs="0" maxOccurs="1" />
   <xs:element ref="smiddel:cert" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="uri" type="xs:anyURI" use="required" />
 </xs:complexType>
</xs:element>
<xs:element name="smiddel">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="smiddel:medevice" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Listing 6.5 shows the complete SMIDDEL XML schema. The most important part (lines 19-22) is the *sequence* and the lower/upper limits of the occurrence of all the URIs.

6.5 Overall Analysis of SMDPP

This section describes the details about the overall analysis of SMDDP protocol, in terms of *matchmaking* process on the medical devices and their response time, as well as a *comparative analysis* of SMDDP with other existing network discovery protocols.

6.5.1 Performance Evaluation

The advantages of semantic discovery are pretty evident, since it gives us more refined results by interpreting the information relationships, while on the other hand, the normal lexical *attribute-value* based discovery gives us a subset of those results or even none. We are quite satisfied with the performance of SMDDP, which is at core dependent on μOR that enriches the medical devices with the capabilities of semantic querying, reasoning and interpretation of the information and its relationships. In order to evaluate the performance of Java based implementation of SMDDP protocol, we tested 10 different SEARCH queries (see Appendix B) against the 5 instances of SMDDP servers running on a single PC (Pentium 4, 2.4 GHz, 1GB RAM).

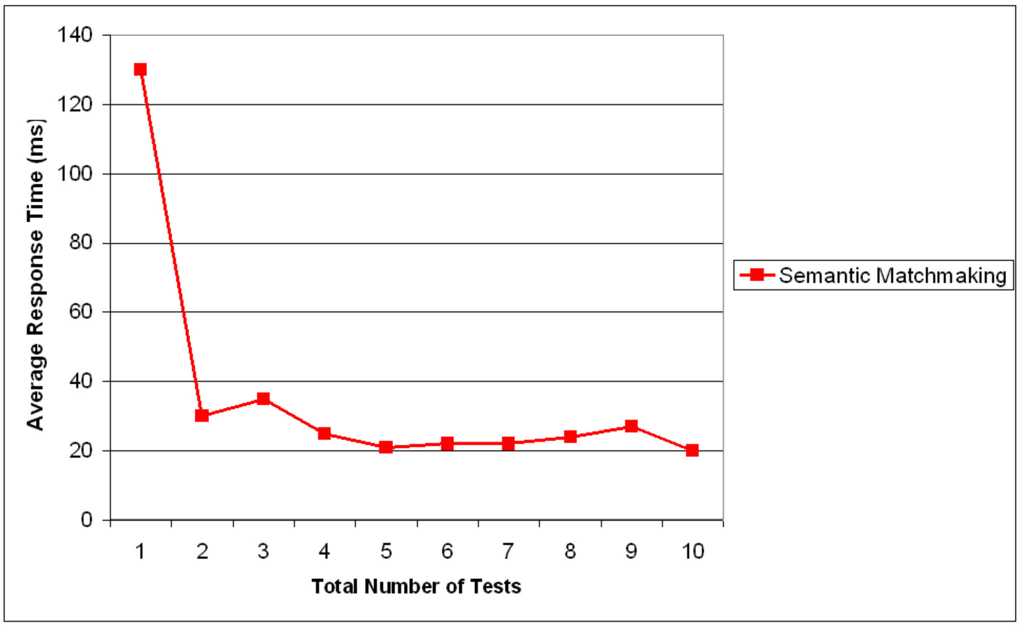
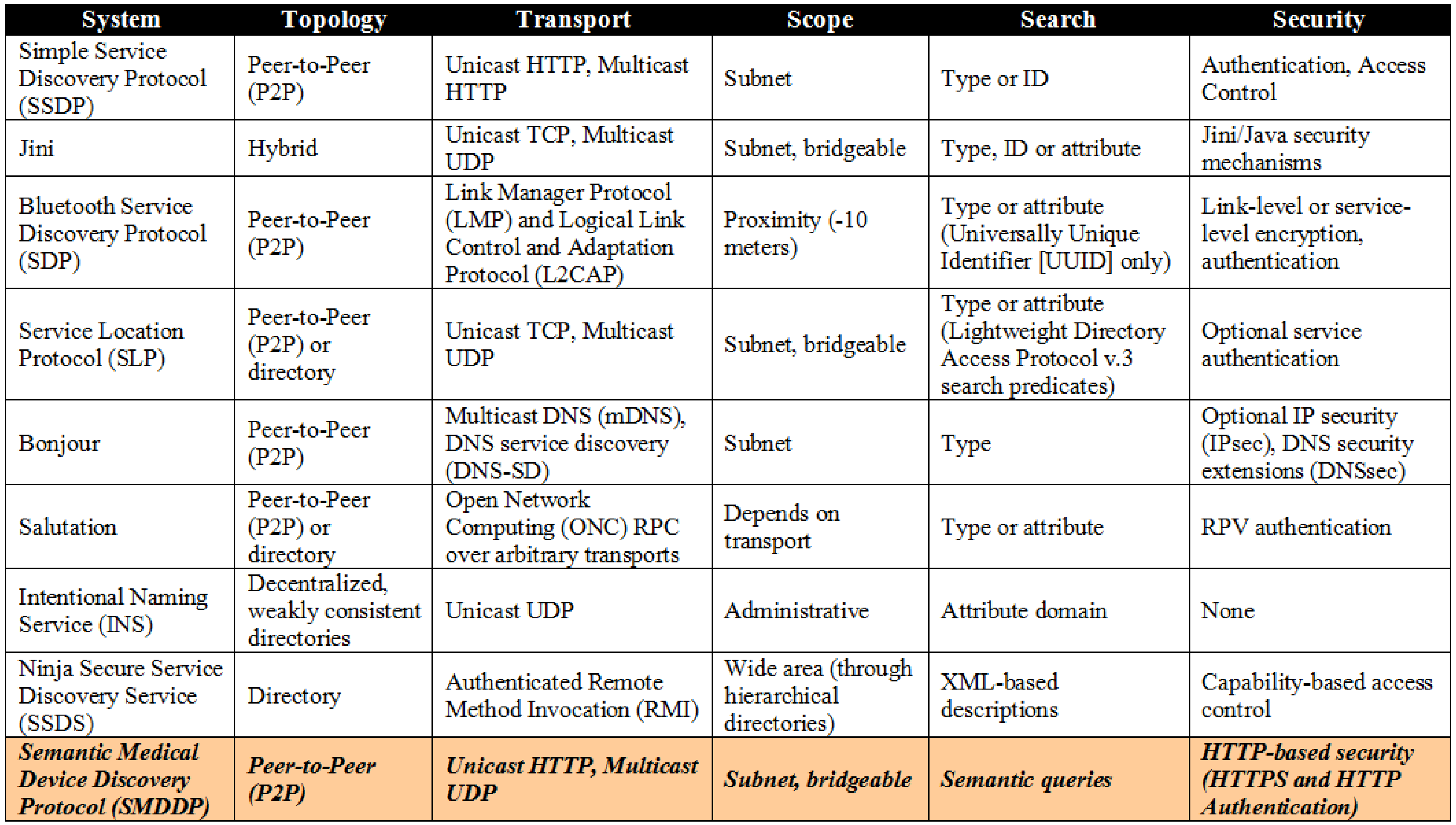


Figure 6.3: Semantic Matchmaking Performance of SMDDP Discovery

During the first test, the SMDDP servers parsed the local application/domain ontologies and created the *rules* as RDF triples from them, performed the description logic reasoning against the available knowledge base and created the *inferences* using μOR. This step will be performed *only* when the medical device is booted, or if there is a change in the knowledge base or ontologies. Finally, the whole knowledge base is searched for the matched triples against the searching criteria (*conditions*) of the received query.

Table 6.1: Comparison of SMDDP with State-of-the-art Discovery Protocols



System	Topology	Transport	Scope	Search	Security
Simple Service Discovery Protocol (SSDP)	Peer-to-Peer (P2P)	Unicast HTTP, Multicast HTTP	Subnet	Type or ID	Authentication, Access Control
Jini	Hybrid	Unicast TCP, Multicast UDP	Subnet, bridgeable	Type, ID or attribute	Jini/Java security mechanisms
Bluetooth Service Discovery Protocol (SDP)	Peer-to-Peer (P2P)	Link Manager Protocol (LMP) and Logical Link Control and Adaptation Protocol (L2CAP)	Proximity (-10 meters)	Type or attribute (Universally Unique Identifier [UUID] only)	Link-level or service-level encryption, authentication
Service Location Protocol (SLP)	Peer-to-Peer (P2P) or directory	Unicast TCP, Multicast UDP	Subnet, bridgeable	Type or attribute (Lightweight Directory Access Protocol v.3 search predicates)	Optional service authentication
Bonjour	Peer-to-Peer (P2P)	Multicast DNS (mDNS), DNS service discovery (DNS-SD)	Subnet	Type	Optional IP security (IPsec), DNS security extensions (DNSsec)
Salutation	Peer-to-Peer (P2P) or directory	Open Network Computing (ONC) RPC over arbitrary transports	Depends on transport	Type or attribute	RPV authentication
Intentional Naming Service (INS)	Decentralized, weakly consistent directories	Unicast UDP	Administrative	Attribute domain	None
Ninja Secure Service Discovery Service (SSDS)	Directory	Authenticated Remote Method Invocation (RMI)	Wide area (through hierarchical directories)	XML-based descriptions	Capability-based access control
Semantic Medical Device Discovery Protocol (SMDDP)	***Peer-to-Peer (P2P)***	***Unicast HTTP, Multicast UDP***	***Subnet, bridgeable***	***Semantic queries***	***HTTP-based security (HTTPS and HTTP Authentication)***

As shown in Fig. 6.3, the first execution took on average **130ms** on each of the five SMDDP servers, since Java takes longer time to initialize the internal data structures. However, the *response* of SMDDP to the subsequent executions for rest of the nine queries was much more faster, produced more stabilized measures and took on average **20-30ms**, since no rules generation from ontologies or any form of semantic reasoning was performed.

6.5.2 Comparative Analysis

SMDDP provides a simple and powerful way for the semantic discovery of ambient intelligent medical devices in pervasive (healthcare) environments, utilizing and exploiting the advantages of HTTP protocol. Although, several discovery mechanisms for pervasive computing have been proposed in the past, but none of them is widely adopted. Edwards has published a comparative study [111] about SSDP, Jini, Bluetooth SDP, SLP, Bonjour, Salutation, INS and Ninja SSDS, including also infrared and RFID mechanisms. The comparison was based on different parameters, i.e. topology, transport, scope, search and security.

Table 6.1 shows a reproduction of the Edwards' comparison table with the additional row of SMDDP, which exhibits some distinct factors by comparing it with other alternatives, such as its powerful semantic discovery capabilities and the use of widely-adopted and reliable HTTP-based security mechanisms. This comparison shows and promotes SMDDP to be the best candidate for the semantic discovery of ambient intelligent medical devices in pervasive computing scenarios within the healthcare domain, and also a valuable and promising alternative in other networking environments as well.

Chapter 7

Implementation

This chapter provides the details about the programming languages, libraries, environments and the tools used for the development of complete *Semantic Medical Devices Space* (SMDS) framework and its constituent components, including *Semantic Medical Device Discovery Protocol* (SMDDP), *Micro OWL Description Logic and Rules based Reasoner* (μOR) and security framework. Also, it provides the details about the hardware platforms that we have used to host our SMDS framework and to validate its performance results, particularly in connection with the scenarios of SmartHEALTH Project [21]. The UML class diagrams of complete SMDS framework are given in Appendix A.

7.1 Programming Languages

7.1.1 Java (J2SE 1.5)

The complete SMDS framework is developed using Sun's Java Standard Edition (SE) 1.5 programming language. Apparently, it seems strange why SMDS is *not* developed using Java Micro Edition (ME) (MIDP[1] or CLDC[2]) which is particularly suited for resource-constrained devices. However, there are various reasons, few of them are given below, to prefer Java SE over Java ME for the development of SMDS framework.

- **Java Native Interface (JNI):** MIDP does not support JNI and therefore it is not possible to extend Java APIs beyond those that come with the device. On the other hand, we *do* want to provide support through SMDS for utilizing the existing software functionality of medical devices, for which JNI is inevitably required.

- **Reflection/Serialization:** Java ME does not support reflection, object serialization or Remote Method Invocation (RMI), which are usually required in Web Services communication while transferring user-defined serialized objects.

- **Accessing Native Application Data:** MIDP 2.0 alone does not allow accessing native S60 Symbian OS or other mobiles' OSs application data, rather FileConnection and Personal Information Management APIs (JSR-75) have to be used to access native application data.

[1]Mobile Information Device Profile; `http://java.sun.com/products/midp/`
[2]Connected Limited Device Configuration; `http://java.sun.com/products/cldc/`

- **File Handling:** CLDC does not support file access, rather FileConnection API (JSR-75) has to be used to access the file system.
- **Class Un/Loading:** MIDP does not support mechanism for class loading or unloading of both MIDlet classes and the platform's API classes. All the classes loaded during the running of a MIDlet will occupy the heap for its lifetime.
- **Pre-verification:** MIDP applications have to be pre-verified before runtime. CLDC does not support a full Java SE type byte-code verifier, but instead a different kind of byte-code verifier that takes less memory in the device, but requires application class files to be pre-verified during application development, more specifically in the compile process.

7.1.2 Microsoft Visual C#.NET

Microsoft's Visual C#.NET language is used to develop the graphical (gateway) applications for Windows Xp and Windows Mobile 5.0/6.0 platforms, and clearly show the advantages of using Semantic Web Services technology to get the heterogeneous medical or mobile devices interoperable. These applications are developed to realize different pervasive healthcare scenarios of the SmartHEALTH project.

7.2 Hardware Platforms

This section describes *off-the-shelf* hardware platforms that we have used to host the SMDS framework for its evaluation, and to use them for different pervasive healthcare scenarios in connection with SmartHEALTH Project. These hardware platforms include Gumstix, Vodafone VPA 4, Viliv Net-Tablet PC and several desktop computers.

7.2.1 Gumstix

The Gumstix[3] is a full functional open embedded computer which is available in various configurations and can be individually extended through expansion boards. The mainboard has two connectors for expansion boards on both sides, which allows the use of up to two expansion boards at the same time leading to a *sandwich-like* hardware configuration. It offers various types of expansion boards allowing the user to align the hardware configuration to his/her needs. The available interface types are for example Ethernet, WiFi 802.11b/g, Bluetooth, GPS, USB, serial, GPIO, I2C, SPI, SD or Compact Flash card slots, Audio in/out and even LCD interfaces.

We have selected *Verdex Pro XL6P* mother-board (Marvell™ PXA270 with XScale™ 600MHz, 128MB RAM, 32MB Flash), *NetPro-WiFi-Vx* (10/100baseT Ethernet with 802.11b/g WiFi module) and *Console-Vx* (3 * RS-232 ports on miniDIN8 connectors) together to host the SMDS framework. Fig. 7.1 shows the assembled Gumstix of all these components, while Fig. 7.2 shows a Urine Analyzer device to which the Gumstix is attached to enhance its capabilities and turning it into an *ambient intelligent* Urine Analyzer. The Gumstix serves as a *gateway* between the connected Urine Analyzer and SMDS framework, and offers Semantic Web Service based interface through Ethernet or WiFi 802.11b/g to the clients, which could be other medical or mobile devices or even health information systems.

