DOCTORAL THESIS

Context-aware Model Driven Software Agents in Healthcare Information Systems

Perrot, Cedric

Award date:
2013

Thesis submitted in total fulfilment of the requirements of the degree of Doctor of Philosophy

Context-aware Model Driven Software Agents in Healthcare Information Systems

Cedric Perrot

MBA, M.IT (Hons)

SID: 12842964

Degree Program: PhD in Information Technology (IT 33020)
Date of Submission: 5. September 2012

Bond University
Faculty of Business, School of IT
University Drive, Robina QLD 4229, Australia
http://www.bond.edu.au

STATEMENT OF ORIGINALITY

This thesis has not been submitted, in whole or in part, for a degree at this or any other university. To the best of my knowledge this thesis contains no material previously published or written by another person except where due acknowledgement is made in the thesis itself.

Cedric Perrot

Burleigh Waters, 5. September 2012

Publications

Accepted not yet published:

Cedric Perrot, Gavin Finnie, Iain Morrison (2012) “Establishing Context for Software Agents in Pervasive Healthcare Systems”, The 15th International Conference on Network-Based Information Systems (NBiS-2012), Melbourne Australia

Submitted:

Cedric Perrot, Gavin Finnie, Iain Morrison (2012) "Context-aware Model Driven Software Agents in a Healthcare Setting" for consideration for publication in Artificial Intelligence in Medicine. Submitted on 1. July 2012.

Acknowledgments

This thesis, like most, was written by one author, yet it has been influenced by many groups of supporters, friends, colleagues and family.

I hereby would like to thank Professor Gavin Finnie and Professor Iain Morrison for their continued support and guidance throughout my time writing this thesis. Thank you both for giving me the opportunity to conduct this research and sharing your knowledge, experience and insight.

I would like to thank Patrizia my wife for her continued support and giving me extra impulses when I needed them most.

Many thanks to all my friends and colleagues who have helped by reading drafts of this thesis and giving me valuable ideas and vivid discussions while writing this thesis.

Abstract

eHealth has been an area of much change in recent years. New ways of interacting with patients and caregivers are presented every year. Improving the effectiveness of patient treatment and interaction between caregiver and patient is certainly a frequently discussed topic.

Today data is stored in different data repositories such as in the general practitioner's office, hospitals, research institutions and others. Synchronising and coordinating this plethora of information is difficult, making this information generally available equally so.

In today's world we are presented with a numerous sources of contextual information which could be used to improve our everyday lives. In healthcare, the trend is to give patients the possibility to participate in their treatment and that technology is used to minimize disruption in their everyday lives. With the dawn of a new era of technology in sensors and mobile computing devices, user context has become more relevant than ever before. User context can now be used to help in the patient's treatment and improve the outcome.

This thesis concentrates on user context and proposes a new federated approach to improve patient – caregiver interaction, thus enabling the patient to be an active participant in his or her health and wellbeing.

This thesis has focused on creating a new framework and software using different proven technologies, patterns and models, resulting in an amalgamation of best practices engineered to provide a new approach to how user context in eHealth is processed.

The framework created in this thesis facilitates information flow and healthcare decision support in a federated environment allowing making decisions where context information is made available. It further provides a measure of flexibility and transparency in rapid changes to device capability.

This thesis will show that by applying this framework a novel way of executing decisions with the correct context and information is demonstrated. Participants in the healthcare process can seamlessly interact with each other without prior knowledge of the problem domain and the decision making rules. All information in terms of decision making and problem domain information are given to the participant when the participant's context information is needed.

The final proof-of-concept artefact was designed to run on a wide variety of hardware, from mobile devices to high performance servers, all providing healthcare context and information.

For software agents to be useful in helping to guide clinical care and healthy behaviors, they must incorporate or leverage specific details captured via sensors about the patient and his or her environment.

The software agent is built using standard components found in the marketplace. The implementation of the proposed framework provides the glue between these components. The components are open source projects such as Jade Agent, Sesame Semantic Web components, Drools business rules engine and commercial components from Oracle Service Registry to name a few.

The following work will explore healthcare needs for ubiquitous/context aware computing, as well as present a model that will lead to the creation of an agent-oriented piece of software, using Protégé and OWL, that will provide for the context specific needs of a wide range of healthcare stakeholders.

Contents

Table of Figures

List of Abbreviations

OWL	Web Ontology Language
UML	Unified Modeling Language
CASA	Context Aware Software Agent
J2EE	Java 2 Platform, Enterprise Edition
EA	Enterprise Architecture
MDA	Model Driven Architecture
JESS	Java Expert System Shell and Scripting Environment
RuleML	Rule Markup Language
ebXML	Electronic Business using eXtensible Markup Language
RosettaNet	Globally supported standards organization in promoting collaborative commerce
WSRID	Web Service Rule Interface Description
OWL-S	Semantic Markup for Web Services
DAML+OIL	Semantic Markup language for Web resources
BOM	Business Object Model
ECG	Electrocardiogram
eHealth	Health supported by electronic processes and communication also named Health Informatics
BPMN	Business Process Modelling Notation
DS	Design Science
PCMH	Patient-Centered Medical Home
WSFL	Web Service Flow Language

WSDL	Web Service Definition Language
XLang	Extension of WSDL. It provides both the model of an orchestration of services as well as collaboration contracts between orchestrations.
BPEL	Business Process Execution Language
BPML	Business Process Modelling Language
JSR-94	Java(tm) Rule Engine API
DBMS	Database Management System
SRML	Simple Rule Markup Language
RuleML	Rule Markup Language
DCOM	Microsoft's Distributed Component Model
CORBA	Common Object Request Broker Architecture
OMG	Object Management Group’s
XML	eXtensible Markup Language
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
HTTP	Hypertext Transfer Protocol
XSD	XML Schema Definition
JAXR	Java(tm) API for XML Registries
WSCI	Web Service Choreography Interface
ACL	Agent Communication Language
FIPA	IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies
RDF	Resource Description Framework
JADE	Java(tm) Agent Development Framework

SPARQL	Query Language for RDF
XStream	Simple library to serialize objects to XML and back again
LDAP	Lightweight Directory Access Protocol
API	Application Programming Interface
WS-HumanTask	OASIS WS-BPEL Extension for People (BPEL4People)
WS-Coordination	OASIS Web Services Coordination
WS-Transaction	OASIS Web Services Transaction (WS-TX)
JAAS	Java Authentication and Authorization Service
HIPAA	Health Insurance Portability and Accountability Act
CCD	Continuity of Care Document
HL7	Health Level Seven International (non-profit organisation)
CPRS	Computer-based Patient Record System
CCR	Continuity of Care Record
CDA	Clinical Document Architecture
RIM	HL7 Reference Information Model

1 Introduction

In September 1991, Mark Weiser outlined the future development of the miniaturization of computer technology and its integration into everyday objects in his influential article "The Computer for the 21st Century" [1]. Instead of communication through mouse, keyboard and screens we will communicate through pens, watches, furniture, clothes and these objects will communicate amongst each other and with other people's objects [2]. Weiser stamped the term "Ubiquitous computing" referring to devices that invisibly serve people in their everyday lives at home in an unobtrusive capacity, operate in the background at the workplace and work unobtrusively freeing people from tedious work and routine tasks. In its 1999 statement, the European Union's Information Society Technologies Program Advisory Group (ISTAG) introduced a vision where "people will be surrounded by intelligent and intuitive interfaces embedded in everyday objects around us and an environment recognizing and responding to the presence of individuals in an invisible way" [3].

It is this incremental environmental connectedness or awareness that is encompassed within the terms situational or context awareness. We adopt in this thesis the latter term.

It is no stretch to claim that context-aware computing is destined to be the next major breakthrough in computing hardware and software in the world today. Particularly when combined with the ideas of ubiquity that Weiser outlined, context-awareness becomes almost mandatory: After all, what is the value of a computer that is indistinguishable from any other tool if it is not seamlessly intertwined with the environment and needs of the user? As described by Schilit, Adams and Want (1994) in their pivotal early research, context-aware software is that which must adapt according to many different factors, including, but not limited to, "location of use, collection of nearby people, hosts, and accessible devices," as well as changes to these factors over time [4]. While context-aware software can be implemented to serve in a handheld device, it can also be placed in stationary devices and be made aware of situations and other variables.

This situational awareness is where context-driven software can be most valuable in healthcare applications. The idea behind context awareness is that the software can satisfy a

number of functions based on its environment and other data which it receives. Based on changes in the environment, the software will alter its output accordingly. This automatic process of contextual reconfiguration can be informed by a wide variety of sources, but it generally consists of adding or removing components (e.g. sensors) or altering the connections between them (interdependencies in domain model), and changing commands and output based on the information received by these software systems (reasoning component resulting in different actions based on context value).

There are several other important applications for context-aware computing. These include the contextualized delivery of information and alerts e.g. patient receives only relevant information and alerts when being monitored, but can also yield important action with regard to the coordination of various services toward the goal of actionable interpretations of commands delivered from a high level. For instance, in a healthcare setting, a high-level board member might hand down a recommendation or order that could be seamlessly integrated into the rules organization of the hospital or healthcare organization in a way that each stakeholder at each level receives orders tailored to him or her. In addition, this software stands to provide comprehensive capture, representation, communication, gathering, and brokering of user context that is solving large-scale problems at a granular level for each participant in a network. This software also provides a route to resolving differences rife with ambiguity between several participants, finding and rectifying inconsistencies in facts, and for providing an automated body of reasoning consistent with institutional necessity. By tailoring computing and the computing experience to the local level, this system will sharpen computing utility and relevance in many crucial fields.

1.1 Background

The following work will explore ubiquitous computing and context-awareness, as they are related and can be implemented in one of the most important fields: Healthcare. The dissemination of information in the healthcare domain is of crucial interest to many stakeholders in the world today. If this information can reach chronically ill patients in their everyday lives in a more efficient manner, or if doctors' advice and orders can be better spread among the nursing

staff and to families, the patient stands to gain a great deal with regard to his or her own health, wellbeing and peace of mind. In addition, any system which stands to enhance the level of communication and involvement at all levels of healthcare is of enormous importance to the various stakeholders at all levels of the healthcare organization. After all, it is the collaboration of the healthcare team at every level which can ensure a more successful healthcare outcome. If computing could be enhanced to take into account a wide variety of contextual information, and include the patients' data with such a consideration, it is highly likely that the patient will stand to gain a great deal in terms of healthcare outcomes.

Looking at the goal of this thesis and reducing the research goal to one core statement it would be: “How can context-aware computing and model driven architecture along with agent-based technology be combined to enable a distributed approach where distributed context in various systems is made available in a standardized fashion to other participants?”. This proposed approach will design a new application framework enabling the underlying concept of reasoning where context is available. In more detail all reasoning is executed where context is stored until either context is missing and the reasoning is propagated to the place where required context is stored or the reasoning concludes.

In considering the implementation the main goal was to reuse existing software components available in the marketplace (open source or commercial) and to make these components work together. The application frameworks logical components were borrowed from open source projects such as Jade Agent, Sesame Semantic Web components, Drools business rules engine and commercial components from Oracle Service Registry to name a few. The focus was to combine these standard components and to create the glue between these components to enable the core functionality of this framework.

Given these considerations, the following work will explore healthcare needs for ubiquitous/context aware computing, as well as present a framework that will lead to the creation of an reusable agent-oriented software application, using Protégé and OWL to provide for the context-specific model and runtime environment of a wide range of healthcare stakeholders.

1.2 Justification for this research

In the last few years context information has increasingly received more attention. New technological possibilities along with the availability of new sensors have triggered a wide interest into the processing of contextual information [5-7].

Several approaches have focused on sharing context information and on using ontologies to share information amongst the different participants [8-13], some focussing on context middleware [14-18] and others researching how to acquire and describe context as well as how to complete missing contextual information [15, 19-22].

Other concepts such as the execution of business logic (reasoning), software agents, the usage of directory components such as ebXML[23], UDDI [24] and so on, have been researched in relation to context itself or in conjunction with other concepts and approaches[25-27].

This thesis's framework is combining different approaches and concepts to create a novel approach to registering, finding and retrieving context in a distributed environment while processing all business logic where context information is stored. This framework focuses on the usage of decentralised context information. Along with the decentralised storage of context information decision making is also decentralised. Because decision making is also decentralised to the place where context is stored, the synchronisation of context information between participants or synchronisation of context information in a central repository is rendered obsolete in this approach.

Other approaches in context aware computing are either focussing on maintaining context information on a central server using middleware, or fetching the required context information from a known location, while one composes web services into a business process and very few focus on ontologies [28-30].

The approach of this thesis differentiates itself from other approaches in that it contains only context information which is registered in a context repository. There is no business knowledge or any other type of business know-how coded in the framework. All information needed for decision making is modelled during design time as a commonly understood ontology and passed into the framework during execution time for decision making. The major difference

of this approach in comparison to other approaches is that it uses many approaches researched to some extent in isolation and combines them in one comprehensive framework. The advantage of such an approach is that the framework remains completely generic, the only exception is the conversion converting sensor context inside the context adapter data into the right format. However, although not part of this thesis the conversion of context information into required ontology format could be externalised as well using business rules, scripts or similar technologies.

This topic is expanded in more detail in Chapter 2.

1.3 The research problem

Along with the rise of new possibilities in technology, sensors and personal mobility, more information will be available. The abundance of information both in breadth, detail and quality will be difficult to synchronise amongst millions of participants. It can be anticipated that current architecture frameworks will see a limited applicability to these new challenges. As mentioned in the previous section many approaches are synchronising context information or maintaining context information in a central place accessible by all participants. These approaches will no doubt reach their limit of applicability once context information is available from millions of devices and users.

This research showcases an alternate approach to tackle local decision making without the need to synchronise context information but able to execute decision making where context is stored. In addition communication only takes place between the relevant participants in the decision making process and does not need a central coordinator. This new approach alleviates the need to synchronise context information among participants or central repository.

The common understanding of contextual information along with local decision making are aspects that need to be considered heading into these new challenges. New approaches (architectures) are needed to tackle this wealth of decentralised information.

1.4 Contribution of this research

The purpose of this thesis is to define an alternative approach to distributed context and to create an application framework supporting this new approach and then research its utility in addressing weaknesses in other approaches. Best of breed open source and commercial components have been combined to achieve maximum flexibility. The main aspects of flexibility were to keep each individual component self-contained and to communicate through a clearly defined interface with each other and thus remain interchangeable.

This new framework allows looking at distributed context and local decision making in a novel way. Current approaches promote the usage of middleware to synchronise context amongst each participant or to maintain and store context centralised on a server [14-18]. This approach promotes a design in which context is kept locally, business logic is executed locally and only escalates the decision making to the next participant providing necessary context (e.g. General Practioner, Hospital, Research Lab etc.) if context is missing locally. More detailed information is provided in Section 2.4.

1.5 Scope of study

This thesis's framework is solely focussed on architecting a new framework enabling distributed context and distributed reasoning where context is made available. No consideration has been made in terms of transaction management and security, although some aspects of security are mentioned throughout the thesis in relation to this framework.

Other aspects not covered by this framework is the topic of context aggregation as described by Dey [31]. It proceeded on the assumption that contextual information is either aggregated or interpreted using business logic or aggregated to the required granularity when receiving contextual information through sensors or otherwise. The thesis's scope and limitation are explained in more detail in Chapter 2.

This research aims to create an architecture and framework which distributes rules execution based on user context information, and not on service context. In more detail, user context is used to determine whether a participating node (rules engine) is capable of executing the rule set based on the needed information. If all facts are present, then the entire rule set is

executed on that particular node. Should any contextual information be missing, a query is launched against a contextual service repository (which stores information about contexts and services). After retrieving the information about which service is capable of further executing the business rules based on its capabilities to retrieve contextual information, context information (facts) and reactive behaviour (rules) are then propagated to this node for further execution of the business rules.

This scenario is a step away from classically declaring services which execute a predefined functionality, towards asking the infrastructure “Where do I find the necessary information to further process the business logic and complete the business process?" “. This new approach will be implemented using agents using a service oriented approach. These agents will contain all necessary information to process rules based on its context, find other agents with the required context and to be able to invoke other agents to continue the processing of rules based on their own context.

1.6 Thesis outline/layout of the rest of this thesis

This thesis will in Chapter 2 explain different dimensions of the research subject followed by justification and the confirmation of the research gap. More background information on the research topic relevant to this thesis is provided which highlights the different dimensions of this thesis. The chapter finishes with the research question and its limitations.

Chapter 3 will explain the healthcare domain in more detail followed by sections covering different technological and architectural approaches relevant to this research.

Chapter 4 focuses on the aspects of research methodologies and why design science was chosen as the research methodology of choice.

In chapter 5 the methodology is reviewed in conjunction with the selection of individual components used to build the artefact and an explanation why design science methodology was chosen.

Chapter 6 drills deeper into the systems engineering part creating a runnable framework and the evaluation of the use case part of the first step. How ontologies, agent technologies,

service registry and business model information are combined to apply the use case is explained in the same chapter. The framework components are explained in more detail with their functionality and role in the framework in section 6.7.

Chapter 7 will review the research question and conclude with recommendations, suggestions and issues to be resolved in future extensions of the research.

2 Background & Significance

2.1 Research Subject Area

As this research touches on multiple disciplines this following section elaborates several topics. These disciplines encompass technological aspects such as information technology architecture, business models, knowledge as well as context information, the healthcare domain and in particular mobile healthcare in support of telemedicine.

This chapter starts with a background on ubiquitous computing which is particularly relevant to this research as it is a major enabler of telemedicine and of this research. It is then followed by several topics relevant to this research such as the health care domain in general, mobile healthcare and other possible application areas of this research. Other topics looked in more detail are today's enterprise architecture and the shortcomings of current approaches to tackle mobile healthcare. Context information, knowledge and business models are examined as all are relevant building blocks of this research's new approach to decentralise decision making, context information and medical data. One of the core aspects of this research is the model driven architecture approach (MDA)[32] which provides a possibility to share business model information as well as contextual information across several participants in the decision making process.

2.1.1 Background on ubiquitous computing

Modern computing devices are becoming increasingly powerful. The technological advances in mobile computing have led to new challenges and local capabilities which encompass user context such as sensor data, location information and local computing resources such as available display, processing power, memory and storage capacity [33].

With mobile devices becoming more versatile and given the demand of seamless mobile system integration, new approaches are needed for more complex interaction between mobile and server-based software [34, 35].

Currently, processes or parts of processes, such as monitoring of chronically ill patients, which used to operate on server based infrastructure are now being executed in distributed mobile environments. This poses not only technological challenges but one of integration as well, namely how these distributed units of work are being integrated into existing business processes or workflow architectures and how they share information and context [18, 35, 36].

Some of these challenges include how to distribute and share data among participants or actors, in a business process and networked environment. Particular to this framework challenges include the handling of:

- Service & User context (Computing environment, Location etc.) and Instance data
- Business know-how (Rules)
- Business object model

The agents are designed to run as a self-contained work unit or to be integrated into a higher level business process orchestration scenario. The use case in this thesis is focussing on agents communicating with each other. Scenarios in which agents are triggered from business process engines can be integrated and tasks such as delegation, responsibility, processing and so on could be managed in the business process environment.

In the e-commerce domain we have seen a widespread adoption of service oriented architecture (SOA), achieving interoperability on a functional level as well as process synchronization. Further advances have been seen on data models of and information exchange between independently implemented systems [37].

This research is focused on how to use Model Driven Architecture (MDA) [32] and how agent-based technologies are used to communicate between participants to enable the execution of business rules in a distributed context environment. The following work will describe how to define such context to create a mutually understandable data model of context as well as the required functionality and semantics to execute business rules against context which is made available in different locations.

2.1.2 The healthcare domain

The last decade has seen a widespread shift towards consumerism regarding medical and pharmaceutical decisions. While it is still too early to make any strong assessment of this phenomenon, it has been said to be fuelled by scepticism of governmental, corporate, and professional dominance; alongside with unprecedented economic prosperity which has reduced individuals tolerance for interference in individual autonomy [38].

Historically, healthcare was largely government controlled. With today´s shift towards privatisation, patients are becoming better informed and educated. Healthcare is changing from offer driven to demand driven which requires flexibility as regards how healthcare is applied and provisioned. With an aging society, more efficient and cost effective ways to apply healthcare are important issues. In addition, it was recognized that healthcare does not exist in isolation and needs to be integrated in the value chain (e.g., hospital – care centre – general practitioner). This integration in turn requires more transparency and flexibility between the participants within the value chain to effectively integrate them [39].

This change towards consumerism demands a change in how patients are monitored and escorted during the treatment cycles. Alternative scenarios are currently being developed to empower the patients to participate and contribute to their own treatment process and improve the treatment outcome [40].

The increasing prevalence of chronic diseases and the growing ubiquity of technology motivate the consideration of the potential role of information technology (IT) for the greater empowerment of consumers and their advocates in healthcare. Areas in which the consumer may be included to contribute to their own health outcomes in chronic disease management is both adherence to a healthcare plan and the broader topic of constantly being an active partner in care planning [41]. Increasingly healthcare focuses on putting the patients in control of their wellbeing. Patients are encouraged to think critically and to make autonomous and informed decisions [42]. This patient empowerment is further promoted through home and community based care encompassing monitoring and prevention allowing the patients to create informed long-term care choices [43].

The rising costs in the healthcare system have been a major concern for individuals and governments as well as healthcare providers. The sharp increase in healthcare costs has caused the industry to look for treatment alternatives in order to improve cost containment as well as to improve the treatment outcome [44]. Alternative scenarios are currently being developed to be able to empower the patient to participate and contribute within the context of their own treatment process and improve the treatment outcome [40, 43].

2.1.3 Current approaches in healthcare

The healthcare industry today is looking to advance using decision support to achieve better treatment outcomes and to provide a better support in treatment cycles. One such advancement is the Infobutton standard to deal with context-aware knowledge retrieval. This new approach allows clinicians and patients to retrieve information about a specific disease allowing them participate in the treatment cycle. This approach represents a promising approach towards providing relevant clinical knowledge to the point of care to further the advances in patient-centered care. This technological advancement was achieved by providing a set of specifications such as "Knowledge Request Standard" which was approved in September 2010 as a normative ANSI/ISO Health Level Seven (HL7) standard to access knowledge resources. The "Knowledge Request Standard" provides a URL-based implementation using the HTTP protocol to specify knowledge requests with latest amendments of supporting web-services [45]. One such approach is the OpenInfobutton project providing a runtime environment for Infobutton in a Computer-based Patient Record System (CPRS) [46].

Another essential project providing terminology services across different platforms is the project Apelon. This project allows the conversion of text based information into codification schemes such as SNOMED CT, CPT, LOINC, NDC, RXNorm and ICD-9-CM and acts as a single common resource for an organization’s terminology [47].

Other areas in which healthcare has increased standardization is in the field of decision support systems. HL7 is providing Decision Support Service which provides access to the clinical decision support by receiving patient data as input and returning patient-specific conclusions as output. Provisioning and consumption of such services is defined by the HL7 decision support standard interfaces. These interfaces rely on the data being structured according

to the HL7 Clinical Document Architecture (CDA) and Continuity of Care Document (CCD). These services provide structured recommendations regarding the patient's health regime and chronic disease management needs [48]. One such project implementing the HL7 decision support specifications is opencds.org [49] open source implementation. This project OpenCDS uses a standards-based, service-oriented approach [50] to expose its functionality. Components/technologies used in this framework are the previously mentioned terminology provider Apelon as well as JBoss Drools business rules or inferencing engine and JBoss jBPM open-source business process management suite to enable the use of flow diagrams to represent clinical decision logic. The project further supports the mapping of structured data from standards such as CCD, Continuity of Care Record (CCR) [51], CDA release 2, and C32 (a type of Continuity of Care Document) into the HL7 virtual medical record (vMR) structure which is a simplified view onto the HL7 Reference Information Model (RIM) [49].

Another project implementing decision support adhering to the HL7 standards is GELLO. GELLO is a standard expression and query language for decision support. The syntax of the GELLO language can be used with any object-oriented data model, and is based on the Object Constraint Language (OCL). OCL was developed by the Object Management Group (OMG) as a constraint and query language for UML class models. Given that the HL7 Version 3 Reference Information Model (RIM) and associated Refined Message Information Models (R-MIMs) are based on UML, GELLO was designed to leverage the semantics of these HL7 models, in combination with HL7 Vocabulary and Data Types, for clinical decision support [52].

2.1.4 Mobile Healthcare

Technological advances in integrated circuits, wireless communications, low power devices, miniature physiological sensing and intelligent monitoring devices have made it possible to integrate these devices into a Wireless Body Area Network (WBAN). These new technologies are enabling new pathways in monitoring patients from rehabilitation applications to ambulatory monitoring. Some of the challenges which remain to be resolved are areas such as system design, configuration and customization and standardization [53].

These new technologies are enabling early detection of abnormal conditions and prevention of serious consequences. They further facilitate the monitoring of chronic conditions or during supervised recovery from an acute event or surgical procedure [53].

Broens et al. [39] presented their application framework for context-aware mobile applications in health in which they identified context and adaptation mechanisms playing an important role and they noted that further investigation was mandated into how applications exchange context, how context can be acquired and higher level context can be inferred. Another issue identified was how applications can exchange their context with other applications and with the service/communication network infrastructure [39]. Their approach to mobile health is server driven and is based on a m-health back end support system due to the reason that not all technological advances are available on mobile devices (e.g. J2EE) [39]. Today such implementations on mobile devices have become possible by enabling as the main requirement of this framework the usage of the Java reflection API on mobile devices [54] allowing the implementation of agents on mobile devices. This research's application framework implementation heavily relies upon the Java reflection API to enable the core functionality of intercepting calls to and from the context storage.

2.1.5 Domain examples

There are many other areas which could benefit from such context-aware computing: In the retail environment, storage management to effectively store and retrieve merchandise using location information and stocking frequency [55-57]. In private environments, "smart houses" which automatically track consumption of goods, control the intake of nutritional value and use seasonal preferences to propose a shopping list [58, 59]. In healthcare, blood pressure & blood glucose might be monitored and transmitted to a physician with access to all historical information of the patient [60], check whether the patient at home is adhering to the prescribed diet [58] or the location tracking of patients with Alzheimer's disease [61]. The list of context sensitive distributed computing opportunities are numerous, and with these new applications, new challenges will inevitably arise.

2.1.6 Technical enterprise architecture

The technical enterprise architecture (EA) approach relies on focusing mainly on enterprise computing. Enterprises use this architecture to coordinate their business processes with this architecture and functionality. Business services with the required business granularity are created. These business services are then orchestrated, meaning that they are placed in sequence to each other and executed [62-64].

Popular approaches today encompass the creation of SOAP [65] or REST [66] services to expose business functionality as services and then orchestrate these services through another layer of software such as Activiti [67], Intalio [68] for business process orchestration and MuleESB [69] for technical integration of foreign systems and technical service coordination. These orchestration layers provide visual designer tools in which business services are used inside a modelled business process or processes. When a business process is triggered the process calls the modelled business services executing the business functionality they expose and the business services persist the business data to the database as depicted in Figure 1. This service oriented architecture illustrates how business processes are executed through the different layers starting at the orchestration layer coordinating business processes, including and triggering one or several business services. The business services are then persisting the data, typically to the database. Although this SOA has been simplified to illustrate the core components of such an architecture, this enterprise architecture is widely deployed and supported by the software industry. Some of them divide the service layer into several more detailed layers such as a separation of enterprise layer and domain layer and others add technology enablers such as an Enterprise Service Bus to allow the integration of different technologies e.g. Microsoft, SAP, Oracle and so on [70, 71].

In some instances, enterprises have extended their enterprise SOA to integrate other businesses and their services in the overall business orchestration. In this approach, context is shared and made available globally within the enterprise or through clearly defined access methods [72, 73]. This enterprise architecture becomes problematic when there is a shift from process or service driven architecture to a context driven architecture because this context aware computing necessitates an alternate approach which allows for decentralised decision making

and to provide access to decentralised context information and its description (ontology) without the need of central coordination as illustrated in Figure 1.

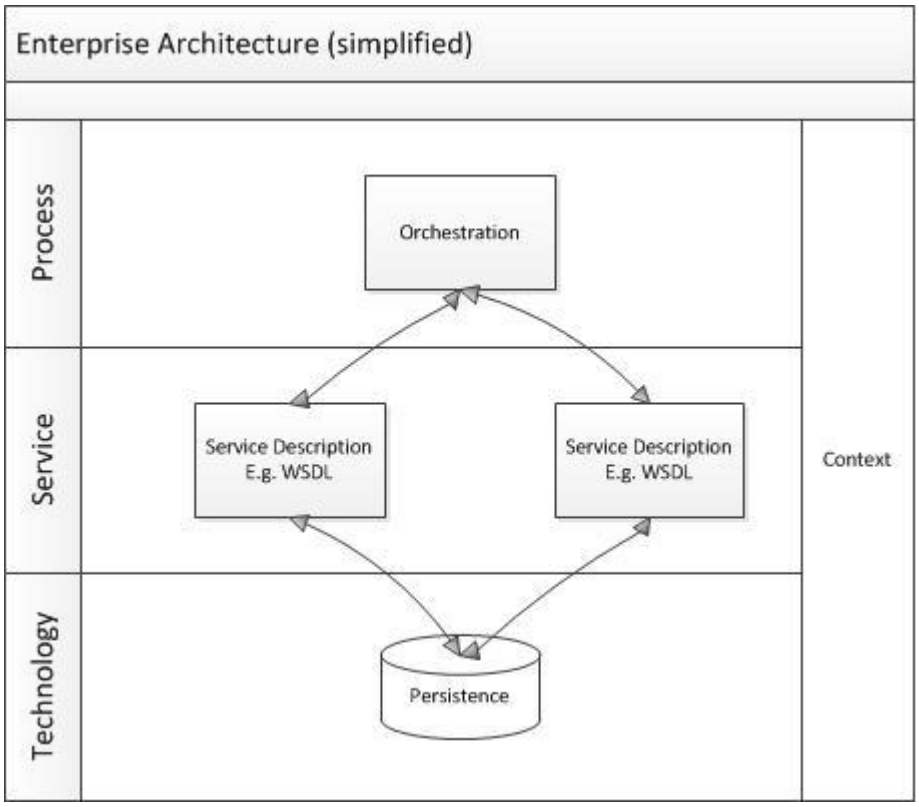

Figure 1 - Service-oriented enterprise architecture (simplified) [72, 73]

2.1.7 Context information

In a world where people move and context becomes available one minute and obsolete the next, context is no longer shared globally but may be stored in devices where events occur and are shared on an as-needed basis; yet solutions are still found within the realm of local reasoning. Such an approach could monitor a heart patient through the sensor readings of an ECG sensor and only escalate an alarm when the readings are getting to a critical level. The decision process of escalating the alarm to the nearest hospital would take place on the patient's mobile phone. Events are fired locally based on sensor readings, and local reasoning takes place to find a solution. In cases where more contextual information is required from other participants, participants will communicate with each other solving the problem without the need of a central coordinator.

Context information encompasses any information about an individual, object, place or time. Context is information about demographics, diseases, medication, allergy and the like, an individual's role, level of formal education, natural language and so on. Therefore any information that is used to characterise an individual, object, place or time for the purpose of this thesis is contextual information. This contextual information can be stored in different places, such as the patient's sensor data on the patient's mobile phone, the patient's medical record at the general practitioner's office and the treatment options of a disease at the hospital or research institute.

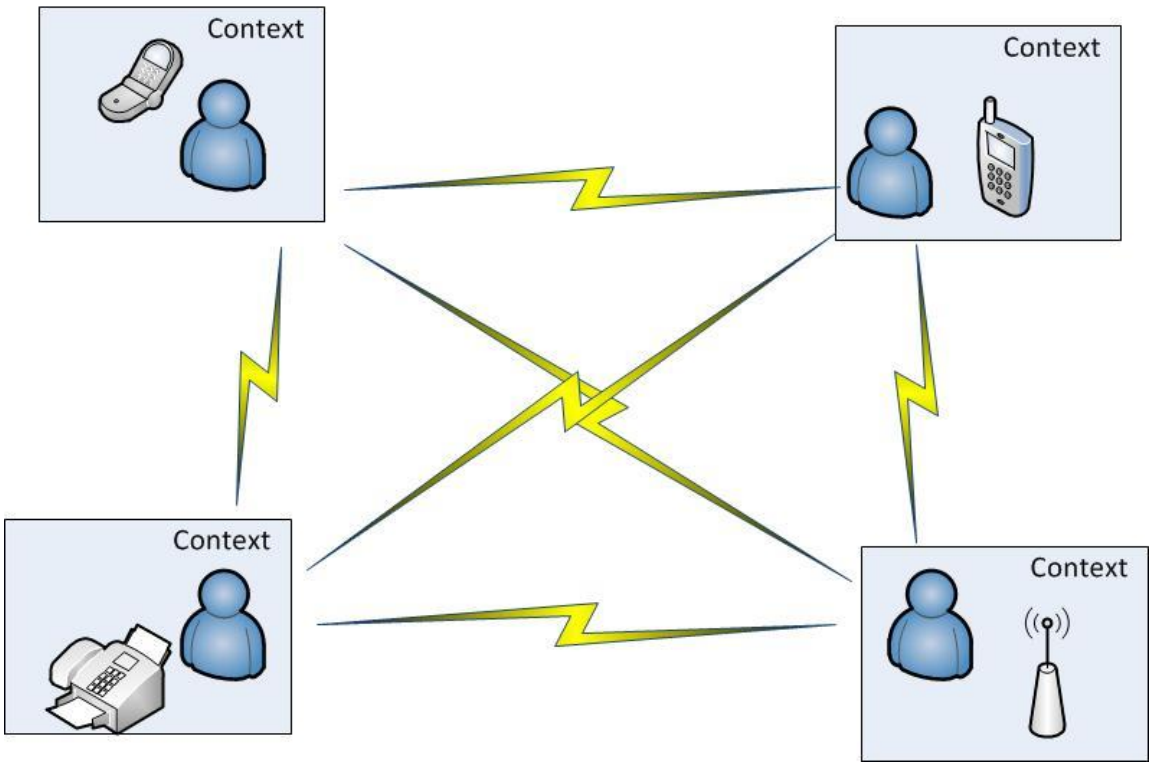

Figure 2 - Mobile user context overview

Due to the nature of context, that is information which occurs in different locations, it is subject to lifecycles. Some context information is valid in perpetuity while other context information is valid only for a short period of time.

Looking at Figure 2 one participant uses the mobile phone and receives a sensor message containing context information, another participant uses a global positioning device while yet another participant is sensing his/her the activity level with a pedometer.

This different contextual information needs to be standardized to enable a mutual understanding of the context and to allow exchange of such context between participants. The data produced by the sensors are sent in different data formats and need interpretation and conversion into one commonly understood data format (ontology). Additionally, some mechanism is needed for storing and retrieving context information. Such operations would most likely overload any existing enterprise architecture today (see Figure 1). Looking at context on a large scale, it becomes apparent that alternate approaches which are different to EA are needed to tackle this scenario.

One such approach could encompass the use of a network oriented and more federated approach to let the participants coordinate themselves, allowing a participant to only interact with actors relevant to itself. Such an approach would help reduce the load on a central coordinator and reduce communication of context information to this central coordinator and reduce synchronisation and management of different context between participants in a transaction.

For this reason, a framework is needed where agents register their available context once for each context type for the lifetime of the context. Such registration allows other agents to find needed context and subsequently enables them to start communicating directly with each other without centralized process coordination.

2.1.8 Knowledge

The world economy has become more knowledge driven and heavily reliant on knowledge and information, particularly in domains such as health, transportation, manufacturing, banking, and insurance. These domains are a few examples highlighting areas which are intensely knowledge driven[74]. The complexity of software is steadily increasing and the focus has begun to shift from specific implementations written in languages such as Java or .NET to frameworks which focus on domain specific knowledge management, to a more general approach. These approaches use ontologies as their common information model, thus sharing technology neutral information and data between participants [36, 75].

Many of the current software applications are still designed in a way that all domain knowledge remains an integral part of the software and lack the ability to change the application

without altering the software. Such classic monolithic approaches to software engineering practices have been supplanted by a segregated approach where technology and knowledge management are determined separately [76, 77].

Knowledge comes in several forms, such as domain knowledge consisting of relations, constraints, concepts and rule-based knowledge. Rules typically represent knowledge about policies, preferences, decisions, recommendations and advice to mention a few types [78]. Today modern Business Rules Engines have seen widespread adoption and acceptance of object oriented programming to pursue a separation of concerns between technology and implementation. Business Rules Engines remain however very limited when used to model real world constraints, concepts and relations [79-81].

This thesis will focus on rule-based knowledge implemented as a set of software-based business rules as well as domain knowledge represented through segregated ontologies.

2.1.9 Business model

Knowledge domains are constantly changing and must be adapted to suit various business requirements and market pressures. Let us assume an ideal world of properly implemented separation of concerns using technological core functionality and business rules. The problem herein is that business rules engines are limited in reflecting business object models, constraints and relations. In such a situation, it is still the responsibility of the application developers to reflect any changes to the domain knowledge, and for the business rules engineer to adapt the rules accordingly. If not tracked properly, any changes made to the domain knowledge can lead to inconsistencies in the business rules. Business objects may change their signature and become unusable for any business rules relying on an old business object signature. When considering the literature on developing software applications with rule-based knowledge we find that all support the separation of having explicit rules and the object oriented core functionality [79-81].

Despite the available choice, we find that rule-based knowledge and object-oriented functionality are heavily intertwined, and the underlying problem of separation of concerns is not solved by any of them. Although, the rule definitions are separated from the workings of object oriented programs, the code still is very dependent on the inner workings of the programs

themselves which makes the rule-based knowledge and object-oriented design extremely dependant on each other. The maintenance of such a systems development is expensive, prone to error and difficult.

The same objective is present for model driven architecture (MDA) approaches: the information model and the business rules ought to be synchronized at the design stage of the development process. Changes in the information model must evoke changes on the business rules constraints.

2.1.10 This approach

With emerging technologies, approaches and concepts numerous approaches can be amalgamated to build an application framework. Separation of concerns is often neglected or not addressed at all in making these different approaches work with each other. This thesis will show that combining Model Driven Architecture (MDA), business rules engines and a generic core application design can be used to achieve maximum flexibility and separation of concerns on multiple levels. Although we use an explicit business rule engine (JESS) [82] and Domain modeller (Protégé 4.x) [83] using OWL these can be easily substituted with alternative applications or generic languages such as UML, RuleML [84] or other proprietary Rules Engines such as Drools [85] and iLog [86].

Several approaches exist which add a description layer on top of Web Services. Substantial research has been conducted in the area of contract management, logistics and supply chain management over Web Services. Two well-known standards supporting these previously mentioned levels of service are ebXML [23] and RosettaNet [87]. These two standards provide a meta-data information layer on top of web services which enables automated web service invocation [23, 87]. Another approach of web services description is presented by [88] which introduces Web Service Rule Interface Description (WSRID) which focuses on describing the business rule-set through the web service interface, and thus allows for a more detailed description of the web service and the underlying rule-set.

Two other standards focusing on Web Service semantics are OWL-S [89]and DAML-OIL [90], its predecessor. These two standards focus on wider description possibilities in order to allow for automated Web Service discovery, composition and invocation.

Multiple approaches have emerged to satisfy the need of describing services in more detail, but one approach has yet to establish itself as a standard. Any standardization would allow for a common approach on how to precisely describe web services which would allow for automated Web Service discovery, composition and invocation.

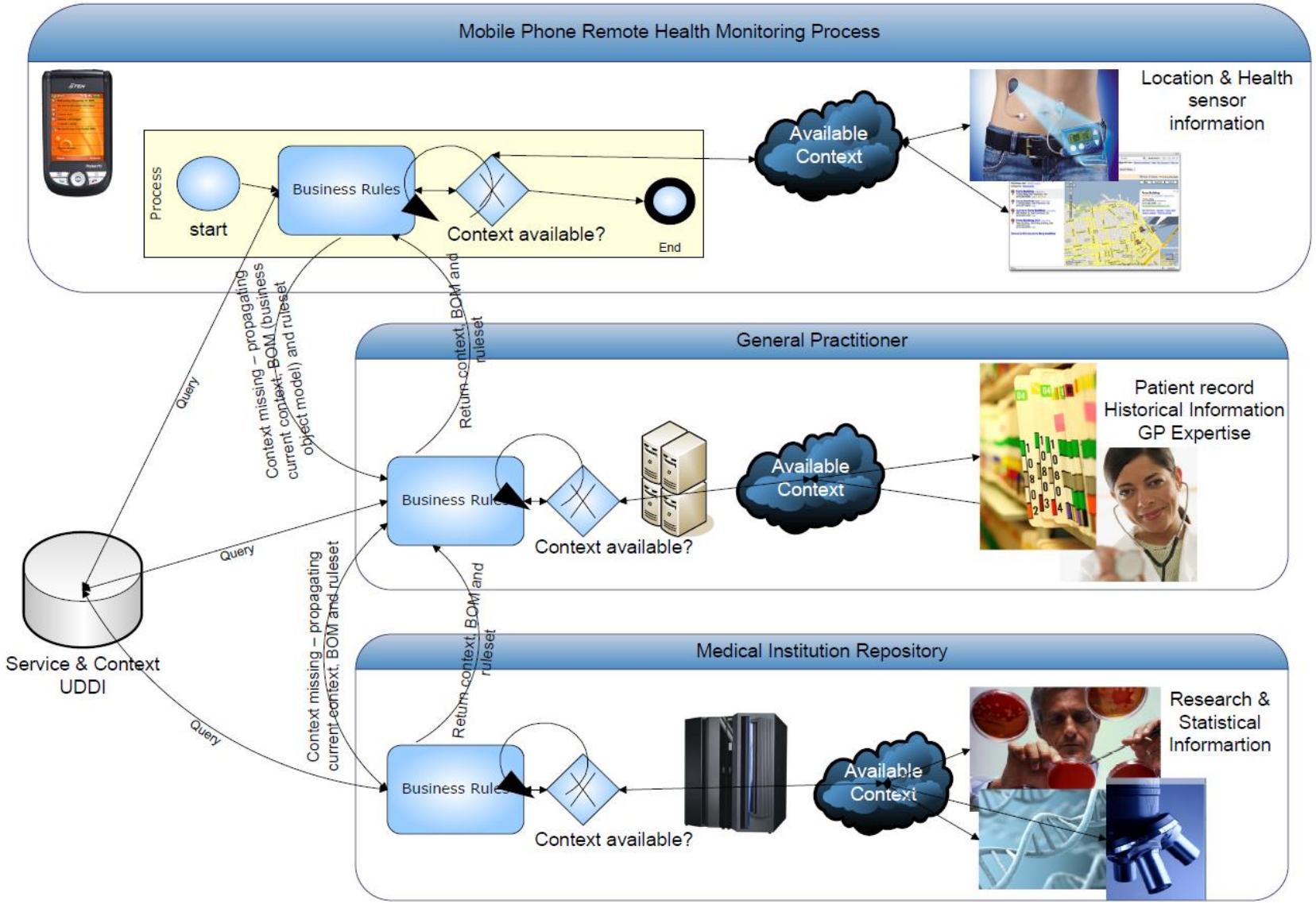

Figure 3 - Context interaction diagram

Figure 3 shows the overall concept of the proposed application framework. Several players are involved when executing business logic. All participants share a common understanding of problem domain described and shared through an ontology. Each participant in that example holds an individual set of context (described in the common ontology) relevant in the execution of the business process. The context of each participant is not shared amongst other participants but made available in the decision making process. In this example the patient's sensor reading triggers an event and starts the business logic on the patient's mobile phone and run against the locally stored context. The business logic or business rules are executed against the context contained in the patient's mobile phone as long as there is context available. Should the execution of the business logic find that context is missing the next participant in the network providing the missing context. In this example the next participant is the GP exposing needed context which is then triggered with the remaining business logic to be executed. In this case the general practitioner is called as second participant and subsequently the medical institution such as research labs or hospitals exposing more needed context needed in the decision making process.

As previously discussed, automated web service discovery relies heavily on an extended web service description capability, and thus one area of complexity emerges on how to encapsulate, describe and find contextual business rules. In addition, there is also the problem of how to modularize business rules functionality to enable distributed business rule execution based on context and available resources and thus enable cascading business rule orchestration across multiple context-aware software agents (CASA) as well as the creation of a suitable architecture enabling this new approach. At present there is no clearly defined standard of business rules semantics and how to distribute business rules across a business architecture. One approach uses RuleML as intermediate language of rule sharing, but no common descriptive language has yet emerged on how to propagate a multitude of information such as business object model, context and rules for distributed rules execution to handle a business interaction such as that shown in Figure 3 [91].

2.2 Background and Justification

In recent years, the concept of context has received more attention, largely due to the increase in mobile technologies and with them, definitions of context [5]. Some mobile applications focus on providing services based on location, helping end consumers to find restaurants, hotels and so on. Other applications use a combination of sensor information and other information such as user preferences and peer recommendations to increase the relevance and usefulness of user and application interaction [5-7].

There are several proponents of context-awareness focusing on modelling context and on finding a common language to define and use contextual information [15, 19-22]. Some focus on context acquisition and efforts to complete contextual information 'pictures' where parts of the information is missing [92, 93]. Still others will use such context as a framework for creation, as toolkits and/or middleware to enable context sharing amongst multiple participants [7, 14-18, 94, 95].

A significant amount of research has been conducted considering issues such as Business Rules, Context, Processes and agent-based architectures in isolation. This research will focus on integrating these technologies as appropriate in our research scope, in a new way of distributing business rules amongst multiple context-aware software agents (CASA) and propagating the decision making to the next context aware software agent which will have the necessary context available. A context-aware software agent is a service provider which in contrast to classically exposing business functionality, only exposes the available contextual information it contains. Propagating information among these stakeholders is necessary to find other agents storing required contextual information and invoke further processing of context dependant business rules.

This research's focus in this new approach is firstly on what information will be shared among the CASA's in terms of context, business object model (BOM) data and rules, as well as the specific information that needs to be passed on from one CASA to another to enable a distributed rules processing approach. The second challenge is how to design function granularity for process atomicity and clean service composition integration. The design of the right granularity is important to both internal execution and external integration with other

agents. Internally the framework depends on business rules to run as a self-contained logic work unit independent from other business rules. Each business rule file (consisting of business rules logically belonging to each other) should remain without any dependency to other business rule files. This allows the tagging of executed business rules and maintaining a list of business rules still to be executed. For example, individual blood pressure measurements ought to be separated from business rules checking the blood pressure history. As these business rules access different context and medical information they are to be separated into individual business rules files.

In other words, every agent focuses on context information only to make gathered contextual information available in a central repository. All other information will be made available during runtime. Hence, agents are stateless in terms of business rules and business object models and only hold stateful information about contextual information they receive.

2.3 Confirmation of Research Gap

Jovanov's project [53] aiming at using intelligent devices and using local processing power to monitor patients is one of many examples showing how health monitoring is being applied in new ways. Jovanov further states that some issues remain to be resolved, which are relevant to this thesis: "system design", "configuration and customization" and "standardization" [53]. The framework of this thesis is providing a new alternative approach in terms of system design, to a large extent providing a solution to customize and configure and proposing the use of ontologies for a common understanding thus enforcing standardization during design time and runtime of the application.

As mentioned previously Broens et.al [39] stated that it is necessary to further research how applications exchange context and how context can be acquired as well as a second challenge of how to handle the volumes of health related data [39].

Identified Gaps	**Solution**
System design	Creation of unified framework.
Standardization	Using a common ontology language to describe the problem domain.
Seamless integration	Embedding framework in technology which allows integration into numerous platforms.
Configuration and customization	Through usage of common ontology language to describe the business domain and with context information along with decision making rules.
Context exchange	Central repository of relationships between related parties in a health care process and the location of the context.

This thesis will address the above issues and provide an alternate scenario showing how to implement patient mobility in a decentralized approach and how contextual information is defined and exchanged. This thesis presents a solution moving away from classical back-end systems by using agent based software which can reason about context and business objects as long as the context is locally available. Should context be unavailable, all the necessary information is extracted and sent to the next software agent providing required context information. It further provides an alternate approach for local decision making using locally stored context to reduce the footprint of regular data transmission to a back-end system.

2.4 Relevance of This Research

2.4.1 Thesis contribution

The expected benefits from this research are an increased understanding of context and its role in application design of distributed systems interacting with user based context as well as an architectural design that allows for a complete declarative approach.

The main goal is to provide a flexible framework which allows using best of breed components (open-source or commercial) available in the marketplace. All software components used in this thesis have been selected due to their ease of use and integration possibility with other components. The design of this framework has been chosen so that every component remains exchangeable. Therefore, any other component fulfilling the same functionality could be used instead e.g. instead of using JBoss's Drools [85] as the reasoning engine it could be substituted by iLog [86] or OWL reasoning engine [96].

Software agent technologies will be used as a communication layer/middleware to be integrated between components such as higher level business process designers, business rules engines and other agents. A use case in the healthcare domain, more specifically targeted at patient centered healthcare delivery, will showcase the application of this newly created framework.

2.4.2 Contribution

The expected contributions of this thesis are:

- Identification of a novel design process for building model driven context aware agents.
- Identification of requirements to support the creation of model- driven context-aware agents, resulting in a conceptual framework that decreases technical complexity and increases the ability of software designers' to build sophisticated applications without coding.
- Building a model- driven context- aware agent framework that supports the above requirements and design process.
- Building a healthcare use case which covers a large portion of the model driven context-aware agent framework to showcase the use, integration and deployment of such a framework.

2.4.3 Benefits

There are several benefits of this research which we can anticipate at this stage. These include several different elements, not limited to

1. a new approach to distributed computing, informed by this new model of distribution and the graphical user interface (GUI) which will control what the user sees;
2. a generic approach to different domains, allowing for rapid changes in different business computing domains without the programming that is so often necessary to compensate for such changes;
3. the achievement of a flexible and extendable approach (with context at its core). Such a flexible approach is beneficial because this will allow for the mitigation of a particularly vulnerable process--the transfer of sensitive contextual information--and replace this step with local-only execution of all contextually-sensitive information.
4. Finally, this research will allow researchers to build upon proven technologies and thus extend current functionality into future domains with an eye toward future business computing needs.

The main goal of this research is to keep the theoretical framework very generic and not specific to any one domain. This focus on broad use will allow for the application of this framework to various problem domains e.g. EHealth, Emergency-Management, Military applications, Retail-Management, Smart Houses and many more.

2.4.4 Deliverables

Upon the completion of this project, the following will be delivered:

1. Concrete and complete, provable design of a specific distributed context-centric model as well as related context driven software agent architecture;
2. A specific and complete design and implementation of a verifiable and repeatable framework which serves to support this new approach to context-driven computing design;
3. A framework which serves to describe the implementation of a given process for chronic disease management, such as an eHealth process, and
4. An evaluation with conclusions which consider a use case which exists in the domain of healthcare and modern computing. This last element will include implementation analysis which has an eye focused on the experience of the end-user, and will represent a departure from the descriptions of the computing and context-awareness which will be a staple of the other project deliverables.

The first deliverable will consist of describing the architecture of a distributed context-centric model and context driven software agent. An integral part of this step will consist of finding and retrieving contextual information, as well as describing such information. The steps involved in the process of finding and retrieving contextual information will be described in turn, with the intention of being repeatable to future programmers and experimenters, as well as experienced users in the field. Additionally, this first deliverable will elaborate on how these software agents can be integrated into an existing or new business process using new approaches such as BPMN.

In order to standardize these methods, users must share a language in which contextual information can be described and interpreted. For this reason, the second deliverable will offer a new design and implementation of a verifiable and repeatable framework that will assist in the

repeatability of this work. The final deliverable implements an eHealth process using the previously created framework to prove the validity and applicability of the framework into the healthcare domain. This step will include a working prototype including an end-user interface displaying results and alerts.

This research due to its goal of building a prototype in the health domain will adopt Nunamaker's systems development research methodology. This methodology consists of three main steps:

- Concept Development
- System Building
- System Evaluation

2.5 Broad Research Question

As mentioned in section 2.3 and 2.4 the framework created in this thesis addresses the issues of "system design", "configuration and customization", "seamless integration" "standardization" and "context exchange". This is attempted through merging different approaches and best practices into one comprehensive framework. The framework will consist of several components addressing individual functionality needed when executing. The most obvious components are the decision support and the context locater component. More research is needed to dissect the framework into individual reusable components. Given the above assumptions, most components of this framework are to remain problem domain neutral and need to be dynamically configured and/or parameterised for this framework to work.

Primary Research Issue: How can Context-aware Computing, Model Driven Architecture and Agent- based technologies help to improve healthcare outcomes?

2.5.1 Problem statement

More and more information will be available in the future and new architectures and approaches are needed to deal with this oncoming plethora of contextual data, available at a greater depth and breadth of quality than ever before. As less time will be spent interpreting and

condensing information, it will be more necessary to make sense of contextual information, interrelating and drawing conclusions from such information, thus enabling the full exploitation of contextual information between people and organizations at a previously unseen scale. Information will include much more depth and breadth to support decision making e.g. non-invasive ECG sensors to monitor heart patients, motion sensors to determine whether the patient has collapsed and many more applications.

New sensors will be able to send their results much more frequently in a previously unseen quality. Context will be rich in information drawn from different sources to allow the best possible decision making. Particularly in the mobile environment where people move and interact, contextual information will become key to being able to provide individuals with relevant and precise information and decision making. New sensors will be necessary to send their results much more frequently in a greater quality in depth and breadth.

2.5.2 Research questions

The research questions are:

"What information and components of the framework are necessary to be able to fully support a design oriented context enabled software agent?"

"***What information is needed for the framework to describe, find and store context information?***"

"Which components of a software agent must be fully dynamic in terms of model driven design to be able to integrate them in the framework?"

2.5.3 Limitations

This research will not cover areas such as transaction management and security, including authorization and authentication as well as data transmission encryption and security, nor Capability Management and IT Governance.

With respect of context, the topic of context aggregation is not covered. This work assumes that all defined context information is available at the depth required and does not need any further aggregation and interpretation.

Due to diverging implementations between business rules engines this research will focus on one business rules implementation. The business rules engine of choice is JESS [82].

3 Key Concepts and Related Information

3.1 Introduction

Design science is essentially divided into two main steps with the first involving the conceptual creation of the artefact to reflect testing of the research objectives anchored in design goals. The second step involves systems engineering and the selection of tools to create the artefact and the evaluation of the artefact's performance in the use context.

In this chapter several areas essential to the conceptual creation of the artefact are studied in more detail. These areas include healthcare aspects and changes in the healthcare domain, in particular with telemedicine which provides an approach of delivering health care services at home. The use case in this research focuses on monitoring a patient's blood pressure in home care.

Other aspects encompass business processes along with business rules, the usage of services, the role of context along with the application of model driven architecture (MDA) using the semantic web.

This chapter will begin with a brief consideration of the features of the framework that can be used by application developers and business analysts in constructing a context-centric software model. Dey et al [31]. defined a set of features which will be modified to consider context as a factor which is both acquired during runtime and a decision-making process which decides what decision will be triggered during runtime. In addition to these concepts, the chapter will also consider the concept of the common model language, a concept that can be supported by all agents, as well as the role that common language will play in both contextualizing and instantiating contextual information. This chapter will also show that, when moving forward to the final software product, the framework base must be stored in a repository which is capable of retrieving and aggregating this content.

This chapter will end with a consideration of the primary contextual framework as used for the software that is created as the end result of this work. Using factors such as agent, context relation storage, and model/context information repository, this contextual framework will be

presented in a straightforward manner. This will allow for 'context' to be explained in greater depth that will be useful in advance of the following chapter which will consider the software, and aspects of that software in more detail.

The following section will explore the key concepts that form the majority of the work in this thesis. As the focus of this work is the healthcare domain, the topics under consideration in this section will begin with a thorough discussion of the current problems with regard to healthcare in Australia and around the world, as well as the issues that can be traced to a lack of technological innovation with regard to comprehensive computing involving context and the problems that have arisen as a result. With that context established, this work will consider various alternatives and solutions that have been proposed in recent years that would allow for at least partial mitigation of many of these problems and issues and address the gaps mentioned previously in section 2.3.

Following the consideration of the care domain and the computing problems therein, as well as the potential solutions that have been floated in recent years, this section will expand into a more developed consideration of business rules, particularly as they relate to healthcare business practices, as well as a more thorough consideration of ontology. This consideration will explore various business rules, and explore the ways in which these can be best integrated into existing software and programming architectures in a way that can lead to the more seamless integration of existing modes of understanding the vast breadth of available medical information.

This discussion of business rules, particularly those of healthcare institutions, will then transition into a consideration of context itself. In this discussion, this work will consider context in a categorical sense, as well as consider model-driven architecture (MDA), the potential uses of MDA, and a unified modelling language (UML). This section will also introduce the OWL language of Web Ontology; as well as explore the ways in which context in programming can be best expressed through use of OWL in the healthcare domain.

3.2 Approach to fill the Research Gap

The newly created framework of this thesis touches on multiple disciplines to create a useable prototype. Several trends, best practices, standards and technologies are investigated in

order to fill the individual components required by this framework with a component capable of delivering the required functionality. One of the main goals set out was to create a framework which could easily exchange individual components in its design. Another goal was to reuse existing components and use the maximum capability of each of these components and to provide the glue between these components in this framework.

Identified Gaps	**Investigated area or discipline to fill the research gap**
System design	Usage and adaptation of other middleware and framework approaches in context awareness.
Standardization	Investigated trends, best practices and standards in healthcare and information technology
Seamless integration	Web Services, Agent Technologies, Business processes (BPMN, BPEL), Distributed computing (CORBA, DCOM)
Configuration and customization	Semantic Web, Business rules, Model driven architecture, Generic services
Context exchange	Semantic Web Ontology, UML

3.3 Grounding in Discipline Knowledge

3.3.1 Health domain

Historically, medical technology innovation has been blamed for rising costs of healthcare and is considered one of the culprits of being unable to contain costs in the healthcare system [97]. It is clear to many sources that the current tools for cataloguing and consolidating information in the healthcare domain are as yet insufficient, both in scale and scope, to handle the vast amount of sensitive and critical data used in a healthcare setting [98-100]. This section will showcase some of the current problems in healthcare computing, as well as identify and evaluate several different tools and services that could be of definite use to healthcare professionals at present and in the future.

The ideas and concepts that will be introduced in this section are crucial for several different reasons: First, communication between healthcare professionals represents a large part of the day-to-day activity of these important professions. Whether direct or indirect, these elements of communication form an enormous part of their work, as well as being key to patient care. These communications range from laboratory results to complex consultation and advice, and are important and useful but at the same time are plagued by a variety of interruptions. These interruptions stand to be mitigated by software communications solutions that this section will elucidate. That said, these interruptions and complications are crucial to consider for the simple reason of medical errors. While such errors are often uncomplicated and easily remedied, they also stand a chance of harming or killing a patient. It is for this reason that all potential gaps in communication at all levels of the medical and healthcare field must be mitigated: Even if an error due to a gap or lapse in communication only leads to one error in a thousand, these are errors with potentially life-threatening implications.

When cooperation between healthcare professionals is mediated through computerized platforms, such as hospital-GP intermediation platforms or homecare coordination platforms, as well as mobile devices and other tools, there can be enhanced cooperation at all levels of the healthcare domain. Due to new tools and software, some of which will be explored in this section and throughout this greater work, healthcare professionals will be able to better manage their tasks and thus decrease the potential for error while increasing the quality of patient care.

Despite these lofty aspirations, current efforts which have aimed at increasing integration and communication between different levels of professional in the healthcare domain have come under increasing levels of scrutiny from healthcare system members mainly concerned with the bottom line and thus likely to provide only profitable medical services [101, 102]. New and expensive devices are not always the most cost-effective solutions at these facilities, and despite the vastly increasing costs of medical care in Australia and around the world, there remains a great disparity between what an (often private) board of directors is willing to fund and what a given hospital may feel it requires, particularly with regard to expensive new communications devices. In this way, many have argued that substantial cost savings can be achieved by investing in new software solutions [62, 103, 104]. This work will go forward from the perspective that software solutions, particularly those which utilize so-called

'context-driven' architectures, will offer an inherently less expensive alternative for hospitals which are already equipped with computer infrastructures. Given the aforementioned problems in healthcare, software solutions will provide new ways to integrate largely existing technology without costs spiralling out of control.

The last decade has seen the retreat of managed care due to pre-selection with regard to who receives what healthcare related services. In this situation, the consumer has emerged as the centre point of priority setting in healthcare [38, 105]. With the awareness that current medical service delivery is lacking quality and desired improved treatment outcomes, healthcare providers are seeking alternatives [106].

One such alternative is Patient-Centered Care, which has made it to the centre stage in quality discussions [107]. Epstein defines this concept as follows: “Patient-centered care is a quality of personal, professional, and organizational relationships. Helping patients to be more active in consultations changes centuries of physician-dominated dialogues to those that engage patients as active participants” [107].

3.3.2 Telemedicine

Telehealth, telemedicine or m-health are synonyms to describe a new area of medicine. It refers to using mobile technologies and remote monitoring using sensor devices all in conjunction with wireless and internet based services [108].

The initial goal of providing medical services to underserved zones is being increasingly sought for urban scenarios in industrialized countries and between urban hospitals across national borders [109]. Such development can be explained in part by the pursuit of reduced costs, profit increase and efficiency enhancement to improve urban patients’ access to healthcare, and the growing demand for home healthcare [110].

With the introduction of telemedicine originally designed to provide a richer data stream to clinicians, new possibilities have evolved in encouraging patients to participate in their treatment of chronic illness and to involve all clinicians, not just specialists, to participate in telemedicine programs and further highlighting the home monitoring as a big growth area in telemedicine allowing conservation of valuable hospital resources [111].

Multiple economic, professional, societal, social and political factors still affect telemedicine's potential in complex ways. Expected evolutions will see significant progress in the quality and accessibility of the basic system building modules. Particularly, some of the necessary advances are related to the gap between urban and rural telemedicine, the ergonomics of sensing devices and the cost of implemented prototypes. Different levels of sophistication of devices and implementation are found in current telemedicine systems [110].

The growing prevalence of chronic illness and the rising ubiquity of technology motivate examination of the potential role of information technology (IT) for greater empowerment of consumers and their advocates in healthcare. Areas where the consumer can contribute to their own health outcome in chronic disease management include both adherence to a healthcare plan and the broader issue of the patient becoming an active partner in their own healthcare planning [41].

There is little or no doubt that this trend of telemedicine or tele-health will become one of the next major trends in health given the current proliferation of progress to date [112]. Numerous projects [39, 40] and research are focussing on telemedicine in order to improve patient treatment outcomes and allowing patients to actively participate and to handle their disease at home. It further helps the physicians to stay on top of the patient's disease and to check health sensor readings in between visits to monitor changes in the patient's health [113].

The framework created in this thesis aims to provide a unified approach to how to handle multiple participants in a healthcare interaction scenario in an open approach involving patients, specialists, physicians and so on providing relevant context information.

3.3.2.1 Telemedicine at work

The Charité University Hospital, part of Berlin University, has opened a centre for cardiovascular telemedicine that will carry out research and provide services to patients. The first big project at the new centre began in March 2008 with the 'Partnership for the Heart' project, a clinical study on telemonitoring for patients with chronic heart failure. The patients are fitted

with a device to perform ECG measurements by themselves, readings which are transmitted through a special mobile phone to a medical doctor. 600 patients participated in that study [114].

The 'Partnership for the Heart' project aimed to show a reduction in mortality rates and hospital admissions for heart failure patients, due to the assistance of telemedicine [114, 115]. Overall, the study sought to prove that more can be achieved with telemedicine while not only reducing costs but at the same time the system proved to be a valuable support of the doctors and more safety for patients was achieved by improving care at home. These savings are likely to be achieved through a reduced number of visits to the GP and particularly how patients will be able to stay at home and use fewer of the expensive hospital resources and the hospital's expensive infrastructure. This study was concluded and results published in 2010 [114-116].

Several projects have emerged throughout the last 5 years. The AWARENESS project rightfully claims that mobile devices, sensors and consumer electronics are increasingly equipped with wireless networking capabilities. This enables these devices to communicate across different networks allowing for a complete new generation of pro-active and context sensitive applications to be developed [117]. Another positive example is the Laboratory for Information Processing Technology (ITIV) at the University of Karlsruhe which is developing a personal health monitoring system. This system aims to improve patient interaction through context-awareness by adapting itself to the user [60]. CARISMA (which is another project for context sensitive middleware) project members argue that devices will increasingly be networked, thus enabling distributed applications to adapt to changes in context, such as changes in network bandwidth, battery power, connectivity, and the availability of services and so on [95].

In the healthcare domain tension exists whether to favour knowledge transfer for local processing or to favour centralised processing. One such project advocating distributed monitoring of patients is AlarmNet as proposed by Wood et al. (2008) [118]. AlarmNet monitors patients on several sensor readings and decides on the appropriate course of action for embedded systems. Wood et al. further states that: "This requires a long-term shift from a centralized, expert-driven, crisis care model to one that permeates personal living spaces and involves informal caregivers, such as family, friends, and community."[118].

3.3.2.2 Patient-centered medical home

As mentioned earlier in section 3.3.2 telemedicine increasingly focuses on health care delivery at home reducing the load on professionals.

Medical Home is not just a place where healthcare is being delivered but a healthcare model that delivers healthcare services which focus on the following features [119]:

- Patient-centered
- Comprehensive
- Coordinated
- Accessible
- Continuously improved through a systems-based approach to quality and safety

Stange et al. defines the Patient-Centered Medical Home (PCMH) as follows: "PCMH aims to personalize, prioritize and integrate care to improve the health of whole people, families, communities and populations" [120]. The PCMH includes new approaches which go beyond current primary care as it is practiced today. These approaches include elements such as same-day appointment, electronic visits, group visits, disease registries and management, greater patient engagement, care coordination, new collaborative relationships, team-based care, quality and safety initiatives, electronic prescribing and medical records [120]. In practice the PCMH model still remains elusive and further research and definitions are needed to create a common understanding of PCMH [121].

Nutting (2009) realized that the transformation to PCMH is a developmental process and consists of two components: a "core component" and its "adaptive reserve". The core component focuses on capabilities to manage basic finances, clinical and practice operations whereas the adaptive reserve was defined as including capabilities such as a strong relationship system within the medical practice, protected group reflection time, shared leadership, and attention to the local environment [122, 123].

PCMH requires more than just a series of small changes. Nutting et al. define the key theme changes as follows: [123, 124]

- Practice adaptive reserve is critical to managing change
- Developmental pathways to success vary considerably by practice
- Motivation of key practice members is critical
- the larger system can help or hinder change
- Practice transformation is more than a series of changes and requires shifts in roles and mental models
- Practice change is enabled by the multiple roles that facilitators play.

3.3.3 Healthcare in motion

Many current approaches in healthcare today are being revisited as regards the delivery of medical services, with most now focused on how medical services are being delivered and with many focused on placing the patient in the centre of their models. Not only are new mental models needed to accommodate more flexible healthcare models but new systems are needed to support a more distributed approach to healthcare [110, 123].

3.4 Business processes

Several attempts have been made to define business processes, which vary in complexity and scope. Some of them cover extensive business process capability from strategic to fine granular business processes while others cover just parts of a business process architecture stack. Very few provide adequate coverage of all levels of business process definitions. Furthermore, specifications such as WSFL [125], XLang [126] and BPEL [127] have not delivered on the notion of flexible ad hoc business processes. These specifications assume that business processes are pre-defined in nature and do not accommodate for the necessary flexibility needed dynamically by the business process [128]. In addition to the lack of formal definitions, there are no standard models for process definition languages, process enactment and for supporting

communication between systems [129]. This approach of centralised business coordination is too restrictive in a decentralised approach with thousands of participants where context information availability and location is the key determinant where decision making is executed.

This research's framework not only allows triggering decision making from local software agents where context changes trigger business rules to execute. This framework is also flexible and generic enough to accommodate the integration of decentralised agents from a central business process execution engine thus still supporting the classic enterprise architecture approach through its generic interface. Using this framework along with a business process execution engine is certainly addressing the above issues by supporting communication between systems because all information necessary for the decision making process is passed to every agent in the decision making process or stored locally (context information).

3.4.1 Business Process Execution Language (BPEL)

The most commonly known composition language is BPEL4WS, which is an amalgamation of two previously known standards from Microsoft's XLANG and IBM's Web Services Flow Language (WSFL). Although so much effort has been put into BPEL by the community, the support for sub-processes and the integration of human interaction have been omitted in the initial design of BPEL. The first is currently addressed in the BPEL-SPE proposal which focuses on sub-processes with life-cycle coupling and the second is addressed by the BPEL4People and WS-HumanTask proposal allowing functional extensions for human interactions [130-133].

3.4.2 Business Process Modelling Notation (BPMN)

BPMN was developed by the Business Process Management Initiative (BPMI) in 2004 [134]. Today the BPMN specification has been adopted by the OMG Group as an official part of their specification portfolio [135].

BPMN consists of two main components, (1) the design component which provides a readily understandable notation for business users, system architects and alike and (2), the underlying implementation layer which provides translations into BPEL[127] and BPML[136].

Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation. Additionally, there are likely to be efforts to fit BPMN into a larger context of higher-level business modelling, which includes, for instance, the modelling of business rules.

3.5 Business rules

According to the Business Rules Group “A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business. The business rules that concern the project are atomic -- that is, they cannot be broken down further.” [137] Date et al. define business rules as statements that are used to discover new information or guide the decision making process [79].

Business Rules are an externalization of knowledge and a separation of rules logically (or perhaps physically) from a system. Business knowledge is sometimes inaccessible or unknown; usually this knowledge is hard coded in software or buried deep down in legacy applications. Business rules integration is an attempt to extract the business knowledge from these sources and make them more transparent and visible for change [138].