[3]Gumstix miniaturized computers; `http://www.gumstix.com/`

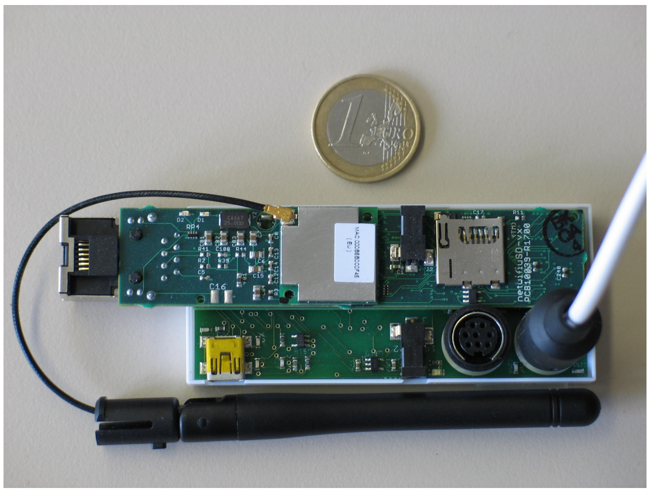


Figure 7.1: Gumstix with XL6P motherboard

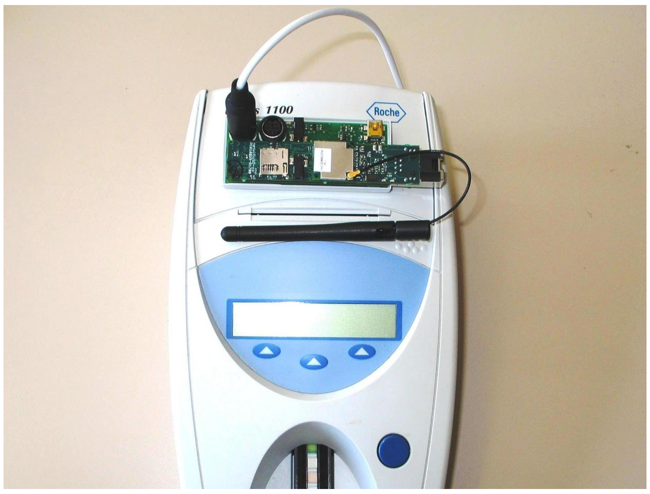
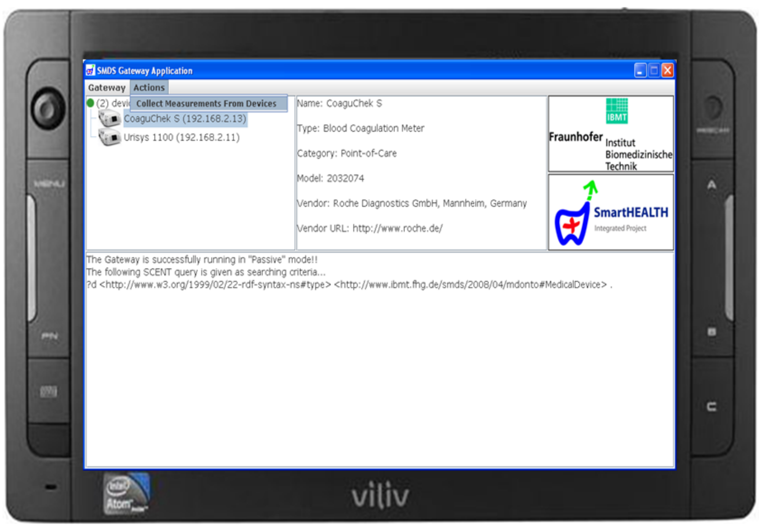


Figure 7.2: Urine Analyzer from Roche® Diagnostics with Gumstix

7.2.2 Viliv Promotion Pack S5 Net-Tablet PC

The Viliv Promotion Pack S5 EXP P 4.8-Inch Net-Tablet PC Viliv has crafted an amazing, ultra-portable, fully-functional Internet device, and rich in features with unabated battery life and power. It features a superfine 4.8-Inch WSVGA display, 16 GB SSD, and an Intel Atom™ Silverthorne 1.2GHz CPU. From a pocket PC standpoint, the S5 has a surreal (for its size) battery life of 7-hours of use, and it supports 720p HD video playback with the features of integrated WiFi, Bluetooth and GPS. Fig. 7.3 shows the Viliv Net-Tablet PC running the GUI based gateway application, which collects the measurements from *all* the medical devices (or *matched* with the provided criteria) in the environment and forwards them to the remote SmartHEALTH information system.


SMDS Gateway Application
Gateway
Actions
Collect Measurements From Devices
(2) devi
CoaguChek S (192.168.2.13)
Urisys 1100 (192.168.2.11)
Fraunhofer
IBMT
Institut Biomedizinische Technik
SmartHEALTH
Integrated Project
viliv

Name: CoaguChek S
Type: Blood Coagulation Meter
Category: Point-of-Care
Model: 2032074
Vendor: Roche Diagnostics GmbH, Mannheim, Germany
Vendor URL: http://www.roche.de/

```
The Gateway is successfully running in "Passive" mode!!
The following SCENT query is given as searching criteria...
?d <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.ibmt.fhg.de/smds/2008/04/mdonto#MedicalDevice> .
```

Figure 7.3: The Viliv Promotion Pack S5 EXP P 4.8-Inch Net-Tablet PC

7.2.3 Vodafone VPA-4

Vodafone VPA-4 (*Vodafone Personal Assistant*) is a 3G-compatible model, having a QWERTZ (German layout) keypad for convenient text entry, and its display is folded out behind the keypad when users are writing or browsing the Web. The VPA-4 is equipped with Windows Mobile Pocket PC Phone Edition 2003 OS, a 520MHz Intel processor and 128MB RAM with various other features, i.e. Bluetooth, infrared, WiFi, MiniUSB and SDIO/MMC card slot etc. Fig. 7.4 shows the VPA-4 device running the GUI based gateway application, which collects the measurements from *all* the medical devices (or *matched* with the provided criteria) in the environment and forwards them to the remote SmartHEALTH information system.

7.3 Runtime Environments

This section describes the third-party *software* runtime environments that we have used for the SMDS framework and SmartHEALTH Project.

7.3.1 JamVM

JamVM[4] is a small Java Virtual Machine (JVM) which conforms to the Java 2 specification, licensed under GPL, and targets the embedded devices. In comparison to most other VMs (free and commercial) it is extremely small, with a stripped executable on PowerPC of only 220K, and Intel 200K. However, unlike other small VMs (e.g. KVM) it is designed to support the full specification and includes support for object finalization, Soft/Weak/Phantom References, class-unloading, the Java Native Interface (JNI) and the Reflection API. JamVM's interpreter is highly optimized, incorporating many state-of-the-art techniques such as *stack-caching* and *direct-threading*. JamVM can be compiled for Linux, MacOS and Solaris platforms, and particularly for ARM processors

[4]JamVM, A Java Virtual Machine for small devices; `http://jamvm.sourceforge.net/`

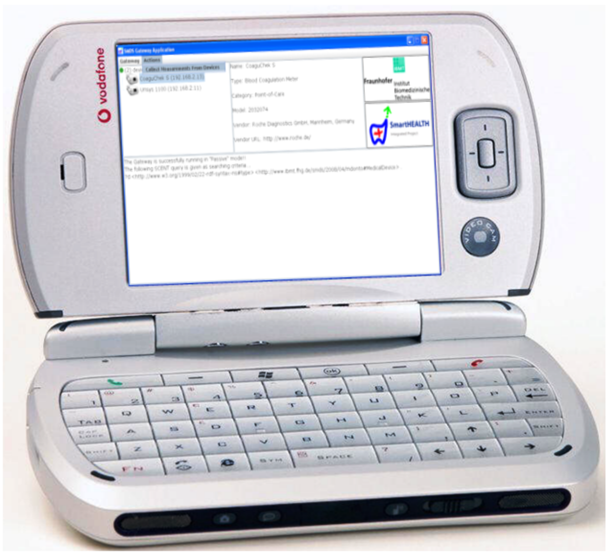


Figure 7.4: The Vodafone VPA-4 with Windows Mobile 2003

which are mainly used in embedded systems due to their low energy consumption and high processing speed. JamVM's major advantages include its widespread use and good support of JNI which is needed to access the hardware interfaces on the embedded platforms through Java. We have used JamVM for embedded Linux running on Gumstix in order to host SMDS framework, as explained under Section 7.2.1.

7.3.2 Mysaifu JVM

Mysaifu[5] JVM is a free, open-source Java Virtual Machine which conforms to Java 2 specification and is targeted for almost all Windows Mobile and Windows CE (Compact Edition) based Pocket PC platforms. Mysaifu JVM is developed in Java and C++ programming languages and is available under GPL License 2.0. We have used Mysaifu JVM in order to host SMDS framework on Windows Mobile based devices, as explained in Sections 7.2.2 and 7.2.3.

7.3.3 GNU Classpath

The GNU Classpath[6] is an almost Java 1.5 compatible implementation of the Java standard libraries that is licensed under GPL. An exact comparison between the original Sun Microsystems JDKs and GNU Classpath can be found at `http://builder.classpath.org/japi/jdk15-classpath.html`. At the time of writing this thesis, the GNU Classpath serves 99.11% of JDK 1.4 and 95.18% of JDK 1.5. In contrast to Sun Microsystems JDKs, the GNU Classpath is ported to several ARM platforms and can be used in combination with several JVMs, such as JamVM, Cacaovm or Kaffe. We have used GNU Classpath in conjunction with JamVM in order to host SMDS framework on Gumstixs, as explained under Section 7.2.1.

[5]Mysaifu, A JVM for Windows Mobile; `http://sourceforge.jp/projects/mysaifujvm/`
[6]GNU Classpath - GNU Project; `http://www.gnu.org/software/classpath/`

7.3.4 CSOAP Web Services Server

CSOAP[7] is a research outcome of OZONE[8] Project, and provides a Java based SOAP engine for resource-constrained devices (mobile or PDAs) and is able to deploy Web Services, and to manage RPCs (Remote Procedure Call) from SOAP clients and dispatch them to the Web Services. The implementation of CSOAP follows the Sun's JAX-RPC Specification, which gives a standard for SOAP-based RPC to support the development of SOAP-based interoperable and portable Web Services. The original CSOAP server supports Java Web Services *only* for Java based clients, but for SMDS framework, we have adapted the CSOAP server to additionally support the clients developed in Microsoft .NET programming languages, i.e. Visual C#.NET and Visual Basic.NET.

7.3.5 Jetty Web Server

Jetty[9] is a compact HTTP server, HTTP client, and a Java servlet container, which is used in a wide variety of projects and products, ranging from embedded devices, tools, frameworks, application servers, and clusters. Jetty is open source and available for free commercial use and distribution under Apache License 2.0. In SMDS framework, Jetty's *servlet container* is used to host the CSOAP server, while the *HTTP Server* is used for hosting the Web Services description files (SA/WSDL files), the device' certificate to be downloaded on the client side, and for *SMDDP protocol* for receiving the HTTP callbacks from the matched medical or mobile devices during the discovery process.

7.3.6 HyperSQL Database

HyperSQL Database (HyperSQL[10]) is the leading SQL relational database engine written entirely in Java. It has a JDBC driver and supports a rich subset of ANSI-92 SQL (BNF tree format) plus many SQL:2008 enhancements. It offers a small, fast database engine which offers both *in-memory* and *disk-based* tables and supports embedded and server modes. Additionally, it includes tools such as a minimal web server, in-memory query and management tools (can be run as applets) and a number of demonstration examples. In SMDS framework, we have used HSQL database to host the device DB (see Appendix A for the class diagram) where each medical device stores its measurements and other relevant information, which are later offered through Semantic Web Services.

7.4 Software Libraries

This section outlines the different software libraries used by the SMDS framework.

7.4.1 SAWSDL4J API

The Semantic Annotations for WSDL and XML Schema (SAWSDL[11]) W3C Recommendation defines mechanisms using which semantic annotations can be added to WSDL components. SAWSDL does not specify a language for representing the semantic models, e.g. ontologies, rather it provides mechanisms by which concepts from the semantic

[7]CSOAP - Web Services Server; `http://www-rocq.inria.fr/arles/download/ozone/csoap-1.0.zip`
[8]OZONE - EU-IST Project; `http://www.hitech-projects.com/euprojects/ozone/`
[9]Jetty - A Compact HTTP Server/Client; `http://www.eclipse.org/jetty/`
[10]HSQL - A 100% Java SQL Database System; `http://hsqldb.org/`
[11]Semantic Annotation for WSDL and XML Schema; `http://www.w3.org/2002/ws/sawsdl/`

models that are defined either within or outside the WSDL document can be referenced from within WSDL components as *annotations*. These semantics when expressed in formal languages can help to disambiguate the description of Web Services during automatic discovery and composition of the Web services. SAWSDL4J[12] API interface is an implementation of the SAWSDL specification, which allows the developers to create SAWSDL based applications by annotating the WSDL document of Web Services. In SMDS framework, the *client* medical or mobile device(s) make(s) use of SAWSDL4J API to *parse* the SAWSDL documents of the *matched* medical or mobile devices in order to extract the the name(s) of the Web Service method(s) to invoke, which *match* with the desired input/output parameters types.

7.4.2 Bouncy Castle Cryptography API

Bouncy Castle[13] provides light-weight cryptography APIs for Java and C#.NET, as well as provider for Java Cryptography Extension and the Java Cryptography Architecture. In SMDS framework, besides using the standard Java cryptography API, an implementation of Bouncy Castle is also developed to provide security for the applications running on Gumstixs and Mobile platforms, developed in Java and C#.NET languages respectively. The user can also use any other third-party security provider, provided s/he has implemented the required cryptographic methods defined in the interface, i.e. *sign*, *verify*, *encrypt* and *decrypt* etc.

7.4.3 Piccolo XML Parser

Piccolo[14] is a small, extremely fast XML parser for Java, which implements the SAX 1, SAX 2.0.1, and JAXP 1.1 (SAX parsing only) interfaces as a non-validating parser and attempts to detect all XML well-formedness errors. Piccolo was developed by Yuval Oren and is released as open source software under the terms of the Apache Software License 2.0. In SMDS framework, Piccolo is used for fast XML processing of WSDL, SAWSDL documents, and for ontologies and knowledge base triples as well.

7.4.4 RDF Filter for SAX2

The RDF Filter[15] is a simple filter layer between SAX (The Simple API for XML) and the higher-level RDF (Resource Description Format), an XML-based object-serialization and meta-data format. The RDF filter library is used by several RDF-based projects. In SMDS framework, RDF Filter is used in conjunction with Piccolo XML parser, while loading the knowledge base or ontologies triples in the memory.

7.4.5 RXTX - A Communication Library

RXTX[16] is a native library, available for various platforms, i.e. Windows and Linux, and provides *serial* and *parallel* communication functionalities to the Java based applications. In SMDS framework, RXTX is used exclusively on Gumstixs to support their serial communication with medical devices. Fig. 7.1 and Fig. 7.2 illustrate how the medical device (RS-232) is attached with serial interface (RS-422) of Gumstix.

[12]SAWSDL4J API; `http://lsdis.cs.uga.edu/projects/meteor-s/opensource/sawsdl4j/`
[13]Bouncy Castle Cryptography APIs; `http://www.bouncycastle.org/`
[14]Piccolo XML Parser for Java; `http://piccolo.sourceforge.net/`
[15]RDF Filter for SAX2; `http://rdf-filter.sourceforge.net/`
[16]RXTX - A Library for Serial/Parallel Communication; `http://users.frii.com/jarvi/rxtx/`

7.5 Software Tools/IDEs

This section describes details about the software tools and IDEs (Integrated Development Environments) used for the software development of SMDS framework/applications.

7.5.1 Microsoft Visual Studio 2005

Microsoft Visual Studio 2005[17] is an IDE from Microsoft. It can be used to develop *console* and *GUI* (Graphical User Interface) applications along with Windows Forms applications, web sites, web applications, and Web Services in both *native code* together with *managed code* for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight. In SMDS framework and for SmartHEALTH project, we have used MS Visual Studio 2005 and Visual C#.NET to adapt the SMDS framework and the development of GUI applications for Windows (mobile) platforms, respectively.

7.5.2 Eclipse IDE

Eclipse[18] is a free, open-source, multi-language IDE (Integrated Development Environment) for various languages, e.g. Java, C/C++, PHP, and Perl etc. Eclipse employs *plug-ins* in order to provide all of its functionality on top of (and including) the runtime system, in contrast to some other applications where functionality is typically hard coded. The complete SMDS framework as well as rich graphics applications, i.e. Gateway application, are developed using Eclipse IDE.

7.5.3 Protégé

Protégé[19] is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. The Protégé platform supports two main ways of modeling ontologies: the *Protégé-Frames* editor enables users to build and populate ontologies that are *frame-based*, in accordance with the OKBC (Open Knowledge Base Connectivity) protocol; while the *Protégé-OWL* editor enables users to build ontologies for the *Semantic Web*, in particular in the W3C's OWL (Web Ontology Language) language. In SMDS, we have used Protégé-OWL to develop *MeDO* (Medical Device Ontology) and *SmdsOnto* (SMDS Ontology) ontologies.

7.5.4 Radiant

Radiant[20] is an eclipse plug-in that provides a graphical user interface to allow the developers to create and publish SAWSDL and WSDL-S service interfaces by adding *annotations* to the existing service descriptions in WSDL via OWL ontologies. In SMDS framework, Radiant is used to annotate the WSDL document of device Web Services via *MeDO* (Medical Device Ontology) and *SmdsOnto* (SMDS Ontology) ontologies.

[17]MS Visual Studio 2005; `http://msdn.microsoft.com/en-us/vstudio/default.aspx`
[18]Eclipse - A Java Integrated Development Environment; `http://www.eclipse.org/`
[19]Protégé - An Ontology Editor and Knowledge base Framework; `http://protege.stanford.edu/`
[20]Radiant Plug-in; `http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1`

Chapter 8

Experimental Evaluation

This chapter provides details about the experimental evaluation of *Semantic Medical Devices Space* (SMDS) framework in different pervasive healthcare scenarios, particularly in connection with SmartHEALTH Project [21]. Sections 8.1 and 8.2 illustrate different types of medical devices and health information systems (HIS), respectively, that we have used as testbed for our experiments, while Section 8.3 illustrates all the healthcare scenarios in detail which make use of these medical devices and HISs.

8.1 Testbed Medical Devices

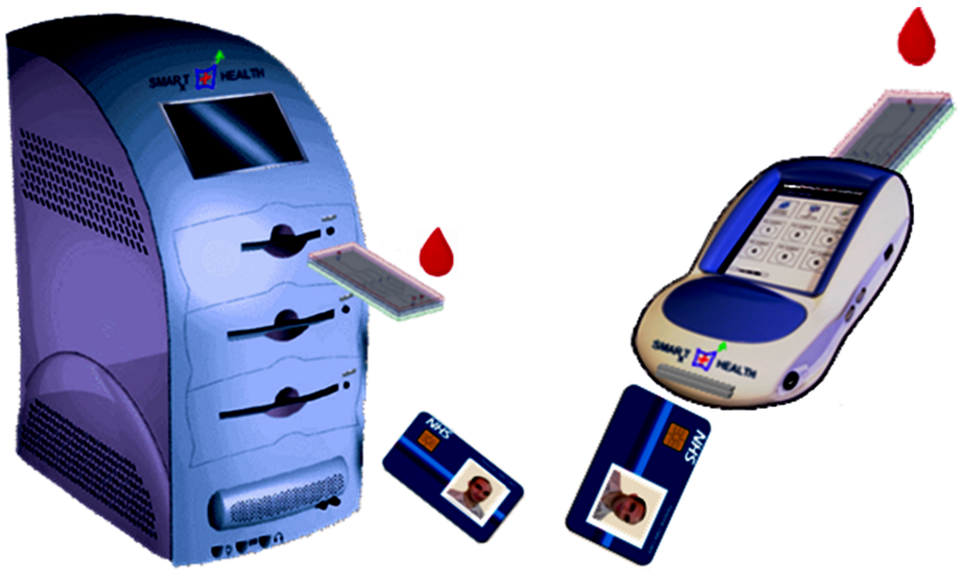
This section provides details about all the medical devices that are used in different scenarios of SmartHEALTH Project. Each of the medical device listed below was attached with a separate Gumstix system through a serial cable, which connected the serial port of medical devices (RS-232) to the serial port of Gumstix (RS-422). The medical devices having no serial interface were connected with through Bluetooth/WiFi interface.