The Business Rules Group defines the following main aspects of business rules [139]:

- Rules are essential for, and a discrete part of, business models and technology models.
- Rules apply across processes and procedures.
- Business rules must be expressed in such a way that they can be validated for correctness by business people and can be verified against each other for consistency.
- Rules should be expressed declaratively in natural-language sentences for the business audience. A rule and its enforcement are separate concerns.

- Rules build on facts, and facts build on concepts as expressed by terms.
- Rules should arise from knowledgeable business people.
- Rules, and the ability to change them effectively, are fundamental to improving business adaptability.

3.5.1 Business rules classification

Substantial research has been conducted on business rule classification [79, 80, 128, 140]. One approach to solving the problem of classifying business rules is to divide business rules into groups. These business rule groups include “Structure related” which addresses the ‘how’ aspect in terms of activity related rules. The “Role related” rules addresses the ‘who’ and ‘where’ aspect and the “Message related” address the ‘what’ which defines what information is being sent and finally, the “Constraint rules” which define the pre- and post-conditions as well as message constraints and event control [79, 128].

Rosenberg and Dustdar classify Rules Types as follows [88]:

- Integrity Rules e.g. authorizing username and password.
- Derivation Rules or Inference Rules e.g. A=B, B=C therefore C=A.
- Reaction Rules or Event-condition- action rules e.g. if a loan applicant salary is below $10’000 then reject the application.
- Deontic Rules (partially identified) or authorization rules. E.g. only Manager is allowed to accept loan applications above $400’000.

Such a system for classification of rules may become useful in a distributed environment (e.g. limiting the need of distributing Deontic rules. Deontic rules might already have all required information to process due to their nature of preconditioning and the already available context).

Classification in relation to the framework presented in the research becomes an important aspect when creating business rules designed to run in a distributed environment.

Rules need to be created as a self-contained work unit. The classification of those rules into groups eases the creation, integration and maintenance of such business rules.

3.5.2 Business rules integration

The classic approach of integrating business rules into an existing application is to register the applications business objects with the business rules engine. The application usually consists of objects which relate to each other such as patient to general practitioner. In the case of Java a plain old java object (POJO) is added to the business rules engine. The business rule engine then accesses the POJO through the public methods it exposes. For example a java class Patient exposes several public methods as shown in Figure 4.

```
public class Patient {
        String name;
        XMLGregorianCalendar birthDate;
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public XMLGregorianCalendar getBirthDate(){
            return birthDate;
        }
        public void setBirthDate(XMLGregorianCalendar birthDate){
            this.birthDate = birthDate;
        }
}
```

Figure 4 - Java Person Class

These public methods such as getName, setName, getBirthDate and setBirthDate can be accessed from the business rules engine during execution time as shown in Figure 5

```
rule "Check Birthyear"
    salience 200
    no-loop true
    when
        $p : HealthOntology.Patient(
            Calendar.getInstance().get(Calendar.YEAR) - $p.birthDate.getYear() < 18)
    then
        System.out.println(
            "Patient is too young");
end
```

Figure 5 – Patient Birth Year Check Business Rule

Business rules change quite frequently and are classically embedded in the source code or are externalised as interpretable scripts (business rules). These scripts are interpreted by business rules engines which map the script's functionality to the runtime business objects registered with the business rules engines. Most business rules engines provide a Java interface (JSR-94) to add the relevant business objects during runtime and trigger these rules processes. A business rules engine can also interface with database management system (DBMS) products, a variety of browsers and other kinds of software. Therefore, these integration possibilities into different software environments is making business rules reusable throughout different systems [138].

The most common implementation of business rule integration is the use of the façade pattern, which serves to hide the underlying implementation of different business rules engines through a common interface. The façade pattern hides one or more complicated subsystem application programming interfaces (API) and simplifies the access through a higher level interface which makes the business rules engine interfaces easier to use [88, 91, 141].

Another approach is to generate web services based on the semantic description of individual rules. Issues with such an approach are that usually typical web services are generated during design time and not during runtime [88, 91, 141] and therefore lack the flexibility to deploy rules during runtime without interface changes.

Another problem area is the exchange of business rules among different business rules engines [88, 141] due to an overall of lack of standardization of a rule representation language [91]. One approach that has been proposed in [141] is to use a Business Rules Broker architecture to exchange design time and runtime information between business rules engines by exposing business rules through web service interfaces for runtime communication and Simple Rule Markup Language(SRML) or Rule Markup Language (RuleML) as an intermediate Language between business rules implementations [88, 142].

Additionally, WS-Coordination and WS-Transactions are standardised by Oasis, which define themselves as: "OASIS (Organization for the Advancement of Structured Information Standards) is a not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society. OASIS promotes industry consensus and

produces worldwide standards for security, Cloud computing, SOA, Web services, the Smart Grid, electronic publishing, emergency management, and other areas" [143].

WS-Coordination and WS-Transactions are used in some projects to synchronize deployed business rules among several rule engines inside a transaction to avoid diverging of active business rules versions. One such aspect of coordination lies in runtime synchronization as to maintain integrity among the different participants in a distributed environment [142, 144, 145].

3.5.3 Business process and rules combination

Charfi and Menzini (2004) argue that: "The lack of flexibility is tightly related to the lack of modularity: If we can break down the business logic underlying the composition into several parts or modules, the composition becomes more flexible since each of these parts can evolve independent of the rest. This motivates the need for a more fine-grained approach; with appropriate support, parts of the composition logic can be created, modified or deleted dynamically at runtime" [146].

Service composition is too complex to handle manually [128] and BPEL does not provide the concept of crosscutting modularity [146]. Crosscutting modularity typically aims to cut across classes, functions, processes and so on to reuse existing functionality across a system and to maintain its modularity. Typical applications of crosscutting modularity are functionalities common to the entire system such as user privilege checking, user auditing, logging and many more. As a consequence functionality such as "Authentication", "Authorization", "Logging" and so on which represent crosscutting aspects of an application would need to be solved elsewhere [147].

An approach to overcome the crosscutting limitation is presented in papers [146, 148] using AO4BPEL which presents an aspect-oriented approach. Aspect-oriented programming facilitates common tasks such as logging, monitoring, performance optimization, business rules, authorization and authentication, transaction management and many more to uniformly handle crosscutting concerns [146, 149].

3.6 Distributed computing

Service orientation or distributed computing is by no means a new approach. Microsoft's Distributed Component Model (DCOM) and the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) were both early attempts to provide service contracts in a loosely coupled way [150].

Although these technologies enjoyed a high market adoption rate, it was apparent that Microsoft's DCOM would remain a vendor specific implementation and OMG's CORBA specification, implemented by several vendors', lacked interoperability among the different CORBA vendor products [150].

In the late 1990's several vendors realized that SOA would never provide true agility unless SOA became implementation independent and that the eXtensible Markup Language (XML) would make a perfect messaging protocol. These advances eventually led to the creation of Web Service Description Language(WSDL) [151], Universal Description Discovery and Integration (UDDI) [24] and the Simple Object Access Protocol (SOAP) [65], all of which formed the building blocks of modern Web Services [150].

3.7 Generic services

Berger defines generic services as follows "The objective of the generic services is to ease the development of agent services by providing generic building blocks, which can be assembled and customized to create complex applications. These generic service components mainly handle most of the agent-related concerns (protocol, conversation, language, ontology, and errors), while allowing the developer to concentrate on the application logic." [6]

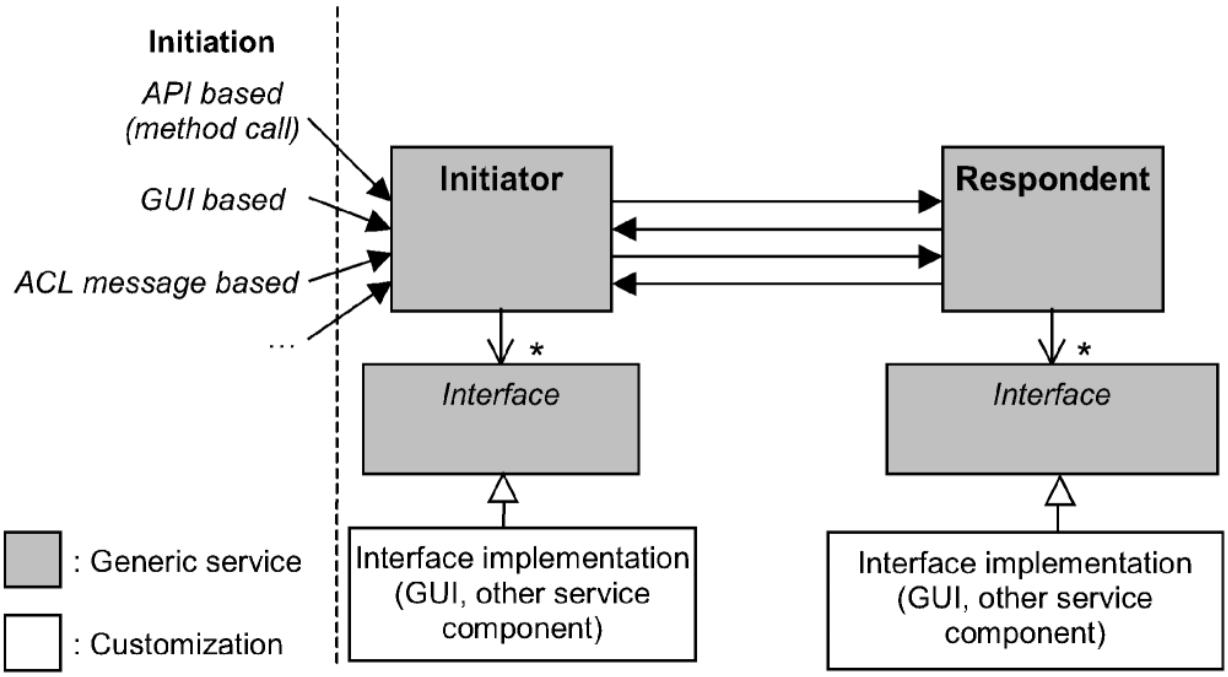

Figure 6 - Berger's generic services diagram [6]

Figure 6 illustrates the interaction between Initiator and Respondent with one Initiator capable of calling many Respondents. An agent in Berger's scenario can consist of both Initiator and Respondent and thus this approach could be scaled to multiple levels of integration [6].

In Berger´s approach "the only prerequisite for an Initiator to interact with a Respondent is to have the information about the protocol, the content language, and the ontology used". In other words, the service or agent must have knowledge of ontology (understanding of the domain and its structures) and is installed in the agent and queried during runtime by other agents (Initiators) to check whether it understands a particular ontology [6].

3.8 Web services

A Web Service is a self-contained, modular unit of application logic. It provides business functionality via the Internet and is designed to support a stateless model and atomic synchronous and asynchronous message exchange [152].

3.8.1 Web service model

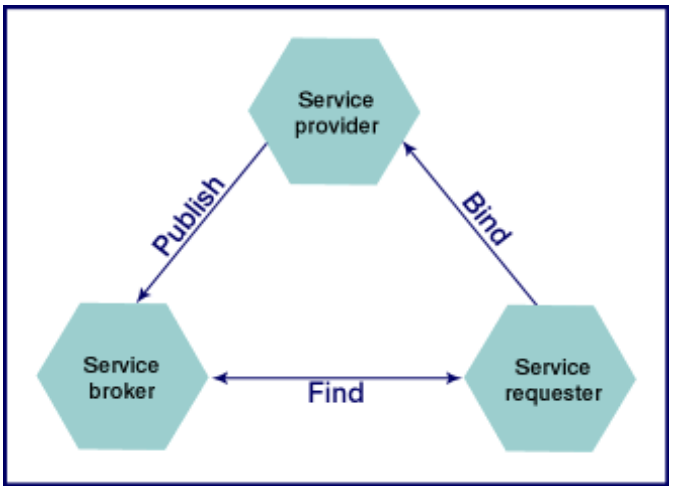

Figure 7 - Publish Bind Find Web Service Model [153]

The framework of Web Services consists of a publish-find-bind cycle shown in Figure 7. Firstly, the service provider publishes the Web Service to Service Broker (UDDI) using the WSDL information. Secondly, the service requester finds the required Web Service in the UDDI repository and retrieves the WSDL information. Thirdly, the service requester binds to the Web Service of the Service provider using the information provided inside the WSDL information [153, 154].

3.8.2 Simple Object Access Protocol (SOAP).

SOAP is essentially the XML-based protocol for exchanging structured information over Internet communication protocols, and serves to enable distributed and decentralized communication. In most cases HTTP is used as transport protocol [65].

3.8.3 Web Service Description Language (WSDL)

A WSDL document is a XML-based document which serves to define services as collections of network endpoints or ports. Endpoints and messages are described in an abstract format to separate the definition and the actual implementation from each other [151, 152].

A WSDL document consists of following elements: [152]

- Types – a container for data type definitions using some type system (such as 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.

3.8.4 Universal Description Discovery (UDDI)

The UDDI repository allows storing of information about Web services, data structures and behaviours of all its instances. UDDI provides a programmatic functionality to publish, find and manage information of web services located inside the repository [24].

The main functionality of the UDDI repository is to provide functionality to [155]:

- publish information about a Web service
- Find web services
- Determine security and transport protocols
- Providing fail-over, intelligent routing or load-balancing functionality

3.8.5 Mapping of WSDL to UDDI

UDDI is structurally different to WSDL, and consists of the notion of businesses, services, binding templates and interfaces (tModel). Therefore, the information provided in

WSDL must be stored in different locations in the UDDI registry. The following image provides an overview on how information is mapped from WSDL structure to the UDDI structure:

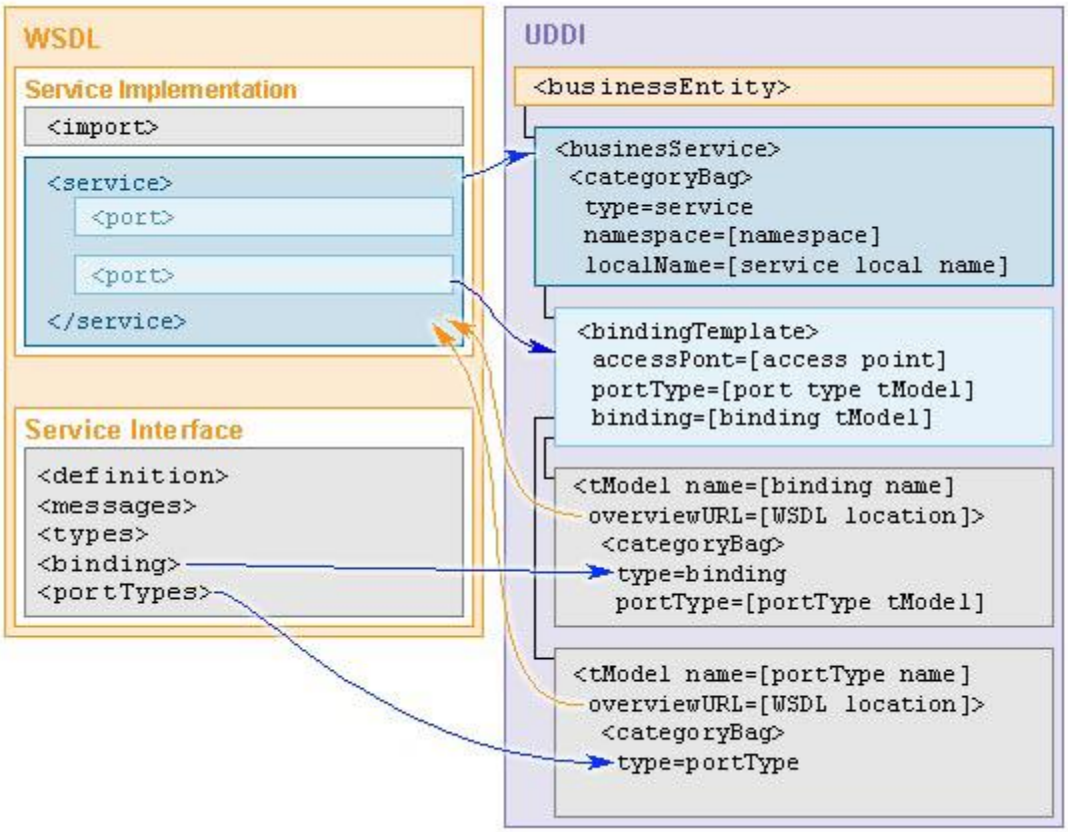

Figure 8 - Mapping WSDL to UDDI[156]

Although it is important to know on how the information is stored, many components or software projects provide an easier solution to register existing Web services to ease the burden of integration converting WSDL information into an UDDI. Some of these projects are UDDI4j (version 2), [157] client libraries contained in Apache's jUDDI which make full use of Java annotation [158] or the Java API for XML Registries (JAXR) which provide an abstraction level on top of XML Registries such as UDDI and ebXML [159].

Because of a lack of expressiveness in UDDI for WebServices some approaches have extended the functionality of UDDI to be able to add more relevant meta-information to context information to enhance the registration, retrieval and storage of semantic web services [160, 161].

3.8.6 Web service composition

In principle, Web service composition or orchestration is the aggregation of simple Web services functionality into more complex or coarse granular Web Services. Thus, Web services can be composed into more complex business processes and then be re-exposed as a more complex business web service. Composite Web services are usually created using languages such as BPEL4WS or short BPEL, ebXML, XPDL, WSFL, WSCI, and BPML [162, 163].

3.9 What is context?

Several attempts have been made to define context. Dey et al. (2002) define context as "any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects." [33]

Schilit et al. (1995) [164] sees context as three categories consisting of

1. Computing environment: available processors, devices accessible for user input and display, network capacity, connectivity, and costs of computing;
2. User environment: location, collection of nearby people, and social situation, and
3. Physical environment: lighting and noise level.

Many issues must be addressed in context-aware systems. Satyanarayanan (2001) identifies some key questions in this regard [165]: How is context represented internally? How is this information combined with system and application state? Where is context stored? Does it reside locally, in the network, or both? What are the relevant data structures and algorithms? How frequently does context information have to be consulted? What is the overhead of taking context into account? What are the minimal services that an environment needs to make context-awareness feasible? What are reasonable fallback positions if an environment does not provide such services? Is historical context useful? What are the relative merits of different location-sensing technologies? These questions will inform a great deal of the following exploration of context-sensitive information technology.

3.9.1 Previous research on context

Context-sensitive applications need data from sensors, devices, and user actions, and might require ad hoc communication support to dynamically discover new devices and engage in spontaneous information exchange as well as to make the actual computing part of it and its enabling technologies essentially transparent [18].

Roman and Campbell identify at least three application aspects that are altered by context: data, composition, and logic. As an example, data might be needing transformation based on the user's location. Composition could change based on the location of close-by peers or if the user is inside a doctor's office or hospital. Finally, logic could change based on activity or time of day adjusting the content or information the user sees [33].

Maamar distinguishes between three contexts, user context, service context or a combination of both. This researcher focuses on service context which addresses issues such as (1) service adaptability, (2) dealing with service availability, and (3) supporting on-the-fly service composition [94]. User context would focus on the state and environment of the user such as location, mood, heart rate and so on [28, 33].

Ranganathan and Campbell argue that ubiquitous computing environments must provide middleware support for context-awareness. To this end, they also propose a middleware that serves to facilitate context-aware agent development, thereby allowing agents to acquire contextual information easily so they may adapt to the changing context. Another key issue is to enable a common semantic of contextual information among heterogeneous agents [17].

Pascoe describes context as "Context could be generally described as the subset of physical and conceptual states of interest to a particular entity" [166].

D-Jess [167] a distributed extension of Jess [82], once implemented as a context-enabled middleware, could distribute both contextual information (facts) and reactive behaviours (rules) among participating nodes. Rules can be executed either locally or remotely using the facts distribution mechanism explained by Cabitza et al. [167, 168].

Prekop and Burnett (2003) classify context into two dimensions, internal and external. The internal context encompasses users' goals, tasks, work context, business processes,

personal events, communication, emotional and physical state while the external dimension covers the elements of the physical environment, including location, proximity to other objects, temperature, time, lighting levels, and so on [169].

Another classification of context has been exemplified by Kunze et al. [60] which distinguishes between two types of context: low level context addresses information which can be directly sensed by sensors and high level context focuses on more complex contextual information which is derived from multiple information sources.

3.9.2 Definition of context used in this research

User context is defined in this work as any information that can describe an entity or person and its surrounding environment, events that occur relevant to that entity or person be it internal or external. Low level context made available from sensors is aggregated or inferred by the business logic into high level context as supported by Zhang et al. [170]. Any derived contextual information is therefore a product of applied business rules logic (e.g. The average blood pressure of the blood pressure history is not made available as a computed figure but is processed and interpreted in business rules to get to that figure).

3.9.3 What is context-awareness?

It is commonly agreed that the first investigation into context-awareness was the Olivetti Active Badge Location System [171]. However, among the first discussions about context-awareness was that by Schilit and Theimer which argued that “Context-aware computing is the ability of a mobile user's applications to discover and react to changes in the environment they are situated in” [172] and in the work of Schilit, who argued that “Such context-aware software adapts according to the location of use, the collection of nearby people and objects, the accessible devices, as well as changes to those objects over time” [164].

3.9.4 Previous research on context-awareness

Schilit, Adams and Want categorize context-aware system as follows [4]:

- Proximate selection, a user-interface technique where the objects located nearby are emphasized or otherwise made easier to choose.

- Automatic contextual reconfiguration, a process of adding new components, removing existing components, or altering the connections between components due to changes in context.
- Contextual information and commands can produce different results according to the context in which they are issued.
- Context-triggered actions, simple IF-THEN rules used to specify how context-aware systems should adapt.

Dey and Abowd define context-aware as: “A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task” [173] and categorize context-aware computing into two categories [174]:

- Using Context
- Adapting to Context

Zhang et al. argue that context-awareness requires more than just capturing the users context environmental changes but that context-awareness requires an interdisciplinary approach, cutting across programming paradigms, operating systems, embedded systems, networking and many application areas [175].

3.9.5 The definition of context-awareness used in this research

Context-awareness is a concept defined by actions triggered by the user’s context. For example, a patient who has been given a heart monitor to track the patient’s heart beat is sending out measurements at a regular interval. These readings trigger actions inside the context-aware software agent which listens to the sensor readings and processes these events against a reasoning component which in this thesis is a business rules engine.

3.9.6 Potential problems and issues

Given these considerations, context-aware computing is a concept which is loosely defined as the ability of a given software application to adapt to circumstances which cannot be

directly programmed or delivered in a comprehensive manner before the time of use. For this reason, this software must be able to adapt to circumstances which are constantly changing. While it is clear that there is eminent use for such software applications, particularly in the healthcare domain, the definition of this still-burgeoning field remains muddled in the current research.

While there is an existing software architecture presented in this thesis (which will be explored in great depth throughout this piece), there remains some difficulty with regard to many important problems with such dynamic software. These include problems which relate to equipment, such as difficulties related to networks, artefacts, and mobile communication, and other issues which relate to security, such as issues of identification, authentication, confidentiality, integrity, or availability of software applications. As the focus of this work will ultimately be on context-awareness in software as it relates to the healthcare domain, it is crucial to consider the potential problems that can be caused by such a system, as well as the potential routes to fixing such problems in the software.

The first problem is the general issue of acceptance of a given piece of software. As explored by Bellotti and Edward (2001), "humans have complex motivations which lead to unpredictable behaviours" [176]. It is for this reason that any software system which is necessarily based on context is, by design, one which presumes to be able to predict these vastly complicated behaviours, including intent and switching between different users based on their behaviour. As Barkhuus and Dey (2003) describe, these systems must function at three different levels of interactivity: (1) personalization, (2) passive context awareness, and (3) active context awareness. They explain that these systems must be custom-tailored to the user, yet argue that there is only a certain level of personalization and dynamic awareness that a user is willing to accept from a piece of software: "Users are willing to accept a large degree of autonomy from applications as long as the application's usefulness is greater than the cost of limited control," that is, users will accept context-aware computing precisely up to the point at which it becomes inconvenient or offensive [177].

Context must also be considered within the realm of imagination. As Brown and Randell (2004) describe, individuals who use tools for a long period of time come to 'dwell' with these

tools, as aptly indicated by, for instance, individuals who use a telephone or doorbell on a daily basis. In these situations, they have adapted a tool which was initially envisioned to serve one purpose, and have adapted it to a wide range of activities. An example of such adaptation by 'living' in a context-aware environment can be seen when individuals will call a particular telephone number twice in a day if the matter about which they are calling is particularly important. These researchers recommend, as a result, that any context-driven software agent be particularly simple and easy to use to facilitate such processes of 'dwelling' with regard to the software [178].

In addition, it is important to remember that the purpose of the tool might deviate from the process for which it was intended. According to Fogarty et al. (2004), when they introduced their context-aware system that was designed to take into account the individual's location, computer, and calendar information, intended to help individuals to avoid interruptions, the tool provided by the software was instead used to identify whether the person was present and led to no real reduction in interruptions [179].

3.10 What is model-driven architecture (MDA)?

As stated by the OMG, “The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns.” [180]. The core idea behind MDA is to raise the abstraction level in software engineering. This allows for the development of software with a simpler approach which separates the complexity of software engineering into different levels of abstraction [181].

MDA is an open and vendor-neutral architectural framework that was created by the Object Management Group (OMG) and introduced in 2001. It leverages OMG standards and technologies to target different application domains and different technologies [182, 183]. MDA attempts to abstract the design of models from the underlying technology, thus avoiding the repetition of rewriting specifications for each technology [184]. MDA’s are used to model a domain with its structures, behaviours and interface definitions. The classic MDA approach is to use modelling languages as programming languages or to generate code from the designed model rather than being a unique design language [181, 184].

Model driven architecture is best described as different levels of specifications of a model. The first level is the “Computation Independent Model” (CIM), also referred to as the domain model or meta-model, which is then transformed into a “Platform Independent Model” (PIM). At present there is at best limited support on converting the CIM to PIM transformation, because of the intense difficulty associated with the mapping between meta-model and model. The MDA market, however, has recognized this issue and is working on increased support to close this gap. Software architects have consequently begun to start modelling the domain on the PIM layer where there are ample software modelling tools available. Consequently PIMs are then translated into a “Platform Specific Model” (PSM) by mapping the PIM to a platform specific implementation language such as Java or .Net Platform. (see Figure 8) [185, 186].

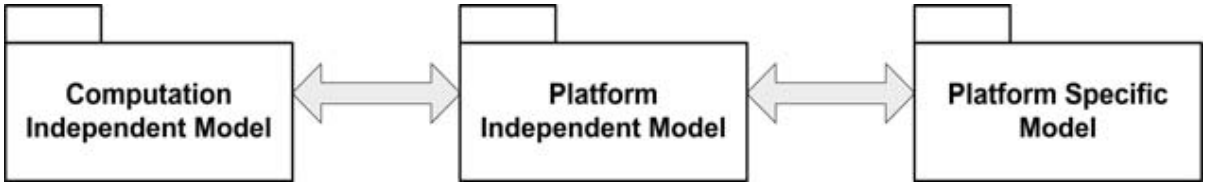

Figure 10 - MDA model transformation [187]

3.10.1 How is MDA used?

Starting with the software development lifecycle which has taken many shapes from iterative, waterfall or extreme programming to name just a few, these approaches consist of several steps. Every approach will undergo, to a larger or lesser extent, the same steps of requirements gathering, analysis, design, implementation and deployment [183]. These steps of the software development lifecycle (SDLC) translate into the MDA approach as seen in Figure 9 [130] which shows that each step in MDA can be mapped into the regular SDLC.

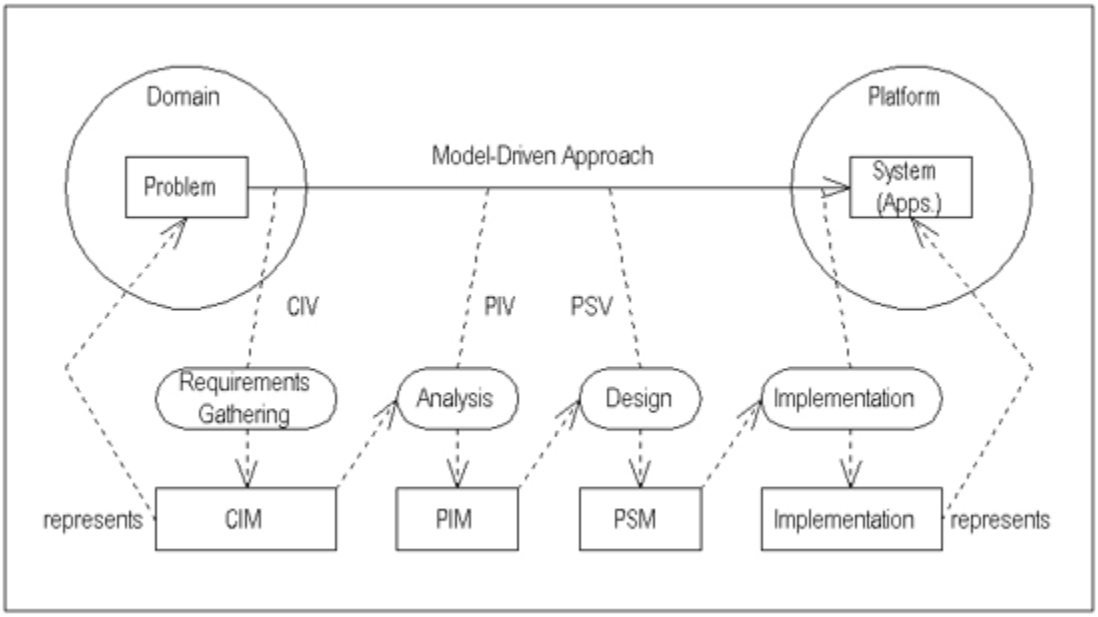

Figure 9 - Software development lifecycle MDA mapping [185]

Many projects today use a model- driven approach, such as the "Unified Modeling Language" (UML), to be able to design a software architecture which can be generated and deployed on several target platforms. A disadvantage of that approach lies in that the architect must have knowledge of all different languages in the different levels of MDA [187]. In addition, complex business rules are difficult to model in any of these modelling languages and thus sometimes require manual coding of such complex or esoteric rules (implied rules or codes of conduct), or the usage of additional tools such as business rules to accommodate those complex business rules. These additions further enhance the complexity and maintainability of such a system architecture design [188].

3.10.2 Unified modelling language (UML)

The Object Management Group first standardized UML in 1997 as part of their Model-Driven Architecture product offering. UML has become a widely accepted standard with improved functionality in the UML 2.0 specifications. Improvements made in UML Version 2.0, include the capability to represent behavioural models, architectural models, business process and the introduction of Object Constraint Language (OCL) rules language to further improve the expressiveness of UML. OCL can be used to implement business rules in any of the layers in the

MDA stack such as the Meta Object Facility (MOF) or UML [189]. In addition, the notion of behaviour and activities and their relation to actions has found its way into the specification. However, no specific action language has been specified and leaves the implementation to the software creator [190].

3.11 Previous research on software agents

There is no clear definition about what agent-based systems are. Ferber defines agent systems as follows: "An agent can be a physical or virtual entity that can act, perceive its environment (in a partial way) and communicate with others, is autonomous and has skills to achieve its goals and tendencies. It is a multi-agent system (MAS) that contains an environment, objects and agents (the agents being the only ones to act), relations between all the entities, a set of operations that can be performed by the entities and the changes of the universe in time and due to these actions" [191].

Wooldridges' definition of this concept defines agents as "An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective" [192]. Stan Franklin and Art Graesser define that: "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future" [193].

Cohen and Cheyer (1994) define both in their Open Agent Architecture and in their follow-up Adaptive Agent Architecture an agent-based system that supports task coordination and execution. They listed the following agent characteristics [194]:

- Delegation – the ability to receive a task to be performed without the user's having to state all the details
- Data-directed Execution – e.g., the ability to monitor local and remote events
- Communication – e.g., the ability to enlist other agents (including people) to accomplish a task

- Reasoning – e.g., the ability to prove whether its invocation condition is true, and determine what are its arguments
- Planning – e.g., the ability to determine which agent capabilities can be combined in order to achieve a goal.

The Hayes Roth definition of an agent is: "Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions" [195].

3.12 Our definition of software agents

Most interpretations of agents put functionality or goal orientation at the centre of every agent. For this work, we solely put context in the centre of our agent, and keep the functionality and goal seeking flexible and generic by means of Model Driven Architecture (MDA) and business rules execution. This means that the only state kept within the agent is that of the context and its description.

Our definition or application of a software agent is that it listens, reacts and exposes context through a context registry to make context available to other agents requesting particular context information. The software agent also has the capability to run as an MDA software agent with ability to interpret given MDA information, generate a class library from the model information, instantiate objects from the given MDA information and recreate an information stream of the MDA information represented in that particular agent. Furthermore, to be able to reason across all objects created from MDA information they are executed against a business rules engine with the capability to access the context location information made available through a context registry, in our case a UDDI registry.

3.13 What is the Semantic Web?

The Semantic Web is about explaining relationships between things and properties of things such as weight, age, size. An example of a relation explained through Semantic Web is

that “John is father of Jeff, Sam is brother of John, and consequently through inferencing one would conclude that Sam is the uncle of Jeff” [196].

The Semantic Web relies heavily on machine understandable ontologies for the purpose of having a mutually comprehensive language across systems. Ontologies provide domain meta-data and data as well as vocabulary concepts as machine processable semantics [197].

Domain vocabularies are declared to have a mutual understanding about the meaning of the domain term. For example, a doctor describes a medication that is sold under different names in different countries. That medication would be named uniquely within a medication-ontology to eliminate ambiguity between systems working with each other.

3.13.1 Background on Ontologies

This section elaborates on ontologies as it relates to the computer science. Gruber (1993) describes the role of ontologies as: “supporting knowledge sharing activities, and then present a set of criteria to guide the development of ontologies for these purposes.” [198]. Borst in 1997 define an ontology as: “formal specification of a shared conceptualization.” [199]. In 1998 Studer et al. merged Gruber’s and Borst’s definition into one concluding definition: “An ontology is a formal, explicit specification of a shared conceptualization.” [200]. This statement as a consequence implies that the conceptualization would express a shared view between several participants and also, that such conceptualization should be expressed in a (formal) machine readable format.[201]

Staab et al. (2009) define ontologies as follows: “We refer to an ontology as a special kind of information object or computational artifact” and conclude that: “Computational ontologies are a means to formally model the structure of a system, i.e., the relevant entities and relations that emerge from its observation, and which are useful to our purposes. An example of such a system can be a company with all its employees and their interrelationships.” [201].

Gomez et al. explain that to enable sharing and reuse of knowledge and reasoning behaviour across domains and tasks, ontologies and problem-solving methods (PSMs) have been developed. Ontologies are concerned with the static aspects of a problem domain, however to implement dynamic knowledge PSM’s are used to implement decision making/reasoning

functionality. PSMs and ontologies can be seen as complementary reusable components to construct knowledge systems from reusable components. In order to build full applications of information and knowledge systems from reusable components, both PSMs and ontologies are required in a tightly integrated way [202].

3.13.2 Web ontology language OWL

The W3C defines the Web Ontology Language OWL as a "semantic mark-up language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language" [203].