8.1.1 Blood Cancer Markers Analyzer - SmartHEALTH Device

In SmartHEALTH Project, a new generation of intelligent *lab-on-chip* bio-diagnostic devices for point-of-care (POC) applications is developed that incorporates advanced capabilities for context awareness, data interpretation through soft computing tools, e.g. Neural Networks, ubiquitous communication in pervasive healthcare environments, and the provision of e-Health services. SmartHEALTH devices are developed for three types of cancers, namely *breast*, *cervical* and *colorectal*, and are intended to operate in all POC driven environments such as hospitals, physicians' offices and ultimately patient self-testing at home or while moving. It is envisioned that SmartHEALTH devices at the POC will be able to *securely* communicate seamlessly and transparently with the local or remote health information system(s). Fig. 8.1(a) and Fig. 8.1(b) show the *desktop* and *portable* types of SmartHEALTH device, respectively.

8.1.2 Pulse Oximeter - Masimo Rainbow

A *Pulse Oximeter* is a medical device that *indirectly* measures the oxygen saturation of a patient's blood (as opposed to measuring oxygen saturation directly through a blood sample) and changes in blood volume in the skin by producing a photoplethysmograph. It is often attached to a medical monitor so the staff could see a patient's oxygenation at



(a) SmartHEALTH Device - Desktop Unit (b) SmartHEALTH Device - Portable Unit

Figure 8.1: Different types of SmartHEALTH Cancer Biodiagnostic Devices

all times. Fig. 8.2(a) shows an example of new generation of battery-operated fingertip pulse oximeters, which is good for home blood-oxygen monitoring, while Fig. 8.2(b) shows portable Masimo Rainbow pulse oximeter, which is used in hospital wards as well as in home-care monitoring. Because of the simplicity and speed, pulse oximeter is of critical importance in emergency medicine and is also very useful for patients with respiratory or cardiac problems, or for diagnosis of some sleep disorders such as *apnea* and *hypopnea*.

(a) A fingertip pulse oximeter (b) A portable pulse oximeter

Figure 8.2: Different types of portable pulse oximeters

8.1.3 Urine Analyzer - Urisys 1100

Fig. 8.3 shows the Urisys 1100 urine analyzer from Roche® Diagnostics, which is a compact, time-saving and easy to use system for standardized, semi-quantitative, and one-at-a-time evaluation of Chemstrip 10MD urine test strips. Simple operation, an attractive design, improved functions and flexible software options make it a suitable system solution for physicians' offices, specialists and occupational health care units as well as for small hospital laboratories and other decentralized testing sites, e.g. hospital wards and decentralized health care units, seeking a cost-effective way to improve standardization and work-flow of urine analysis.

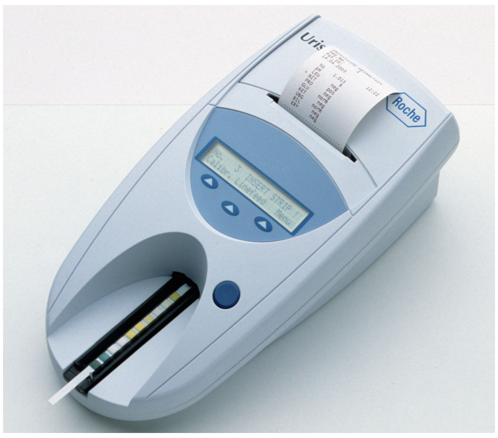


Figure 8.3: Urisys 1100 from Roche® Diagnostics for Urine Analysis

8.1.4 Blood Coagulation Meter - CoaguChek S

Fig. 8.4 shows the CoaguChek® S meter, which is a well-proven and established blood monitoring system from Roche® Diagnostics for *Prothrombin Time* (PT) and *International Normalized Ratio* (INR). It is a reflectance photometer for fast ($\sim$ 1 min) *on-the-spot* measurement from one single drop (10 μl) of capillary, means from the fingertip, or venous whole blood. In addition, it has a large, easy-to-read and icon-based display.

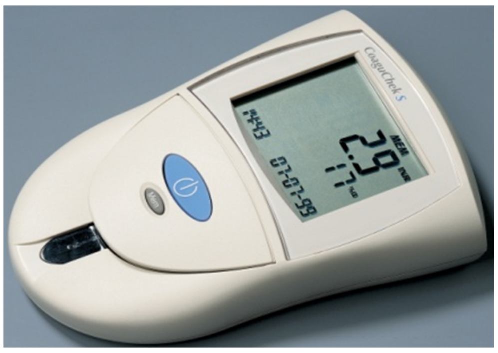


Figure 8.4: CoaguChek S System from Roche® Diagnostics for PT/INR Monitoring

8.1.5 Vital Signs Monitor - VITALMAX 4000 CL

The VITALMAX 4000 CL is simple to set up and the most affordable continuous *multi-parameter* bed-side patient monitor with an outstanding user-friendly software, and features an *eight channel* waveform display on a color LCD with a 12.1 inches effective display area. It is the perfect answer to tight spaces and tight budgets, as it combines the benefits of an anesthetic agent monitor in one compact unit. It is ideally suited for the operating room as it has a two hour battery backup which facilitates patient transfer to the recovery room without interrupting the continuous monitoring of vital signs data.

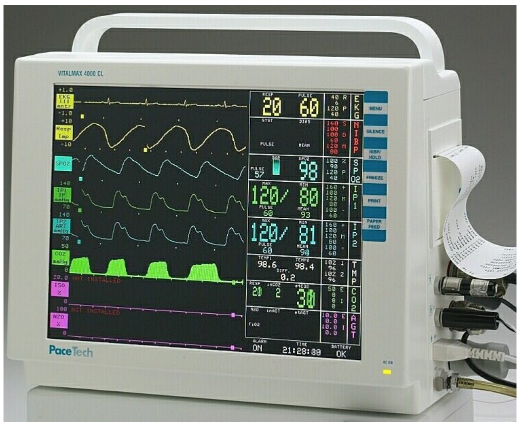


Figure 8.5: Vital Signs Monitor - VITALMAX 4000 CL

8.1.6 Weight Scale - SOEHNLE-Professional 7700

The accuracy, reliability and easy handling are the key factors when weighing patients in hospitals and in GP offices. Fig. 8.6 shows SOEHNLE Professional 7700 scale, which is particularly suitable for specialized hospital wards, as well as at home for periodic weight monitoring. It offers RS-232 interface to transmit the measurement to the Gumstix.

Figure 8.6: Medical Personal Health Scale - SOEHNLE Professional 7700

8.1.7 Digital Blood Pressure Monitor - A&D Medical UA-767PC

Electronic or digital blood pressure monitors for home use are either semi-automatic manual inflation (the user squeezes the bulb to inflate the cuff) or automatic inflation. Automatic monitors have everything contained in one unit, so they're easier to handle than systems with a separate gauge and stethoscope. Most home blood pressure monitors are portable and have a D-ring cuff for one-handed application, which may fit around the wrist or upper arm. Also, expensive monitors have automatic inflation and deflation systems, along with large, easy-to-read digital displays and error indicators, and built-in pulse (heart rate) measurement. Fig. 8.7 shows the UA-767PC automatic blood pressure monitor, which is clinically validated for accuracy and features memory for 280 readings with wired communication (RS-232 or USB) to the computer system.

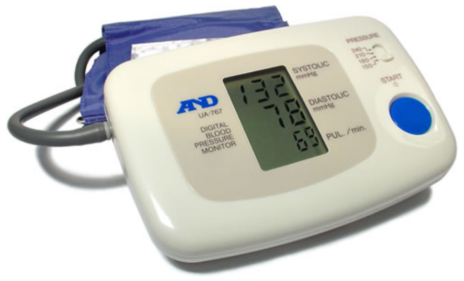


Figure 8.7: UA-767PC Digital Blood Pressure Monitor with Serial Interface

8.2 Testbed Health Information Systems

8.2.1 Laboratory Information System

The laboratory information system (LIS) of the NHS laboratory in New Castle upon Tyne, United Kingdom is taken as testbed for the evaluation of SmartHEALTH analyzers and SMDS framework. Fig. 8.1 reflects this HL7 based LIS, which is connected with SmartHEALTH Information System in the laboratory for the synchronization of patients' medical records.

8.2.2 Hospital Information System

The hospital information system (HIS) of Donostia Hospital in San Sebastian, Spain is taken as testbed for the evaluation of SmartHEALTH analyzers and SMDS framework. Fig. 8.2 reflects this HL7 based HIS, which is connected with SmartHEALTH Information System in the hospital for the synchronization of patients' medical records.

8.2.3 SmartHEALTH Information System

The SmartHEALTH Information System (SIS) is presented as a part of the complete SmartHEALTH ICT platform with the back-end of a database of all relevant patients' data, which is presented to the authorized users (e.g. health professionals) in their Web browsers. The SIS does not claim to be a replacement for any HIS or any electronic health record, rather to assist them significantly in an ambient intelligent infrastructure. It can be used for a number of purposes: it provides a platform for the validation of SmartHEALTH devices, by comparing the measurement results taken from SmartHEALTH devices and from other reference/golden-standard devices; it offers services for the remote home-monitoring of the patients; it validates our proof-of-the-concept, which is to use Semantic Web Services for the communication among SmartHEALTH devices as well as with SIS; it serves as a data management platform for the *clinical validation* of different cancer markers or the interpretation algorithms of multi-parametric cancer marker analysis. In addition, it represents the traditional way how specific *point-of-care* equipment is integrated in a heterogeneous health information infrastructure in a hospital or laboratory, where the medical results are sent from devices to a specific information system that acts as a *bridge* to the HIS or LIS.

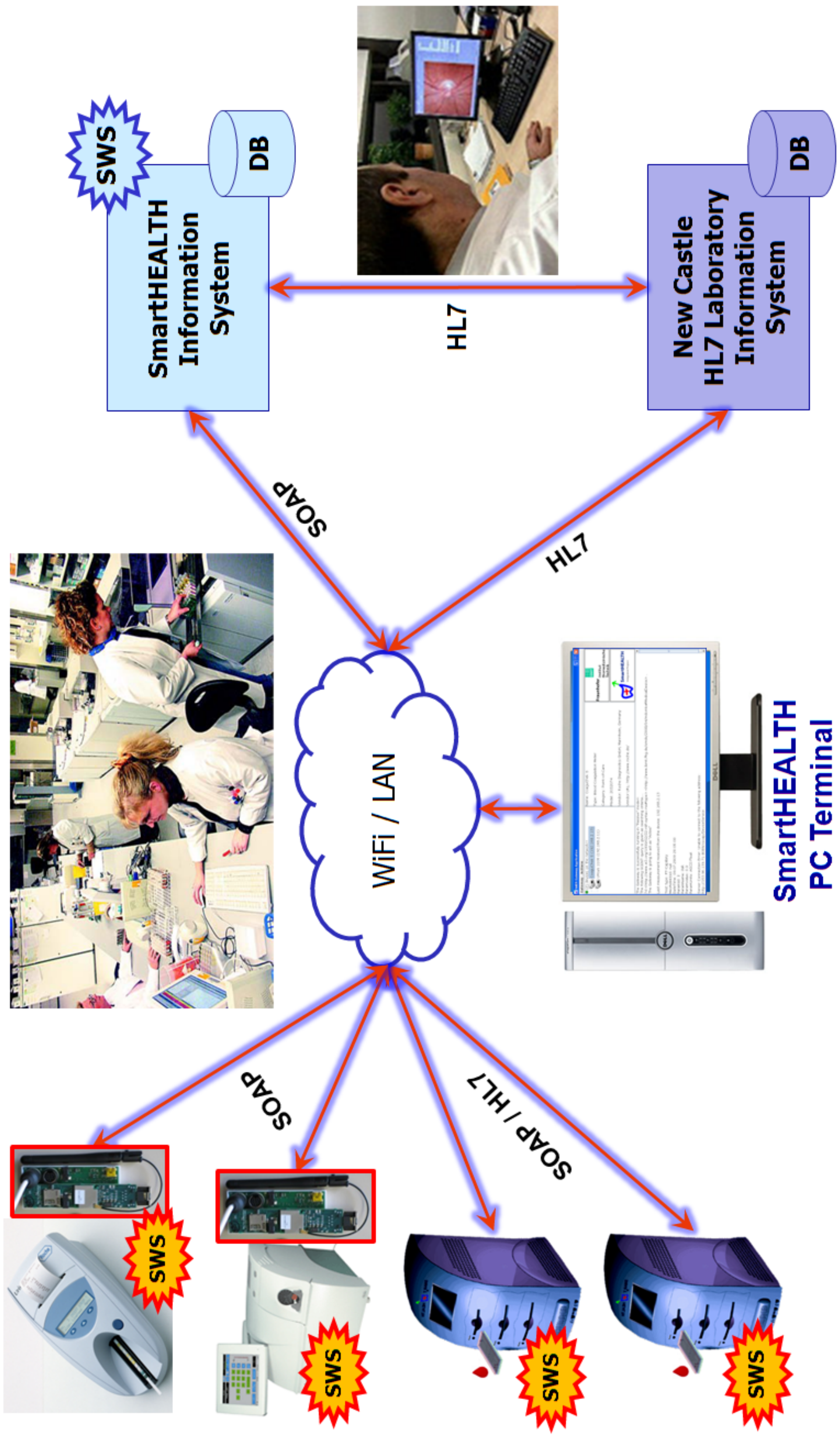
8.3 Testbed Pervasive Healthcare Scenarios

This section provides details about the pervasive healthcare scenarios in which medical devices semantically coordinate with each other and with health information systems through the SMDS framework. For the following scenarios, we define two roles, namely *active gateway* and *passive gateway* for an entity, which could be a SmartHEALTH device or PC/PDA running the SmartHEALTH Terminal application together with SMDS gateway application (GUI or Console based). *Active gateway* means that the entity plays an active role by initiating the process of collecting measurement results from *all* or *desired* medical or mobile devices in the environment, as per the requirements, and then forwards them to the respective information system. *Passive gateway* means that the entity does not initiate any process, rather it facilitates other medical or mobile devices in the environment with its services in order to forward their measurement results to the respective information system. An entity can have both active and passive roles together.

8.3.1 Cancer Diagnosis - Laboratory Scenario

Fig. 8.1 illustrates the *pre-operation cancer diagnosis scenario*, which was carried out as a part of SmartHEALTH user evaluation use-cases at the clinical laboratory of New Castle upon Tyne, United Kingdom. It involves a couple of SmartHEALTH Desktop analyzers, two other types of analyzers, each attached with a separate Gumstix (red en-boxed), SmartHEALTH PC Terminal application (STA) with GUI based SMDS gateway application, an in-house HL7 based laboratory information system and the SmartHEALTH information system (SIS). In this scenario, the STA uses the GUI based gateway application, as shown in detail in Appendix D, as an *active* gateway for all the medical devices in the laboratory and automatically collects the data values from these medical devices, whenever a new measurement of a patient is available. Additionally, it makes a collective intelligent interpretation of all the measurement values using soft computing tools (through trained neural networks and/or support vector machine), and then forwards these measurement values with their interpretation to the SIS through Semantic Web Service. If configured, the STA sends these measurement values to the legacy hospital information system through HL7 communication standard as well, otherwise the SIS automatically synchronizes these measurement results of the patient with the health record stored in hospital information system. The SIS offers a Web based application for the authorized health professionals to view and analyze the patients' disease status through case-reports (cancer's progression or regression) and react accordingly.