OWL was designed to enhance machine interpretability of ontologies and information on the web as opposed to being read by humans. OWL is used to describe the explicit meaning of Terminologies and Concepts and to further describe the relationship between those terms such as properties, hierarchical or lattice relationships as well as constraint restrictions [204]. This description of Terminology and Concepts and their relationship is called "Ontology" [205].

OWL is the next evolutionary step towards a machine interpretable semantic Web which provides ever more capabilities to describe information. This architecture builds upon XML, RDF, RDF-S and its predecessor DAML+OIL web ontology languages, and provides three sublanguages as well: OWL Lite, OWL DL (includes OWL Lite) and OWL Full (includes OWL DL) [205].

3.13.3 Context and OWL

The Resource Description Framework (RDF) [206] and the Web Ontology Language (OWL) [207] are description frameworks which allows expressing of context information. Some languages built on top of RDF and OWL are more tailored to describe context including the Composite Capability/Preference Profiles(CC/PP) [208] and the User Agent Profiling Specification (UAProf). [209] Korpiää used RDF as a context expression language to describe sensor data and derived sensor data on mobile devices [92]. CC/PP used by Indulska was found to be too limited to describe complex context models [210]. On the other hand, OWL allows complex context definition models and is used in different approaches [8, 20, 211-213].

3.13.4 Model-driven architecture and Semantic Web

Attempts have been made to join the Model Driven Architecture approach with the semantic web. Most of these approaches focus on the translation from UML via the XML Metadata Interchange (XMI) format using XSLT or other forms of translation to create an OWL Ontology [214, 215] or taking one level of indirection more such as the "Ontology UML Profile" which uses the Ontology Definition Metamodel (ODM) to translate between models [10, 216]. Translations between the various standards inevitably results in information loss and translation errors due to inadequate meta data storage or the unavailability of good translation parsers [22].

MDA takes a different approach in terms of implementation within the software development cycle. Classically, MDA uses its Stacks of Languages to design the real world in different levels of abstraction and at its last step generate code for a specific target platform [184, 217, 218]. Some implementations of UML have explored interpreting the UML Model in a virtual machine only to conclude that "UML is not defined precisely enough to allow for unambiguous model execution, and second, even if UML was defined precisely enough, one still needs a dedicated runtime architecture on top of which the models run and with which they are tightly integrated" [219].

The Semantic Web takes a different approach towards runtime interpretation of OWL meta information and the instantiation of OWL-Individuals (real world objects). OWL loads information in a runtime repository along with a reasoning or inference engine which exposes functionality to query the loaded OWL document for relationships and properties of instantiated objects (OWL-Individuals) and the associated meta-Information [220, 221].

3.13.5 Our application of model-driven architecture

We have chosen to use OWL instead of UML as object and relationship design language to encompass the design of the meta-model and model as well as the definition of context based on using OWL-S. This is merely a decision based on tool support, OpenRDF's Alibaba is capable of compiling and instantiating java from an OWL definition. This approach of using the semantic web approach using OWL was further supported by Linnhoff-Popien et al. who suggested that that the use of ontologies show the most promise in the area of modelling context [19, 222].

This work will employ a runtime engine to instantiate objects, and infer a set of rules based on the OWL schema as well as use a business rules engine to reason across the objects and context. This is a substantially different approach from the classic MDA approach: instead of generating code during design time, models are interpreted during runtime and supporting code is generated dynamically when executed. Following this approach allows a very flexible model deployment to accommodate changes without any changes to the runtime environment.

Furthermore, this software agent has the requirement of being executed in a stateless manner and to expose context information passed by sensors and other environmental changes. Context is kept stateful inside the agent and made available through a central repository. This agent implementation does not focus on agent communication language (ACL) or FIPA, defines its own protocol and has one uniform interface across all agents for inter-agent communication. Every agent is designed to run independently while reacting to changes in the environment (e.g. sensors data readings) and be able to trigger actions based on the context received as well as registering context and its relation to other participants (e.g. patient-doctor relationship) in a central repository (UDDI).

In addition, this agent software does not depend on agent capabilities but rather the capability of the agent is defined by the context it contains and exposes.

In healthcare the use of the semantic web is about making information available on the internet using machine-readable metadata to be able to uniformly exchange information across organisations. In this way, the research community has come together to acknowledge the importance of establishing strong and explicit models of semantic information that can be used to support the exchange of information in many different contexts. First, ontologies are shared models of some domain that encode a view which is common to a set of different parties, and contexts are local models which are used to encode the view of a domain held by some party. For this reason, it is crucial to understand the importance of context-driven ontologies in the healthcare domain: these are best used in applications where the most important problem facing various stakeholders is the management and various uses of common representations. An example would be environments where knowledge and information must be distributed in a rapid and efficient manner e.g. a chronic heart disease patient suffers from heart arrhythmia allowing

all involved parties to interpret the information the same way without ambiguity and to prepare the necessary services to administer to the heart disease patient, such as ambulance pick up at patient's home and preparation of emergency procedures at the hospital.

In the healthcare domain, as these professionals operate in a peer capacity, albeit with a large degree of autonomy, but still require a great deal of common coordination (particularly to reduce the likelihood of error), it is crucial that any system that aims to solve these problems offer elements of both compatibility (mutual understanding of ontology and model) and local reasoning.

3.13.6 Related works

Context-aware systems building is a complex and time consuming task. One of the major problems is insufficient support in context-aware frameworks, which will support context definition, storage and communication. A context-aware infrastructure ought to provide functionality for most tasks and activities in dealing with context [15].

The CARISMA project stated that middleware for mobile computing must be capable of both deployment-time configurability and run-time reconfigurability to meet the new requirements of mobility and the need for context-awareness and adaptation [95].

3.13.7 Previous implementations of model-driven approach and Semantic Web

Different groups have developed context-aware infrastructure over the years. Xerox's Palo Alto Research Centre has been working on pervasive computing applications since the 1980s [223]. Carnegie Mellon University's Human Computer Interaction Institute (HCII) worked on the project Aura. The project's most important goals were to provide capabilities that support user mobility and serves to shield users from variations in resource availability, regardless of their location [224]. Hewlett Packard's Cooltown project has developed a technology with support of physical-virtual integration where people, places, and things are represented as services and information [225].

CAMUS, a context modelling and reasoning approach to context-aware middleware promotes the use of OWL to define context that uses a server centric approach due to the lack of resources in robotic environments. Robots merely gather sensor information and send the

information for decision making to the central server. Its main focus lies primarily in providing a software architecture for context-aware ubiquitous computing environments [15].

The Oxygen system aims to put humans in the centre rather than focusing on technology. This new approach focuses on human needs and communication, and provides ease of use through interpretation of natural language and visual interfaces, thereby making it easy for individuals to access knowledge, automate repetitive tasks and collaborate with each another. Oxygen uses RDF to support metadata management and manipulation for personalized information management and collaboration [226].

Another project developed at the University of Illinois aims at providing a distributed middleware that provides infrastructure for context-aware agents in smart spaces. At the core of this middleware is a broker-centric agent architecture which provides runtime support to context-aware systems in Smart Spaces such as an intelligent meeting room. One of the key issues of this architecture is that it is aimed at enabling distributed context-aware agents to share a common model of the context using Semantic Web Ontology [17].

Strang and Liennhoff-Popien argue, based on their context modelling survey, that the most promising approach for context modelling for ubiquitous computing environments lies in the ontology section with respect to the following requirements [19]:

- distributed composition
- partial validation
- richness and quality of information
- level of formality
- applicability to existing environments

CONON is a project which seeks to model context in pervasive computing environments using OWL encoded context ontologies. CONON supports an upper level ontology which provides general concepts of basic context constructs (see Figure 10) [20].

```
<owl:Class rdf:ID="ContextEntity"/>
 <owl:Class rdf:ID="Location">
   <rdfs:subClassOf rdf:resource="#ContextEntity"/>
 </owl:Class>
 <owl:ObjectProperty rdf:ID="longtitude">
   <rdf:type rdf:resource="FunctionalProperty">
   <rdfs:domain rdf:resource="Location">
   <rdfs:range rdf:resource="xsd:double">
 </owl:ObjectProperty> ...
<owl:Class rdf:ID="IndoorSpace">
   <rdfs:subClassOf rdf:resource="#Location"/>
   <owl:disjointWith rdf:resource="#OutdoorSpace"/>
 </owl:Class>
 <owl:ObjectProperty rdf:ID="locatedIn">
   <rdf:type="owl:TransitiveProperty"/>
   <rdfs:domain rdf:resource="#Entity"/>
   <rdfs:range rdf:resource="#Location"/>
   <owl:inverseOf rdf:resource="#contains "/>
  </owl:ObjectProperty>   ...
```

Figure 10 - Partial OWL serialization of the upper ontology [20]

Additionally, the project provides extensibility for adding domain specific ontology and the use of logic reasoning to check the context information consistency [20].

Another approach using OWL ontologies to support semantic interoperability and context-aware applications is proposed as Context-embedded Intelligent Hospital Ontology (CIHO) in which ontologies are used to provide a formal description of the healthcare domain and support logic-based context reasoning [227].

3.14 Outcome of Literature Survey

This literature review has shown the key concepts that will form the majority of the work in this thesis, particularly as they relate to the healthcare domain, as well as the problems in this field that can be best addressed through the use of ontological and context-aware software. These are particularly problems of information gathering and awareness, as well as of different end-user agents in a healthcare context. Some concepts such as web services, generic services, and distributed computing were inspected for suitability and applicability and some components of these approaches were borrowed to build this application framework such as registering agents instead of web services in the UDDI registry.

This chapter has also considered various methods and solutions that have been presented thus far, as well as shown that the current approaches, particularly telemedicine and distributed

computing are and will be an important approach to tackle present and future problems. It further shows as well the downsides to these systems with regard to what is most necessary in a healthcare setting.

This consideration transitioned into an exploration of OWL and the Semantic Web, an approach with regard to cataloguing and categorizing information that is of enormous use and benefit in a healthcare context. These systems of categorization and organization of available information demand the semantic sensibility offered by OWL. This chapter considered context in a software environment, and reflected on the possibilities offered by such a consideration when it comes to applications in healthcare and in other important fields. Through orienting the software approach to ontology (through OWL), so-called 'pervasive' computing environments, such as chronic disease patients, hospitals, laboratories, and other healthcare environments, can be made to be more context-aware and supportive of ubiquitous agents, while facilitating cooperation and enhanced communication between different agents. These concepts will be explored in greater detail as this work goes forward.

The following chapter 4 will explore the framework which will explain the final piece of agent-driven software with the aim to transform the healthcare environment into one of greater interoperability and communication through greater context-awareness.

3.15 Constraints on Research Question

To best support the context-centric model driven agent approach we must identify the required features for the framework. At the core of this framework lies the intent to allow application developers or business analysts to develop an application by design only and deploy the application without the need to actually generate and/or compile the source code.

Dey (2000) puts forth a number of features which are general in their application and ought to be supported by this framework as well [174]:

- Context specification (CS)
- Separation of concerns and context handling (SEP)
- Context interpretation (I)
- Transparent distributed communications (TDC)
- Constant availability of context acquisition (CA)
- Context storage (ST)
- Resource discovery (RD)

These framework features are supported by the context-centric model driven agent approach created in this research, by this framework and also add a generic approach to building context-aware applications. Although not all areas will be covered in detail, the framework is flexible enough to accommodate different approaches in the framework modules as proposed by Dey [174] using Context Widget, Context Aggregators and Context Interpreters.

The Context Widget and Interpreter component was omitted due to its specific implementation in converting and interpreting proprietary format of sensor information. Context Aggregators are substituted by the business rules aggregating context information as needed in the business rules scripts.

System developers or system designers will be able to leverage the framework instead of writing ad-hoc specific agents handling specific context in a specific way. Instead this framework enables system designers to use the business object model and defines business rules directly using that business object model without any intermediary steps or code generation, thus reducing interpretation mistakes and errors generating intermediary models.

3.16 Refined Research Focus

Given the considerations made so far in this work, it is important to consider how the framework for our prototypical piece of context-driven software will operate. This paper has considered context so far in only a theoretical domain, so this section will begin a consideration of these concepts and principles in a more practical manner. That said, the following section will offer a brief overview of the principles and characteristics that will define the software to be used in the chapters to come. Chapter 6 will provide more detail of this software, but this section will offer an overview of the framework and structure that will explain the construction of this software.

3.16.1 Context specification

One of the main aspects of the framework is the capability to describe context in its entirety and to be able to describe that context unambiguously. Dey [174] focuses on the description of what context an application needs and deciding what action to take when such context is made available. Our approach dismisses this notion of design time knowledge of which application needs what context and what action to take.

We propose an approach where context is acquired during runtime while the decision-making process is triggered which determines what the decision will be and where more required context can be acquired from. Therefore, this approach eliminates the need to pre-emptively provide context information for a decision process without ever knowing if the decision process will use that context information in its decision making.

3.16.2 Separation of concerns and context handling

Each domain provides its own context ontology which is provided by the caller e.g. the initiating instance, such as a business process (BPMN or BPEL) or an application. All agents are built in a stateless manner with the exception of its own locally stored context and therefore all globally shared contexts and state information is passed to and from every agent.

3.16.3 Conceptual Components of the Research

- **Central Context Registry**

  The central context repository is at the core of the framework, providing all information which is needed to map context to individuals and/or organizations and which allows finding required context at different locations. The information stored in the context repository allows finding the correct software agent holding that particular context.

- **Common model language supported by all agents**

  For an agent to interpret and instantiate model information a mutually understandable model language must be in place. This allows information to be passed between agents without any loss of information.

- **Common language and codification of contextual information**

  All contextual information must be coded to be machine interpretable as well as have a common encoding system to allow sharing a mutual language between agents. This allows for a clear definition of ontological and taxonomic information between agents.

- **Instantiate objects based on model information**

  A software agent must have some capability to create objects based on model information passed to the agent. The same module with the capability to instantiate objects exposes some functionality to access the created objects within the repository during runtime.

- **Access to objects from decision support system**

  In order for decision support systems to run rules against instantiated business objects, business objects need to be registered with the decision support system. Once registered the decision support system is able to access the created business objects to run rules against the business objects.

- **Context information repository**

  Contextual information must be stored in a repository with the functionality of retrieval, storage of context information and possible aggregation of related context.

3.16.4 Framework overview

In Figure 11 we show how the framework is structured and how information is passed between modules.

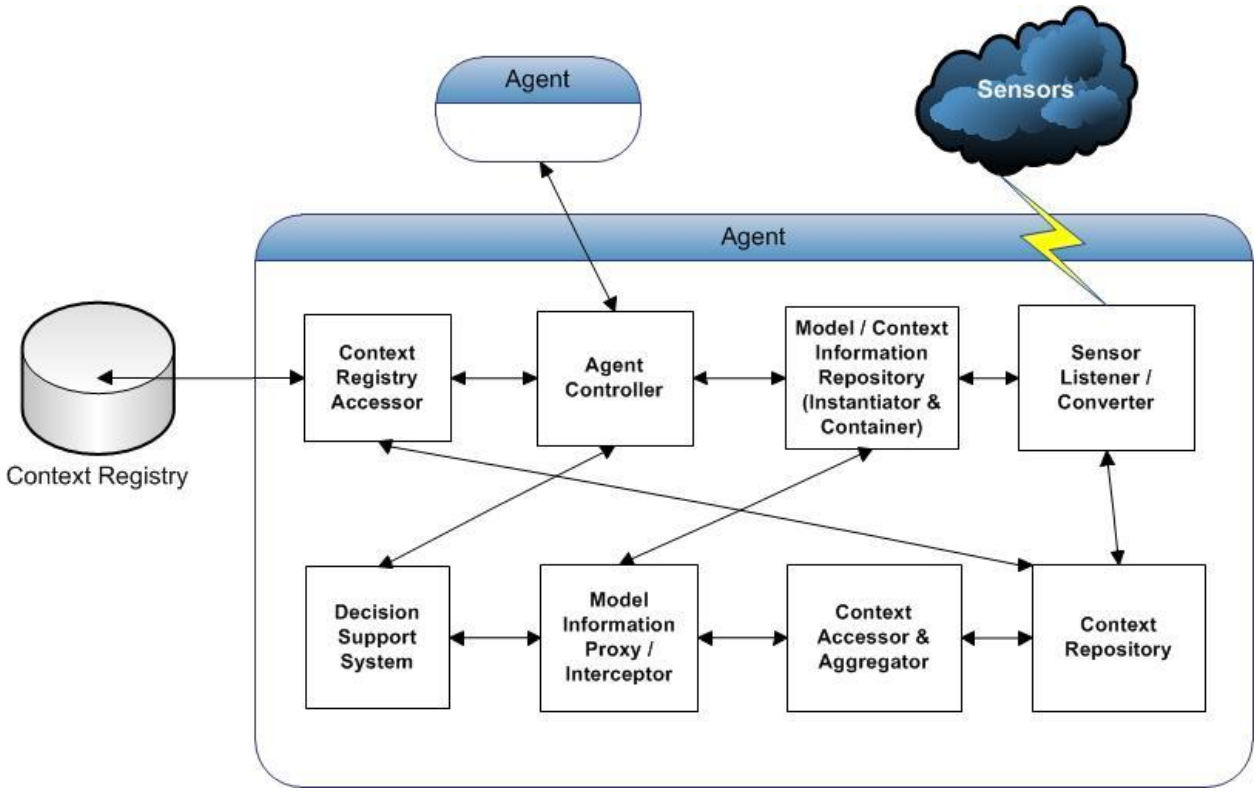

Figure 11 - Conceptual framework

The Agent Controller as the name suggests, coordinates the interaction between the external and internal functionalities. It receives all incoming calls from other agents and controls the sequencing of the different modules inside the agent. In addition it also listens to context changes or additions specific to the agent and triggers the rule processing specific to these new changes in context.

The decision support system wraps business rules components such as Drools, iLog or any other business rules engine. It uses the standard application program interfaces (API) provided by the business rules engine and abstracts the interface into a set of generic function

calls which can be used by the agent. This abstraction allows an easy integration of other reasoning engines.

The model information proxy intercepts calls made to any context or any other model object, and allows for controlled access to functions within the agents. This proxying allows full control of all access to and from the context information enabling runtime monitoring of context and execution control of business rules in other components of the agent.

The context 'accessor' is responsible for providing the information as well as aggregating information if that is needed. Although aggregation such as average, median calculation and so on ought to be made inside the business rule there are instances in which aggregation needs to be modelled and use program code (Java) to aggregate information from lower level information. Although this framework does not promote this approach of aggregating programmatically this functionality could be placed here.

The context repository is a container for all context information stored in one particular agent. For example blood pressure readings would be stored in the context repository whereas the shared model information about the context is stored in the model/context information repository.

As each sensor sends its own protocol containing the sensed data the sensor listeners are specific work units adapted to each individual sensor converting the sensor readings into the commonly understood ontology format. Once converted into the common format the context is then registered with the context registry and stored in the context repository.

3.17 Selected components to fill gap

As discussed in section 3.2 multiple trends, best practices, standards and technologies were investigated to determine if these components were mature and flexible enough to be integrated in the framework proposed in this thesis.

The following table describes which components were chosen to amalgamate into the framework.

Identified Gaps	**Investigated area or discipline to fill the gap**
System design	Several ideas and approaches were borrowed and adapted from Dey [31], Qin [16], Hong [15] and others.
Standardization	Agent technologies (Jade), Semantic Web (OpenRDF's Sesame), Business rules (Drools) and as context directory (UDDI) were chosen to implement the main core components of this framework.
Seamless integration	Agent Technologies were chosen to allow integration into several platforms and to work as an independent piece of software.
Configuration and customization	Semantic Web in particular OWL as an ontology language proved to be powerful enough to express not only business domains and context but to amalgamate them into one comprehensive ontology using Stanford's Protégé Editor alongside with the ontology instantiation engine of OpenRDF (AliBaba).
Context exchange	To describe and exchange context information OWL was used due to its advantages previously described.

3.18 Chapter reviewed

In this chapter we have highlighted different concepts from different sources and described what approaches have been adapted into this framework. The difficulty of this research is that it encompasses different approaches such as the adaptation of UDDI to accommodate the agent's user context registration and the user context's relationship to other entities. Furthermore, this approach allows us to accommodate different technologies and concepts for example, to change how MDA is used classically and how it is adapted to runtime interpretation of the agent, the integration of local reasoning (in this approach business rules) to validate the problem domain as well as the definition of context and the different aspects of how context is stored and retrieved.

4 Methodology

4.1 A Systems Development Approach

Over the last 2 decades a significant number of researchers have pioneered Design Science (DS), yet little has been done within the DS discipline. This can be largely attributed to the lack of a common methodology and a common framework for DS researchers as well as a common presentation template that may have contributed to the slow adoption of DS [228].

Beginning in the early 1990's, when information systems researchers started to develop an interest in design science (DS) research, there was already agreement in prior research about the basic difference between DS and other paradigms, such as theory building and testing, and interpretive research: "Whereas natural sciences and social sciences try to understand reality, design science attempts to create things that serve human purposes" [228, 229].

Information systems, clearly serving human purposes, developed as an applied research discipline which frequently borrows theories from other disciplines, such as computer science, economics and social sciences to solve problems where information technology and organizational issues intersect. However, the dominant research paradigms remain the traditional descriptive research borrowed from the social and natural sciences [228].

Design science also has a long history in Europe where design science dominates in the German speaking countries. A large number of design science researchers can be found in other European countries, particularly Nordic countries, the Netherlands, Italy and France, to name the biggest [230].

According to Hevner et al. (2002) and the 'Editorial Statement and Policy' of Information Systems Research, researchers in the information systems (IS) discipline conduct studies to "further knowledge that aids in the productive application of information technology to human organizations and their management" [231, 232]; IS researchers also aim to develop and communicate "knowledge concerning both the management of information technology and the use of information technology for managerial and organizational purposes" [232, 233].

Hevner further argues that it is “an opportunity for IS research to make significant contributions by engaging the complementary research cycle between design science and behavioural-science to address fundamental problems faced in the productive application of information technology. Technology and behaviour are not dichotomous in an information system. They are inseparable. [234] They are similarly inseparable in IS research. Philosophically these arguments draw from the pragmatists [235] who argue that truth (justified theory) and utility (artefacts that are effective) are two sides of the same coin and that scientific research should be evaluated in light of its practical implications” [232].

Hevner et al. (2004) add that the goal of natural science is truth, whereas the goal of design science is utility. An artefact which initially seems useless may have utility because of some as yet undiscovered truth [232]. Weber recognized that IT research is the study of artefacts as they are adapted to their changing environments and to changes in their underlying components [236].

Not every artefact construction, however, is design research. ‘Research’ implies that problem solutions should be generic to some extent, i.e., applicable to a set of problem situations. The trade-off between the level of solution generated and the problem scope is addressed by situational artefacts. A unique solution is not applicable to many problem situations without generalization, which diminishes its solution power. Situational adaptations should be incorporated in order to preserve application value (i.e., solution specificity) while covering a broad problem scope [230].

March and Smith argue that research is divided into two distinct paradigms, natural science and design science. Natural science is concerned with how and why things are and includes traditional research in physical, biological, social and behavioural domains. Design science, in turn, attempts to create things that serve human purposes, in other words to create utility and to attain goals [237].

March and Smith further argue that design science consists of two basic steps – build and evaluate and these two steps parallel the discovery (known as justification) steps in natural science. These two steps are defined as “Building is the process of constructing an artefact for a specific purpose; evaluation is the process of determining how well the artefact performs.”

Evaluation is complicated by the fact that performance is related to intended use, and the intended use of an artefact can cover a range of tasks [237].

Nunamaker and Chen (1990) argue that in cases where a new way of doing things is proposed, researchers may develop a proof-of-concept prototype system to demonstrate the validity of the solution, based on the suggested new techniques or design. Constructing a system instantiation that automates a process demonstrates that the process can, in fact, be automated. Such automation would provide "proof by construction" [238].

Nunamaker et al. (1990) also advocated the integration of system development into the research process, by proposing a multi-methodological approach that would include theory building, systems development, experimentation, and observations [238].

It is important that we consider the mathematical background of the concept of proof by construction. According to Franklin and Daoud (2011), a constructive proof demonstrates the existence of some bit of mathematics (an object) which has certain properties. The methodology of an experiment that would prove the existence of this mathematical object would work by creating and proving a method through which that object would take form. This source explains that this is in marked contrast to a non-constructive proof, which is a mathematical model which proves the existence of a mathematical object with a certain body of qualities and variables, yet which provides no evidence of a feasible methodology for constructing that object. These non-constructive proofs can often be made by proving a negative, that is, through assuming that the object in question (or its opposite) is non-existent in reality, and use that contradiction to prove their point [239] .

The mathematical model, valid in its own right, is proving a negative or a contradiction in mathematical terms, in which no artefact is being produced. We concluded that the approach of proving a negative is not suitable and in line with the 'constructive' model outlined here.

4.2 Validity of Design Science Research

Design science research is increasingly recognized as a research methodology in the field of information systems. The contributions of design science research are in the novelty and utility of the new constructed artefact. IT artefacts are largely defined as constructs (vocabulary

and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations (implemented and prototype systems) [232].

A conference on design science research in information systems and technology (desrist.org) held in 2010 by the Institute of Management, University of St. Gallen states that: "Design Science Research is becoming firmly established as a key research paradigm in mainstream information systems research" [240] and that "the importance of design was first brought to light by Herbert Simon's famous *'Sciences of the Artificial'*. Since then, an entire body of research embracing goals to design and invent innovative artefacts such as constructs, frameworks, models, methods, processes and systems has emerged. The creation of (IT) artefacts and their evaluation is central to the Design Science Research paradigm" [240].

Zhao et al. (2010) stated that: "As opposed to natural and social research, design research does not crave ultimate truths, grand theories or general laws, but seeks to identify and understand real-world problems and propose appropriate, useful solutions. It is commonly believed that design research involves building, investigating and evaluating innovative artefacts such as constructs, frameworks, models, methods, and information system instantiations in order to solve practical problems. Moreover, the study of methods, behaviours, and best practices related to the problem analysis and artefact development process is encompassed commonly referred to as design science" [241].

The construction process and creation of innovative IT artefacts enable the design science researcher to comprehend the problem addressed by the artefact and the feasibility and utility of the approach to the solution [238].

4.3 Different Design Science Approaches

Looking at different design science approaches in Figure 12 one can see that most design science approaches can be reduced to 3 main steps.

Objectives for a design science research process model	Archer (1984)	(Takeda et al. 1990)	Eekels and Roozenburg (1991)	Nunamaker et al (1991)	Walls et al (1992)	(Rossi et al. 2003)	(Hevner et al. 2004)
1. Problem identification and motivation	Programming Data collection	Problem enumeration	Analysis	Construct a conceptual framework	Meta-requirements Kernel theories	Identify a need	Important and relevant problems
2. Objectives of a solution			Requirements				Implicit in "relevance"
3. Design and development	Analysis Synthesis Development	Suggestion Development	Synthesis, Tentative design proposals	Develop a system architecture Analyze and design the system. Build the system	Design method Meta design	Build	Iterative search process Artifact
4. Demonstration			Simulation, Conditional prediction	Experiment, observe, and evaluate the system			
5. Evaluation		Confirmatory evaluation	Evaluation, Decision, Definite design		Testable design process/product hypotheses	Evaluate	Evaluate
6. Communication	Communication						Communication

Figure 12 - Design Science process elements from IS other disciplines and synthesis objectives for a design science research process in IS [94]

The first step is the identification of the problem, the second step consists of the building of the artefact and the third step is the demonstration and evaluation of the artefact. These 3 main steps are shared amongst the most prominent approaches of design science research.

4.4 Design Science Research Methodology

Due to its goal of building a prototype in the health domain, this research will adopt Nunamaker's systems development research methodology. This constructive methodology can be best expressed using the following diagram:

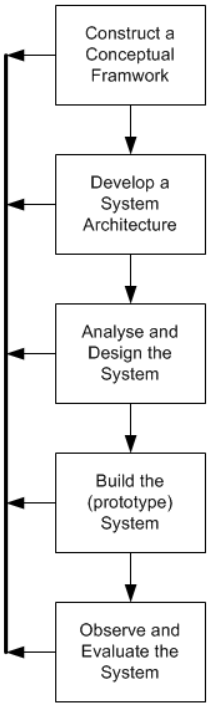

Figure 13 - Research methodology (Nunamaker, 1991)

As discussed earlier in section 4.3 Nunamaker's five steps methodology can be further reduced into three main steps:

1. Concept Development for problem identification and motivation,
2. System Building (Artefact creation), and
3. System Demonstration and Evaluation.

The following section will consider each of these concepts as they relate to the overall methodology of this thesis.

4.5 Concept development

This section is divided into the following four aspects in the construction of the application framework project:

(1) This project must reflect a strong understanding of the issues of data and context and understand the design issues of context, as well as how context can be modelled and shared among multiple participants. This can be achieved through use of various models as well as modelling which takes into account different types of end-users. As described by Yu, Kangas, and Brewster (2003), systems for the visually impaired, for instance, offer huge potential for growth in upcoming years, yet these systems still fail to address the differences between visual impairments and contextual environments, while offering little context-awareness, particularly for users who have lost central and peripheral vision, as well as total vision loss. The methodology followed in this research tries, to some extent on a conceptual level to take into account the contextual basis of all types of users [242].

(2) This model must also produce a reusable business object model which can be used by every CASA. As described by Johnson, such models can take many forms, including those which improve business transaction processing or improve workflow. Of particular importance are the business models which can manage the information and code, while interfacing with the GUI or some database. Examples of these systems are often based on a Dynamic Object-Model. The final model would consider all elements of the business object model in an attempt to show that this model would provide a coherent distillation of all areas in which business objects are used, whether or not this use is in tandem with user input [243].

(3) The model must produce an agent interface which allows business rules, business object model and contextual data to be seamlessly exchanged. At this stage of the thesis, this work will present as the proof-of-concept the software agent which underlies the concept. This will be the proof-of-construction, as well as the most salient aspect of the thesis. This software agent will interact with the underlying system and offer users an efficient way of accessing the underlying objects, facts, and data.

(4) Finally the work will specify which theoretical model best reflects the characteristics of the system that has been described in the preceding steps.

4.6 System building

System building is divided into 3 Phases:

- Phase 1 focuses on analysing the system and building a healthcare use case.
- Phase 2 consists of selecting a business rules implementation. E.g. Open Source such as D-Jess, Jeops, Drools, JRuleEngine and many more [244] or commercial ones such as Oracle Rules or ILog. Furthermore, this step requires determining the feasibility of coupling between the Business Process Management System and Business Rules Engine, and determining to what extent business rules must be decoupled or designed in terms of function granularity, separation of rules which can be run independently from each other and classification (salience) of those rules. In addition, it is necessary to prevent any dependencies on higher level business processes in order to fulfil the requirement of being an atomic transaction and not process dependent to allow simple integration into business processes.
- Phase 3 aims at creating a framework based on the conceptual model which enables business rules to be deployed and executed in a distributed environment using the criteria determined in the conceptual model as well as a working implementation of that model.

4.7 System evaluation

This step of the process will consist of implementing a previously defined use case in the framework created in order to examine whether the framework accommodates all required functionality for the use case.

4.8 Justification of this Methodology

Research is essentially about discovery: It is a process of posing questions, finding methods, and coming to important solutions. When the desired outcome of that inquiry is informed action or the production of a specific outcome, the context is appropriate for a design methodology approach. There are several areas of research and design philosophy that must be considered in order to properly justify the current methodology that has been considered throughout this chapter. This justification will consider primarily the terms 'ontology' and 'epistemology', with ontology the study of the nature of reality and what is real, and epistemology considering the nature of knowledge and the certainty with which knowledge is known to be 'true'.

4.8.1 Qualitative and quantitative research

There are several other areas of research methodology which can be contrasted with the 'design science' approach used in this work. It is first important to present this methodology, as well as justify this approach, particularly as it relates to the design sciences, and relate this point back to the reason that the following work will not be using either qualitative or quantitative methodologies in this exploration of software as they relate to different contexts.

It is necessary to begin by considering qualitative and quantitative research methodologies. The value of, as well as need for, objective and methodologically sound research is without doubt. Only through sound research designs can we eliminate threats to validity and draw scientifically valid conclusions, and this is the truest path to well-informed conclusions and proper practice in all fields.

While there is considerable value in traditional research, a more interactive, reciprocal research and development model, one which combines evidence from previous research studies with researcher’s knowledge plays a critical role in conducting this research.

A good evidence-based research design is a mixed method design that integrates qualitative and quantitative research. This type of design begins with a strong research methodology with quantitative methods that are enhanced with qualitative measures of key processes and outcomes. Qualitative methods, such as interviews and case studies, improve the

design by providing data that can give insights into how findings work and how findings can be translated to practice. By itself, a quantitative method can identify what works, but has limited explanatory power: there is little information about how students learned and how instruction worked, for example. With qualitative designs there is rich information about learners and teaching, but the information about what worked is more subjective and cannot be generalized. By combining the two methods, we can obtain a much richer understanding. In other words, using a rigorous design the quantitative methods can tell us what works, while the qualitative methods can tell us how it works.

In this context, while a mixed-methods approach may yield considerable information with regard to how a given context-aware computing system may operate, these methodologies are less relevant with regard to how to program and establish such a system. While a mixed-methods approach, or even a qualitative research approach, may be relevant at a later date, particularly after the software is being implemented in healthcare and other consumer/professional domains, this rigorous and scientific approach to computing and programming demands a different methodology. The 'design science' approach that will be described and used in this work is far closer to quantitative research in its methodology, albeit with major caveats and deviations from this more common model.

4.8.2 Design Science

By definition, any research which is conducted from a design science perspective must have a strong body of certainty at its core, as it involves the introduction of new artefacts and perspectives into the world. From an epistemological perspective, the design researcher will have a strong foundation in certainty and in real fact, while understanding what a given piece of information means through a comprehensive process of construction and circumscription. That is, an artefact (or new piece of software) will be constructed out of whole cloth, its behaviour and perspective provided to the user is understood as the result of interactions between its core components, and the nature of these interactions are well-understood by the individual who is conducting the design process. For the design science researcher or developer, the behaviour of the final product is both the ultimate goal of the research process, and the functionality which is embodied in the final result of this process is the defining characteristic of that novel artefact.

The meaning and the justification of the final product is the user-based purpose of the final product.

In short, while the end result of a design science development effort may be easily understood, it is the methodology which surrounds the creation process which is often difficult to understand and comprehend. The design science developer may add a practical or functional addition to an existing product or body of knowledge, but the fact of the product and its design and success in that design, is the only true metric of success in this regard. As Zhao et al. (2010) describes, while this product may provide the basis for further exploration, all that is required is a successful project when considered from a design science perspective [241]. In this regard, the justification of this methodology when considered from this perspective is the same as the governing philosophy of design science itself: The point of this development project is to generate a body of software programming that exists to fulfil a certain end-user requirement. The goal of this work is the same as the product of this work, so for that reason, the methodology, and indeed the specific route which is taken to fulfil this design requirement, is not as important an element as the final product which is created.