Table 8.1: Pre-operation Cancer Diagnosis Scenario at Clinical Laboratory of New Castle upon Tyne Hospital, U.K



8.3.2 Cancer Treatment - Hospital Scenario

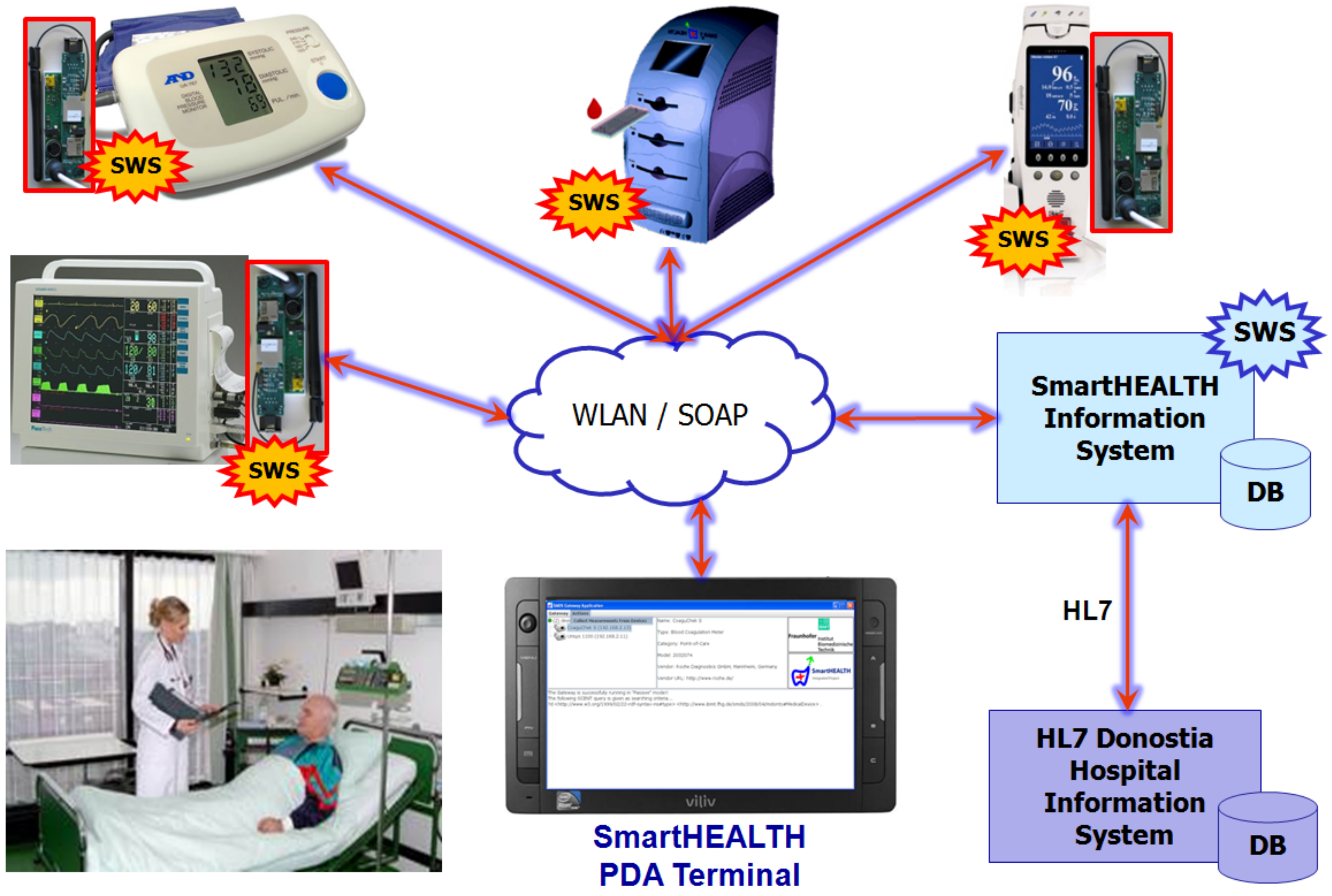
Fig. 8.2 illustrates the *post-operation cancer treatment scenario*, which was carried out at the Donostia Hospital of San Sebastian, Spain, as a part of SmartHEALTH user evaluation use-cases. It involves a SmartHEALTH Desktop analyzer, SmartHEALTH PDA Terminal application (STA), blood pressure device, ECG (Electro Cardiogram) device, pulse Oximeter, each attached with a separate Gumstix (red en-boxed), with GUI based SMDS gateway application, an in-house HL7 based hospital information system and the SmartHEALTH information system (SIS). In this scenario, the STA acts as an *active* gateway for all the medical devices in the hospital ward and automatically collects the data values from these medical devices attached with a patient, whenever a new measurement is available. For the *real-time* ECG and vital parameters data transfer to SIS, the corresponding medical devices use the data forwarding Semantic Web Services offered by the PDA based gateway application. Additionally, a collective intelligent interpretation of all the measurement values is made using soft computing tools (through trained neural networks and/or support vector machine), and then forwards these measurement values with their interpretation to the SIS through Semantic Web Service. If configured, the STA sends these measurement values to the legacy hospital information system through HL7 communication standard as well, otherwise the SIS automatically synchronizes these measurement results of the patient with the health record stored in hospital information system. The SIS offers a Web based application for the authorized health professionals to view and analyze the patients' disease status through case-reports (cancer's progression or regression) and react accordingly.

8.3.3 Cancer Follow-up and Monitoring - Home-care Scenario

The SmartHEALTH Project aims to support independent living at home by providing *post-operation home-care breast cancer monitoring* service, which provides an improved quality of life for the patients and their families, and provides the opportunity to the patient to stay at home yet receive the level of care required for their condition.

Fig. 8.3 illustrates this pervasive healthcare scenario where a SmartHEALTH portable device, besides being blood cancer markers analyzer, also acts as an active or passive gateway for other medical devices (e.g. blood pressure, blood coagulation, weight scale etc.) and offers its services as Semantic Web Services. As an *active* gateway device, the SmartHEALTH device semantically discover other *desired* or *all* medical devices in the environment, as per the requirements of healthcare application, collects the measurement values of a patient from these medical devices, makes a collective intelligent interpretation of all the measurement values using soft computing tools, and then forwards these measurement values with their interpretation to the remote hospital information system and/or doctor's clinical information system as per configuration. As a *passive* gateway device, the SmartHEALTH device only acts as a *data forwarding* device for other medical devices in the environment. Whenever a medical device finishes its measurement, it semantically discovers the SmartHEALTH device as gateway device, and uses its service to forward the measurement results to the respective information system(s), in our case the SmartHEALTH information system. Finally, the patient's family doctor or health professional in the hospital analyzes the results of blood cancer markers, the measurement values of other medical devices, and their intelligent interpretation in terms of the disease status (cancer's progression or regression) in order to discuss with the patient about the further steps to be taken.

Table 8.2: Post-operation Cancer Treatment Scenario at Donostia Hospital of San Sebastian, Spain


SWS
SWS
SWS
SWS
SWS
WLAN / SOAP
SmartHEALTH Information System
DB
HL7
HL7 Donostia Hospital Information System
DB
SmartHEALTH PDA Terminal

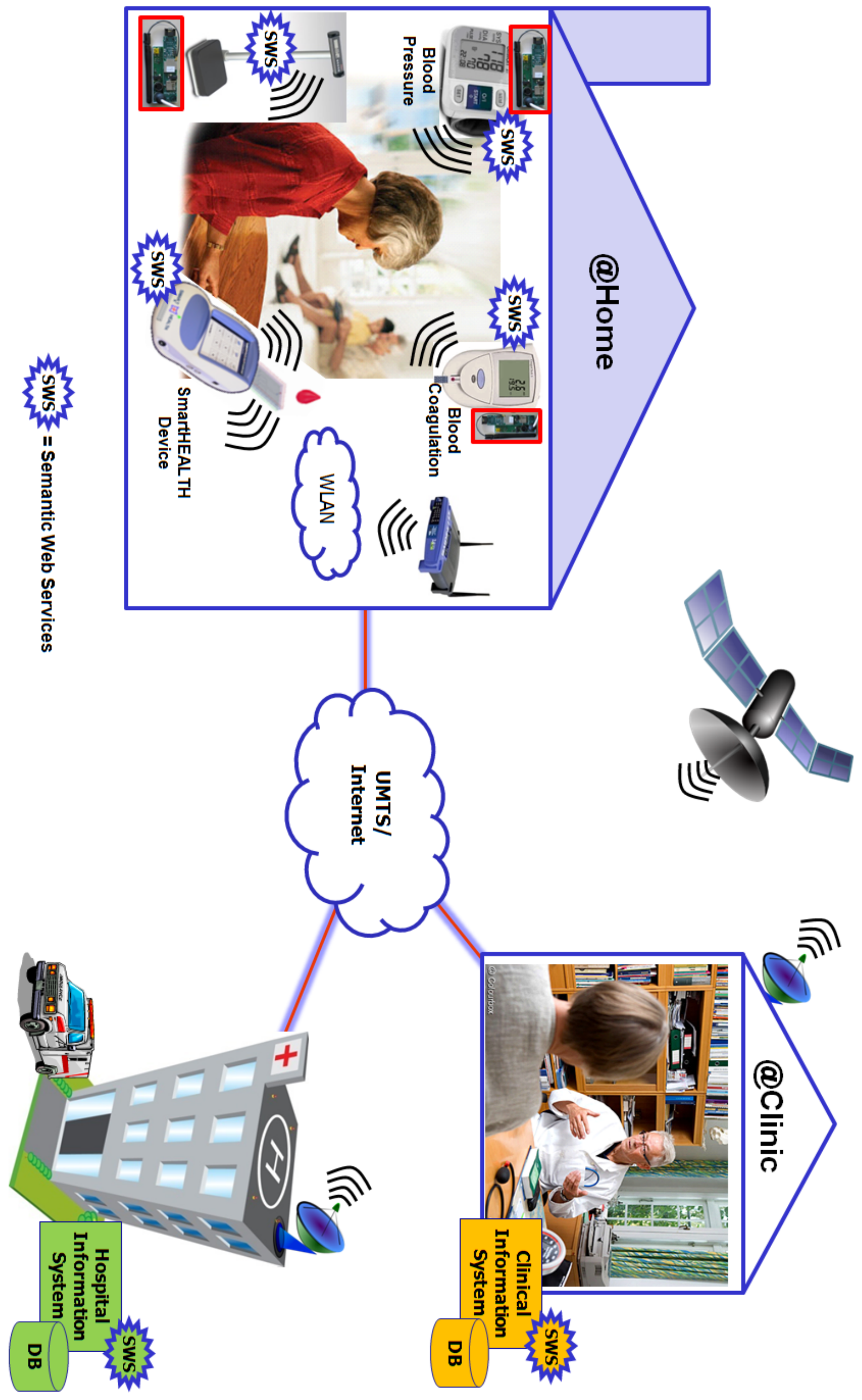


Table 8.3: Post-operation Home-care Breast Cancer Follow-up and Monitoring Scenario

8.4 Conclusions

This chapter presents various medical devices, information systems and the pervasive healthcare scenarios, which are used to invalidate the performance and behavior of SMDS framework. Every medical device used in the healthcare scenarios, except the SmartHEALTH device, was equipped with a Gumstix which hosted the SMDS framework, not only to enable it to semantically discover other desired or all medical devices or information systems in the healthcare environment, but also to get their measurement results through the Semantic Web Services offered by them. We have got satisfactory results, in terms of memory usage, cost, performance and reactivity of the hardware used for the enrichment of state-of-the-art medical devices and their leverage towards the next generation of ambient intelligent medical devices. Our experiments show that the SMDS framework in particular, and other current SOA based approaches in general are not feasible for *real-time* medical devices, e.g. ECG vital signs monitor or Pulse Oximeter, where hundreds of data frames per second are transmitted. However, on the contrary, medical devices working on *store-and-forward* principle gave exceptional results and we believe that the SMDS framework will be a mile-stone towards achieving full-fledged semantic interoperability among resource-constrained medical or mobile devices, not only in the field of pervasive healthcare, but in other ubiquitous computing application in general as well.

Table 8.4 extends the comparison of the related work, which is presented in Section 3.7, and provides the comparative analysis of SMDS framework. The final weight of SMDS (**58**) shows that it is the best candidate to be used on the resource-constrained medical or mobile devices, whose functional characteristics are to be exposed as SWSs, and enriching them with semantic processing capabilities without loosing autonomy with less resources requirements.

Table 8.4: Comparative analysis of related work against the evaluation criteria

Criterion	SMDS	SCALLOPS	HYDRA	SODA	Gaia	Task Computing
Decentralization	*Very High (16)*	*Very High (16)*	*Very High (16)*	*High (12)*	*Low (4)*	*Low (4)*
Reasoning Capability	*High (12)*	*High (12)*	*High (12)*	*Low (4)*	*Very High (16)*	*Low (4)*
Standards Adherence	*High (6)*	*High (6)*	*High (6)*	*High (6)*	*High (6)*	*High (6)*
Lightness	*Very High (12)*	*Low (3)*	*Low (3)*	*Very High (12)*	*Low (3)*	*Low (3)*
Technological Consistency	*High (9)*	*High (9)*	*High (9)*	*High (9)*	*Low (3)*	*High (9)*
Context Awareness	*High (3)*	*Very High (4)*	*High (3)*	*Low (1)*	*High (3)*	*Low (1)*
Total	**58**	**50**	**49**	**44**	**35**	**27**

Chapter 9

Conclusion and Outlook

We have considered Semantic Web and Web Service technologies to develop a decentralized semantic middleware infrastructure for the medical devices communication in order to escalate them towards the next generation *Ambient Intelligent* (AmI) medical devices, which are semantically aware of the presence of other medical or mobile devices and their services in an e-Healthcare environment. The salient features of this middleware infrastructure, namely *Semantic Medical Devices Space* (SMDS) set it apart from the related work in several ways. First, the SMDS infrastructure is deployable on both new and existing networks of distributed wireless and wired medical devices, which operate with limited resources in terms of computing power, energy and memory usage. Second, the SMDS framework facilitates the medical devices' manufacturers to turn their *dumb* medical devices into AmI medical devices, by exposing their functional capabilities as *Semantic Web Services* (SWSs) which enable them to dynamically and semantically *discover & invoke* the Web Services of other medical devices, and could use their medical data or measurement in a context-aware application.

We have presented, as integral part of SMDS, a lightweight HTTP-based semantic discovery protocol, namely SMDDP (*Semantic Medical Device Discovery Protocol* to support the semantic discovery of AmI medical devices based on their *physical* characteristics (i.e. device vendor, device group/type etc.) and/or *functional* characteristics (i.e. what methods does a SWS provides and their semantic descriptions etc.). Also, we have presented, as integral part of SMDS, a lightweight but powerful knowledge base querying and Description Logics and/or Rules based reasoning system, namely μOR (*Micro OWL Querying and Reasoning System*, which can be integrated on resource-constrained medical or mobile devices. It uses our own SCENTRA algorithm, which is a simple *variables' unification* and *patterns matching* algorithm and used for the SCENT query evaluation as well as for the inferences generation.

Our experiments reflect that SOA architectures, in general, are not suitable for the *real-time* medical devices, where the life-critical data, e.g. ECG, has to be immediately transferred to the e-Health system. The reason is evident, because every frame of real-time data is encapsulated in a SOAP message, which is an unacceptable overhead for the resource-constrained medical devices to process the whole message and extract the relevant data from it. However, we have developed a workable alternative, which gives us *near-real-time* transfer of real-time data, by buffering it in chunks of *configurable* intervals of time, and then sending each chunk one by one to the e-Health system for each interval. In this way, though, there would be a *lag* of that time interval, but on the receiving side, this lag will be unnoticeable and it would seem as the receiving e-Health system is getting continuous real-time data.

On the other hand, we got promising results for our experiments with the *non-real-time* medical devices that usually work on the *store-and-forward* principle, which means that such medical devices perform a measurement, store it in their local memory and then (optionally) forward it to the e-Health system they are connected with. So, in our opinion, it is worth using the SMDS infrastructure, and SOA in general, on such medical devices, which allow them to expose their functionality through Semantic Web Services, allow other medical devices to dynamically and semantically discover these devices and their services, retrieve their measurement results, and then use them for further processing in the respective healthcare application(s).
However, there are issues to be resolved in SMDS framework regarding different aspects as **future work**:

- The current implementation of SMDS framework is targeted for CDC[1] compliant devices only, because of using java.util.HashSet and java.util.HashMap classes for better performance, which are missing from the specifications designed for CLDC[2] and MIDP[3] compliant devices. In order to support SMDS on the latter compliant devices, which have even lesser computing and memory resources, a substitutionary version can be developed using java.util.Vector or java.util.ArrayList classes, but its performance would be comparatively slower.

- The *reliability* of a medical device or its Web Services needs to be determined by its client device(s), as this factor plays a key role in scenarios where more than one medical devices match with the given criteria, and only the best reliable medical device is chosen. This factor is also important when the client is a PC based system and needs to compose the Web Services offered by the *most* reliable medical devices. The SMDS framework still lacks this feature of providing *reliability scores* of medical devices and their Web Services.

- The *authenticity* of a medical device or its Web Services needs to be provided to the respective clients, as this factor is used during the *selection* of medical devices or Web Services, and their *composition*, if required. The SMDS framework lacks the feature of providing authenticity level of a medical device or its Web Services.

- A simple *security mechanism* is developed for the SMDS framework, which is built on the paradigm of asynchronous PKI (*Public Key Infrastructure*) and supports a *handshake of digital certificates* among medical devices as well as (configurable) encryption/decryption and signing/verification functionalities. But, our experiments show that the use of the PKI methods significantly affects the overall performance of resource-constrained medical devices, so an alternative (synchronous) security mechanism should be developed for SMDS framework to enhance its performance.

- The SMDS framework, although the μOR engine supports both *implicit* and *explicit* rules, still lacks support for standard rules definition languages, e.g. SWRL [112] for wide industry adoption. So obvious future work for μOR would be to extend its support for SWRL by developing an appropriate wrapper. Furthermore, the *performance evaluation* of μOR can be extended by studying some of the available benchmarking systems, e.g. LUBM [105]. Last, but not the least, if medical or

[1]Connected Device Configuration; `http://java.sun.com/javame/technology/cdc/index.jsp`
[2]Connected Limited Device Configuration; `http://java.sun.com/products/cldc/`
[3]Mobile Information Device Profile; `http://java.sun.com/products/midp/`

mobile devices are equipped with more computing/memory resources in the future, the remaining set of OWL-Lite axioms, as shown in Table 5.3, could be implemented to enhance the axiomatic expressiveness of μOR, if needed as per the requirements of new pervasive healthcare applications.