This same author [241] explains that design science researchers will create a reality through constructive intervention, but upon the creation of that 'reality', or product, effectively become a passive yet positivist observer of their own product. That is, after the creation of their work, they will step back to record the behaviour of the system and compare it to their initial expectations and goals for their work. In effect, the path to the goal is not as important under this methodology, effectively the 'methodology' is not as important under this design philosophy, as the elements which bookend the 'work' which goes into the project itself. The design scientist will consider the project at hand from the perspective of the observer, both at the front end and after the work is done. The initial (abductive) phase will give way to the interpretations of observations and understanding of the ultimate impact of the software being created, as compared against the initial expectations of the software. This then gives way to new processes of theorizing and new interventions, but ones which focus on the creation of a new product which becomes the focus of the next round of development in its own right. In this sense, design science research shares a great deal with action research methodology in its consideration and interpretation of various elements, but differs in that the design research perspective demands far

less consideration of the individual elements of construction. Instead, the work reflects an artful consideration of the often-haphazard, though inspired, elements of design. These are crucial to any consideration of programming as an art form, as it is under a design science methodology.

Ultimately, design science is the choice of methodology for this work because it allows for the greatest procedural freedom in the pursuit of a goal. While a different perspective, perhaps action research, would allow for the consideration of many different perspectives in development, another research methodology would not afford this designer the kind of freedom which is inherent in design science research. This perspective, with its focus on the end-result (and the user experience), and on experimentation by goals, rather than by degrees, is a methodology which allows for the greatest possible testing and consideration of different factors in advance of a goal. If the goal is not achieved, or if the ultimate user experience differs from that which was initially intended by the developer, the design research mentality does not penalize the developer. This flexibility is its strength, and the ultimate reason why it was chosen as the foundational methodology of this development work. While incremental advancement offers many advantages to the developer, or team of developers, only design science development provides the strong freedom which is most necessary for the kind of software development used in this research.

5 Systems Design and Engineering

5.1 Introduction

Now that this work has presented a framework for the software, it is time to consider the software itself. As the focus of this research is on determining the format of how context must be best defined for the purpose of distribution, as well as storage and service directory querying, the following chapter 6 will consider the relevant software components in granular detail. This chapter will begin with a consideration of context classification that will be necessary, particularly in regard to context attributes within a given organization, and then head into a solution for methods of how to store such information in a directory that can be utilized by software which is part of a federated context-aware architecture. Chapter 6 will also consider a series of business object models (BOM) and how they can be best tacked on to the current piece of software in a way that allows them to be properly added to a context-aware model using the OWL structure, as well as the conceptual framework which was described in detail in section 3.13.

This chapter will begin with a discussion of the various assumptions that were made throughout the construction of this software, and which will have a significant impact on the resulting software. These include both OWL ontology assumptions and those which lie with Business Object model information, and show the differences between the following framework assumptions and the MDA approach. This chapter will also consider limitations of the framework and the programming of the centrepiece software, limitations which will be generally informed by considerations of both other frameworks and best practices. In short, chapter 6 will focus on showcasing an alternative approach to mobile decision making which places context at its focus and which allows for processes to be executed at the location where context is most easily accessible.

This chapter will consider the agent environment, namely Jade 3.7. In addition, this work will explore the difficulties of converting between OWL and UML , as well as the shortcomings and difficulties inherent in each of these 2 approaches to the overall design of the centrepiece

software. This work will consider the reasons why we chose to use Stanford University's Protégé Ontology Editor, version 4.1 Alpha, and identify the reasons why this version of Stanford's standard ontology editor has become the standard which is used by many others in this field. In addition, this work will consider why we chose to use Openrdf.com's Sesame in version 2.3.1 as our OWL runtime component, as well as why we chose to use a Universal Discovery and Directory Index (UDDI) called Oracle Repository over many other interesting and qualified choices.

After this consideration of our framework, this chapter will shift into an implementation mode, describing the steps and decisions involved in developing the software created as part of this thesis. The centrepiece of this consideration will be the context registry, that is, the piece of the software that allows users and agents to register data to be used in the context-aware software processes. We will provide evidence of the UDDI taxonomy we authored toward the production of this context registry, along with considerations of keyValues and methods which will cause registration of unrecognized values to fail.

It is also crucial to be able to resolve context between related partners. In the health context, this may come down to the difference between a general practitioner(GP) and the patient, both of whom have relevant status in the system, and who should be allowed to contribute to the context-awareness elements of this system, but who also have a relevant contextual relationship with one another. These interdependencies between GP and patient resolving user context require the modelling of a certain relationship 'signature'.

This work will then consider the benefits provided by Stanford's Protégé as it relates to the integration of the various ontologies. This consideration will include a discussion of the primary AgentRoot ontology which is the primary focus of the software during runtime. This chapter will show the flexibility of the framework using the example of a generic bit of context which will be shown to integrate into the primary software with a maximum degree of flexibility.

This work will then move into a discussion of the domain ontology which we are using in this software, as well as how to best merge these ontologies with one another using Protégé. For instance, how ontologies from different sources can be merged into one ontology and how to

reuse existing work from other projects, such as the conceptualization of a generic context model in OWL from University of Basel [245] and the software Agent's own ontology.

This chapter will then consider both mapping of context (under UDDI taxonomy) and the routing paths with regard to a basis of context storage in the groupings organized by the software author.

Following an exploration of the various benefits to be gained from use of this agent, this work will move on to an exploration of a runtime example. This use case example will include an exploration of the various agents at play in a typical healthcare use scenario. These will include a starter agent, a general practitioner agent, and information which includes but is not limited to a blood pressure history. An explanation of the running example and an evaluation of these results will follow. At this time it will be important to consider the difference between the model agent program provided here and the current state of healthcare. While this has been considered in other chapters, this context will provide a reminder of the current limitations of medical software, as well as the current problems facing the industry. This consideration of the current healthcare approach will showcase the utility of this program. Conclusions and implications for future research will follow.

5.2 Reiteration of Research Focus

This framework puts user context at the centre of its functionality. Context available in different locations is registered in a central registry, made available to all others. This framework uses semantic web to describe context in a uniform way understood by every other user of that context as well as to describe the problem domain through semantic web ontologies. In this particular framework decision making is covered through business rules which interact with the problem domain model.

The focus of this research is on determining the format to best describe context for the purpose of information distribution, as well as for storage and service directory queries. In addition, some form of context classification is necessary to determine the relation of context attributes with another person or organisation and to be able to store that information in a context-aware service directory [25]. Another aspect will be how to define a common schema of

business object models (BOM) across the different business rules and how to store and retrieve the information.

So far, most previous chapters have explored context-aware architectures, service discovery, orchestration and compatibility, while little focus has been on context-aware agent synchronization [9, 25, 246-248] let alone context-centric software agents using an MDA approach in a context-centric distributed environment.

Due to the selection of only just one business rule engine across all actors/platforms, the complexity of translation between business rules can be ignored. Although we are choosing a business rules engine, this choice does not imply that any other reasoning engine could not be used. One obvious choice would be to choose an inferencing engine supported in many projects such as Jena [249] or Openrdf's Sesame [221]. This framework is aimed at being modular and allows for modules to be replaced with alternative solutions. Emphasis is placed upon a good design service granularity in terms of service reusability and composition as well as business atomicity and last but not least a solid architecture enabling the above issues.

5.3 Approach to Methodology

To best support the context-centric model driven agent approach we must identify the required features for the framework. At the core of this framework lies the intent to allow application developers or business analysts to develop an application by design only and deploy the application without the need to actually generate and/or compile the source code.

These framework features are supported by the context-centric model driven agent approach, by this framework and by adding a generic approach to building context-aware applications. The parts of dynamically sensing context along with context conversion, context aggregation and mapping context to taxonomies have been intentionally left out because these context readings are very specific to sensors, mobile devices and other devices and can be easily integrated as part of a project. Although these areas are not be covered in detail as part of the software solution, the framework is flexible enough to accommodate different approaches in the framework modules as proposed by Dey [174] using Context Widget, Context Aggregators and Context Interpreters or an alternate component which triggers configurable scripts converting the

context information into the required format. System developers or system designers will be able to leverage the framework instead of writing ad-hoc specific agents handling specific context in a specific way. For the purpose of this use case implementation we have created OWL files containing the necessary context which is then passed to the software agent during runtime.

5.4 Review of Applicable Research Methodologies

5.4.1 Assumptions

As previously discussed, we focus on context as the centrepiece of all agents. Context is described in an OWL Ontology alongside all necessary metamodel, Business Object Model information and facts (OWL individuals or Business Objects). Along with the creation of the OWL Ontology, a UDDI taxonomy is created to be able to map the context tags in the UDDI to the OWL Ontology. Furthermore, all agents are not classic goal seeking agents but agents with the capability to instantiate the given metamodel and facts information and run business rules against them.

This framework is generally different from a classic MDA approach which focuses on generating code from the model data. Instead this framework focuses on interpreting the model data and instantiates the OWL Model and OWL individuals in a runtime environment at the beginning of each call to the agent. If the OWL model is out of date the framework generates Java access classes on the fly based on the OWL model to be able to access the instantiated OWL individuals as normal Java objects.

5.4.2 Limitations

This framework does not aim to provide an alternative for the entire MDA stack but rather to provide an alternative reuse of existing approaches in terms of business modelling and decision making. The framework does not aim at providing a workflow and activity diagram replacement. Instead, this work aims to accommodate best practices and aims to be integrated into other frameworks such as BPMN applications or any other process orchestration software. The gathering of contextual information is not part of this thesis; we propose to use widgets as proposed by Dey et al. [31] which handles particular sensor information in widgets, aggregators

and interpreters. It facilitates the separation of application semantics and context metadata from low level input details.

5.4.3 Research aims and objectives

This research focuses on showcasing an alternate approach of mobile decision making putting context in the centre. The developed framework allows for processes to be executed at the location where context is made available. This is in contrast to classical approaches that request context, through web services on central servers, local application programming interfaces or any other form of remote procedure calls (such as CORBA, Java remote method invocation and so forth).

5.5 Tying it all Together

The preceding section has considered several different variables with regard to context-sensitive computing and the different methodologies that surround the usage of this type of computing in the runtime environment, and which make use of a design science-centric methodology. The following chapter 6 will tie together these elements and consider each of these elements in a wider context of the application of design science methodologies. In order to consider each of the preceding elements in turn, it is necessary to review the elements of design science research that are on display, which necessitates a full review of the tenets of design science.

As described by Vaishnavi et al. [250], design science as a research paradigm differs from traditional research because this is a methodology and approach to systems engineering which is fully based in the multi-paradigmatic world of information and communications technology. As there is far more than one simple way to go about constructing an information system, it is necessary to consider these human-oriented systems in a way which is focused, but which acknowledges the necessary chaos in the design process. Nevertheless, these researchers argue that design science will, of necessity, require a thorough and complex grounding in many different research philosophies. The key element that breaks design science from other more traditional methodologies, and particularly those which are not found in the world of information systems development, is the reliance upon and importance of concrete results and evidence of

success: As the ultimate product of this research, particularly in this work, with context-based computing, is a system which people will be using, it is mandatory that the systems methodology produce a system which is concrete in its final deliverable [250].

Other researchers, including Zhao et al. [241] explain that the design science approach to constructing an information systems product is ultimately about the results which are provided by the system alone, and argue that the strict methodology which allows for continuous improvement of all areas of the system, is often less important in the overall process. These researchers cite the example of boats and bridges as extraordinarily important and beneficial technologies which were used by people long before the mechanics and physics that surrounded their development, and ultimately the rules which predicted their behaviour, were contemplated and understood by people in a substantial manner. In the same manner, the relative chaos with which people will assemble information systems is a descriptive factor in this regard: While the programmer will ultimately seek to understand every element of the system, design science demands that this programmer ultimately not lose sight of ‘The forest for the trees', and make sure to concentrate on the concrete deliverables that are provided by the system [241].

In this case, while there are many ways in which context elements can be used to develop a specific runtime environment, the specific techniques and procedures that result in this outcome are not always clear, or are difficult to present in a clear way. While this work will attempt, and has attempted to show these procedures, through 'flow' graphics for instance, this is only one, admittedly flawed, way to describe the procedures inherent to the construction of such a system. Ultimately, while design science methodology requires a strong basis in methodical processes at the front-end, it is the *product*, the key deliverable and ultimate user environment, which is the goal and objective of any programming effort, and particularly one governed by tenets of design science.

The methodologies considered throughout this chapter must all be considered in the context of the context-aware mobile user experience, yet each of the elements that have been considered throughout this chapter, including the agent environment, aspects of the semantic web, the modelling software, OWL, UDDI, and other elements, must be considered in the context of a design science 'philosophy'. That is, the agent environment and experience is the

ultimate factor which each of these other elements works towards, so the specific processes which determine this programmer's use of the various elements (as well as the methodologies and understanding which determines the individual use of these elements) must be considered secondary to the larger dynamic and ontological process which ensures a comfortable user experience. In effect, while future programmers may seek to recreate the methodological elements which have resulted in the creation of this user experience, they must begin from the user experience and seek to reform various elements of that experience.

According to design science methodology, future programmers will not begin from the individual elements (OWL and UDDI, for instance), but rather from 'reverse engineering' the existing properties of a system which needs to be changed and reformed to comply with the new realities and exigencies of the user runtime environment. Particularly with context-sensitive computing, there is a strong demand for elements of software and design which place the user experience (and reflecting elements of the user reality) on a pedestal, and argue that the user experience is the most important element of the software product being created.

By devoting less time and energy to careful transcription and consideration of the various processes involved in creating software, the design science methodology applied in this work will help not only to ensure that the final software created in the runtime environment is one which places context awareness as its primary goal but also exposes a new approach to context awareness in terms of a federated approach i.e. stateless design along with local reasoning. This does not mean that there is no place for 'ground-up' software engineering, but rather that future software based on this software architecture, will begin by first considering elements of the federated approach that was not considered by this work, and go forward and program through first correcting for the missing features of this framework.

6 Implementation

6.1 Introduction

After this lead-up, this chapter will consider the system design and implementation, beginning with a thorough consideration of the various packages used in this work. The centrepiece of this software is the ch.pesto.agent, which orchestrates all operations of the agent module and which triggers all important functions of the program. This work will consider the various functions of this agent. This chapter will also consider (1) ch.pesto.owlcompiler, (2) ch.pesto.classloader, (3) ch.pesto.registry, (4) ch.pesto.uddi.tools, (5) ch.pesto.repository.context, (6) ch.pesto.rules and (7) ch.pesto.proxy. Each of these elements is integral to the presentation of this agent and its proper function.

This chapter will then continue into a discussion of the methods which will describe the ways in which business rules are incorporated into this body of context-aware software. This will include a discussion of rules sequencing, as well as properly-timed rules execution, and how the business rules are stored, in relation to how such storage relates to agent action. Specific code will be presented that will show functionality and application in Protégé relevant to the use case. This will lead to a consideration in greater detail of the Agent Runtime Environment, including explanations of which elements of this environment will be held in a stateful and stateless manner, as well as a consideration of various serialising and deserialising processes.

6.2 Work Plan

6.2.1 Implementation

In the previous chapter we described the different components used to build the framework. This chapter will focus on how these components interact with each other, why they were chosen and where context and business model ontology are being used.

6.2.2 Semantic Web components

6.2.2.1 Modeller / Designer

Initially we looked into using an industry standard UML Designer and converting UML models into OWL-S. After investigating the feasibility of UML – OWL-S translation, we came to conclude that one standard does not allow a clean translation from design approach to another. Some major problems consist of features being not completely implemented or of features being entirely missing in either UML or OWL.

The following are examples of functionalities missing in UML [216]:

- Unique name assumption within a namespace
- No support for predicate definition such as intersectionOf, unionOf, complementOf
- And other not yet finalized developments such as SWRL Semantic Web Rule Language and OWL services

And some example functionalities missing in OWL [216]:

- Behavioural and Related Features which declare capabilities and dynamic aspects of the system
- Complex Objects which support various flavours of the part-of relationship between classes (aggregation, composition)

6.2.2.2 OWL modeller/designer

Due to aforementioned translation problems we opted for a pure implementation of an OWL. The OWL modeller of choice became Stanford University's Protégé Ontology Editor 4.1 alpha version [83]. Protégé has become a widely accepted standard modelling ontology. It allows domain experts to create and modify reusable ontologies as well as to integrate problem solving methods. Protégé is used to generate applications and domain-specific knowledge acquisition tools from ontologies. Protégé OWL Editor provides functionality to define classes and class

hierarchies, slots and slot-value restrictions as well as build relationships between classes and to create properties for these relationships [251].

In recent years, Protégé has emerged as the de facto standard among developers who are seeking to develop ontology-based software for use on the semantic web. As a plug-in extension to the already-extensive Protégé ontology development platform, this program is one of the most useful and effective tools for the creation of context-aware software. Given this consideration, it is important to consider the benefits which are provided by this platform, as well as the ways in which it is superior to its competitors. According to Rubin et al. [252], the chief reason why Protégé OWL is of such enormous benefit to users in the healthcare and bio-medicine fields is that it allows that "many of the problem-solving tasks they seek to automate can be construed as classification tasks." In addition, the wide variety of storage formats, data acquisition and visualization tools, and graphical user interfaces (GUIs) provided in Protégé has seen it come to the front of the pack with regard to available software frameworks for existing problems in the context realm.

This section will describe the advantages of Protégé in greater detail, as well as the usefulness which is provided by OWL in conjunction with Protégé with regard to the purposes of this larger project. First, it is important to note that OWL, or the Web Ontology Language, has recently become a standard-bearer of sorts for the categorization and organization of the semantic, context-aware web. As developers of intelligent applications in biomedicine face significant challenges with regard to representing, managing, sharing, and reusing the extensive base of knowledge that is required by their systems, there is a need for both a common context-driven language and an easily-understood graphical language that can be used to understand it. While this field is still ruled by reasoning systems which make use of inefficient problem-solving approaches based on outdated reasoning methods, these older systems, particularly in the healthcare realm, don’t focus intensely on context and user cues. Rubin et al. explain that these systems are "difficult to build and maintain because knowledge is contained in domain ontologies as well as in the application code," so for this reason, comprehensive methodologies that take advantage of contextual information and present it in a way that is easily-understood, become highly important and necessary [252].

To that end, this project has decided to use Protégé not least because it is free and thus easily-disseminated. It is also an easy to learn piece of software, and can be run on multiple platforms because it runs in Java, therefore can be used on a wide range of computers. In addition, benefitting from the Stanford community and internet users around the world, Protégé (and OWL) have an active support community of hundreds. This community has proven invaluable in supporting this research and the goals that we have met over the course of this project.

There are other reasons why we have chosen to go with Protégé for this project. We have found that not only is this program highly efficient and well-suited for creating effective GUIs, but the methodology which surrounds its API allows this program to be seamlessly integrated into other applications and toward other purposes. In addition, the expression editor is well-suited for assembling different expressions, and the defined classes which come with this program are highly useful for constructing simple organization trees. Finally, the 'wizards' that come with this program are also very useful with regard to streamlining tasks which are highly complicated, such as creating groups of classes, causing a set of classes to disjoin meaning preventing classes to be added to which are mutually exclusive to each other, or creating a matrix. In particular, the support for OWL provided by this program is superb.

6.2.2.3 OWL Compiler and runtime components

As the underlying OWL runtime component we used Openrdf.com's Sesame in version 2.3.1. It can be seen as providing all required functionality similar to Jena which is integrated in Protégé. Jena was considered as a framework based on its tight integration with Protégé and its wide adoption. Nonetheless, the Jena open source project was abandoned by Hewlett Packard and subsequently a new open source project was launched to continue support for Jena as OpenJena with many members joining from the old project team.

Nonetheless, due to the interruption of development of Jena many add-ons would only work on older Jena versions. To make matters more difficult several needed add-ons would not support the same Jena version making it nearly impossible to use Jena at this stage to complete a working framework. Several other Semantic Web components were evaluated but later dismissed to due to their overall lack of functionality and progress in development. One very promising

implementation which later had to be dismissed because some implemented functionality in the area of object instantiation from OWL to Java was incomplete was the open source project OpenAnzo.org.

To best support most required functions in the semantic web realm, we used openrdf.com's AliBaba 2.0 alpha 4 version as an OWL to Java compiler and an Object Repository. AliBaba compiles the given OWL-S information and generates Java classes which are then used in the Object Repository to map the semantic web information to java classes and provide an easy programmatic access. It maps Java objects to and from RDF resources and OWL classes to Java classes in a non-intrusive manner that enables developers to work with resources stored in an RDF Repository as objects. The Object Repository is an extension to the Sesame RDF Repository that allows an RDF store to function as an object store [220].

6.2.3 Universal Discovery and Directory Index (UDDI)

We initially expected to use Apache's jUDDI [158], due to its wide open source adoption and acceptance in the community. At the time of implementation we discovered that jUDDI in version 3.0.1 was not adhering to the UDDI v3 specifications and was supporting alternate namespace names other than those defined in the specification. This problem was later rectified after escalating this issue with the project team in version 3.0.2. Due to this inconsistency with the UDDI v3 specifications it was therefore decided to opt for a commercial grade UDDI repository software called Oracle Repository which is fully based on the previously open source Systinet WASP UDDI server version 6.5. Systinet's UDDI implementation has been the de facto standard for years. Systinet has since been taken over by Hewlett Packard and integrated in their SOA suite and is no longer Open Source.

In addition, Oracle Repository fully supports all necessary functions such as resolving relationships between UDDI entries which is a crucial part for validation purposes and uniquely resolving for required context registered in the repository.

6.2.4 The context registry

At the centre of this framework is the Context Registry, allowing all agents to register particular contexts in relation to another person or organization (in future referred to as a business entity). No context is stored in the Context Registry; every agent only stores the availability of the context it holds about a business entity. Every agent retrieves and persists context in itself or knows on how to persist and retrieve that information to and from an existing context storage. Every business entity must be registered in that registry for this framework to create interrelationships between related entities. Every entry in the Context Registry is given a unique key when created, which is used to uniquely identify an entry representing a business entity.

This framework emphasises interchangeability of the individual components in the framework. We have chosen UDDI which was originally designed as a directory for Web-services. Although designed for another purpose, one its core functions is to provide service directory access and it was thus suitable to be easily integrated and adapted to the needs of this framework. Although UDDI was chosen as context registry of choice other implementations such as ebXML [23], Microsoft's ActiveDirectory [253], LDAP Directories such as OpenLDAP [254] or any other Directory software can used to substitute UDDI as context registry.

Each of those directory software components could be adapted, some with more and some with less effort, to provide the functionality to resolve for registered services with a relationship between entities as well as providing functionality such as adding, modifying and deleting agent context registrations inside the Context Registry.

6.2.5 Registering services in context registry

For the registration to validate the creation of services with a specific signature, some form of taxonomy must be created to register context in a uniform fashion to be able to retrieve the information efficiently at a later stage. In this particular case we have chosen Oracle Repository as the registry of choice. This Context Registry is implemented in UDDI and all examples are based on this premise.

Figure 14 is an example which illustrates an extract of a UDDI taxonomy used in the use case. The use case will be explained in more detail later in this document.

```
<?xml version="1.0" encoding="UTF-8"?>
<taxonomy xmlns="http://systinet.com/uddi/taxonomy/v3/5.0" xmlns:uddi="urn:uddi-org:api_v3" check="true">

    <tModel tModelKey="uddi:pesto.ch:taxonomy:chronicDisease">
        <uddi:name>Chronic Disease Taxonomy</uddi:name>
        <uddi:description>Specifies information for X care plan</uddi:description>
    </tModel>

    <compatibilityBag>
        <compatibility>businessEntity</compatibility>
    </compatibilityBag>

    <categorizationBag>
        <categorization>categorization</categorization>
    </categorizationBag>

    <validation>
        <categories>
            <category keyName="Patient" keyValue="PAT000000" disabled="false">
                <category keyName="Blood Pressure" keyValue="PAT000100" disabled="false">
                    <category keyName="Systolic" keyValue="PAT000110" disabled="false"/>
                    <category keyName="Diastolic" keyValue="PAT000120" disabled="false"/>
                </category>
                <category keyName="Weight" keyValue="PAT000200" disabled="false">
                    <category keyName="Weight/kg" keyValue="PAT000210" disabled="false"/>
                </category>
            </category>
```

Figure 14 - Chronic disease management example of UDDI taxonomy

Considering the above XML definition of the taxonomy used to register context categories in UDDI, the keyValue attribute of the category tag is essential to map the taxonomy to the ontology used to define context and its use within the agent framework. Specifically, this allows the user to describe context categories or specific context in a coded fashion which is needed in the UDDI Registry as well as modelling the context in OWL.

We initially created a taxonomy which allows the user to validate context code while registering in the UDDI Context Registry. Every taxonomy is uniquely identified through the tModelKey, which can be created for every problem domain or at a more granular level for specific areas in a problem domain. By uploading the taxonomy into the UDDI a validation service is created in UDDI. This validation service is triggered with every registry entry created using the taxonomy's tModelKey (see Figure 14), thus removing the burden of runtime checking

of context code integrity from the framework. If an agent tries to register with a code not present in the taxonomy the registration will fail.

In Figure 15 we show the registered service with its relation to a specific context. In this example a general practitioner has registered Blood Pressure History with the taxonomy code CAG000100 under the Patient Jeremy Sick.

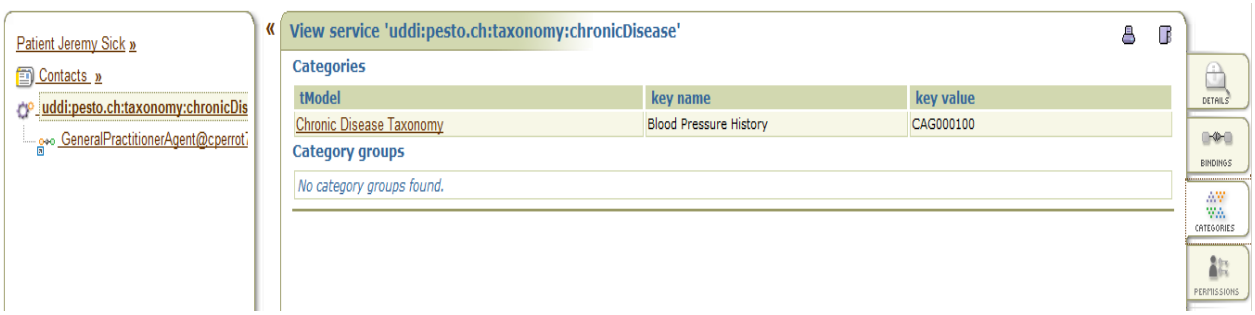

Figure 15 - Context and agent registration/taxonomy view

In Figure 16 we show the same service with its access point. The access point in this example is representing the JADE Agent connection string which allows initiating communication with a JADE agent.

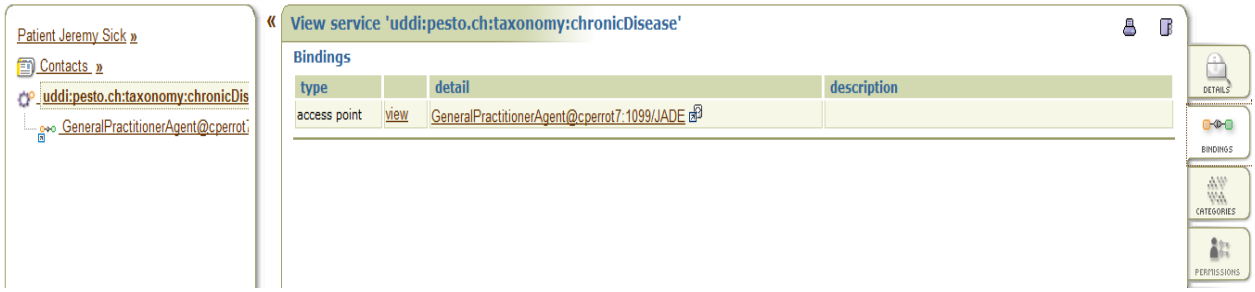

Figure 16 - Context and agent registration/agent bindings

6.2.6 Resolving relations in context registry

One essential aspect of the context registry in this example UDDI is to allow resolving context between related partners. For example, a general practitioner might use such a system to enter a medical history of the patient and register that specific context code in the context

registry. The general practitioner's relation with the patient as well as the context code must be registered with the context registry. This chapter will show how this has been achieved by using UDDI as the context registry.

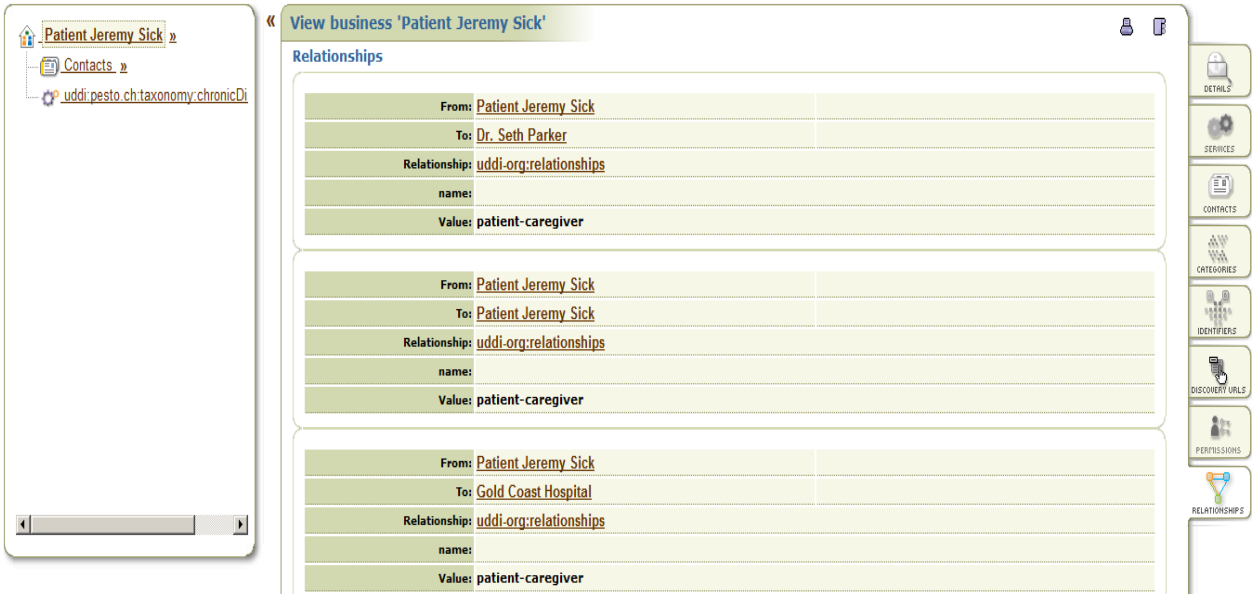

Figure 17 - UDDI relationship of related entities

As previously described services are registered with a tModelKey of the taxonomy allowing the UDDI repository to validate for consistency. The tModelKey is a unique key used to identify the entire taxonomy. This unique tModelKey is then used to find and register services (agents) providing context about business entities (see Figure 17).

The first step is to initiate resolving the relationship between two individuals in UDDI with a specific relationship signature. This specific relationship is resolved as shown in this example, using a chosen signature named "patient-caregiver" as literal (see Figure 17). This signature can be set to whatever value is required to support uniqueness in resolving for relations. Another example of such a relation could be to resolve for a patient's artificial hip supplier which would probably be name 'patient-supplier', to resolve for next of kin for alerts or simply to design these relationships on a more granular level such as 'patient-doctor', 'patient-nurse', 'patient-hospital' and so on. With these created relationships it is possible to resolve for exposed context services (tagged with the taxonomy identifier) from related business entities

(doctor, patient, nurse and so on). It furthermore also allows reading from a focused result-set which provides efficient querying and matching with context services.

This result-set then needs to be further checked against registered services. The second step is to query the UDDI Registry for services registered with a tModelKey as shown in Figure 14. The first result-set is then matched with the relationship result-set to see whether a service matches with one of the related persons e.g. the general practitioner has a relationship with a patient.

6.2.7 Modelling

The ontology modeller of choice was Protégé provided by Stanford University. Protégé allows a seamless integration of different ontologies by merging these ontologies into one comprehensive ontology. This functionality was used to create one ontology from different ontologies such as AgentRoot ontology, a context ontology and a domain specific ontology. This consolidated ontology is then used to provide the necessary meta information to the deployed context agents. This merging of different ontologies is particularly useful when integrating with other third party providers which use their own ontology for context.

Using the merging functionality can result several types of conflicts identified by Taboada et al. [255] such as multiple paths to the same ontology, missing ontology and concepts defined multiples times differently in ontologies to be merged [255].

Although merging of ontologies is technically possible it includes several challenges when doing so. Some of these challenges include the initial selection of a terminology, ontologies being based with different terminologies which cannot be merged and different levels of granularity which do not fit on top of each other. Smith et al. (2006) quotes in relation to ontologies, taxonomies and terminologies that: “In the ideal case a representation would be such that all portions of reality salient to the purposes for which it was constructed would have exactly one corresponding unit in the representation, and every unit in the representation would correspond to exactly one salient portion of reality. Unfortunately, in a domain like biomedicine, ideal case will likely remain forever beyond our grasp.” [256]. This shows that every problem domain sometimes has shared understanding and ontologies can be shared or in other cases parts

can be reused with the addition of more specific ontological definitions applicable only to one problem domain.

6.2.8 Agent modelling

At the root of this framework lies the AgentRoot Ontology, which the framework depends on to find context during runtime. The framework relies on 3 classes with 2 functions in order to resolve all required contextual information during runtime. This allows the runtime environment to intercept calls made to class instantiations based on AgentRoot as depicted in Figure 18.