- Since SMDS does not support (semi)automatic composition or planning of Semantic Web Services, future work could be the implementation of a small scale composition planer for the small devices. Mechanism for *run-time* compilation of Web Service client stubs could be pursued, perhaps with the help of a proxy system which compiles classes for a device, in order to support fully automatic composition/planning of Semantic Web Services on a device.

List of Figures

List of Tables

Appendix A

UML Class Diagrams of SMDS Software Framework

This appendix shows the UML (Unified Modeling Language) class diagrams of the complete SMDS software framework, to understand the classes' hierarchy, their associations and package dependencies. These diagrams are generated using Omondo[1] plugin.

Fig. A.1 shows the class diagram of SAWSDL wrapper that we developed around the SAWSDL API to support semantically annotated WSDL processing. Fig. A.2 shows the unfolded package hierarchy in the Eclipse environment. Fig. A.3 shows the complete structure of general Web Services offered by a medical or mobile device, though this can easily be extended if further methods need to be developed. Fig. A.4 shows the package structure ofμOR while Fig. A.5 shows the package structure of SMDDP protcol. Fig. A.6 shows the package structure of the security sub-framework based on PKI (Public Key Infrastructure), while Fig. A.7 shows the database schema, used to store the measurements of medical devices and other relevant information.

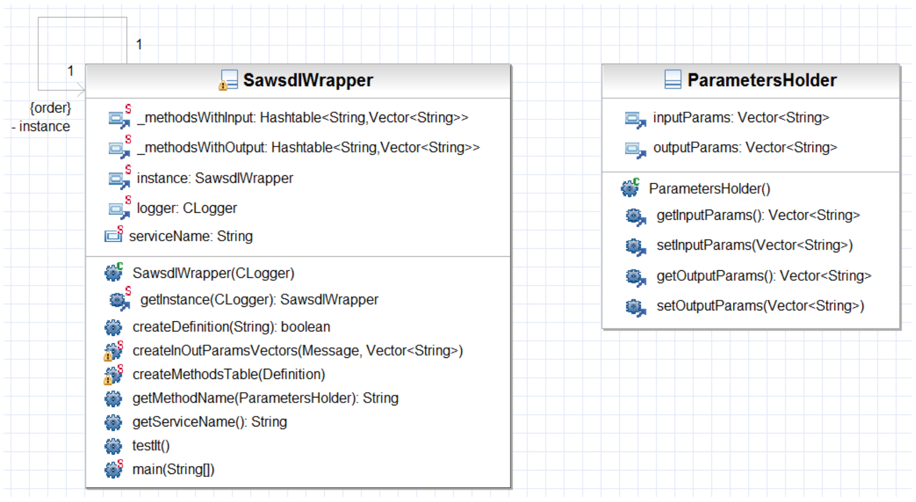
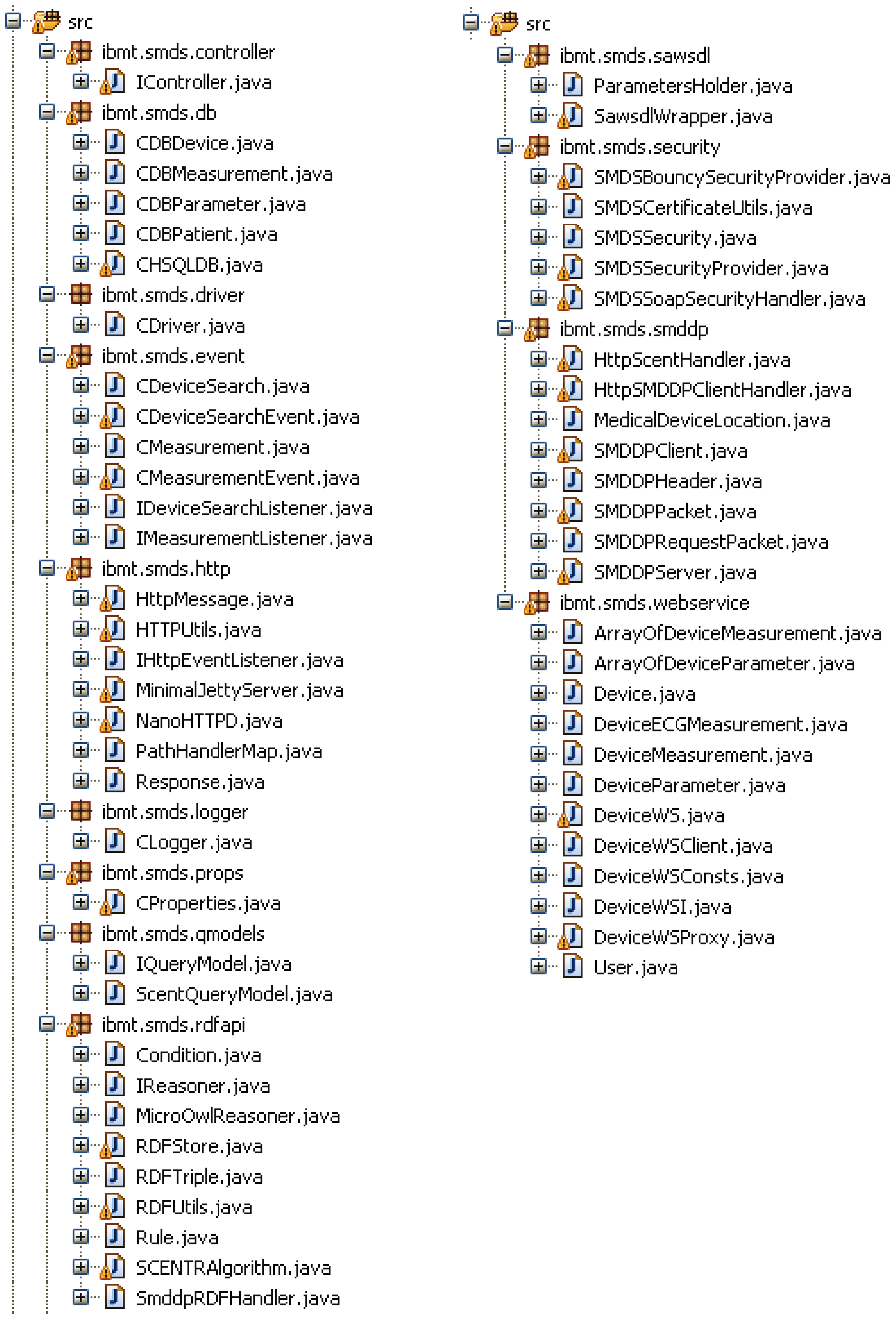

1
1
{order}
- instance
SawsdlWrapper
_methodsWithInput: Hashtable<String,Vector<String>>
_methodsWithOutput: Hashtable<String,Vector<String>>
instance: SawsdlWrapper
logger: CLogger
serviceName: String
SawsdlWrapper(CLogger)
getInstance(CLogger): SawsdlWrapper
createDefinition(String): boolean
createInOutParamsVectors(Message, Vector<String>)
createMethodsTable(Definition)
getMethodName(ParametersHolder): String
getServiceName(): String
testIt()
main(String[])
ParametersHolder
inputParams: Vector<String>
outputParams: Vector<String>
ParametersHolder()
getInputParams(): Vector<String>
setInputParams(Vector<String>)
getOutputParams(): Vector<String>
setOutputParams(Vector<String>)

Figure A.1: UML class diagram of SAWSDL wrapper package

[1]Omondo plugin for Eclispe; `http://www.omondo.com`

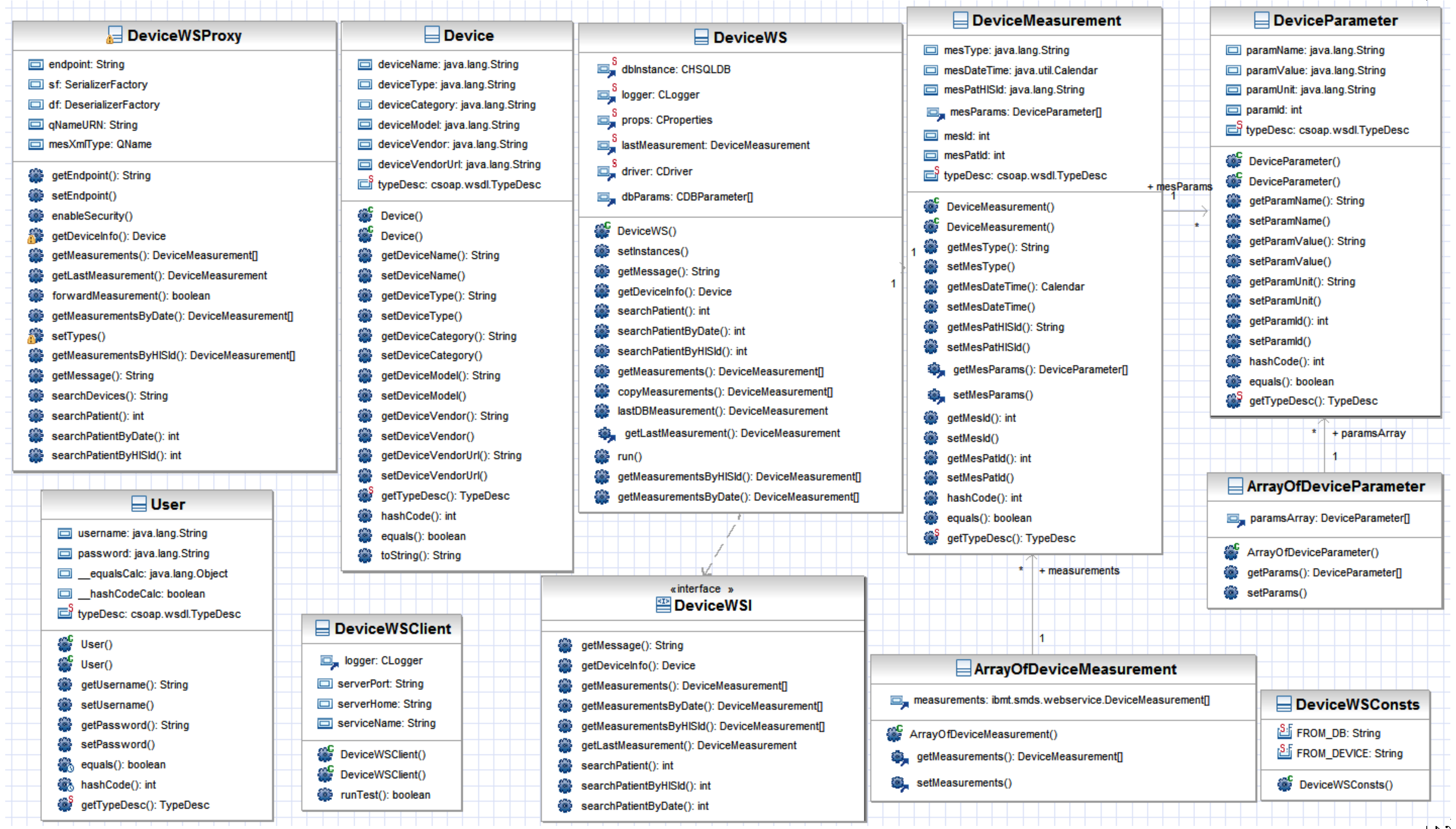


```
src
  ibmt.smds.controller
    IController.java
  ibmt.smds.db
    CDBDevice.java
    CDBMeasurement.java
    CDBParameter.java
    CDBPatient.java
    CHSQLDB.java
  ibmt.smds.driver
    CDriver.java
  ibmt.smds.event
    CDeviceSearch.java
    CDeviceSearchEvent.java
    CMeasurement.java
    CMeasurementEvent.java
    IDeviceSearchListener.java
    IMeasurementListener.java
  ibmt.smds.http
    HttpMessage.java
    HTTPUtils.java
    IHttpEventListener.java
    MinimalJettyServer.java
    NanoHTTPD.java
    PathHandlerMap.java
    Response.java
  ibmt.smds.logger
    CLogger.java
  ibmt.smds.props
    CProperties.java
  ibmt.smds.qmodels
    IQueryModel.java
    ScentQueryModel.java
  ibmt.smds.rdfapi
    Condition.java
    IReasoner.java
    MicroOwlReasoner.java
    RDFStore.java
    RDFTriple.java
    RDFUtils.java
    Rule.java
    SCENTRAlgorithm.java
    SmddpRDFHandler.java
```

```
src
  ibmt.smds.sawsdl
    ParametersHolder.java
    SawsdlWrapper.java
  ibmt.smds.security
    SMDSBouncySecurityProvider.java
    SMDSCertificateUtils.java
    SMDSSecurity.java
    SMDSSecurityProvider.java
    SMDSSoapSecurityHandler.java
  ibmt.smds.smddp
    HttpScentHandler.java
    HttpSMDDPClientHandler.java
    MedicalDeviceLocation.java
    SMDDPClient.java
    SMDDPHeader.java
    SMDDPPacket.java
    SMDDPRequestPacket.java
    SMDDPServer.java
  ibmt.smds.webservice
    ArrayOfDeviceMeasurement.java
    ArrayOfDeviceParameter.java
    Device.java
    DeviceECGMeasurement.java
    DeviceMeasurement.java
    DeviceParameter.java
    DeviceWS.java
    DeviceWSClient.java
    DeviceWSConsts.java
    DeviceWSI.java
    DeviceWSProxy.java
    User.java
```