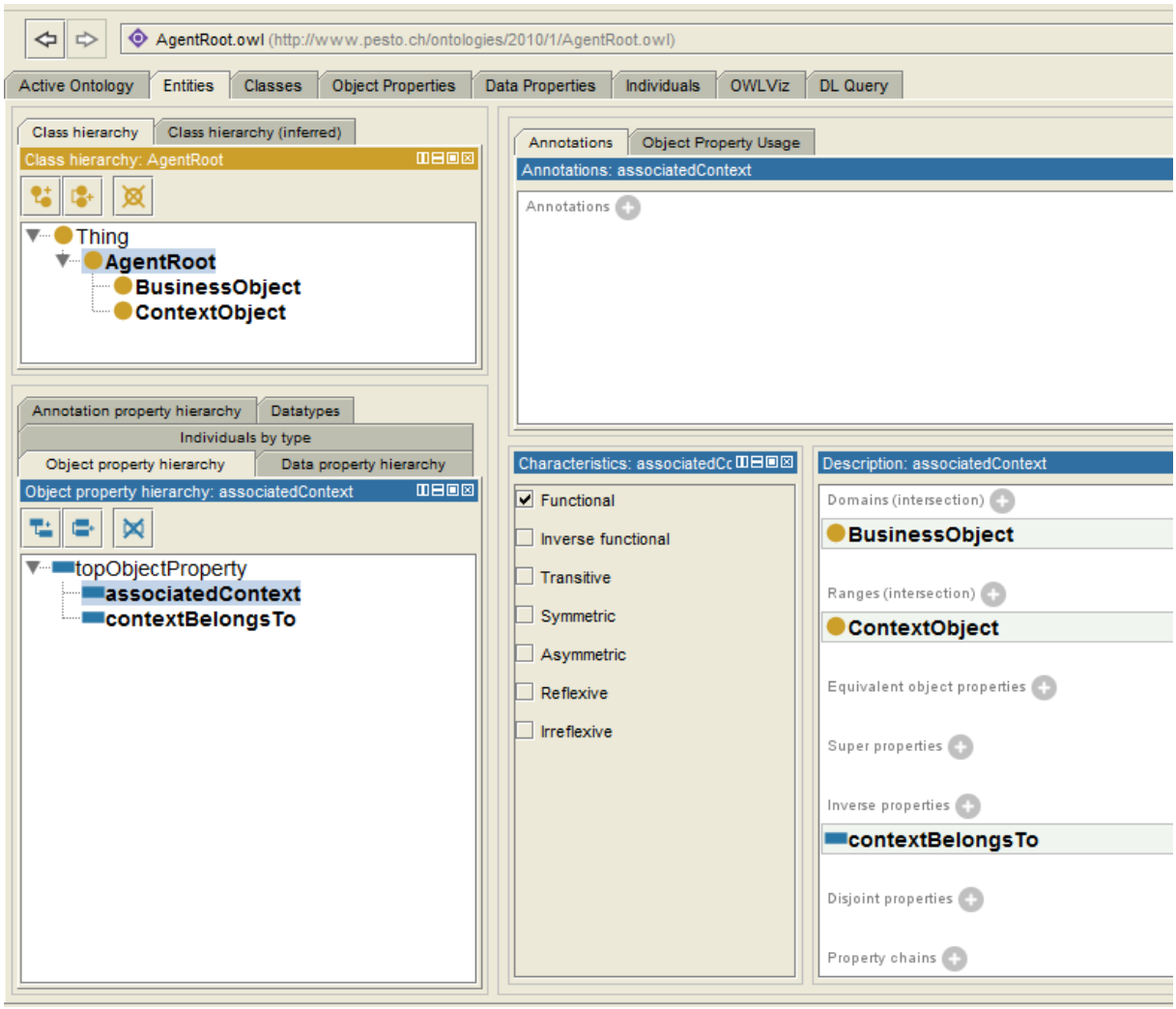

Figure 18 - AgentRoot meta ontology

6.2.9 Context ontology modelling

After categorizing context categories, the next step was to demonstrate the flexibility of the framework using a generic content example of context which could be integrated into this approach as well as allowing for maximum reusability. A generic template in OWL was found created by Thorsten Möller of the University of Basel in Switzerland provided through the Protégé Ontology Library [245, 257]. Although, this context meta-model was chosen any other meta definition of context described in OWL could be taken to start the creation of context modelling for this framework.

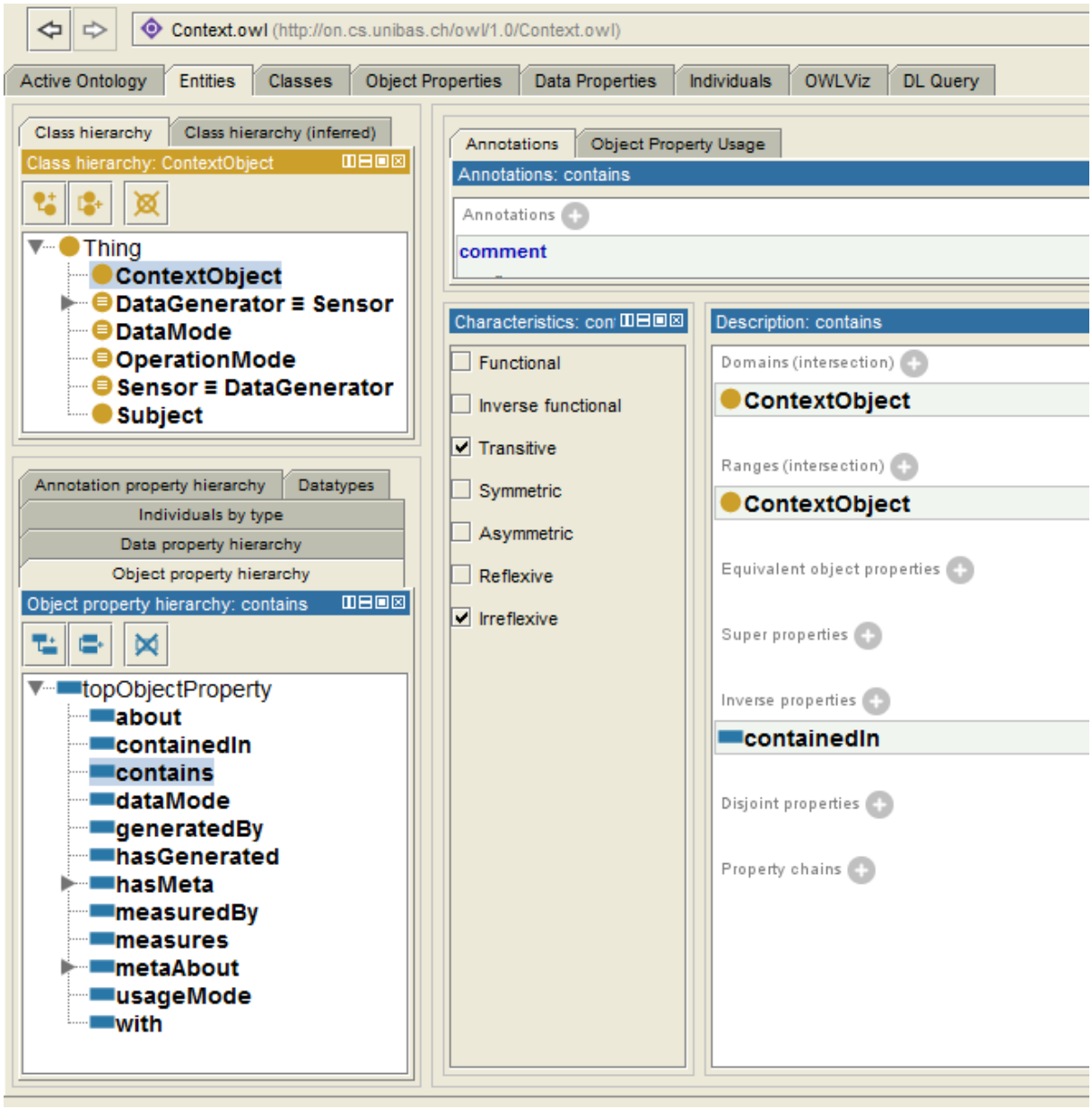

Figure 19 - Context meta definition

6.2.10 Domain ontology

Domain ontologies are created to describe the relations and properties of entities in a particular domain such as health, military, disaster management and others. Many templates can be found which today provide a good starting point to model a domain-ontology and adjust them to the specific user needs one might have. One of the most prominent places cataloguing over 10’000 ontologies is Swoogle [258].

Another good starting point to initiate domain modelling is to reuse existing models which are predominantly available in UML format. Although, we previously described the translation problems of converting UML into OWL the functionality provided by these tools is sufficient to convert an initial UML domain model into OWL to be further modelled and adjusted to suit the requirements.

6.2.11 Merging of ontologies

As previously described Protégé provides a function which merges all loaded and used ontologies into one common ontology. This merging of ontologies has been used in this framework to reduce the complexity instead of having several independent ontologies linked into one. Although not used in this framework it is anticipated that one would be able to change the approach of using linked ontologies instead of merging all used ontologies into one.

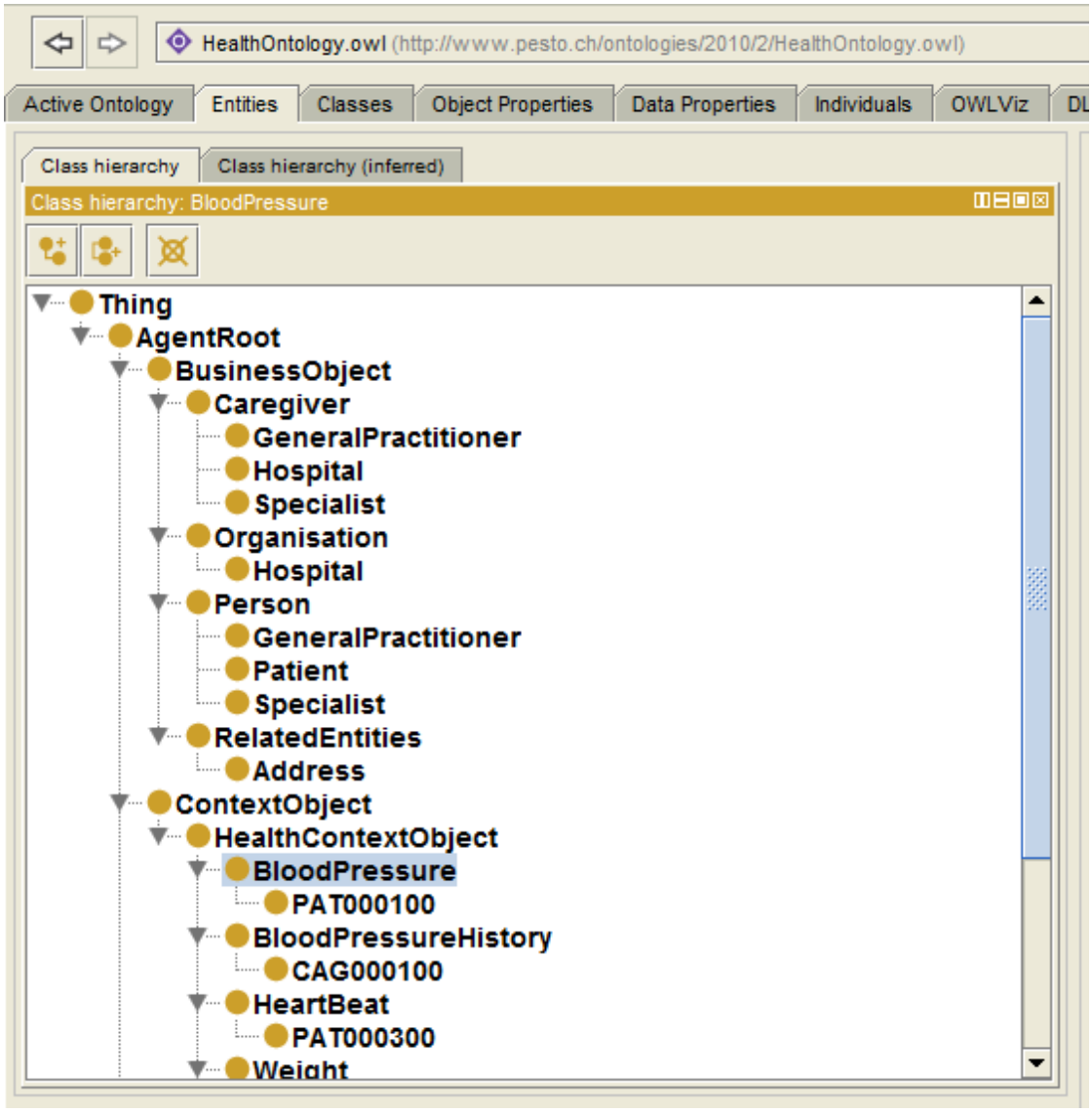

Figure 20 - Consolidated ontology

As shown in Figure 20 context objects are created under root class ContextObject and HealthContextObject. This consolidation process is use of the AgentRoot, context and a domain specific ontology to create a common ontology which is used by the software agents. All domain classes are placed underneath either BusinessObject or ContextObject to have a specific ancestry, thus enabling finding objects and providing the ability for the programmer to easily invoke functions upon them.

6.2.12 Mapping of context and UDDI taxonomy

When mapping the categorization of the UDDI taxonomy described in section 6.2.5, the classes are shifted to suit the framework and are to be mapped correctly inside the newly created ontology. When considering Figure 20 we see that under HealthContextObject several context classes are defined in the ontology. For example BloodPressure derives from HealthContextObject and PAT000100 derives from BloodPressure which is identical with the UDDI taxonomy seen in Figure 14. This allows for a simpler modelling and mapping of the taxonomy categories and the context classes. Although this showcases one possible example of modelling, the PAT000100 class is not necessarily used during runtime to resolve the mapping but only to simplify modelling and to make it more transparent during the modelling phase of the problem domain.

To best map context and classes to each other, functions in the ontology are required to resolve these dependencies. Figure 21 illustrates how the association between a BusinessObject (which all domain specific business classes derive from) is associated with a context function. In this case, “associatedContextPAT000100” is associated with BusinessObject which allows the runtime functionality to intercept this call and to inspect returned results of that call and to redirect the call if necessary.

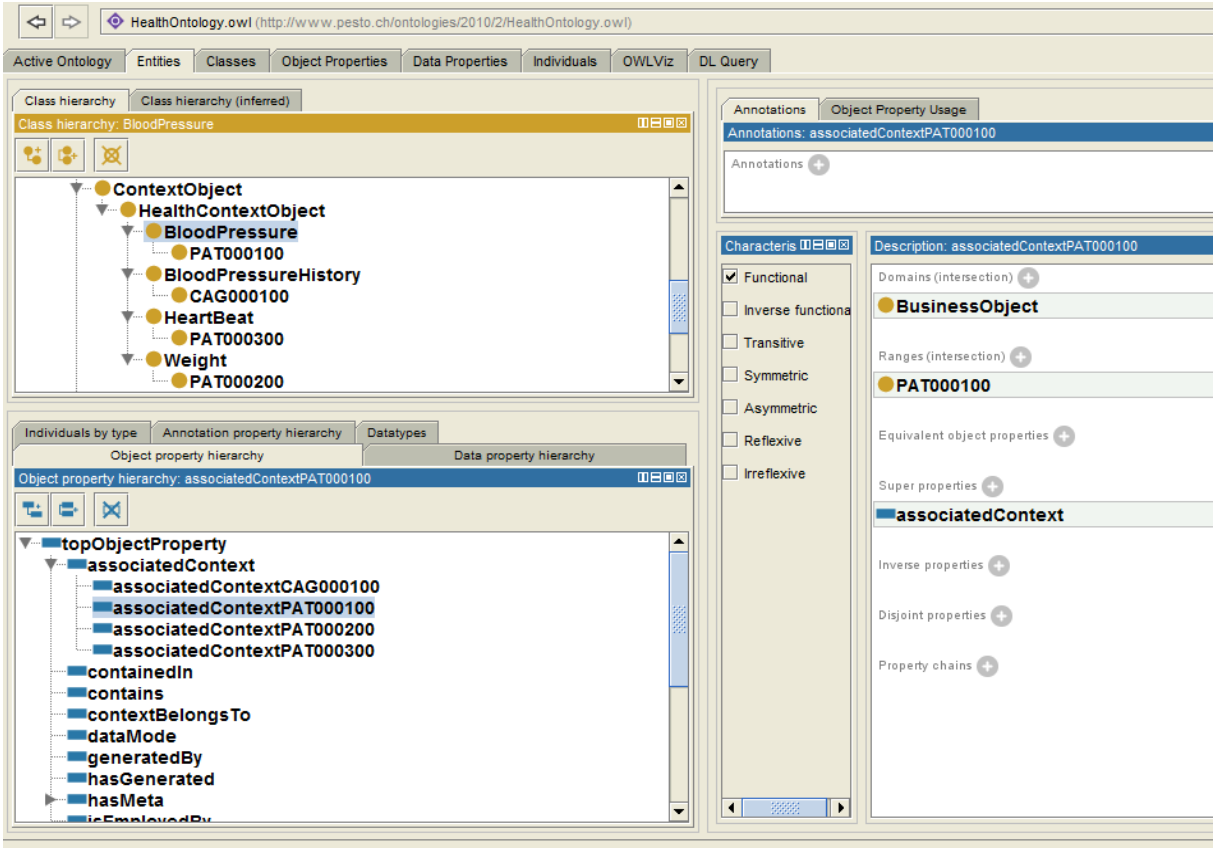

Figure 21 - Modelling context associations

The framework scans for all calls which have a preceding “associatedContext” in their function name and intercepts them before routing them to the internal context storage. The context storage locates the context and returns the context to the module which intercepted the call to the context. If no context could be found, the agent will start any number of rollback procedures, tagging all rules as either executed or not executed followed by streaming all domain information and propagating all information to the next agent exposing needed context. More on how this is done in detail will be explained later in this thesis.

6.3 Tools, Instruments, and Frameworks

6.3.1 Business rules

As already mentioned in section 3.5.2 once a decision support system is selected, the newly created business rules are specific to that decision support system and thus are difficult to

be interchanged. Changing the decision support system involved manual conversion or the integration of a rules broker architecture to support multiple decision support systems.

Another central part of this framework is the structuring of the business rules to allow for a fine granular control of the business rules in terms of which rule has been executed and which rule has not been executed as well as the sequencing of those rules. This thesis's approach is to separate the rules into numbered files which allows for clear execution sequencing, keeping the rules small and with contextual integrity.

Numbering the rules steers the sequencing of the rules. A rules file called 0001-BloodPressure.drl would precede 0004-WeightCheck.drl. This mechanism is used to mark rules as executed. It is vital that all rules are divided into smaller, more manageable intrinsically self-contained business rules files. In more detail thematically solving different self contained problems, as a consequence separating blood pressure related issues from weight checking issues if these issues have no correlation, is fundamental to this framework.

The entire state of the software agent is saved in between every execution of every individual business rule file to guarantee a facility to roll back to the state before the business rule file was executed. This mechanism will only rollback to the previous state information in the case of missing context information while other errors and messages will be propagated to the initiating application or agent. This allows the runtime environment to propagate a consistent state to the next software agent providing the missing context information.

Such a granular approach becomes more important when contextual information is being resolved during runtime. Looking at the alternate possibility of creating few files with numerous business rules contained in them would make it very difficult to control the state of the execution. In order to pursue such an approach the user would need a subtle knowledge of the internals of the business rules engine deployed in that framework to be able to intercept and prevent execution of already processed rules. In light of this complexity of controlling the business rules engine inner workings, the decision was made to therefore opt for a more business rules engine-neutral approach, which served to circumvent the problem of controlling the business rules engine internals and dividing the business rules in small, more thematically-organized files thus allowing full control from outside the business rules engine.

This framework does not limit how business rules are stored and retrieved as long as the sequence of the individual business rules is properly added to the agent parameter (see Figure 35 in the Appendix). The following rules file (Figure 22) shows a simple rule concerning blood pressure:

```
rule "Check Bloodpressure"
    salience 100
    no-loop true
    when
        $p : HealthOntology.Patient(
            $bp : associatedContextPAT000100,
            $bp.systolic > 139 || $bp.diastolic > 99 )
    then
        System.out.println("Blood pressure is outside norm");
end
```

Figure 22 - Blood pressure simple business rule

Note that the function invoked to retrieve the patient's blood pressure is associatedContextPAT000100 which coincides with the function modelled in Protégé (see Figure 21). The modelled classes and function are exposed by the java wrapper classes and passed to the business rules engine during runtime.

```
rule "Check Bloodpressure Patient History"
    salience 100
    no-loop true

    when
        $p : Patient( $weight : associatedContextPAT000200, $weight.weightInKG > 90 )
    then

        System.out.println("Weight has increased beyond the recommended level: " +
                           "Weight in kg " + $weight.getWeightInKG());
end
```

Figure 23 - Weight check simple business rule

The same underlying principle of context codification is shown in Figure 23 in which weight can be retrieved using the modelled function associatedContextPAT000200.

6.3.2 The agent runtime environment

As an environment to develop software agents we decided to use Jade 3.7 [259] due to its popularity in the developer community, support, versatility in terms of add-ins supplied by

Tilab and other 3rd party developers, making it highly adaptable and suitable to meet our requirements.

We would like to reemphasize that the framework was designed to be all of the following:

- Stateful only holding contextual information
- Stateless in holding domain information
- and model driven instantiating all domain data and executing rules when an agent is initiated

It is important to reiterate the above criteria i.e. the only information created or intercepted by an agent will be stored in the agent's context storage, which in this implementation is OpenRDF's Sesame OWL repository. All other information is handled statelessly and is passed from the initiator to the first agent and all subsequent agents involved in the decision making process. More precisely, we have adopted different technologies to send information to and from the agents. For passing information back and forth between agents, we have opted for xStream provided by xstream.codehaus.org to serialise and deserialise information. Business rules and the OWL information are part of the serialised parameters passed to and from the software agent. This approach of serialising and deserialising information looked to be the most promising in terms of integration with web-services and BPMN and the uptake of XML based data exchange in recent years [134, 151, 260]. It can also be easily substituted through other serialising technologies.

6.3.3 Introducing a running example

In order to showcase our approach, we have chosen to focus on contextual computing in the healthcare domain. In this particular example we show how contextual information is being used from three different sources. Firstly, the patient's contextual information which has been stored in the patient's software agent (e.g. mobile phone). Secondly, the general practitioners' software agent which holds all the contextual information held by the general practitioner from observations and diagnosis from the patient's past visits and lastly the hospital's contextual

information which provides contextual information about the chronic disease parameters and treatment options.

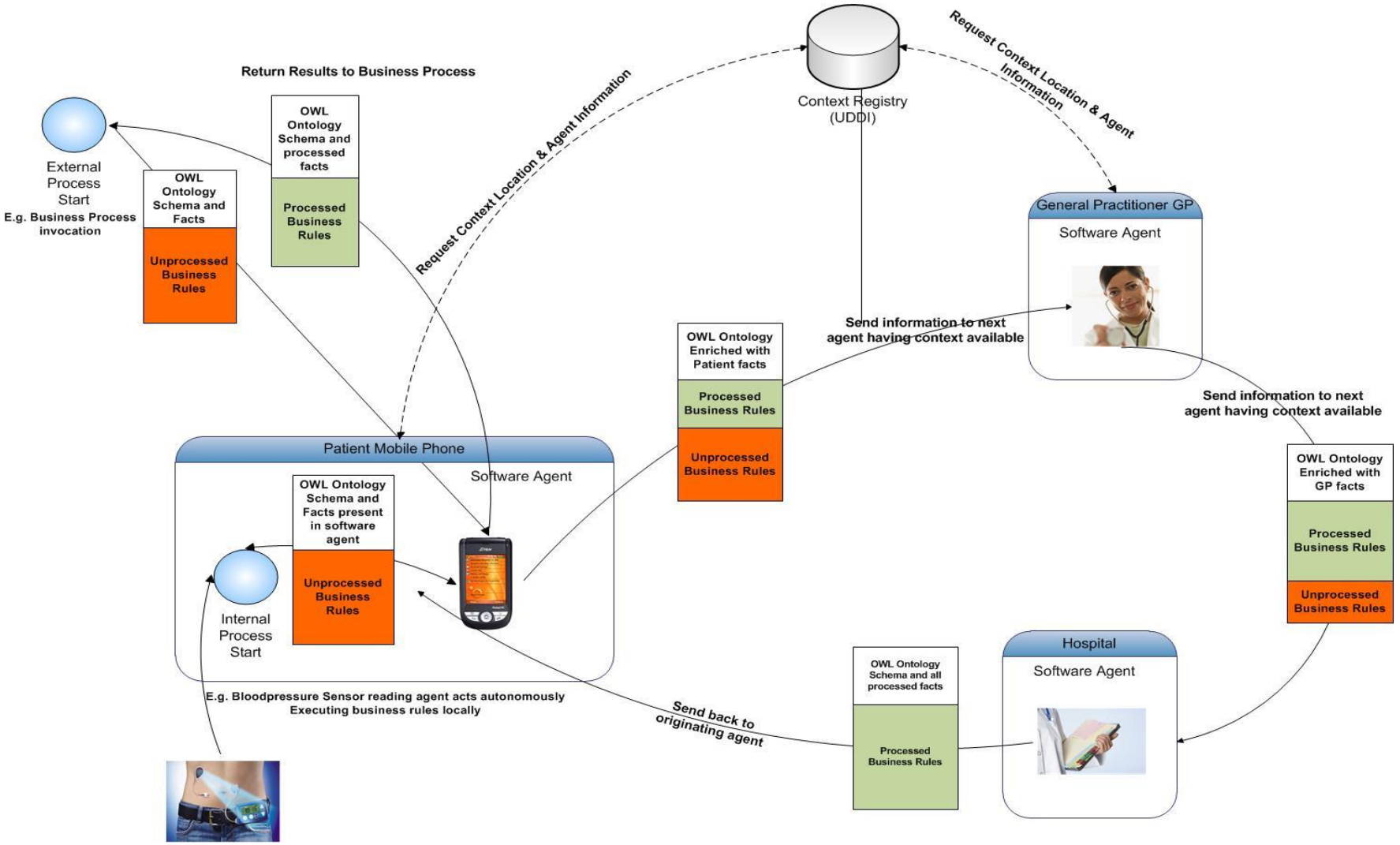

Figure 24 - Use Case Example

To simplify the approach without compromising the expressiveness of the results, we decided to showcase the use case shown in Figure 24 using only the software agent of the patient and general practitioner interaction and creating a software agent which starts the process. The starter agent could be substituted with a BPMN or Workflow solution. In considering Figure 24 the only agent omitted in this running example would be the hospital agent. It is crucial to reiterate that the following example does not focus on gathering, converting and storing contextual information inside the software agent but instead illustrates the communication between the agents and how rules are executed by the agent providing relevant context in relation to a business entity.

The starter agent was created to substitute for a more complex situation where the application already provides business model and individuals information. The starter agent stores the business model information and individuals along with the business rules stored inside the agent. In a more complex environment, business rules would be associated with a particular business process and retrieved from a central repository or similar. For this example, we have stored all necessary information inside the starter agent which allows for a seamless loading of all data needed to start the process, which otherwise would be provided by the BPMN or calling application.

6.3.4 Starter agent

The starter agent could be seen as the overall process trigger and merely simulates a new sensor reading and sends the values to the first agent. In a real world scenario this could be an internal sensor reading such as location or a blood pressure reading triggering a process/event inside an active agent. For demonstration purposes we have chosen to create a full agent with all messaging capabilities and a graphical user interface to display returned results and to simulate a blood pressure reading of the patient which triggers the decision making process across the different agents.

The starter agent initializes and loads all data required to start the chain of software agents involved in executing the business rules against the contextual and model/individual information. It then creates a message holding all the information to send to the patient agent holding the initial set of context information as an OWL model and OWL individuals (instances of context). All initially necessary information is created inside the starter agent such as ontology

and individuals, business rules and the version information of the ontology. The version information of the ontology helps to decide whether to regenerate and recompile the ontology java wrapper files in each agent in the execution queue.

6.3.5 Patient agent

The patient agent receives the initial call of the starter agent and deserialises all passed information into xml using the XStream library [260]. It then compiles and recreates the java wrapper classes for the ontology if the version information has changed. The java classes then have to be dynamically loaded and act as a wrapper to the real classes contained in the OWL semantic web container (OWL storage) to make these classes accessible to the framework during runtime. The separate OWL storages are then created, one to load all individuals (business objects) and the other for all instances of context passed to the agent. At this stage the environment has been prepared to run the business rules against the business objects using the given context information of the starter agent along with the local context information only available to the patient agent. The business rules are then executed while doing some sanity checking for looping conditions, missing context and to check whether the execution of business rules has completed.

6.3.6 General practitioner agent

The general practitioner agent consists of the same code as the patient agent. The main differentiating factors between the patient agent and general practitioner agent are the parameters passed to the agent and the loaded context inside the agent. Therefore, all the steps made in terms of initializing and deserialising are repeated in this agent. The main difference between the agents consist of different available local context and different parameter states such as the execution state of the business rules and changed state information of business objects.

In this example we have used three Business Rules Files which contain rules pertaining to blood pressure, weight check and blood pressure history. One simple example of such a business rule file has been given in Figure 22. Of course in the real world business rules are more complex and Figure 25 shows a more in depth example e.g. investigating a series of blood pressure readings.

```
rule "Check Bloodpressure Patient1 History"
    salience 100
    no-loop true

    when
        $p : Patient(   $weight : associatedContextPAT000200,
                        $bp : associatedContextPAT000100,
                        $weight.weightInKG > 120 && $p.age > 50 && ($bp.systolic > 139 || $bp.diastolic > 99) )
    then

        BloodPressureHistory bphist = $p.getAssociatedContextCAG000100();
        if (bphist != null) {
            int entries = 0;
            double diastolicAverage = 0;
            double systolicAverage = 0;
            Set<ContextObject> ctxset = bphist.getContains();
            for (ContextObject ctx : ctxset) {
                entries++;
                BloodPressure ctx1 = (BloodPressure) ctx;
                XMLGregorianCalendar cal = ctx1.getMeasurementTimeStamp();
                diastolicAverage += ctx1.getDiastolic();
                systolicAverage += ctx1.getSystolic();
                System.out.println("General Practioner History measurement date: " + cal.toString() +
                                   " Systolic: " + ctx1.getSystolic() + " Diastolic:" + ctx1.getDiastolic());
            }
            if(entries > 0){
                //check for increase more than 5%
                diastolicAverage = (diastolicAverage / entries) * 1.05;
                systolicAverage = (systolicAverage / entries) * 1.05;
                if( $p.getAssociatedContextPAT000100().getDiastolic() > diastolicAverage ||
                    $p.getAssociatedContextPAT000100().getSystolic() > diastolicAverage){
                    HealthOntology.Notification noti = ContextRepository.createObject(HealthOntology.Notification.class);
                    noti.setNotificationMessage("weight and increase in blood pressure need medical attention");
                    HashSet notifications = new HashSet();
                    notifications.add(noti);
                    $p.setNotificationBelongsTo(notifications);
                }
            }
        }
end
```

Figure 25 - Business rule—patient blood pressure history

This business rule in Figure 25 showcases the check for an increase of 5% in blood pressure over a series of past blood pressure readings and writing back a notification message if the threshold has been exceeded.

6.4 Architecture and Justification

6.4.1 Explanation of the running example

We have explained what each individual agent does in relation to this example. This section will focus on illustrating the interaction between the agents and the workflow between them. As previously explained, we created an interactive starter agent to create all domain

objects and to provide the business rules to start the next patient agent which is based on the premise of a model driven software agent being completely controlled through the data passed to the agent. We created 2 simple rules, the first of which checks for blood pressure as shown in Figure 22 not exceeding a particular predefined threshold. The second rule (see Figure 23) was aimed at the patient weight. The two simple rules could be executed using the patient agent's locally stored context information without the need to further propagate the running state to the next agent containing other context information.

We then introduced the business rule checking for a 5% increase (see Figure 25) in blood pressure over past readings, requesting information not available in the patient agent and thus forcing the agent to stop executing at the business rule requesting the missing context. The next step is then to roll back all changed information to the state it contained before the business rule with the missing context was executed. This is done before the execution of each business rule file by saving all information every time one business rule file is run. This further reemphasizes the need to create business rules files as a self-contained unit of work e.g. one example of such a rule contained in one rule file would be weight checking which could be a combination of age and weight. This level of granularity allows the efficient execution of business rules across all agents without the need to re-execute large business rules files when propagated to the next agent. In this example the two simple business rules (see Figure 22 and Figure 23) would be tagged as executed and only the blood pressure history would need to be run at the general practitioner's software agent.

We have now one rule that was executed which requested context information not available to the patient software agent. Now as previously described the agent rolls all information back to the state before the business rule was executed and streams all information into a common format understood by all agents to propagate all this information to the next agent providing the blood pressure history context of this patient. In this case, the general practitioner agent was invoked recreating the patient agent's state and further processing the business rules exposing its own locally stored context to the remaining business rules.

6.5 Utility and Applicability

6.5.1 Current healthcare approach

Today's healthcare approach encompasses several problems. Considering a simple medical process illustrates the shortcomings of today's process.

- Patient becomes ill and seeks medical attention
- Doctor interviews patient and writes down notes on paper or in an application
- Patient is given health regime and medication
- Patient feels unwell and calls doctor
- Doctor is not available and nurse cannot read notes
- Nurse asks patient to call again the next day
- Patient calls the next day and doctor is not available because he/she is treating patient
- Patient still feels unwell and needs to go back for another appointment to see doctor

This process description of a patient – doctor interaction is simple and certainly everyone has experienced something similar in the past. This is also a woefully inefficient system which must be improved.

This process is prone to error due to the fact that to this day, many medical records are still being kept as paper based notes. In addition, paper based notes tend to vary in legibility depending on the individual's hand writing and thus, interpretation and codification of such information is nearly impossible. Furthermore, in case of an emergency where this information could be relevant or even vital in treating the patient, valuable time is lost in retrieving the information due to the fact that someone has to physically get the paper notes and transmit the findings to the next caregiver. There is little or no standardized encoding used to write down notes which would enable machine interpretability of such notes. This creates the problem that

information is not readily available when needed and patient care is deferred to when the caregiver is available for consultation. This results in potentially disastrous delays which may decide the patient's fate.

Instead of writing findings on paper, the decision ought to be made when the patient parameters are not within range and medical information procured through context made available by the doctor. This would allow for immediate response and advice to the patient to be provided as well.

6.5.2 New codification possibilities

Notes on the patient's history ought to be entered electronically which allows for further processing of them semantically enabling a standard codification of those notes.

One such approach is the Semfinder approach which analyses the notes semantically resulting in a standard ICD-10 codification [261].

A general practitioner's text on a patient might look like this "*Dislocated fracture of the rt. lateral clavicula*". This text is then analysed and codified using only those concept atoms that are required for generating the right code. (see Figure 26) Alternate words/synonyms such as clavicle instead clavicula can be used to obtain the same code.

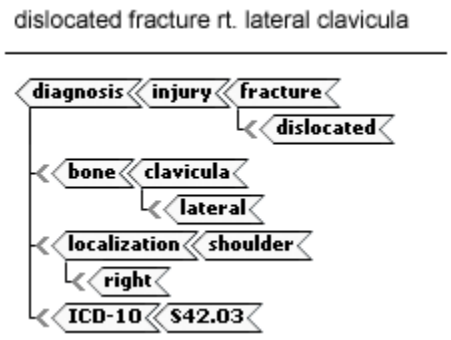

Figure 26 - Semantic Text Analysis / ICD-10 codification [262]

These codes could be used to classify context and register the ICD-10 codified contexts in the context registry. Instead of registering context as defined in this example as PAT000100 (see Figure 21) one could design context around the ICD-10 codes. Context would be registered

as S4203 and the business object would modelled in OWL to expose context as "associatedContextS4203".

One advantage of this new approach is certainly that several context ontologies can be merged into one big ontology and therefore, this approach allows the accommodation and use of many different context coding schemes be it diagnosis coding, therapy coding, medication coding and so on. Looking at the ICD-10 coding scheme the large number of entries certainly could become challenging in designing a system [263]. Approaches could therefore encompass to work only with a subset of codes relevant to the problem domain. An alternate approach could be to map context to the archetypes or templates of OpenEHR [264]. It is important to reemphasize that this approach can function with any form of coding scheme thus allowing the designer to choose a fine granular coding scheme or a more abstract coding scheme. The ICD-10 example previously mentioned was only taken to illustrate one possibility of many and can be changed to whatever coding scheme the system designer envisaged.

6.5.3 Healthcare approach using new framework

A new healthcare approach could look different after deploying the newly created framework:

- Patient becomes ill and seeks medical attention
- Doctor interviews patient and writes down notes in a standardized form in an application or the doctor's comments are automatically semantically checked and coded [261].
- Context is automatically written to the context repository and registered in the context registry
- Patient receives a given health regime, medication and body sensors e.g. blood pressure gauge
- Patient feels unwell, body sensors pick up reading and start processing decision making

- Doctor's notes e.g. blood pressure readings are queried
- Decision making has found a sudden blood pressure increase of more than 5% and recommends that patient should seek medical attention
- Doctor and Hospital are alerted
- Patient is picked up by ambulance or drives to caregiver
- Patient is treated

This process illustrates the inherent advantages of deploying such a framework over the existing system of healthcare delivery. The decision-making process begins immediately so no time is lost either due to information loss and/or misinterpreted information or due to the unavailability of the doctor to diagnose a problem when it occurs.

The time difference between these two processes that illustrate how medical services are delivered may only be days, but this is often time that a patient may not have in critical situations. This new approach could be especially useful when monitoring people with chronic diseases or who might have life threatening conditions such as cardiac problems.

6.6 Framework security

Although security is not in the scope of this research it is still important to touch on that subject. Several issues could be determined while creating this framework in relation to security.

6.6.1 Healthcare Regulations

To safeguard healthcare related information the USA adopted the Health Insurance Portability and Accountability Act (HIPAA) in 1996. This law protects the privacy of medical records on several levels with compliance required by April 14, 2003 [265]. HIPAA is divided into two distinct sections: one addresses privacy and the other security issues. The privacy rule mandates the implementation of appropriate safeguards to protect the privacy of personal health information and further limits the uses and disclosures of health information without the patient authorization. The security rule requires appropriate administrative, physical and technical

safeguards to ensure the confidentiality, integrity, and security of electronic protected health information[266]. Australia covers healthcare information privacy as part of the privacy act [267] and the United Kingdom as part of the data protection act [268].

The common aspect of all these data privacy laws is to protect healthcare related information of individuals from unauthorized access. Australia is further promoting the use of person controlled electronic health records (PCEHR) owned by the person with the right to assign healthcare providers to use the electronic health record [269].

6.6.2 Technical Implementation

In general security can be divided into several categories. These categories are privacy / confidentiality, integrity, authentication, authorization and non-repudiation.

- Privacy or confidentiality protects information from unauthorized third party access.
- Integrity is about keeping the original message not tampered with and unaltered.
- Authentication identifies the sender and recipient to each other.
- Authorization allows access to particular parts of functionality (program code) or data (in this framework contextual information).
- Non-repudiation proves that a message has been received.

Due to the interchangeability of each individual component most of the framework's security implementation depends on the selected agent environment, in this research Jade 3.7. Jade includes an add-on Jade-S which addresses most security issues mentioned above. The Jade environment provides security aspects such as user authentication, agent action authorization and message signature and encryption. Jade uses Java Authentication and Authorization Service (JAAS) to authenticate users. The authorization is checked through a policy file which adheres to the default Java/JAAS syntax, but uses an extended policy model to control actions performed inside the agent [270, 271]. The security integration of Jade still has shortcomings such as very coarse permission rights only providing access to the agent's actions. One other shortcoming very relevant to this framework is that agents only apply security to the agent container but do not control access to information stored inside the agent [270, 272]. This has consequences if

security would need to be implemented at the context code access level, controlling the access to the context repository from inside the agent's runtime environment.

To address the issue of context level permission security (accessing one particular context) an extra layer of functionality is mandated. Several approaches could be integrated to achieve full access control of context using the runtime container's security infrastructure and extending the policy to accommodate these permission extensions. One such approach could encompass the usage of aspect oriented programming including e.g. AspectJ [149] in conjunction with Apache Shiro [273] security framework to handle permission to access the context repository. These two software components are open and mature enough to be integrated into the agent framework and work in conjunction with existing security policies.

Securing context on a fine granular level could be implemented at the context code level. This would allow controlling the access to a specific context taxonomy code such as CAG000100 shown in Figure 15. Moreover, this approach could also be extended to not only control access to context taxonomy code but to check the permission to the content of the context.

6.6.3 Application framework and privacy

The main privacy and security issue identified in relation to this framework is what information is accessed, transmitted and processed and how it can be protected. The application framework internally works with unique identifiers to identify individuals or organisations. These unique identifiers are used to create relations between linked parties later used in the healthcare decision making process. These relationships are stored in the central registry accessible from each agent as shown in Figure 17.

How much sensitive healthcare information is passed to and from other participants in this framework is largely defined by the business model design. For example this applies if individual patient information is part of the design (such as social security number, name, address and so on) and thus transmitted from agent to agent. This transmitted information allows identifying an individual person, therefore more stringent security measures do apply. This is in comparison to a business model design where sensitive patient data is anonymised or not

included at all then security measures can be implemented less stringently. The framework itself is designed not to transfer any health context information outside its own scope.

In relation to this framework the application of data privacy is a function of the technical implementation of security in this framework as well as the design approach of the business model. Some fundamental questions have to be asked at the start of the design stage. How much protectable data is designed into the business model and what level of security does that protectable data need!

Another privacy and security aspect is the access to the central registry where relationships between individuals and organisations are stored. This registry could be designed to only store relationships based on the unique identifiers and context relation classifications without any identifiable personal information. For the implementation in this thesis all personal information about the relationship between patient and healthcare providers is stored in the central repository which calls for a deeper investigation on how to protect the relationship data in the central repository from unauthorised access if that is mandated by the privacy laws.

6.7 Framework in Detail

This chapter will look in more detail into the inner workings (programming fragments) and structure of the framework. Java packages are prefixed with inverse domain names to keep software from name clashing with other software packages. The package name “ch.pesto” was chosen because the domain was owned by the writer of this thesis and bears no relevance to the framework itself but was merely used to make the software packages unique.

6.7.1 Recapitulation of used Technology

The tools and standards used to implement this approach are:

- Jade Agent Environment Version 3.7
- XStream to serialize and deserialize java objects into xml Version 1.3.1
- Oracle’s UDDI (originally Systinet) Version 11.1
- JBoss Drools business rules Version 5.01
- Stanford’s Protégé OWL Modeler Version 4.1 Alpha

- OWL as semantic web language to define the business model as well as the context information
- OpenRDF's Sesame Version 2.3.1and AliBaba Version 2.0 alpha 4 framework to compile model information into java objects and as a runtime container for semantic web information.
- OWLIM RDF Version 3.0 beta 12 engine[274] as query engine and inference engine in conjunction with Sesame.

6.7.2 Package explanation

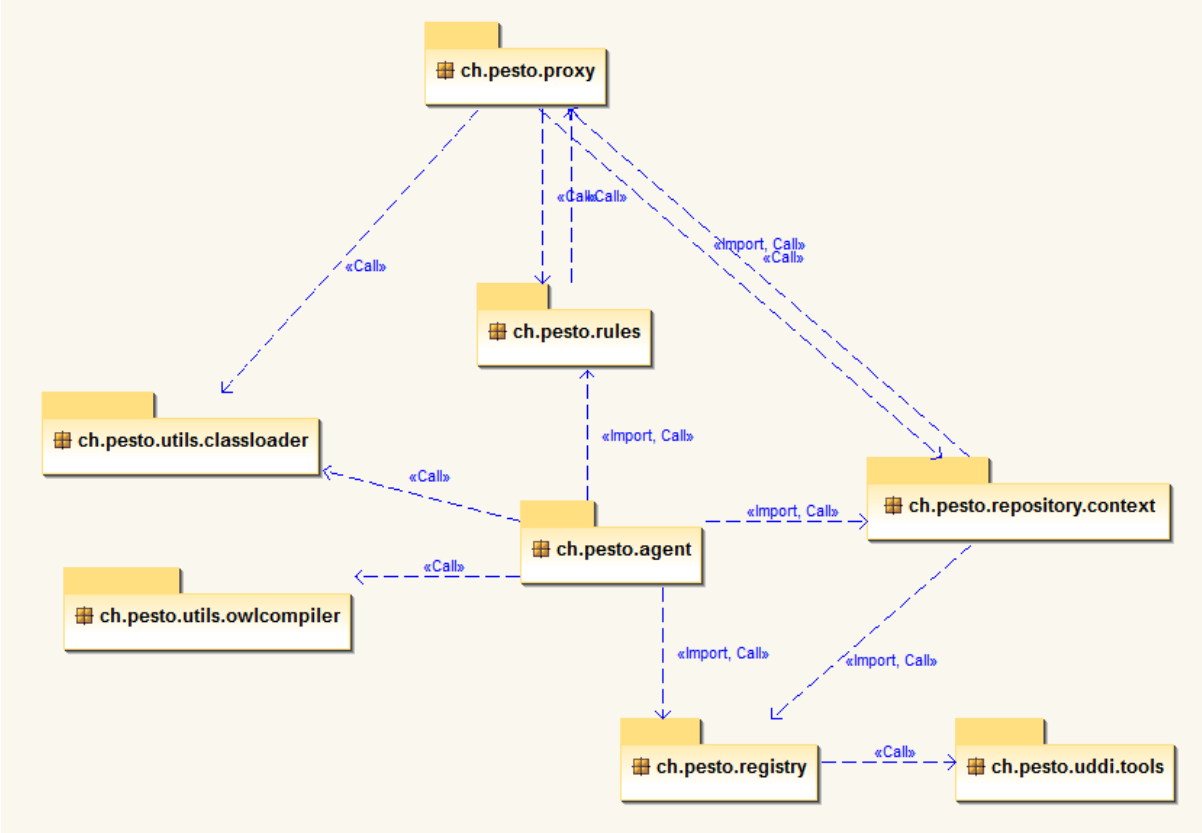

Figure 27 – Software Package Overview

6.7.2.1 ch.pesto.agent

All operations are orchestrated in the agent module which triggers all necessary functions for the agent to work.

The ch.pesto.agent coordinates the different components such as:

- Triggering the generation of Java access classes to the OWL individuals.

  This step involves the use of Sesame's AliBaba OWL Compiler to create Java classes in order to access the OWL individuals (instantiated business objects) passed into the agent.
- Instantiating the model information using the previously generated access classes for usage within the agent.
- Coordinating the business rules engine via a common interface JSR-94 as well as the sequencing of the business rules to be run, which includes tracking the rules which have already been executed and marking them as such.
- Monitoring access to the context repository.

  Through java reflection all calls containing a particular method signature are intercepted and routed to the context repository. In the case that context is not found it follows into the next step of breaking the business rules execution.
- Controlling if missing context was detected to rollback business objects and globally shared context to the state before the business rule was executed.

  Serialising and deserialising the agent, contextual and model information to and from agents.

  Here all information which has been validated and written as a business object, context, messages and so forth is serialised in preparation for sending it to the next agent providing the missing context.
- Serializing context and business objects using XStream to create xml structures and calling the next agent exposing needed context with the model, context and business object information.

The following picture shows the code steering the entire agent in the main controlling function:

```
public void action() {
    MessageTemplate mt = MessageTemplate.and(
            MessageTemplate.MatchConversationId(AgentParametersIn.ONTOLOGY_NAME), MessageTemplate
                    .MatchPerformative(ACLMessage.REQUEST));
    ACLMessage msgRx = myAgent.receive(mt);

    if (msgRx != null) {
        String content = msgRx.getContent();
        AgentParametersIn parmIn = XStreamDeSerializer.getInParametersFromXStream(content);
        if (parmIn == null) {
            String errMsg = "xml parameter could not be parsed - returning with nothing to do";
            log.error(errMsg);
            parmIn.agentParam.errorMessage = errMsg;
        }else{
            parmIn.agentParam = ctxAgent.execute(parmIn);
        }

        ACLMessage reply = msgRx.createReply();
        reply.setPerformative(ACLMessage.CONFIRM);
        String xmlResponse = XStreamDeSerializer.getXMLStream(parmIn.agentParam);
        reply.setContent(xmlResponse);
        ctxAgent.send(reply);
    } else {
        block();
    }
}
```

Figure 28 - Agent main controlling function—action

```
protected AgentParametersInOut execute(AgentParametersIn agentParamIn) {
    // Initialize new repository and Rules runner
    RunRules rr = RunRules.getNewInstance();

    try {
        compileAndLoadOWLSchema(agentParamIn);
        ContextRepository ctxRep = getNewFactsRepository(agentParamIn);

        // get all the objects derived from the same base
        List<Object> objs = ctxRep.getObjects("AgentBase.AgentRoot");
        // load objects into Drools and run rules
        boolean ret = true;
        ret = runAllRules(objs, agentParamIn);

        while (!ret) {
            // find another agent with the needed context
            RegistryConnector regCon = RegistryConnector.getSingletonInstance();
            ServiceList serviceList = regCon.findServiceWithRelations(
                    agentParamIn.agentParam.missingObjectID, agentParamIn.agentParam.missingContext, regCon
                            .gettModelKey(), agentParamIn.uddiKeyValueRelationship);

            String sMissingContextKey = ContextRepository.createRegistryDelimitedKey(
                    agentParamIn.agentParam.missingContext, agentParamIn.agentParam.missingObjectID);
            String seekingContextKey = ContextRepository.createRegistryDelimitedKey(
                    agentParamIn.agentParam.seekingContext, agentParamIn.agentParam.seekingObjectID);
            if (!sMissingContextKey.equals(seekingContextKey)) {
                // protection to prevent calling ourselves
                // same context is being iterated over by the callee
                List<String> endPoints = getAllEndPoints(regCon, serviceList);
                // call it with the remainder of the rules to be run
                callAgentsContainingRequiredContext(endPoints, agentParamIn, rr, objs);
            }

            // check if we processed all rules and if not continue processing
            if (agentParamIn.agentParam.moreRulesToProcess()) {
                ret = runAllRules(objs, agentParamIn);
            } else {
                ret = true;
            }
        }
    } catch (Exception e) {
        log.error(e.getMessage());
        agentParamIn.agentParam.errorMessage = e.getMessage();
    } finally {
    }
    return agentParamIn.agentParam;
}
```

Figure 29 - Agent main controlling function—execute

In the top of the function we instantiate a new business rules engine which ensures that the agent provides re-entrant capabilities and is not using the same rules engine instance again. Looking back at the design in Figure 20 all business objects are derived from AgentRoot. All instantiated business objects derived from AgentRoot are filtered out and registered with the business rules engine as previously described in section 6.2.8, therefore all business objects relevant to the domain model and agent have to be derived from AgentRoot. The function call ctx.getObjects(“AgentBase.AgentRoot”) returns all objects of the given type as a result set. This

result set is then loaded into the business rules engine to make these objects known to the rules engine. The business rules engine can then be triggered with business rules to validate the registered objects and run rules against them.

Should the function rules return unfinished with context missing a new agent exposing that missing context is located and all information transferred to this new agent. This includes all state information available about processing the rules, domain facts information (instantiated objects), ontology and taxonomy information as well as available contextual information.

The parameters passed to and from the agent are shown as Java structures in Figure 34 and Figure 35 in the Appendix. All the information contained in the agent parameters is being streamed as XML structure between agents.

6.7.2.2 ch.pesto.owlcompiler and ch.pesto.classloader

All the OWL information passed will be compiled just in time to create a .jar file containing access classes to OWL individuals contained in the OWL runtime environment. In the background all the semantic web constructs remain intact but are wrapped by java classes generated by the OWL compiler. The OWL compiler's previously generated Java jar file is then loaded into the agent's classpath in order for the runtime environment to detect the newly created OWL access classes.

6.7.2.3 ch.pesto.registry

This is a wrapper for any type of registry such as UDDI or any other registry supporting a similar functionality. As previously mentioned in section 6.2.5 alternate components/technologies can be used to substitute Oracle's UDDI registry. Applications supporting a similar functionality to UDDI would be ebXML [23], Microsoft's ActiveDirectory [253], LDAP Directories such as OpenLDAP [254] or any other Directory software. The wrapper provides the functionality to resolve for services with a relationship between entities as well as providing functionality such as adding, modifying and deleting agent context registrations inside the Context Registry.

```
public void loadContextAndRegisterWithUDDI(URL contextInstances, String queryURI, URL uddiTaxonomy)
        throws Exception {
    RegistryConnector regCon = RegistryConnector.getSingletonInstance();
    saveContextInstances = contextInstances;
    loadRDF(contextInstances);

    // http://www.pesto.ch/ontologies/2010/2/HealthOntology.owl#HealthContextObject
    // through looking up the file and using it here
    final String s1 = "SELECT REDUCED ?id ?belongsToId \n";
    final String s2 = "WHERE { \n";
    final String s3 = "{ ?subj a <";
    final String s4 = ">} \n";
    final String s5 = "OPTIONAL { ?subj <http://www.pesto.ch/ontologies/2010/1/AgentBase.owl#id> ?id} \n";
    final String s6 = "OPTIONAL { ?subj <http://www.pesto.ch/ontologies/2010/1/AgentBase.owl#belongsToId> ?belongsToId} \n";
    final String s7 = "}";

    final String select = s1 + s2 + s3 + queryURI + s4 + s5 + s6 + s7;
    try {
        try {
            TupleQuery tupleQuery = connection.prepareTupleQuery(QueryLanguage.SPARQL, select);
            TupleQueryResult result = tupleQuery.evaluate();
            try {
                while (result.hasNext()) {
                    BindingSet bindingSet = result.next();
                    Value id = bindingSet.getValue("id");
                    Value belongsToId = bindingSet.getValue("belongsToId");
                    String businessKey = belongsToId.stringValue();

                    if (!businessKey.isEmpty()) {

                        String registeredContextKey = createRegistryDelimitedKey(businessKey, id.stringValue());
                        // check for already registered contextKeys - somehow the SPARQL query returns the same instance twice
                        if (!registeredBusinessAndContext.contains(registeredContextKey)) {
                            String authInfo = regCon.getAuthInfo();
                            String serviceKey = generateServiceKey(regCon.getAgentBinding());
                            try {
                                regCon.createService(businessKey, regCon.gettModelKey(), serviceKey, regCon
                                        .getAgentBinding(), authInfo);
                                try {
                                    regCon.addCategoryToService(
                                            uddiTaxonomy, serviceKey, id.stringValue(), authInfo);
                                    registeredBusinessAndContext.add(registeredContextKey);
                                    registeredServices.add(serviceKey);
                                    log.info("successfully registered context: " + id.stringValue()
                                            + " on business object: " + belongsToId.stringValue());
                                } catch (Exception e) {
                                    log.error("addCategoryToService failed: " + e.getMessage());
                                    try {
                                        regCon.deleteService(serviceKey, authInfo);
                                    } catch (Throwable ex) {
                                        // ignore any exception
                                    }
                                }
                            } catch (Exception e) {
                                log.error("createService failed: " + e.getMessage());
                            }
                            regCon.discardAuthInfo(authInfo);
                        }
                    }
                }
            } finally {
                result.close();
            }
        } finally {
            connection.close();
        }
    } catch (OpenRDFException e) {
        log.error(e.getMessage());
    }
}
```

Figure 30 - Controlling Function of registering Context individuals contained in OWL Ontology in Context Registry

In Figure 30 above, the main controlling function shows how context is read from the ontology and registered with the context registry (UDDI). SPARQL is used as a query language to extract OWL context individuals. If context is available we generate a service key which identifies the agent. This key is used to register the software agent and the context identifier with the context registry.

6.7.2.4 ch.pesto.uddi.tools

This package provides the low level functionality and connectivity to the actual UDDI repository supporting the required UDDI API's along of course with the java client libraries provided by the UDDI software. It further aids relating different registered entities in the UDDI repository and resolving their relationship to each other.

6.7.2.5 ch.pesto.repository.context

This package uses the functionality from ch.pesto.owlcompiler and ch.pesto.classloader to access the context which was passed as an OWL model and OWL individuals. It includes a separate instantiation of an OWL runtime engine instantiating all context information either passed in and statically loaded during start-up or dynamically during runtime through new context readings, through sensors or from user input. Context can be added to the agent's repository either statically as a previously mentioned OWL file or dynamically through sensor readings, user input and so on.

As described earlier OWL business object individuals and OWL context individuals are maintained in two different OWL runtime containers. Due to that fact, context which is loaded statically through an OWL file requires some form of synchronisation between the OWL context repository and the instantiated OWL business object instances to take place. This is achieved by intercepting calls to the business objects to getContext(*) to retrieve the context from the repository and to set the value into the business object as shown through a code fragment in Figure 31.

```
private void loadContextToBusinessObject(String belongsToObjectID, String contextKey, Object original) {
    // get relevant context and put them into the hashMap
    String methodName = "setAssociatedContext" + contextKey;
    Method meth = null;
    Object ctx = null;
    try {
        if (ctx == null) {
            Class cls = Class.forName(ctxBusinessObject);
            org.openrdf.result.impl.ResultImpl it = (org.openrdf.result.impl.ResultImpl) connection.getObjects(cls);

            while (it.hasNext()) {
                Object bo = it.next();
                try {
                    // make sure its really the required object
                    if (cls.isInstance(bo)) {
                        meth = cls.getMethod("getId");
                        String id = (String) meth.invoke(bo);
                        if (id.equals(belongsToObjectID)) {
                            try {
                                Class<?> parameterClass = findMethodParameter(methodName, cls);
                                if (parameterClass != null) {
                                    meth = cls.getMethod(methodName, parameterClass);
                                } else {
                                    throw new Exception("no parameter found to set context");
                                }
                                try {
                                    //loop through children contexts
                                    meth = parameterClass.getMethod("getContains");
                                    Set set = (Set) meth.invoke(original);

                                    for(Object child : set){
                                        connection.addDesignation(child, child.getClass());
                                        connection.addObject(child);
                                    }
                                    connection.addObject(original);
                                    Object bpHist = connection.addDesignation(original, parameterClass.getClass());
                                    meth.invoke(bo, parameterClass.cast(bpHist));
                                    // found the object and set the context we can return now
                                    return;
                                } catch (Exception e) {
                                    log.error("invoking method name to set context failed, reason: "
                                            + e.getMessage());
                                }
                            } catch (Exception e) {
                                log.error("getting method name to set context failed, reason: " + e.getMessage());
                            }
                        }
                    }
                } catch (Exception e) {
                    log.error("error setting context to business object, reason: " + e.getMessage());
                }
            }
        }
    } catch (Exception e) {
        log.error("error setting context to business object");
    }
}
```

Figure 31 - Synchronising Context Repository with Business Object when accessing context

6.7.2.6 ch.pesto.rules

This is a package which allows the integration of different rules engines such as JBoss Drools, iLog or others. This package simply maps internal interfaces into standard JSR-94 or business rules engine specific interfaces which allow the encapsulation of the inner workings to

the reasoning engine and simplifies the switch-over to an alternate reasoning engine. Figure 32 depicts the controlling function of the Drools business rules engine. This code could be easily substituted to interface with another reasoning engine such as iLog.

```
public boolean runRules(List<Object> allObjects, RuleEntry rule) throws AgentException, MissingContextException {
    haltCalled = false;
    try {
        RuleBase ruleBase = RuleBaseFactory.newRuleBase();
        PackageBuilder builder = new PackageBuilder();

        org.drools.io.Resource res = ResourceFactory.newByteArrayResource(rule.ruleContent.getBytes());
        builder.addKnowledgeResource(res, mapRulesType(rule.ruleDialect), null);

        if (!builder.getErrors().isEmpty()) {
            String errMsg = builder.getErrors().toString();
            log.error(errMsg);
            throw new AgentException("Drools rules build errors, reason: " + errMsg);
        }
        Package pkg = builder.getPackage();
        ruleBase.addPackage(pkg);

        workingMemory = ruleBase.newStatefulSession();

        addObjects(workingMemory, allObjects);
        workingMemory.fireAllRules();

    } catch (Exception e) {
        log.error("exception in runRules, reason: " + e.getMessage());
        throw new AgentException(e);
    }
    return true;
}
```

Figure 32 - RunRules controlling function

6.7.2.7 ch.pesto.proxy

This package solves the interception of business rules calls requesting specific context information in the context repository. This is achieved using reflection in java. Every object retrieved from the facts repository mentioned in section 6.7.2.1 in ch.pesto.agent is wrapped with a ShadowProxy class. The ShadowProxy is simply mirroring all functionality of the original Java class through Java reflection which allows the interception of every call made to the context exposed by the business object. This call to the context can then be redirected to the context repository and thus allow full control over the availability of context and the execution of the agent.

Looking at Figure 22 blood pressure is requested from the business object. The ShadowProxy intercepts the call to the context and simply checks whether the business object

provides the context and if the context is not returned by the business object it then redirects the call to the local context repository.

If any context is missing all execution will cease and a rollback will be initiated to the last successfully executed business rule along with all facts and context information available at this point in time. The missing context is queried in the Context Registry to find another suitable agent providing that context which will be able to fulfil the next steps of the business rules execution. Figure 33 shows the implementation of the ShadowProxy where all calls to the business object are monitored and only calls which start with “getAssociatedContext” are intercepted and redirected to the context repository. The reason why the method call “getAssociatedContext” can be hard coded is that it is mandated by the model and was defined in Figure 21.

```
public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    Object result = null;
    final String ctxStr = "getAssociatedContext";
    try {
        result = m.invoke(sourceObject, args);
        //if null object is returned alibaba wraps the null value in a CachedPropertySet instance
        if(result instanceof CachedPropertySet && ((CachedPropertySet)result).isEmpty()){
            result = null;
        }
        //intercept any call to context which retuns a null object
        if(result == null && m.getName().startsWith(ctxStr)){
            if(m.getName().equals(ctxStr)){
                //Sanity check calling this function is not allowed
                log.warn("*** DO NOT CALL getAssociatedContext directly use specialised function such as getAssociatedContextXXX");
                return null;
            }
            //must be a valid call for a context such as getAssociatedContextPAT000100()
            String methodName = m.getName();
            String contextName = methodName.substring(ctxStr.length());

            //get businessObject id
            Method meth = sourceObject.getClass().getMethod("getId");
            String id = (String) meth.invoke(sourceObject);

            ContextRepository ctxRep = ContextRepository.getContextSingletonInstance();
            result = ctxRep.getContext(id, contextName);
            //already proxified object from ContextRepository
            if(result == null){
                RunRules.getInstance().haltRulesProcessing();
                throw new MissingContextException(id, contextName);
            }
            return result;
        }else{
            Class<?> cls = Class.forName("AgentBase.AgentRoot");
            if(cls.isInstance(result)){
                return ShadowProxy.newInstance(result);
            }else if (result instanceof java.util.Set<?>){
                Set retList = (java.util.Set)result;
                Set newSet =  new HashSet();
                for(Object item : retList){
                    newSet.add(ShadowProxy.newInstance(item));
                }
                return newSet;
            }
        }
    } catch (InvocationTargetException e) {
        log.error("Invocation error in ViewProxy reason: " + e.getMessage());
    } catch (MissingContextException e) {
        //just propagate the exception
        throw e;
    } catch (Exception e) {
        log.error("unexpected invocation exception: " + e.getMessage());
    } finally {
    }
    return result;
}
```

Figure 33 - ShadowProxy intercepting calls to Context

6.8 Chapter reviewed

This chapter filled in the conceptual framework components with individual technology and showed how these components communicate with each other. Some attention was given to the inner workings of the software agent as to illustrate the dynamic properties of the software agent interpreting and instantiating the OWL model and facts during runtime as well as to highlight the prerequisites to make this approach work.

One of the goals of this framework was to keep every component of this framework interchangeable. The most apparent component which could be exchanged would be the reasoning component, implemented today with the business rules engine Drools from JBoss [85]. As the semantic web is getting momentum so will the individual components of the semantic web get more mature and in the foreseeable future one might replace a business rules engine with an OWL reasoning component [15, 20, 252].

7 Discussion and Conclusions

We have successfully shown that the software architecture presented that enables a federated approach using software agents, context registry, semantic web and a reasoning engine to access and reason against distributed context is feasible. Through a simple use case which reacts to the patient's blood pressure and weight sensor readings we have shown that the framework handling distributed context performs as we initially expected.

The use case illustrates how specific context is stored in the software agent's context repository and only context type (context codification) and its relation to the context owner and other involved parties (caregiver) is stored in and retrieved from a central context registry (UDDI registry).

All CASAs are designed to communicate directly with each other and to send all information, except the context owned by each CASA, to the next CASA which further processes business rules to resolve for a decision. Sensitive contextual information is not transmitted through the network and no added context security to access context sensitive information is needed for the entire decision making process to finish as the context is only available to the local agent.

This approach is both deployment-time configurable and run-time reconfigurable as outlined in the CARISMA project [95], due to the fact that all information, except the agent's local context, is transmitted to the context agents during runtime and thus makes this approach highly flexible in terms of what rules, ontologies and facts are given to the context agent.

7.1 Research Question Revisited

"What information and components of the framework are necessary to be able to fully support a design oriented context enabled software agent?"

This thesis answered this question by creating the framework containing the different components. The abstract components used were:

- **Definition and description of context** in a standardized manner.
- **Design component** which allowed creating the model of the problem domain and its interrelationships with other models.
- **Instantiation engine** allowing instantiation of objects passed into the software
- **Reasoning engine** to run rules against those objects
- **Context Registry** which enables the storage of context of an entity and the relationship between entities.
- **Runtime Environment** enabling and coordinating the above components

"What information is needed for the framework to describe, find and store context information?"

This question was answered in section 6.2 which describes how context is defined and modelled and how with the same design approach business models can be merged into one comprehensive ontology. It further shows how the context registry taxonomy is related to the modelled context. This chapter then showed how context is registered and how the location of context is retrieved.

"Which components of a software agent must be fully dynamic in terms of model driven design to be able to integrate them in the framework?"

This thesis's framework has successfully implemented different runtime components which are all fully dynamic and are parameterized by the business model, context model, business rules and runtime environment parameterization such as the location of the context registry.

The runtime environment of the software agent can host different problem domains and is designed to be completely agnostic of the problem domain it is running in.

7.2 Conclusions

Although successful, we have encountered one bottleneck in the implementation of the central context registry. Coordinating the registering and unregistering of context information in relation to its software agent location could become a major problem simply due to the fact that there are so many potential queries and updates from all software agents. If the network would grow to reach a global scale and encompass millions of users, the update frequencies of all agents would certainly limit the usefulness of such an approach. Alternate scenarios how to implement and deploy the context registry in a federated approach would be mandated. A possible approach to tackle this problem is to clone the central registry and synchronise these instances with each other. This functionality is already built into many UDDI registries such as Oracle Registry (originally Systinet) which was the selected technology component for the central registry.

There are many challenges and open issues in mobile- or tele-health. These include the advances in mobile technologies, shared taxonomies and ontologies, usability of sensors, privacy and security and other more high-level issues such as governance, payment services and management issues.

We have shown an implementation of an eHealthcare use case. This implementation illustrates that apart from the technical approach there are some benefits to future application in several problem domains. One domain is certainly in healthcare, where local decision making will become more important in the future. Patients will want the results immediately and will increasingly seek to want to participate in their healthcare plan. Thus one of the benefits will be that of local decision making and limited interaction with participants who hold further required context information. The patient will get feedback immediately when no context from outside sources is needed. Information is only passed between software agents when required, thus eliminating the need of higher level coordination as the exchange data and context will be implicit between software agents.

Through the usage of a commonly shared context model, interpretation of context is unambiguous. Context will always represent the same information among all agents.

Based on the stateless design and runtime capabilities of the software agent, this research shows that such a software design allows a flexible adaptation to new business constraints. Due to the fact that the agent does not store any business relevant information apart from shared context information, business constraints can be changed centrally and pushed to other agents when their context information is required.

Other application domains of such an approach might be disaster management where participants in the field are relying on close by location of context as well as military applications, stock trading and other problem domains which could benefit from a generic federated context-aware software architecture presented in this thesis. Disaster management and military scenarios with their characteristic of many local participants and stock trading due to its nature of rapidly changing context could benefit from such an approach. Because context information is not passed from one agent to the other but only the proceeds and the current state of the agent no sensitive information has to be transmitted across the wire. These three problem domains are examples in which business models and instantiation as well as context models are shared and sensitive context information is hidden by the CASA and thus prevents unauthorized access by third parties but still makes context available for local decision making. Local context information is stored locally, processed locally and always kept local.

7.3 Consolidation and Summary

This framework has been designed to accommodate the technical aspects of business models, business rules execution and context registration. Further investigation could research the integration of this approach in terms of IT governance, capability management as well as security management in general. One such security integration could encompass the integration of context access control to specific context taxonomy codes or to control access to context content.

Another research area of interest would be to investigate the integration of other reasoning engines and to create a uniform interface to integrate alternate business rules engines supporting the JSR-94 interface or generally another form of reasoning engine such as Semantic Web Reasoning.

One area of improvement would be to use business rules to react to sensor readings and interpret, aggregate and convert sensor readings into the format given by the OWL Ontology instead of having to customise program widgets in order to convert readings to a specified format.

Additionally, looking at the previous statement UDDI could be extended to support the federation approach as one of its core functionalities. This would allow segregation of registries and dissemination by problem domain, geographically, socially or otherwise to reduce the load on one single context registry. Those federated repositories could be used in environments such as disaster management which are relying heavily on federated systems running independently, allowing runtime execution in small geographical pockets without the need to register with one big central context registry.

We further propose that UDDI Registry could be extended with a function called "findServicesFromRelatedBusinesses" which might increase the framework performance of this framework.

Another area of interest is whether it is feasible to have a distributed approach for the context registry, in light of the intense and time-dependant needs of disaster management. Contextual information might only be available and relevant geographically. This could also reveal and address a potential issue such as performance bottlenecks when too many participants share one common context registry.

Many projects focus mainly on service context and have a very limited focus on user context with location being the most prominent user context information. Ubiquitous computing will become more reliable and future frameworks and/or service providers will integrate and provide alternate forms of fallback options to address issues in service contexts.

One main goal of this software architecture was to transparently determine where data and context is available to be able to propagate decision making to a context-aware service provider in the hierarchy chain where the data and context is available. Additionally, this approach will be able to execute work items where events occur and where high-level or low level contextual user information is available.

A sudden weight loss or gain can be fatal for a patient with a chronic heart disease or a high blood glucose reading. Although, high readings from a recently-taken meal often cannot be related with that meal and thus should not be ignored and naturally an alarm would be issued as a result, but put in context this alarm might be mitigated in the context of that meal. In every circumstance, more information is needed to be able to make an accurate decision. The patient with the chronic heart disease might have had a physical workout without replenishing his body with the necessary fluids. Correlating contextual information is vitally important to be able to obtain a clearer picture about the patient’s wellbeing.

This work has shown the importance of context-aware computing and software in a medical/healthcare context, but there are a wide variety of other places where this computing can be of use. These include commercial applications, such as those which take information about a given person's immediate environment to provide them with, for instance, custom weather messages when they reach a destination. While online retailers already use context-aware software to track customers on their websites, this software can also be used to provide retail information tailored to a person's location at a store or in a mall. This can help save money, too, as companies can use this information as a means for optimizing and economizing a process of companywide software-configuration, such as through analysing their users' context to recognize the software they use, and in particular, the software they don't use, and configuring their workspaces accordingly. While healthcare is clearly the most beneficial area in which this software can be used, for its life-changing implications alone, context-awareness is clearly leading the way in modern software applications. It will be up to the next generation of developers to test out the implications of this new paradigm.