Figure A.2: Classes hierarchy in different packages of SMDS framework

Figure A.3: UML class diagram of Web Services package



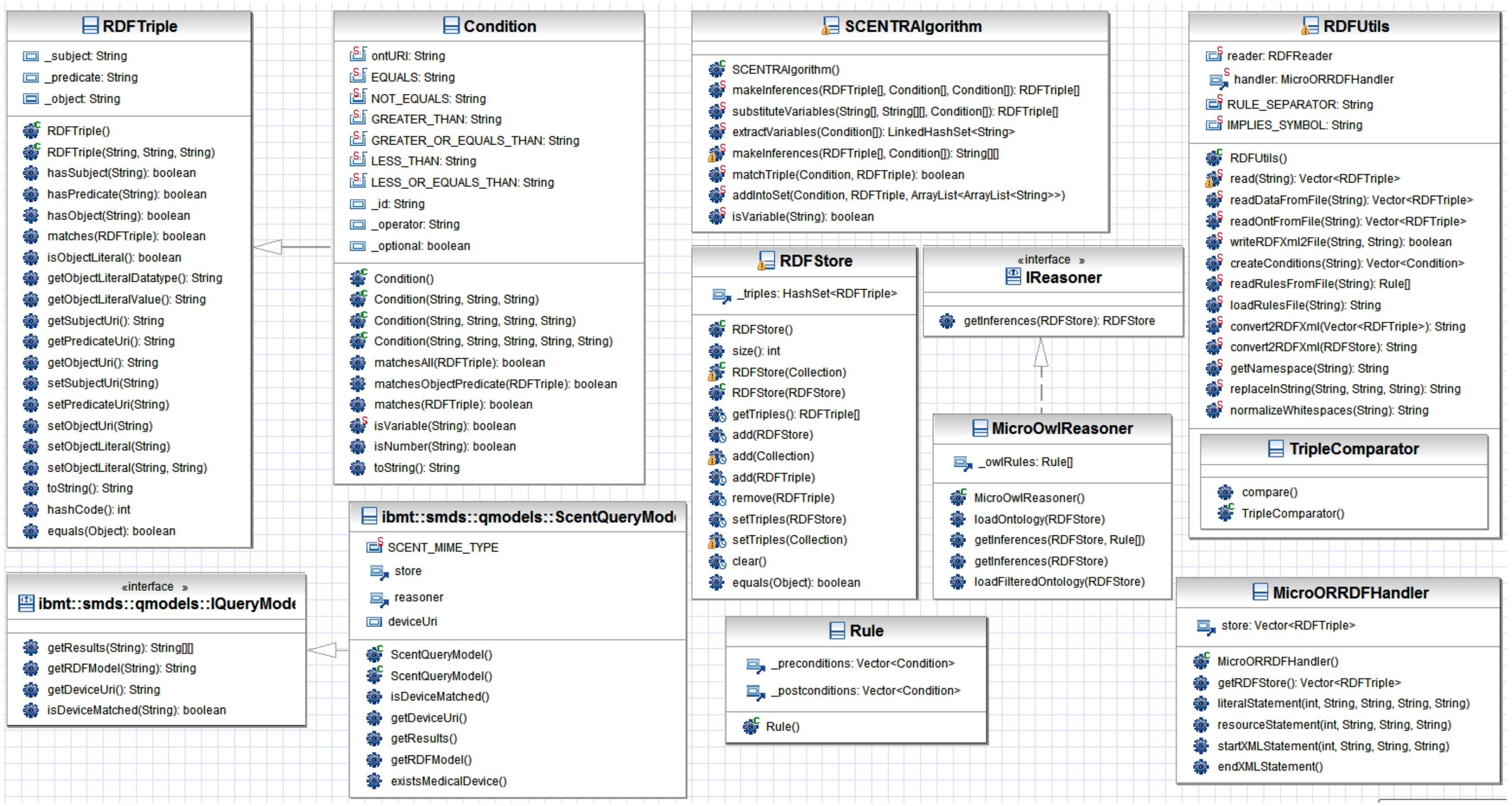

RDFTriple
_subject: String
_predicate: String
_object: String
RDFTriple()
RDFTriple(String, String, String)
hasSubject(String): boolean
hasPredicate(String): boolean
hasObject(String): boolean
matches(RDFTriple): boolean
isObjectLiteral(): boolean
getObjectLiteralDatatype(): String
getObjectLiteralValue(): String
getSubjectUri(): String
getPredicateUri(): String
getObjectUri(): String
setSubjectUri(String)
setPredicateUri(String)
setObjectUri(String)
setObjectLiteral(String)
setObjectLiteral(String, String)
toString(): String
hashCode(): int
equals(Object): boolean
Condition
ontURI: String
EQUALS: String
NOT_EQUALS: String
GREATER_THAN: String
GREATER_OR_EQUALS_THAN: String
LESS_THAN: String
LESS_OR_EQUALS_THAN: String
_id: String
_operator: String
_optional: boolean
Condition()
Condition(String, String, String)
Condition(String, String, String, String)
Condition(String, String, String, String, String)
matchesAll(RDFTriple): boolean
matchesObjectPredicate(RDFTriple): boolean
matches(RDFTriple): boolean
isVariable(String): boolean
isNumber(String): boolean
toString(): String
SCENTRAlgorithm
SCENTRAlgorithm()
makeInferences(RDFTriple[], Condition[], Condition[]): RDFTriple[]
substituteVariables(String[], String[][], Condition[]): RDFTriple[]
extractVariables(Condition[]): LinkedHashSet<String>
makeInferences(RDFTriple[], Condition[]): String[][]
matchTriple(Condition, RDFTriple): boolean
addIntoSet(Condition, RDFTriple, ArrayList<ArrayList<String>>)
isVariable(String): boolean
RDFUtils
reader: RDFReader
handler: MicroORRDFHandler
RULE_SEPARATOR: String
IMPLIES_SYMBOL: String
RDFUtils()
read(String): Vector<RDFTriple>
readDataFromFile(String): Vector<RDFTriple>
readOntFromFile(String): Vector<RDFTriple>
writeRDFXml2File(String, String): boolean
createConditions(String): Vector<Condition>
readRulesFromFile(String): Rule[]
loadRulesFile(String): String
convert2RDFXml(Vector<RDFTriple>): String
convert2RDFXml(RDFStore): String
getNamespace(String): String
replaceInString(String, String, String): String
normalizeWhitespaces(String): String
RDFStore
_triples: HashSet<RDFTriple>
RDFStore()
size(): int
RDFStore(Collection)
RDFStore(RDFStore)
getTriples(): RDFTriple[]
add(RDFStore)
add(Collection)
add(RDFTriple)
remove(RDFTriple)
setTriples(RDFStore)
setTriples(Collection)
clear()
equals(Object): boolean
«interface »
IReasoner
getInferences(RDFStore): RDFStore
MicroOwlReasoner
_owlRules: Rule[]
MicroOwlReasoner()
loadOntology(RDFStore)
getInferences(RDFStore, Rule[])
getInferences(RDFStore)
loadFilteredOntology(RDFStore)
TripleComparator
compare()
TripleComparator()
ibmt::smds::qmodels::ScentQueryMod
SCENT_MIME_TYPE
store
reasoner
deviceUri
ScentQueryModel()
ScentQueryModel()
isDeviceMatched()
getDeviceUri()
getResults()
getRDFModel()
existsMedicalDevice()
«interface »
ibmt::smds::qmodels::IQueryMode
getResults(String): String[][]
getRDFModel(String): String
getDeviceUri(): String
isDeviceMatched(String): boolean
Rule
_preconditions: Vector<Condition>
_postconditions: Vector<Condition>
Rule()
MicroORRDFHandler
store: Vector<RDFTriple>
MicroORRDFHandler()
getRDFStore(): Vector<RDFTriple>
literalStatement(int, String, String, String, String)
resourceStatement(int, String, String, String)
startXMLStatement(int, String, String, String)
endXMLStatement()

Figure A.4: UML class diagram of μOR querying and reasoning system package

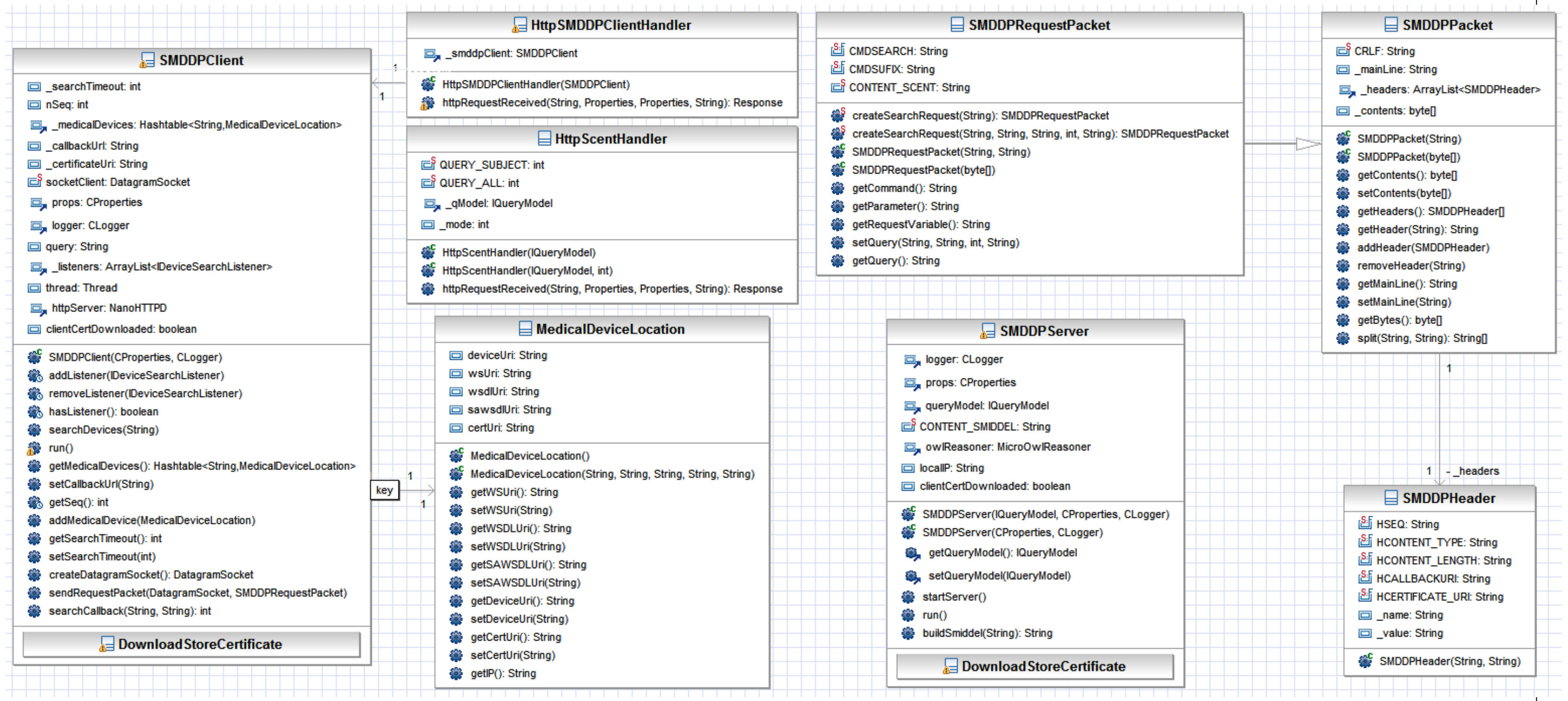

SMDDPClient
_searchTimeout: int
nSeq: int
_medicalDevices: Hashtable<String,MedicalDeviceLocation>
_callbackUrl: String
_certificateUri: String
socketClient: DatagramSocket
props: CProperties
logger: CLogger
query: String
_listeners: ArrayList<IDeviceSearchListener>
thread: Thread
httpServer: NanoHTTPD
clientCertDownloaded: boolean
SMDDPClient(CProperties, CLogger)
addListener(IDeviceSearchListener)
removeListener(IDeviceSearchListener)
hasListener(): boolean
searchDevices(String)
run()
getMedicalDevices(): Hashtable<String,MedicalDeviceLocation>
setCallbackUrl(String)
getSeq(): int
addMedicalDevice(MedicalDeviceLocation)
getSearchTimeout(): int
setSearchTimeout(int)
createDatagramSocket(): DatagramSocket
sendRequestPacket(DatagramSocket, SMDDPRequestPacket)
searchCallback(String, String): int
DownloadStoreCertificate
HttpSMDDPClientHandler
_smddpClient: SMDDPClient
HttpSMDDPClientHandler(SMDDPClient)
httpRequestReceived(String, Properties, Properties, String): Response
1
1
HttpScentHandler
QUERY_SUBJECT: int
QUERY_ALL: int
_qModel: IQueryModel
_mode: int
HttpScentHandler(IQueryModel)
HttpScentHandler(IQueryModel, int)
httpRequestReceived(String, Properties, Properties, String): Response
key
1
1
MedicalDeviceLocation
deviceUri: String
wsUri: String
wsdlUri: String
sawsdlUri: String
certUri: String
MedicalDeviceLocation()
MedicalDeviceLocation(String, String, String, String, String)
getWSUri(): String
setWSUri(String)
getWSDLUri(): String
setWSDLUri(String)
getSAWSDLUri(): String
setSAWSDLUri(String)
getDeviceUri(): String
setDeviceUri(String)
getCertUri(): String
setCertUri(String)
getIP(): String
SMDDPRequestPacket
CMDSEARCH: String
CMDSUFIX: String
CONTENT_SCENT: String
createSearchRequest(String): SMDDPRequestPacket
createSearchRequest(String, String, String, int, String): SMDDPRequestPacket
SMDDPRequestPacket(String, String)
SMDDPRequestPacket(byte[])
getCommand(): String
getParameter(): String
getRequestVariable(): String
setQuery(String, String, int, String)
getQuery(): String
SMDDPServer
logger: CLogger
props: CProperties
queryModel: IQueryModel
CONTENT_SMIDDEL: String
owlReasoner: MicroOwlReasoner
localIP: String
clientCertDownloaded: boolean
SMDDPServer(IQueryModel, CProperties, CLogger)
SMDDPServer(CProperties, CLogger)
getQueryModel(): IQueryModel
setQueryModel(IQueryModel)
startServer()
run()
buildSmiddel(String): String
DownloadStoreCertificate
SMDDPPacket
CRLF: String
_mainLine: String
_headers: ArrayList<SMDDPHeader>
_contents: byte[]
SMDDPPacket(String)
SMDDPPacket(byte[])
getContents(): byte[]
setContents(byte[])
getHeaders(): SMDDPHeader[]
getHeader(String): String
addHeader(SMDDPHeader)
removeHeader(String)
getMainLine(): String
setMainLine(String)
getBytes(): byte[]
split(String, String): String[]
1
1
- _headers
SMDDPHeader
HSEQ: String
HCONTENT_TYPE: String
HCONTENT_LENGTH: String
HCALLBACKURI: String
HCERTIFICATE_URI: String
_name: String
_value: String
SMDDPHeader(String, String)

Figure A.5: UML class diagram of SMDDP semantic discovery protocol package

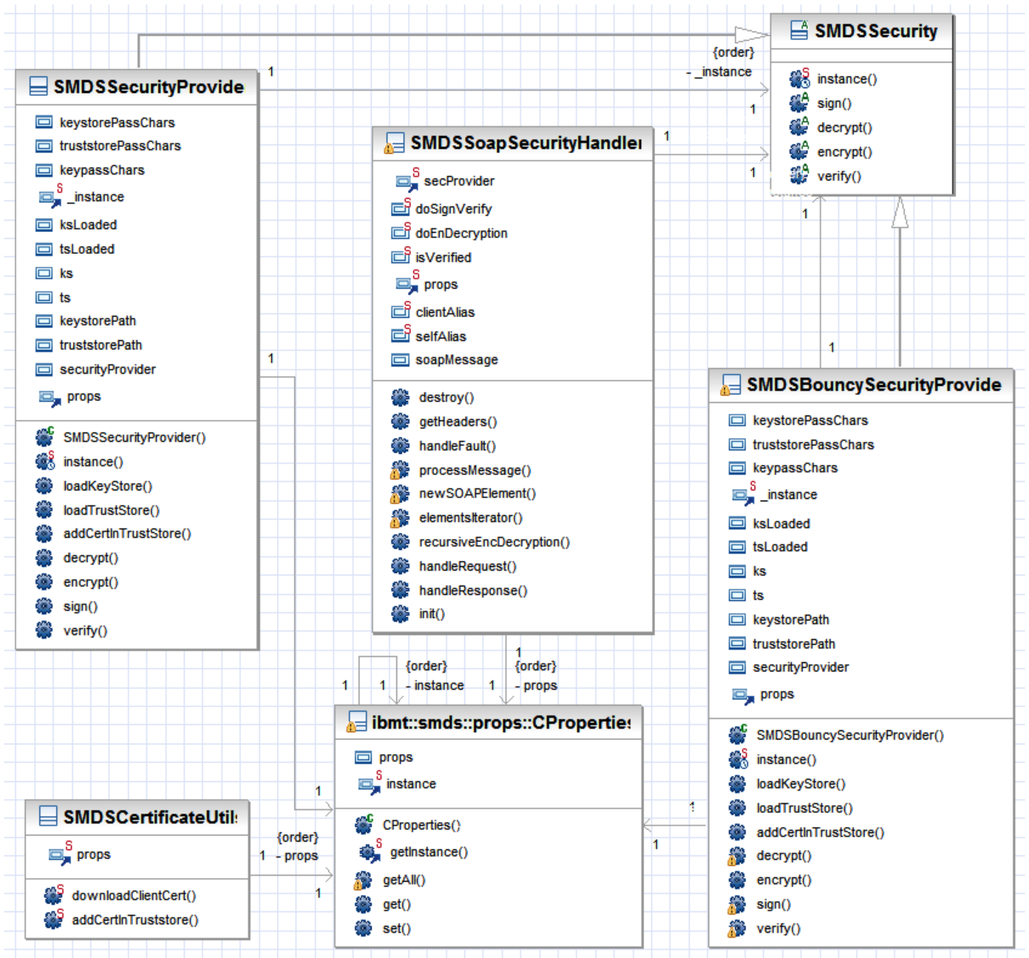


Figure A.6: UML class diagram of PKI based security package

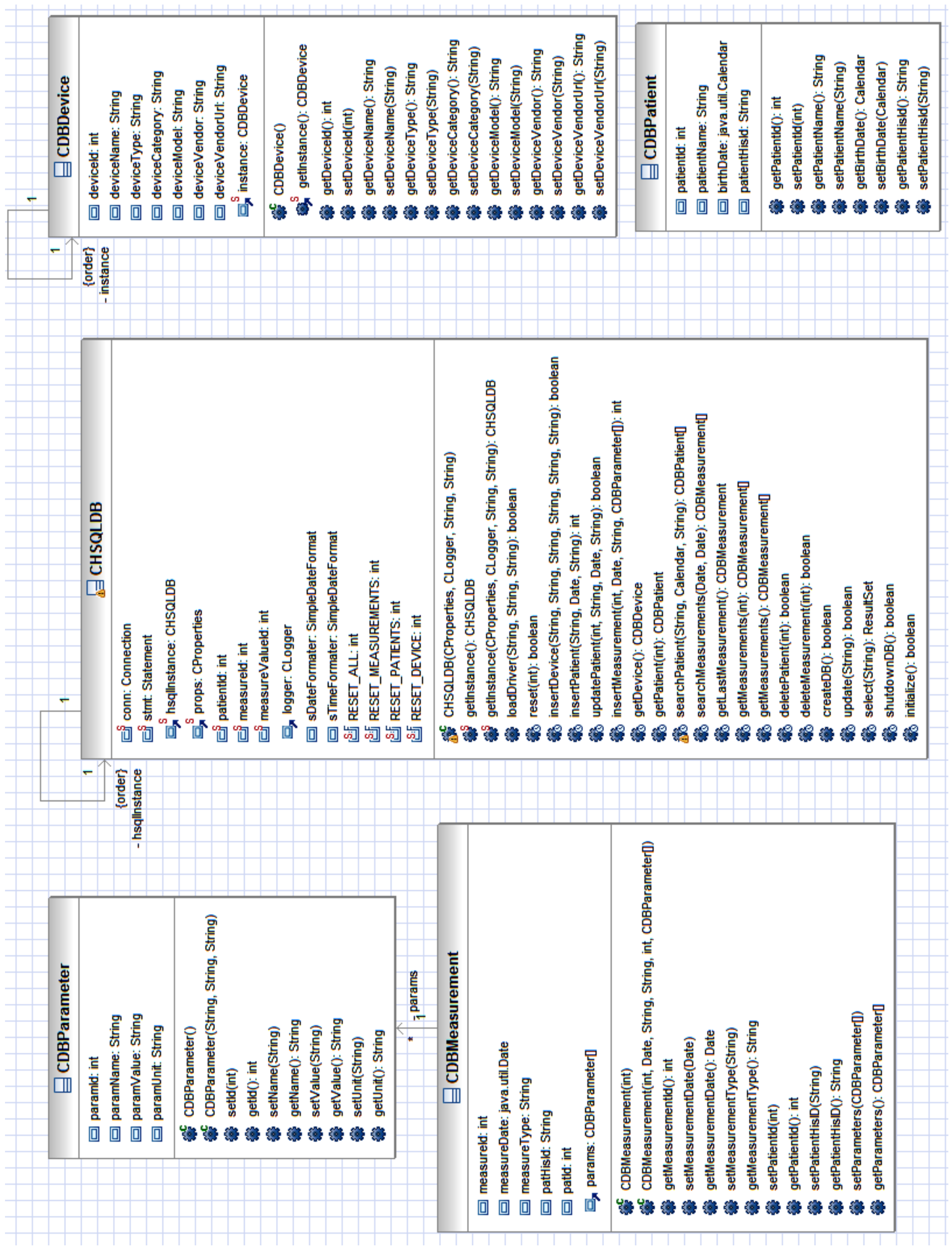


Figure A.7: UML class diagram of medical device database

Appendix B

List of Test Queries

This appendix provides a list of ten different queries in plain textual form, which we used to evaluate the performance of μOR in comparison with other reasoning systems. Although, this list is not exhaustive and can be increased with the combination of each other, but it gave us with sufficient analytical data. The *italicized* words show the *concepts* defined in the respective ontologies.

- Find all the *devices* (medical or non-medical).
- Find all the *medical devices* (of all types of all groups in a hospital/lab).
- Find all the *medical devices* in the *Room xxxx* (the room could be a hospital ward, operation theater etc.).
- Find all the *medical devices* belonging to the *group xxxx* (the group could be urine analysis, blood pressure, weight scale, ECG etc.).
- Find all the *medical devices* from the *vendor xxxx* (the vendor could be Roche, Dräger, BD etc.).
- Find all the *medical devices*, who have performed the *measurement* of *patient* with *id xxxx* (the unique patient id in hospital/laboratory).
- Find the *urine analyzer* having the *measurement* of *patient* with *id xxxx* (the unique patient id in hospital/laboratory).
- Find the *blood coagulation* device *associated* with the *patient xxxx* (the patient's id in home-care/hospital/laboratory).
- Find all the *gateway devices* having *Internet connectivity.*
- Find the *gateway device* having *connectivity* with *SmartHEALTH Information System.*

Appendix C

SCENT Rules Definition

This appendix provides an example of two SCENT rules that we defined *explicitly* using SmdsOnto ontology. Lines 1-2 shows the first rule[1], the Line 3 is the rule separator, while Lines 4-6 shows the second rule.

The first rule has only one pre-condition (Line 1) and one post-condition (Line 2), where the post-condition is always followed by the *implies sign* (->). After the first rule, and before starting the second or further rules, every rule is separated with the rule separator sign "====" (four times *equal* sign). Please note that if there is only one rule in rules.txt, then there is no need to add the rule separator sign at the end. The second rule has two pre-conditions (Lines 4-5) and one post-condition (Line 6).

```
1 ?d <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#hasCompliance>
     <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#SmartHEALTHCompliance> .
2 -> ?d <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#hasCompliance>
   <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#NeuroblastomaCompliance> .
3 ====
4 ?d <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#hasCapability> ?c .
5 ?c <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#mesType> "B. Analyzer" .
6 -> ?d <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#hasCompliance>
   <http://www.ibmt.fhg.de/smds/2008/04/smdsonto#NeuroblastomaCompliance> .
```

Both rules are good examples of showing the use of different productions at the position of *object*, e.g. URI node (Rule 1), *variable* and *literal* (Rule 2). The first rule means that if a device has compliance with the *SmartHEALTHCompliance* (is capable of performing breast/cervical/colorectal cancers analysis), then it is also compliant with *NeuroblastomaCompliance* (is capable of performing Neuroblastoma cancer analysis).

In the current example, both rules have similar post-condition, which means that if we perform a query based on this post-condition on the current knowledge base, the result would be same. So, it does not matter whether both rules exist or only one rule exists, because the result would be same in either case.

[1]Apparent line breaks are due to the space (width) limitation

Appendix D

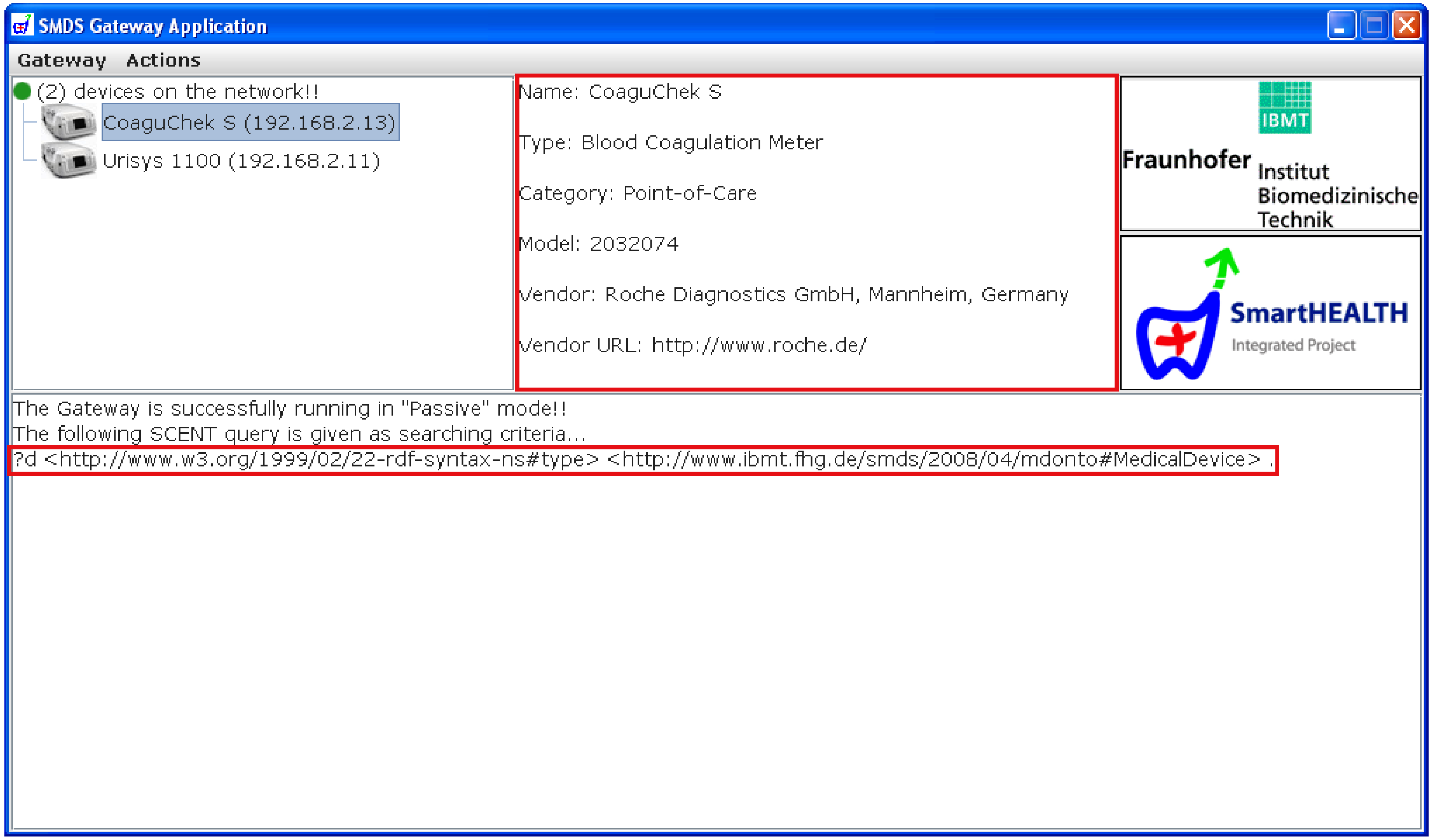
Graphical Gateway Application

This appendix shows the GUI based Gateway application that is developed in Java using Swing. As shown in Fig. D.1, this application broadcasts a SCENT query to find all the medical devices in the environment and it finds two medical devices, namely blood coagulation device and urine analyzer.

When a user clicks on the found medical devices, e.g. blood coagulation device in this case, on the right-hand box, some of the physical characteristics of the corresponding medical device are shown, which were retrieved through Semantic Web Service of that medical device.

Fig. D.2 illustrates the behavior of the gateway application as *active* gateway, where it retrieves the measurement from the selected or all medical devices. Fig. D.3 illustrates by displaying what measurement results it has retrieved from the selected or all medical devices.

Figure D.1: SMDS Gateway application - showing medical devices with the matching criterion



(2) devices on the network!!
CoaguChek S (192.168.2.13)
Urisys 1100 (192.168.2.11)

Name: CoaguChek S

Type: Blood Coagulation Meter

Category: Point-of-Care

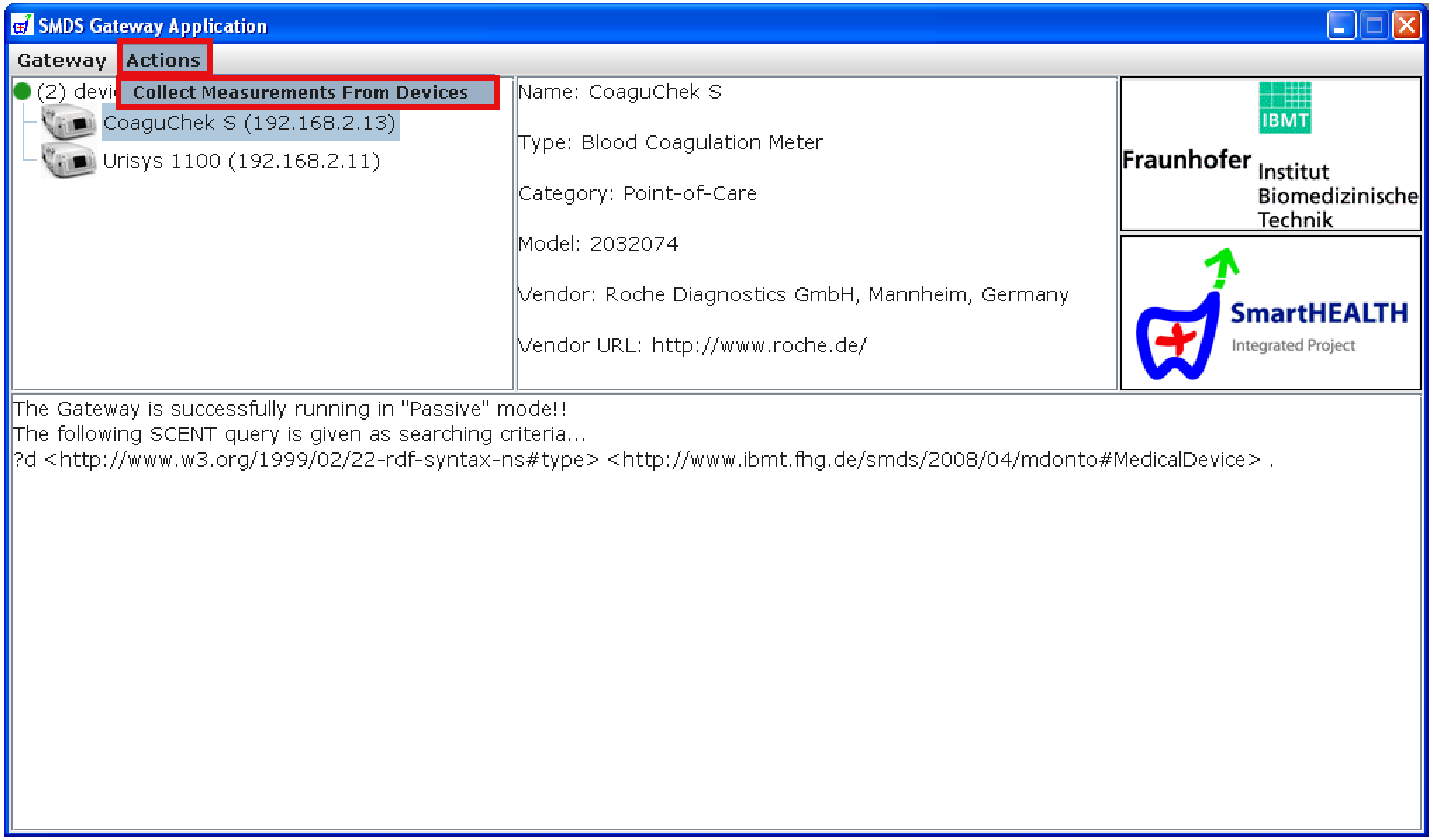
Model: 2032074

Vendor: Roche Diagnostics GmbH, Mannheim, Germany

Vendor URL: http://www.roche.de/

```
The Gateway is successfully running in "Passive" mode!!
The following SCENT query is given as searching criteria...
?d <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.ibmt.fhg.de/smds/2008/04/mdonto#MedicalDevice> .
```

Figure D.2: SMDS Gateway application - retrieving measurement from the selected or all medical devices


SMDS Gateway Application
Gateway
Actions
Collect Measurements From Devices
(2) devi
CoaguChek S (192.168.2.13)
Urisys 1100 (192.168.2.11)
IBMT
Fraunhofer Institut Biomedizinische Technik
SmartHEALTH
Integrated Project

Name: CoaguChek S

Type: Blood Coagulation Meter

Category: Point-of-Care

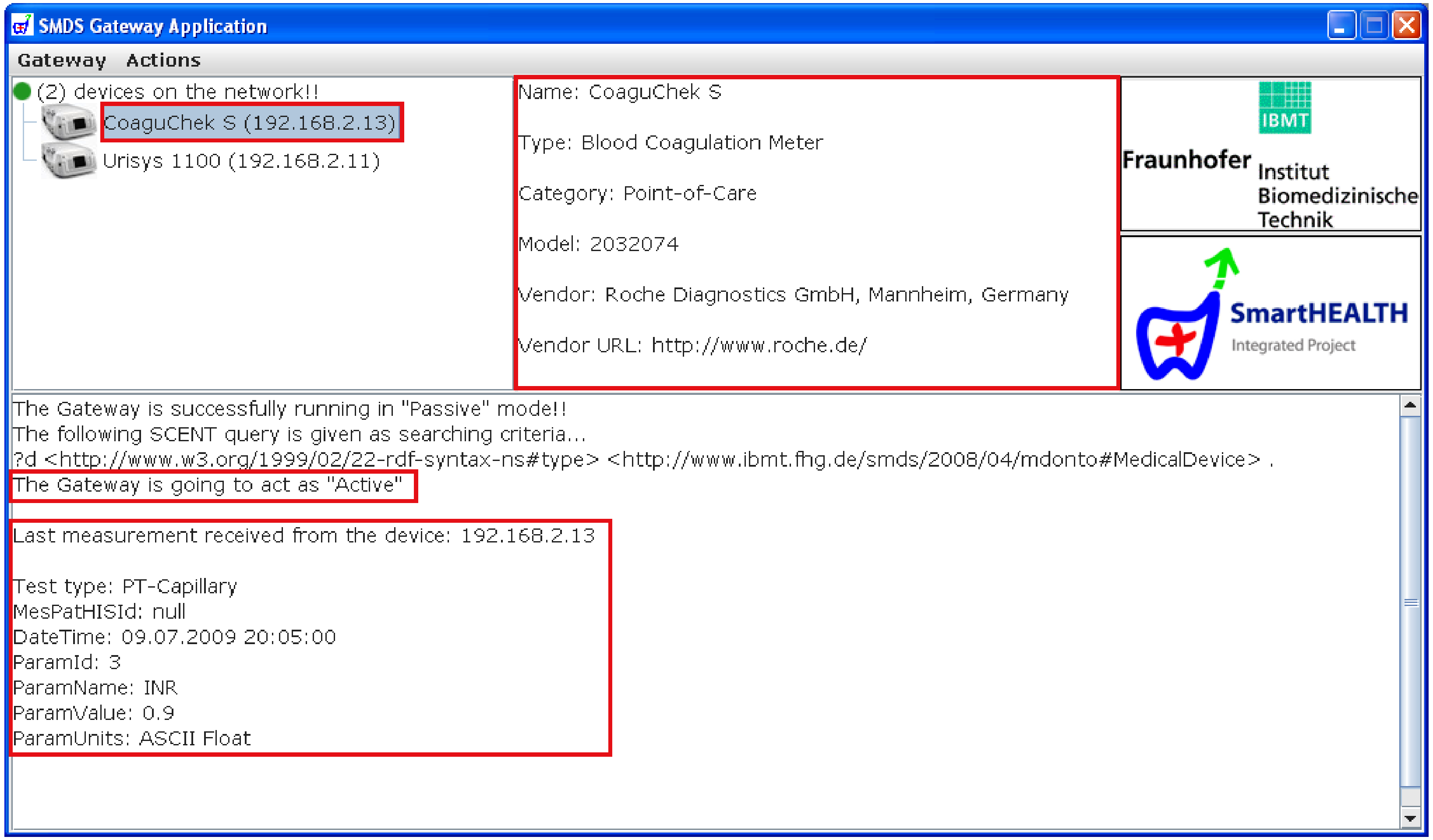
Model: 2032074

Vendor: Roche Diagnostics GmbH, Mannheim, Germany

Vendor URL: http://www.roche.de/

```
The Gateway is successfully running in "Passive" mode!!
The following SCENT query is given as searching criteria...
?d <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.ibmt.fhg.de/smds/2008/04/mdonto#MedicalDevice> .
```

Figure D.3: SMDS Gateway application - displaying measurement results retrieved from the selected medical device


SMDS Gateway Application
Gateway
Actions
(2) devices on the network!!
CoaguChek S (192.168.2.13)
Urisys 1100 (192.168.2.11)
IBMT
Fraunhofer Institut Biomedizinische Technik
SmartHEALTH
Integrated Project

Name: CoaguChek S

Type: Blood Coagulation Meter

Category: Point-of-Care

Model: 2032074

Vendor: Roche Diagnostics GmbH, Mannheim, Germany

Vendor URL: http://www.roche.de/

```
The Gateway is successfully running in "Passive" mode!!
The following SCENT query is given as searching criteria...
?d <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.ibmt.fhg.de/smds/2008/04/mdonto#MedicalDevice> .
The Gateway is going to act as "Active"

Last measurement received from the device: 192.168.2.13

Test type: PT-Capillary
MesPatHISId: null
DateTime: 09.07.2009 20:05:00
ParamId: 3
ParamName: INR
ParamValue: 0.9
ParamUnits: ASCII Float
```

Bibliography

[1] IEEE Standard Computer Dictionary: A compilation of IEEE standard computer glossaries. IEEE Computer Society Press, New York, NY, USA, jan. 1991.

[2] Glass M., Costa L.; IEEE 1073 (MIB) Standardized Connectivity for patient connected devices; Medical Electronics. Pittsbugh, Pa. Measurement & Data Corp., September 1996.

[3] Stephan Pöhlsen et. al; A Concept for a Medical Device Plug-and-Play Architecture based on Web Services; In Joint Workshop on High Confidence Medical Devices, Software and Systems and Medical Device Plug-and-Play Interoperability 2007.

[4] John J. Garduilo, S. Martinez, R. Rivello, M. Cherkaoui. Moving Towards Semantic Interoperability of Medical Devices. In Joint Workshop on High Confidence Medical Devices, Software and Systems and Medical Device Plug-and-Play Interoperability 2007.

[5] EU SAPHIRE Project, Intelligent Healthcare Monitoring based on Semantic Interoperability Platform; `http://www.srdc.metu.edu.tr/webpage/projects/saphire/`

[6] SCALLOPS Project; `www.dfki.de/scallops`

[7] EU CAALYX Project, Complete Ambient Assisted Living Experiment; `http://caalyx.eu/`

[8] EU Amigo Project, Ambient intelligence for the networked home environment; `http://www.hitech-projects.com/euprojects/amigo/`

[9] EU WSAMI Project, A Middleware Infrastructure for Ambient Intelligence based on Web Services; `http://www-rocq.inria.fr/arles/work/wsami.html`

[10] EU InHome Project, An intelligent interactive services environment for assisted living at home; `http://www.ist-inhome.eu/`

[11] IMPRONTA Project, Industrial Manufacturing Platform for Reconfigurable, Agent-Based Production; `http://www.pe.tut.fi/impronta/`

[12] Ivan M. Delamer, J. L. Martinez Lastra, A Peer-to-Peer Discovery Protocol for Semantic Web Services in Industrial Embedded Controllers, 2005.

[13] SOCRADES EU Project; `http://www.socrades.eu/Home/default.html`

[14] SAMIA Research Project, Service-bAsed Monitoring for Industrial Ambients; `http://www.samia-project.info/`

[15] The ITEA SODA Project; `http://www.soda-itea.org/Home/default.html`

[16] Network-centric Middleware for GrOup communication and Resource Sharing across Heterogeneous Embedded Systems; `http://www.ist-more.org`

[17] Sarnovsky, M., Butka, P., Kostelnik, P., Lackova, D.; HYDRA - Network Embedded System Middleware for Ambient Intelligent Devices, In: ICCC' 2007: Proceedings of 8^{th} International Carpathian Control Conference, Strbska Pleso, Slovak Republic, May 24-27 (2007) 611-614

[18] SemProM, Semantic Product Memory Project; `http://www.semprom.org`

[19] Varshney, Upkar; Pervasive Healthcare Computing, EMR/EHR, Wireless and Health Monitoring, ISBN: 978-1-4419-0214-6, 2009

[20] Healthcare Information and Management Systems Society (HIMSS); `http://www.himss.org`, 2007.

[21] Integrated Project SmartHEALTH; `http://www.smarthealthip.com`

[22] Safdar Ali, Stephan Kiefer, Semantic Medical Devices Space: An Infrastructure for the Interoperability of Ambient Intelligent Medical Devices, In Proc. of International IEEE/EMBS Conference on Information Technology in Biomedicine, Greece, 2006

[23] Safdar Ali, Stephan Kiefer, μOR - A Micro OWL DL Reasoner for Ambient Intelligent Devices, In Proc. of 4^{th} International IEEE Conference on Grid and Pervasive Computing, Geneva, Switzerland, Lecture Notes in Computer Science 5529, pp 305-316, 2009

[24] Safdar Ali, Stephan Kiefer, Semantic Coordination of Ambient Intelligent Medical Devices - A Case Study, In Proc. of ACM SIGCHI, IEEE, EMB International Conference on Pervasive Computing Technologies for Healthcare, London, U.K, 2009

[25] Safdar Ali, Stephan Kiefer, Semantic Coordination of Ambient Intelligent Medical Devices in Future Laboratories, MASAUM Journal of Basic and Applied Sciences (MJBAS), Volume 1 Issue 2, September 2009.

[26] Safdar Ali, Stephan Kiefer, Neuroblastoma Screening through Semantic Coordination of Ambient Intelligent Medical Devices, In Proc. of International Workshop on Internet of Things & Services (IoTS), Sophia-Antipolis, France, 2008

[27] Safdar Ali, Stephan Kiefer et. al.; Personal Health Systems for Home Patients after Stroke Rehabilitation Ű Experiences from Pilot Projects, In Proc. of German Congress for the Exhibition, Technologies, Applications and Management for Ambient Assisted Living, Berlin, Germany, 2008

[28] Safdar Ali, Aitor Uribarren et. al.; Applications of Ambient Intelligence in Medical Devices and Clinical Environments, International IEEE Conference on E-Medical Services, Morocco, 2007

[29] Safdar Ali, Stephan Kiefer, Enhancing the interoperability of disparate e-Health Systems through Web Services, In Proc. of International Conference of Medical Physics and Biomedical Engineering (ICMP/BMT), Nuremberg, Germany, 2005

[30] Safdar Ali, Stephan Kiefer, Architecting disparate e-Health systems interoperable through Web Services, In Proc. of International Med-e-Tel Conference, Luxembourg, 2005

[31] L. Cabral et al.; Approaches to Semantic Web Services: An Overview and Comparison; LNCS 3053, pp. 225-239, 2004

[32] ARMUS Corporation, Introduction to Web Services

[33] SOAP 1.2 Part 1, W3C Working Draft; `http://www.w3.org/TR/soap12-part1/`

[34] Web Service Description Language, `http://www.w3.org/TR/wsdl`

[35] Universal Description, Discovery and Integration specifications; `http://www.uddi.org/specification.html`

[36] S. Chan, et. al, Devices Profile for Web Services, MSDN Library, February, 2006, `http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf`

[37] Elmar Zeeb et.al, Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services, 21*st* International Conference on Advanced Information Networking and Applications Workshops, AINAW'07

[38] T. Berners-Lee, J. Hendler and O. Lassila; The Semantic Web, *Scientific American* 284(5):34-43, 2001

[39] Klyne, G., et al. Resource description framework (RDF): Concepts and Abstract Ssyntax, W3C recommendation 2004, `http://www.w3.org/TR/rdf-concepts/`

[40] RDF Schema Description Language (RDF-S); `http://www.w3.org/TR/rdf-schema/`

[41] Web Ontology Language (OWL); `http://www.w3.org/2004/OWL/`

[42] DAML+OIL Ontology Markup Language; `http://www.daml.org`

[43] Matthias Klusch, On Agent-Based Semantic Service Coordination, Cumulative Habilitation Script 2008.

[44] Semantic Web Services; `http://www.daml.org/services/`

[45] Data, Information and Process Integration with Semantic Web Services (DIP); `http://dip.semanticweb.org`

[46] Semantic Web Service Initiative (SWSI); `http://www.swsi.org`

[47] BPEL4WS Consortium. Business Process Execution Language for Web Services; `http://www.ibm.com/developerworks/library/specification/ws-bpel/`

[48] Semantic Annotations for WSDL (SAWSDL); `http://www.w3.org/2002/ws/sawsdl/`

[49] OWL-S Semantic Markup for Web Services; `http://www.w3.org/Submission/OWL-S/`

[50] Fensel, D., Bussler, C. The Web Service Modeling Framework (WSMF). Eletronic Commerce: Research and Applications. Vol. 1. pp. 113-137, 2002

[51] Web Service Semantics (WSDL-S); http://www.w3.org/Submission/WSDL-S/

[52] METEOR-S: Semantic Web Services and Process; http://lsdis.cs.uga.edu/projects/meteor-s/

[53] Semantic Web enabled Web Services (SWWS); http://swws.semanticweb.org/

[54] Web Service Modeling Ontology (WSMO); http://www.wsmo.org/

[55] Blerta Bishaj, Comparison of Discovery Protocols; TKK T-110.5190 Seminar on Internetworking, 2007.

[56] UPnP Specification, http://www.upnp.org/resources/default.asp

[57] Sun Microsystems; Jini Architecture Specification, version 1.2, December 2001. http://www.sun.com/software/jini/specs/jini1.2html/jini-title.html, 09 April 2007.

[58] Choonhwa Lee, Sumi Helal, Protocols for Service Discovery in Dynamic and Mobile Networks. International Journal of Computer Research, 2002.

[59] HL7 Health Level 7 Communicaiton Protocol; http://www.hl7.org/

[60] DICOM Home Page; http://medical.nema.org/

[61] ASTM International; http://www.astm.org

[62] McDonald, C., Laboratory Observation Identifier Names and Codes (LOINC) Users Guide vs. 1.0, Regenstrief Institute, Indianapolis, 1995

[63] Computer-based Patient Record Institute, CPRI-Mail, Vol. 3/1, February 1994, Chicago, IL.

[64] M. Glass, *ANSI/IEEE 1073: Medical Information Bus (MIB)*, Health Informatics Journal, Vol. 4, No. 2, pp. 72-83, SAGE Publications 1998

[65] CEN/ISO/IEEE 11073 Medical Device Communication Standard; http://standards.ieee.org/announcements/pr_ieeep11073.html

[66] Integrating the Healthcare Enterprise (IHE); http://www.ihe.net/

[67] Strategies for harmonization and integration of device-level and enterprise-wide methodologies for communication as applied to HL7-LOINC and ENV 13734; Final document approved by CEN/TC 251 2001-09-18

[68] Coiera, E., Guide to Medical Informatics, The Internet and Telemedicine, Oxford University Press, ISBN 0-412-75710-9.

[69] University of Twente, The Netherlands CTIT, Mobile Telemedicine and eHealth ; http://www.ctit.utwente.nl/research/sro/ehealth/index.html

[70] Maheu, Whitten, Allen, e-Health, Telehealth, and Telemedicine, A Guide to Start-Up and Success. Jossey-Bass A Wiley Company; ISBN 0-7879-4420-3.

[71] E. Igras; Towards the 3rd Wave e-Health Technology Platform, A white paper. January 2003

[72] The ITEA SIRENA Project; `http://www.sirena-itea.org/Sirena/Home.htm`

[73] SODA Technical Framework Description, 2007; `http://www.soda-itea.org/Documents/objects/file1176731057.06`

[74] OASIS Devices Profile for Web Services (DPWS) Version 1.1, 2009; `http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf`

[75] SODA General Ontology for Process Modeling, 2007; `http://www.soda-itea.org/Documents/objects/file1228418088.76`

[76] Ryusuke Masuoka, Yannis Labrou, Bijan Parsia, and Evren Sirin; Ontology-enabled pervasive computing applications, IEEE Intelligent Systems, 18(5): 68-72, September-October 2003.

[77] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003, Angers, France, April 2003.

[78] Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Tim Finin, and Yelena Yesha; A reactive service composition architecture for pervasive computing environments, Technical report, University of Maryland, Baltimore County, March 2002.

[79] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou; Task Computing - The Semantic Web meets Pervasive Computing. In Proceedings of 2^{nd} International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, October 2003.

[80] Zhexuan Song, Ryusuke Masuoka, Jonathan Agre, and Yannis Labrou; Task Computing for ubiquitous multimedia services. In MUM'04: Proceedings of the 3^{rd} international conference on Mobile and ubiquitous multimedia, pages 257-262, New York, USA, 2004. ACM Press.

[81] Zhexuan Song, Yannis Labrou, and Ryusuke Masuoka; Dynamic service discovery and management in Task Computing. In First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04), pages 310-318, 2004.

[82] Manuel Román and Roy H. Campbell; Gaia: Enabling Active Spaces, In Proceedings of the 9^{th} workshop on ACM SIGOPS European workshop, pages 229-234, New York, NY, USA, 2000.

[83] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt; A middleware infrastructure for active spaces; IEEE Pervasive Computing, 1(4): 74-83, 2002.

[84] Anand Ranganathan and Roy H. Campbell; A middleware for context-aware agents in ubiquitous computing environments; In Proceedings of the ACM/IFIP/USENIX International Middleware Conference, 2003.

[85] D. Sacchetti et. al., Seamless Access to Mobile Services for the Mobile User; *Demonstration at the IEEE International Conference on Software Engineering (ASE)*, September 2004

[86] OASIS Web Services Security (WSS) Specification; `http://www.oasis-open.org/committees/wss/`

[87] Peter Haase, Jeen Broekstra, Andreas Eberhart and Raphael Volz. A Comparison of RDF Query Languages. *In Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004.*

[88] W3C Consortium, SPARQL Query Language for RDF, April 2006, W3C Recommendation; `http://www.w3.org/TR/rdf-sparql-query/`

[89] SPARQL Query Engine; `http://sparql.sourceforge.net/`

[90] W3C Consortium, RDF Test Cases, 10 Feb. 2004, W3C Recommendation; `http://www.w3.org/TR/rdf-testcases/#ntriples`

[91] Tim Berners-Lee, *Notation 3: An readable language for data on the Web*, W3C Consortium, March 2006; `http://www.w3.org/DesignIssues/Notation3.html`

[92] JUAN IGNACIO VÁZQUEZ GÓMEZ; A reactive behavioral model for context-aware semantic devices; Ph.D Thesis, 2007.

[93] Jorge Pérez et.al, On the Semantics of SPARQL; `http://web.ing.puc.cl/~jperez/papers/chapter09.pdf`

[94] C. Gutierrez, C. Hurtado and A. Mendelzon, *Foundations of Semantic Web Databases.* In Proceedings of the Twenty-third ACM Symposium on Principles of Database Systems (PODS), pages 95Ű106, 2004.

[95] Forgy, C.L.; Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem; Artificial Intelligence, 19(1982) 17-37

[96] Robert B. Doorenbos. Production Matching for Large Learning Systems. Ph.D Thesis, 1995.

[97] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for expressive description logics, Proceedings of the International Conference on Logic for Programming and Automated Reasoning (LPAR'99), 1999 number 1705 in LNAI, pp. 161Ű180.

[98] OWL Web Ontology Language Overview, W3C Recommendation 2004, `http://www.w3.org/TR/owl-features/`

[99] E. Sirin, B. Parsia et. al., "Pellet: A Practical OWL DL Reasoner", International journal of web semantics, 2007

[100] T. Kleemann, A. Sinner, KRHyper - In Your Pocket, System Description; In proceedings of conference on automated deduction, LNAI 3632, pp. 452-457, 2005.

[101] J. Minsu, J. Sohn; Bossam: An extended rule engine for OWL inferencing, In proceedings of RuleML 2004

[102] I. Horrocks, D. Tsarkov, FaCT++ Description Logic Reasoner: System Description, In proceedings of 3^{rd} international joint conference of automated reasoning 2006.

[103] R. Möller, V. Harrslev; Racer: A Core Inference Engine for the Semantic Web, In proceedings of 2^{nd} international workshop evaluation of ontology based tools, pp.27-36, 2003.

[104] P. Baumgartner, Hyper Tableaux - The Next Generation; Technical Report 32-97, Universität Koblenz-Landau, 1997

[105] Y. Guo, Z. Pan, J. Heflin; LUBM: A Benchmark for OWL Knowledge Base Systems; Journal of Web Semantics 3(2), 2005, pp 158-182

[106] Zhu, F., Mutka, M., and Ni, L., ”Classification of Service Discovery in Pervasive Computing Environments”, *MSU-CSE-02-24*, Michigan State University, East Lansing, 2002

[107] Kozat, U. C. and Tassiulas, L., ”Network Layer Support for Service Discovery in Mobile Ad Hoc networks”, *IEEE INFOCOM 2003*, San Francisco, USA, 2003.

[108] Paul Overell and Dave Crocker, Augmented BNF for Syntax Speficiations: ABNF, October 2005. IETF RFC 4234; `http://www.ietf.org/rfc/rfc4234.txt`

[109] Tim Berners-Lee, Roy T. F, James T, et. al. Hyptertext Transfer Protocol - HTTP/1.1, 1999. IETF RFC 2616; `http://www.ietf.org/rfc/rfc2616.txt`

[110] S. Olson et. al. Support for IPv6 in Session Description Protocol (SDP). RFC3266; `http://rfc.net/rfc3266.html`

[111] W. Keith Edwards, Discovery Systems in Ubiquitous Computing, IEEE Pervasive Computing, 5(2): 70-77, 2006.

[112] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission 21 May 2004;