8 Appendix

```java
package ch.pesto.agent.ontology;

import com.thoughtworks.xstream.annotations.XStreamAlias;

@XStreamAlias("AgentParametersIn")
public class AgentParametersIn {
    public static final String ONTOLOGY_NAME = "CONTEXT-AGENT-ONTOLOGY-V1";
    public static final String AGENT_NAME = "CONTEXT-AGENT";
    public static final String AGENT_NAME_TYPE = "context-enabled-agent";

    @XStreamAlias("ontologyURI")
    public String ontologyURI = "";


    @XStreamAlias("owlSchema")
    public String owlSchema = "";

    @XStreamAlias("owlSchemaVersion")
    public String owlSchemaVersion = "";

    @XStreamAlias("uddiTaxonomy")
    public String uddiTaxonomy = "";

    @XStreamAlias("uddiTModelKey")
    public String uddiTModelKey = "";

    //e.g. "patient-caregiver" a string used in UDDI to identify a relationship
    @XStreamAlias("uddiKeyValueRelationship")
    public String uddiKeyValueRelationship = "";

    @XStreamAlias("AgentParametersInOut")
    public AgentParametersInOut agentParam = new AgentParametersInOut();
}
```

Figure 34 - Agent parameters In

```java
package ch.pesto.agent.ontology;
@XStreamAlias("AgentParametersInOut")
public class AgentParametersInOut {

    @XStreamImplicit(itemFieldName = "rules")
    public LinkedList<RuleEntry> rules = new LinkedList<RuleEntry>();

    public boolean moreRulesToProcess() {
        if (rules != null) {
            for (RuleEntry re : rules) {
                if (!re.processed) {
                    return true;
                }
            }
        }
        return false;
    }

    @XStreamAlias("owlInstances")
    public String owlInstances = "";

    @XStreamAlias("missingObjectID")
    public String missingObjectID = "";

    @XStreamAlias("missingContext")
    public String missingContext = "";

    @XStreamAlias("seekingObjectID")
    public String seekingObjectID = "";

    @XStreamAlias("seekingContext")
    public String seekingContext = "";

    @XStreamAlias("errorMessage")
    public String errorMessage = "";
}
```

Figure 35 - Agent parameters inout

9 References

1. Weiser, M., *The computer for the 21st century.* Scientific American, 1995. **272**(3): p. 78-89.
2. Bohn, J., et al., *Social, economic, and ethical implications of ambient intelligence and ubiquitous computing.* Ambient intelligence, 2005: p. 5-29.
3. Ahola, J., *Ambient intelligence.* ERCIM News, 2001. **47**.
4. Schilit, B., N. Adams, and R. Want. *Context-aware computing applications*. 1994.
5. Gellersen, H.W., A. Schmidt, and M. Beigl, *Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts.* Mobile Networks and Applications, 2002. **7**(5): p. 341-351.
6. Berger, M., et al., *An Approach to Agent Based Service Composition and Its Application to Mobile Business Processes.* IEEE TRANSACTIONS ON MOBILE COMPUTING, 2003: p. 197-206.
7. van Setten, M., S. Pokraev, and J. Koolwaaij, *Context-Aware Recommendations in the Mobile Tourist Application COMPASS.* LECTURE NOTES IN COMPUTER SCIENCE, 2004: p. 235-244.
8. Chen, H., T. Finin, and A. Joshi, *An ontology for context-aware pervasive computing environments.* The Knowledge Engineering Review, 2003. **18**(03): p. 197-207.
9. Flury, T., G. Privat, and F. Ramparany, *OWL-based location ontology for context-aware services.* Proceedings of the Artificial Intelligence in Mobile Systems (AIMS 2004), 2004: p. 52–57.
10. Gasevic, D., *Model driven architecture and ontology development.* 2006.
11. Hepp, M. and D. Roman, *An ontology framework for semantic business process management.* Proceedings of Wirtschaftsinformatik, 2007. **2007**.
12. Isern, D., D. Sánchez, and A. Moreno, *An ontology-driven agent-based clinical guideline execution engine.* Artificial Intelligence in Medicine: p. 49-53.
13. Jovanoviæ, J. and D. Gaševiæ, *Ontology-based automatic annotation of learning content.* Int'l J. on Sem. Web and Inf. Sys, 2006. **2**(2).
14. Capra, L., *Reflective Mobile Middleware for Context-Aware Applications*, 2003, University of London.
15. Hong, C.S., et al., *Context modeling and reasoning approach in context-aware middleware for urc system.* International Journal of Mathematical, Physical and Engineering Sciences, 2007. **1**(4): p. 208-212.
16. Qin, W., Y. Shi, and Y. Suo, *Ontology-based context-aware middleware for smart spaces.* Tsinghua Science & Technology, 2007. **12**(6): p. 707-713.
17. Ranganathan, A. and R.H. Campbell, *A Middleware for Context-Aware Agents in Ubiquitous Computing Environments.* LECTURE NOTES IN COMPUTER SCIENCE, 2003: p. 143-161.
18. Yau, S.S., et al., *Reconfigurable Context-Sensitive Middleware for Pervasive Computing.* IEEE Pervasive Computing, 2002. **1**(3): p. 33-40.
19. Strang, T. and C. Linnhoff-Popien. *A context modeling survey*. 2004. Citeseer.
20. Wang, X.H., T.G. Da Qing Zhang, and H.K. Pung, *Ontology based context modeling and reasoning using OWL.* 2004.
21. Evermann, J. and Y. Wand, *Towards ontologically based semantics for UML constructs.* Conceptual Modeling—ER 2001, 2001: p. 354-367.
22. Evermann, J., *A UML and OWL description of Bunge's upper-level ontology model.* Software and Systems Modeling, 2009. **8**(2): p. 235-249.
23. www.ebxml.org. *ebXML - Enabling A Global Electronic Market*. 2009 23 March 2009]; Available from: http://www.ebxml.org/specs/index.htm.

24. uddi.org. *UDDI Version 3.0.2*. 2006 4 November 2008]; Available from: http://uddi.org/pubs/uddi_v3.htm.
25. Doulkeridis, C., E. Valavanis, and M. Vazirgiannis, *Towards a Context-Aware Service Directory.* LECTURE NOTES IN COMPUTER SCIENCE, 2003: p. 54-65.
26. Doulkeridis, C. and M. Vazirgiannis. *Querying and updating a context-aware service directory in mobile environments*. 2004.
27. www.prlog.org. *Context Connect Awarded China Patent for Directory Services*. 2009 27 July 2009]; Available from: http://www.prlog.org/10263137-context-connect-awarded-china-patent-for-directory-services.html.
28. Maamar, Z., S.K. Mostéfaoui, and H. Yahyaoui, *Toward an Agent-Based and Context-Oriented Approach for Web Services Composition.* IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2005: p. 686-697.
29. Hong, J., E. Suh, and S.J. Kim, *Context-aware systems: A literature review and classification.* Expert Systems With Applications, 2009. **36**(4): p. 8509-8522.
30. Paganelli, F. and D. Giuli. *An ontology-based context model for home health monitoring and alerting in chronic patient care networks*. 2007. IEEE.
31. Dey, A.K., G.D. Abowd, and D. Salber, *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications.* Human-Computer Interaction, 2001. **16**(2, 3 & 4): p. 97-166.
32. Miller, J. and J. Mukerji, *Model driven architecture (mda).* Object Management Group, Draft Specification ormsc/2001-07-01, 2001.
33. Roman, M. and R.H. Campbell, *A User-Centric, Resource-Aware, Context-Sensitive, Multi-Device Application Framework for Ubiquitous Computing Environments.* Urbana, 2002. **51**: p. 61801.
34. Steele, R. *A Web Services-based System for Ad-hoc Mobile Application Integration*. 2003.
35. Muller-Wilken, S., F. Wienberg, and W. Lamersdorf. *On integrating mobile devices into a workflow management scenario*. 2000.
36. Fensel, D., *Ontologies: a silver bullet for knowledge management and electronic commerce*2004: Springer Verlag.
37. Morrison, I., et al., *Modelling the Clinical Processes of Prescribing (MCPOP)–Information, Clinical Workflow and Processes.* HIC 2004: Proceedings: p. 194.
38. Robinson, J.C., *The end of managed care.* JAMA: the journal of the American Medical Association, 2001. **285**(20): p. 2622.
39. Broens, T., et al., *Towards an application framework for context-aware m-health applications.* International Journal of Internet Protocol Technology, 2007. **2**(2): p. 109-116.
40. Istepanian, R.S.H. and J.C. Lacal. *Emerging mobile communication technologies for health: some imperative notes on m-health*. 2003. IEEE.
41. Warren, J., *IT-Empowered Consumers and Advocates for Chronic Disease Management.* Health Care and Informatics Review Online, September, 2006.
42. Anderson, R.M. and M.M. Funnell, *Patient empowerment: myths and misconceptions.* Patient education and counseling, 2010. **79**(3): p. 277-282.
43. Grabowski, D.C., et al., *Supporting home-and community-based care: Views of long-term care specialists.* Medical Care Research and Review, 2010. **67**(4 suppl): p. 82S-101S.
44. Bodenheimer, T., *High and rising health care costs. Part 3: the role of health care providers.* Annals of Internal medicine, 2005. **142**(12 Part 1): p. 996-1002.
45. wiki.hl7.org. *Product Infobutton - HL7Wiki*. 2012 15 February 2013; Available from: http://wiki.hl7.org/index.php?title=Product_Infobutton.

46. www.openinfobutton.org. *OpenInfobutton*. 2013 15 February 2013; Available from: http://www.openinfobutton.org/.
47. apelondts.org. *Apelon DTS > Home*. 2013 15 February 2013; Available from: http://apelondts.org/.
48. www.hl7.org. *HL7 Standards Product Brief - HL7 Decision Support Service (DSS), Release 1*. 2013 15 February 2013; Available from: http://www.hl7.org/implement/standards/product_brief.cfm?product_id=12.
49. www.opencds.org. *Home*. 2013 15 February 2013; Available from: http://www.opencds.org/.
50. hssp, d.w.c. *HSSP-DSS - home*. 2013 15 February 2013; Available from: http://hssp-dss.wikispaces.com/.
51. wiki.medpedia.com. *Continuity of Care Record (CCR) Standard - Medpedia*. 2013 15 February 2013; Available from: http://wiki.medpedia.com/Continuity_of_Care_Record_(CCR)_Standard.
52. gello.org. *GELLO Expression Language*. 2013 15 February 2013; Available from: http://gello.org/.
53. Jovanov, E., et al., *A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation.* Journal of neuroengineering and rehabilitation, 2005. **2**(1): p. 6.
54. developer.android.com. *java.lang.reflect | Android Developers*. 2012 15 March 2012; Available from: http://developer.android.com/reference/java/lang/reflect/package-summary.html.
55. Yang, W.S., H.C. Cheng, and J.B. Dia, *A location-aware recommender system for mobile shopping environments.* Expert Systems With Applications, 2008. **34**(1): p. 437-445.
56. Resatsch, F., et al., *Do Point of Sale RFID-Based Information Services Make a Difference? Analyzing Consumer Perceptions for Designing Smart Product Information Services in Retail Business.* Electronic Markets, 2008. **18**(3): p. 216-231.
57. Kwon, O.B. and N. Sadeh, *Applying case-based reasoning and multi-agent intelligent system to context-aware comparative shopping.* Decision Support Systems, 2004. **37**(2): p. 199-213.
58. Gu, H. and D. Wang. *A content-aware fridge based on RFID in smart home for home-healthcare*. 2009. IEEE.
59. Kok, K., et al. *Smart houses for a smart grid*. 2009. IET.
60. Kunze, C., et al., *Application of ubiquitous computing in personal health monitoring systems.* Biomedizinische Technik, 2002. **47**: p. 360–362.
61. Giraldo, C., S. Helal, and W. Mann. *mPCA–A mobile patient care-giving assistant for Alzheimer patients*. 2002.
62. Khoumbati, K., M. Themistocleous, and Z. Irani. *Integration technology adoption in healthcare organisations: a case for enterprise application integration*. 2005. Ieee.
63. Schekkerman, J., *How to survive in the jungle of enterprise architecture frameworks: Creating or choosing an enterprise architecture framework*2006: Trafford.
64. Lankhorst, M., *Enterprise architecture at work: Modelling, communication and analysis*2009: Springer-Verlag New York Inc.
65. www.w3.org. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 2007 4 November 2008]; Available from: http://www.w3.org/TR/soap12-part1/.
66. jcp.org. *JSR-000311 JAX-RS: The Java API for RESTful Web Services - Final Release*. 2012 22 March 2012; Available from: http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html.
67. activiti.org. *Activiti*. 2011 15 May 2012; Available from: http://activiti.org/.
68. www.intalio.com. *Business Process Management System (BPMS) - Intalio The Private Cloud Company*. 2012 11 March 2012; Available from: http://www.intalio.com/bpms.

69. www.mulesoft.org. *Mule ESB - Open Source ESB Community w. Documentation | Mule ESB Community*. 2012 5 Feb 2012; Available from: http://www.mulesoft.org/.
70. docs.oracle.com. *Introduction Oracle® Fusion Middleware Concepts and Architecture for Oracle Service Bus*. 2011 27 March 2012; Available from: http://docs.oracle.com/cd/E21764_01/doc.1111/e15020/introduction.htm.
71. msdn.microsoft.com. *An Overview of Service-Oriented Architecture in Retail*. 2012 13 April 2012; Available from: http://msdn.microsoft.com/en-us/library/bb264584.aspx.
72. Fowler, M., *Patterns of enterprise application architecture*2002: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
73. Endrei, M. and I. Books24x7, *Patterns: service-oriented architecture and web services*2004: IBM Corp., International Technical Support Organization.
74. Kuusisto, J. and M. Meyer, *Insights into services and innovation in the knowledge intensive economy.* Technology Review, 2003. **134**: p. 2003.
75. Davies, J., D. Fensel, and F. Van Harmelen, *Towards the semantic web: ontology-driven knowledge management*2003: Wiley.
76. Channabasavaiah, K., K. Holley, and E. Tuggle, *Migrating to a service-oriented architecture.* IBM DeveloperWorks, 2003. **16**.
77. Dagger, D., et al., *Service-oriented e-learning platforms: From monolithic systems to flexible services.* Internet Computing, IEEE, 2007. **11**(3): p. 28-35.
78. Schreiber, G., *Knowledge engineering and management: the CommonKADS methodology*2000: the MIT Press.
79. Date, C.J., *What not how: the business rules approach to application development*2000: Addison-Wesley Professional.
80. Von Halle, B., *Business rules applied: building better systems using the business rules approach.* 2001.
81. Ross, R.G., *Principles of the business rule approach*2003: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
82. herzberg.ca.sandia.gov. *Jess, the Rule Engine for the Java Platform*. 2008 11 November 2008]; Available from: http://herzberg.ca.sandia.gov/.
83. protege.stanford.edu. *The Protégé Ontology Editor and Knowledge Acquisition System*. 13 September 2010]; Available from: http://protege.stanford.edu/.
84. ruleml.org. *RuleML Homepage*. 2012 11 April 2012; Available from: http://ruleml.org/.
85. www.jboss.org. *Drools - JBoss Community*. 2012 9 April 2012; Available from: http://www.jboss.org/drools/.
86. www, i.c. *IBM - IBM ILOG*. 2012 9 April 2012; Available from: http://www-01.ibm.com/software/websphere/ilog/#.
87. www.rosettanet.org. *RosettaNet Home*. 2009 23 March 2009]; Available from: http://www.rosettanet.org/cms/sites/RosettaNet/.
88. Rosenberg, F. and S. Dustdar. *Design and Implementation of a Service-Oriented Business Rules Broker*. 2005.
89. www.w3.org. *OWL-S: Semantic Markup for Web Services*. 2004 23 March 2009]; Available from: http://www.w3.org/Submission/OWL-S/#Hendler-McGuinness.
90. www.daml.org. *DAML.org*. 2006 23 March 2009]; Available from: http://www.daml.org/.
91. Nagl, C., F. Rosenberg, and S. Dustdar, *Vi DRE–A Distributed Service-Oriented Business Rule Engine based on RuleML.* Technical University of Vienna Information Systems Institute Distributed Systems Group Technical Report TUV-1841-2006, 2006. **38**.

92. Korpipaa, P., et al., *Managing context information in mobile devices.* Pervasive Computing, IEEE, 2003. **2**(3): p. 42-51.
93. Henricksen, K. and J. Indulska. *Modelling and using imperfect context information*. Citeseer.
94. Charfi, A. and M. Mezini. *An aspect-based process container for BPEL*. 2005. ACM New York, NY, USA.
95. Capra, L., W. Emmerich, and C. Mascolo, *CARISMA: Context-Aware Reflective Middleware System for Mobile Applications.* IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2003: p. 929-945.
96. www.w3.org. *Implementations - OWL*. 2012 9 April 2012; Available from: http://www.w3.org/2007/OWL/wiki/Implementations.
97. Bodenheimer, T., *High and rising health care costs. Part 2: technologic innovation.* Annals of Internal medicine, 2005. **142**(11): p. 932.
98. Robu, I., V. Robu, and B. Thirion, *An introduction to the Semantic Web for health sciences librarians.* Journal of the Medical Library Association, 2006. **94**(2): p. 198.
99. Rubin, D.L., N.H. Shah, and N.F. Noy, *Biomedical ontologies: a functional perspective.* Briefings in bioinformatics, 2008. **9**(1): p. 75-90.
100. Bodenreider, O., *Biomedical ontologies in action: role in knowledge management, data integration and decision support.* Yearb Med Inform, 2008. **67**: p. 79.
101. Albee, G.W., *The Boulder model's fatal flaw.* American Psychologist, 2000. **55**(2): p. 247.
102. Horwitz, J.R., *Making profits and providing care: Comparing nonprofit, for-profit, and government hospitals.* Health Affairs, 2005. **24**(3): p. 790-801.
103. Catalino, J., *Software solutions can trim rising costs. Providers can take a number of actions right now to reduce healthcare costs, and improve efficiency and effectiveness of the US healthcare system.* Health management technology, 2010. **31**(3): p. 10.
104. Robinson, S., *Urgent care software drives GP cost savings.* Haymarket Business Publications Ltd. - Advertising And Public Relations, 2011: p. 1.
105. Lesser, C.S., P.B. Ginsburg, and K.J. Devers, *The End of an Era: What Became of the "Managed Care Revolution" in 2001?* Health Services Research, 2003. **38**(1p2): p. 337-355.
106. Ferlie, E.B. and S.M. Shortell, *Improving the quality of health care in the United Kingdom and the United States: a framework for change.* Milbank Quarterly, 2001. **79**(2): p. 281-315.
107. Ronald M. Epstein, M.R.L.S., Jr, PhD, *The Values and Value of Patient-Centered Care.* Annals of Family Medicine, 2011. : p. 9:100-103.
108. Rao, S., et al., *Health information technology: transforming chronic disease management and care transitions.* Primary care, 2012. **39**(2): p. 327.
109. Moore, S.K., *Extending healthcare's reach.* Spectrum, IEEE, 2002. **39**(1): p. 66-71.
110. Puentes, J., et al., *Telemedicine trends and challenges: a technology management perspective.* International Journal of Biomedical Engineering and Technology, 2007. **1**(1): p. 59-72.
111. Lawrence, D., *Telemedicine.* Healthcare Informatics, 2009. **26(2)**: p. 42-44.
112. Bashshur, R., et al., *The Taxonomy of Telemedicine.* Telemedicine and e-Health, 2011. **17**(6): p. 484-494.
113. *More home monitoring seen as future of heart failure treatment.* Healthy Years, 2012. **9**(3): p. 3.
114. www.ehi.co.uk. *E-Health Insider :: Berlin University opens telemedicine centre*. 2008 9 April 2008; Available from: http://www.ehi.co.uk/news/primary-care/3630.
115. www.tagesschau.de. *"Telemedizin" soll Gang zum Arzt ersparen*. 2008 10 January 2008; Available from: http://www.tagesschau.de/inland/telemedizin2.html.

116. www.partnership for the, h.d. *Medica: Klinische Studie von Partnership for the Heart vorgestellt: Partnership for the Heart*. 2011 1 June 2011; Available from: http://www.partnership-for-the-heart.de/presse/newsdetails/tt_news/23/1/?cHash=ced59f9c00.
117. Wegdam, M. *AWARENESS: A project on Context AWARE mobile NEtworks and ServiceS*. 2005.
118. Wood, A., et al., *Context-aware wireless sensor networks for assisted living and residential monitoring.* Network, IEEE, 2008. **22**(4): p. 26-33.
119. www.pcmh.ahrq.gov. *PCMH: Home*. 2011 28 March 2011; Available from: http://www.pcmh.ahrq.gov/portal/server.pt/community/pcmh__home/1483.
120. Stange, K.C., et al., *Defining and measuring the patient-centered medical home.* Journal of general internal medicine, 2010. **25**(6): p. 601-612.
121. Bitton, A., C. Martin, and B.E. Landon, *A nationwide survey of patient centered medical home demonstration projects.* Journal of general internal medicine, 2010. **25**(6): p. 584-592.
122. www.annfammed.org. *Initial Lessons From the First National Demonstration Project on Practice Transformation to a Patient-Centered Medical Home*. 2012 7 April 2012; Available from: http://www.annfammed.org/content/7/3/254.full.
123. Nutting, P.A., et al., *Journey to the patient-centered medical home: a qualitative analysis of the experiences of practices in the National Demonstration Project.* The Annals of Family Medicine, 2010. **8**(Suppl_1): p. S45.
124. Nutting, P.A., et al., *Initial lessons from the first national demonstration project on practice transformation to a patient-centered medical home.* Annals of Family Medicine, 2009. **7**(3): p. 254.
125. xml.coverpages.org. *Cover Pages: Web Services Flow Language (WSFL)*. 2002 11 April 2012; Available from: http://xml.coverpages.org/wsfl.html.
126. www.ebpml.org. *XLANG*. 2008 11 April 2012; Available from: http://www.ebpml.org/xlang.htm.
127. bpel.xml.org. *BPEL XML.org | Online community for the Web Services Business Process Execution Language OASIS Standard*. 2011 11 April 2011; Available from: http://bpel.xml.org/.
128. Orriens, B., J. Yang, and M.P. Papazoglou, *A Framework for Business Rule Driven Service Composition.* LECTURE NOTES IN COMPUTER SCIENCE, 2003: p. 14-27.
129. Machado, M.T., M.R.S. Borges, and R.M. Araujo. *A non-intrusive infrastructure for the integration of business processes*. in *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on*. 2004.
130. Kloppmann, M., D. Konig, and G. Pfau, *Business Process Standards-Current Landscape (Landkarte aktueller Standards zur Beschreibung von Geschaftsprozessen: WS-BPEL, BPEL4People, BPEL-SPE, BPELJ, SCA).* it-Information Technology (vormals it+ ti), 2008. **50**(2/2008): p. 93-98.
131. Peltz, C., *Web Services Orchestration and Choreography.* COMPUTER, 2003: p. 46-52.
132. van der Aalst, W.M.P., *Don't go with the flow: Web services composition standards exposed.* IEEE Intelligent Systems, 2003. **18**(1): p. 72-76.
133. www, i.c. *WS-BPEL Extension for People*. 2007 5 November 2008]; Available from: http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/.
134. White, S.A., *Introduction to BPMN.*
135. www.bpmi.org. *Business Process Management Initiative*. 2009 3 January 2009]; Available from: http://www.bpmi.org/.
136. www.ebpml.org. *BPML*. 2009 11 April 2012; Available from: http://www.ebpml.org/bpml.htm.
137. www.businessrulesgroup.org. *Defining Business Rules ~ What Are They Really? (Chapter 1)*. 2006 2 February 2009]; Available from: http://www.businessrulesgroup.org/first_paper/br01c1.htm.
138. Von Halle, B., *Business rules applied*2002: John Wiley & Sons New York.

139. www.businessrulesgroup.org. *BRG: The Business Rules Approach*. 2008 5 November 2008]; Available from: http://www.businessrulesgroup.org/bra.shtml.
140. www.businessrulesgroup.org. *BRG-whatisBR_3ed.pdf (application/pdf Object)*. 2004 20 June 2012; Available from: http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf.
141. Rosenberg, F. and S. Dustdar. *Business Rules Integration in BPEL–A Service-Oriented Approach*. 2005.
142. Rosenberg, F. and S. Dustdar. *Towards a Distributed Service-Oriented Business Rules System*. 2005. IEEE Computer Society Washington, DC, USA.
143. www.oasis, o.o. *About Us | OASIS*. 2012 16 May 2012; Available from: http://www.oasis-open.org/org.
144. Papazoglou, M.P., *Web Services and Business Transactions.* World Wide Web, 2003. **6**(1): p. 49-91.
145. docs.oasis, o.o. *OASIS Web Services Coordination Specification*. 2009 7 April 2012; Available from: http://docs.oasis-open.org/ws-tx/wscoor/2006/06.
146. Charfi, A. and M. Mezini. *Hybrid web service composition: business processes meet business rules*. 2004. ACM New York, NY, USA.
147. Kuhlemann, M., et al., *A multiparadigm study of crosscutting modularity in design patterns.* Objects, Components, Models and Patterns, 2008: p. 121-140.
148. Charfi, A. and M. Mezini, *Aspect-Oriented Web Service Composition with AO4BPEL.* LECTURE NOTES IN COMPUTER SCIENCE, 2004: p. 168-182.
149. www.eclipse.org. *The AspectJ Project*. 2009 12 February 2009]; Available from: http://www.eclipse.org/aspectj/.
150. www.zapthink.com. *ZapThink :: Research - Divorcing SOA and Web Services*. 2008 4 November 2008]; Available from: http://www.zapthink.com/report.html?id=ZAPFLASH-2007618.
151. www.w3.org. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. 2003 4 November 2008]; Available from: http://www.w3.org/TR/2003/WD-wsdl20-20031110/.
152. www.zurich.ibm.com. *icaps-ws.pdf (application/pdf Object)*. 2003 4 November 2008]; Available from: http://www.zurich.ibm.com/pdf/ebizz/icaps-ws.pdf.
153. www.ibm.com. *Web Services architecture overview*. 2008 4 November 2008]; Available from: http://www.ibm.com/developerworks/library/w-ovr/.
154. java.sun.com. *Overview of SOAP*. 2008 4 November 2008]; Available from: http://java.sun.com/developer/technicalArticles/xml/webservices/.
155. xml.coverpages.org. *Cover Pages: OASIS Consortium Members Approve UDDI Version 3 as an OASIS Standard*. 2005 4 November 2008]; Available from: http://xml.coverpages.org/ni2005-02-02-a.html.
156. download.oracle.com. *UDDI from Developer Tools*. 2009 18 October 2010]; Available from: http://download.oracle.com/docs/cd/E13173_01/alsr/docs20/help/dev_guide/pr.uddi.dev.html .
157. sourceforge.net. *UDDI4J | Download UDDI4J software for free at SourceForge.net*. 18 October 2010]; Available from: http://sourceforge.net/projects/uddi4j/.
158. juddi.apache.org. *Chapter 8. UDDI Annotations*. 18 October 2010]; Available from: http://juddi.apache.org/docs/3.0/userguide/html/chap-UDDI_annotations.html.
159. glassfish.dev.java.net. *glassfish: GlassFish 3.0.1 Final*. 18 October 2010]; Available from: https://glassfish.dev.java.net/downloads/3.0.1-final.html.
160. Paolucci, M., et al., *Importing the semantic web in UDDI.* LECTURE NOTES IN COMPUTER SCIENCE, 2002: p. 225-236.

161. Pokraev, S., J. Koolwaaij, and M. Wibbels. *Extending UDDI with context-aware features based on semantic service descriptions*. 2003.
162. Chafle, G.B., et al. *Decentralized orchestration of composite web services*. 2004. ACM New York, NY, USA.
163. Roman, D., *Web Service Modeling Ontology.* Applied Ontology, 2005. **1**(1): p. 77-106.
164. Schilit, W.N., *A system architecture for context-aware mobile computing*, 1995, Columbia University.
165. Satyanarayanan, M., *Pervasive computing: vision and challenges.* Personal Communications, IEEE [see also IEEE Wireless Communications], 2001. **8**(4): p. 10-17.
166. Pascoe, J. *Adding generic contextual capabilities to wearable computers*. 1998. Citeseer.
167. Cabitza, F. and B. Dal Seno, *DJess–A Knowledge-Sharing Middleware to Deploy Distributed Inference Systems.* Proceedings of ENFORMATIKA'05 Joint Conferences: p. 25-27.
168. Cabitza, F., M. Sarini, and B. Dal Seno. *DJess-a context-sharing middleware to deploy distributed inference systems in pervasive computing domains*. 2005.
169. Prekop, P. and M. Burnett, *Activities, context and ubiquitous computing.* Computer Communications, 2003. **26**(11): p. 1168-1176.
170. Zhang, D., Z. Yu, and C. Chin, *Context-aware infrastructure for personalized healthcare.* Studies in Health Technology and Informatics, 2005. **117**: p. 154-163.
171. Want, R., et al., *The active badge location system.* ACM Transactions on Information Systems (TOIS), 1992. **10**(1): p. 102.
172. Schilit, B.N. and M.M. Theimer, *Disseminating active map information to mobile hosts.* IEEE network, 1994. **8**(5): p. 22-32.
173. Dey, A.K. and G.D. Abowd. *Towards a Better Understanding of Context and Context-Awareness*. in *Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness*. 2000. ACM Press, New York.
174. Dey, A.K., *Providing Architectural Support for Building Context-Aware Applications*, 2000, Georgia Institute of Technology.
175. Zhang, D., et al., *Survey on context-awareness in ubiquitous media.* Multimedia Tools and Applications, 2011: p. 1-33.
176. Bellotti, V. and K. Edwards, *Intelligibility and accountability: human considerations in context-aware systems.* Human-Computer Interaction, 2001. **16**(2): p. 193-212.
177. Barkhuus, L. and A. Dey. *Is context-aware computing taking control away from the user? Three levels of interactivity examined*. 2003. Springer.
178. Brown, B. and R. Randell, *Building a Context Sensitive Telephone: Some Hopes and Pitfalls for Context Sensitive Computing.* Computer Supported Cooperative Work, 2004. **13**(3-4): p. 329-345.
179. Fogarty, J., J. Lai, and J. Christensen, *Presence versus availability: the design and evaluation of a context-aware communication client.* International Journal of Human-Computer Studies, 2004. **61**(3): p. 299-317.
180. www.omg.org. *MDA_Guide_Version1-0.pdf (application/pdf Object)*. 2009 11 October 2010]; Available from: http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf.
181. www.omg.org. *MDAFAQfinal1.pdf (application/pdf Object)*. 2009 13 September 2010]; Available from: http://www.omg.org/mda/mda_files/MDAFAQfinal1.pdf.
182. www.omg.org. *OMG Document -- omg/03-06-01 (MDA Guide V1.0.1)*. 15 September 2010]; Available from: http://www.omg.org/cgi-bin/doc?omg/03-06-01.

183. www.methodsandtools.com. *Understanding the Model Driven Architecture (MDA)*. 15 September 2010]; Available from: http://www.methodsandtools.com/archive/archive.php?id=5.
184. Kleppe, A.G., J. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*2003: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
185. Poole, J.D., *Model Driven Architecture: Vision, Standards And Emerging Technologies.* 2001.
186. Zhu, L., *Model Driven Architecture.* Essential Software Architecture, 2006: p. 197-215.
187. Sendall, S. and W. Kozaczynski, *Model transformation: The heart and soul of model-driven software development.* IEEE software, 2003. **20**(5): p. 42-45.
188. technology.amis.nl. *Model Driven Architecture (MDA) « AMIS Technology blog*. 15 September 2010]; Available from: http://technology.amis.nl/blog/1178/model-driven-architecture-mda.
189. www.cas.mcmaster.ca. *Feb06-OCL-WenYu.pdf (application/pdf Object)*. 2007 22 September 2010]; Available from: http://www.cas.mcmaster.ca/~sartipi/course/cas707/w07/slides/Feb06-OCL-WenYu.pdf.
190. Bjerkander, M. and C. Kobryn, *Architecting systems with UML 2.0.* IEEE software, 2003. **20**(4): p. 57-61.
191. Mostéfaoui, S.K. *Towards a Context-Oriented Services Discovery and Composition Framework*. 2003.
192. Woolridge, M. and M.J. Wooldridge, *Introduction to Multiagent Systems*2001, : John Wiley & Sons, Inc. New York, NY, USA.
193. Franklin, S. and A. Graesser. *Is it an agent, or just a program*. 1996.
194. Cohen, P.R., et al. *An open agent architecture*. 1994.
195. Hayes-Roth, B., *An architecture for adaptive intelligent systems.* Artificial Intelligence, 1995. **72**(1-2): p. 329-365.
196. www.w3schools.com. *Semantic Web*. 28 September 2010]; Available from: http://www.w3schools.com/semweb/default.asp.
197. Maedche, A. and S. Staab, *Ontology learning for the semantic web.* Intelligent Systems, IEEE, 2005. **16**(2): p. 72-79.
198. Gruber, T.R., *Toward principles for the design of ontologies used for knowledge sharing.* International Journal of Human Computer Studies, 1995. **43**(5): p. 907-928.
199. Borst, W.N., *Construction of engineering ontologies for knowledge sharing and reuse*1997: Universiteit Twente.
200. Studer, R., V.R. Benjamins, and D. Fensel, *Knowledge engineering: principles and methods.* Data & knowledge engineering, 1998. **25**(1): p. 161-197.
201. Staab, S. and R. Studer, *Handbook on ontologies*2009: Springer.
202. Benjamins, V.R. and A.G. Pérez, *Knowledge-System Technology: Ontologies and Problem-Solving Methods.* 1999.
203. www.w3.org. *OWL Web Ontology Language Reference*. 2009 11 October 2010]; Available from: http://www.w3.org/TR/owl-ref/.
204. www.omg.org. *07-2_A_UML_Profile_for_the_Web_Ontology_Language_OWL.pdf (application/pdf Object)*. 2009 11 October 2010]; Available from: http://www.omg.org/news/meetings/workshops/UML2002-Manual/07-2_A_UML_Profile_for_the_Web_Ontology_Language_OWL.pdf.
205. McGuinness, D.L. and F. Van Harmelen, *OWL web ontology language overview.* W3C recommendation, 2004. **10**: p. 2004-03.
206. www.w3.org. *RDF - Semantic Web Standards*. 2010 11 January 2010; Available from: http://www.w3.org/RDF/.

207. www.w3.org. *OWL 2 Web Ontology Language Document Overview*. 2009 11 October 2010]; Available from: http://www.w3.org/TR/owl2-overview/.
208. www.w3.org. *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0*. 2003 1 February 2011; Available from: http://www.w3.org/TR/2003/PR-CCPP-struct-vocab-20031015/.
209. Alliance, O., *User Agent Profile (UAProf) specification*.
210. Indulska, J., et al. *Experiences in using CC/PP in context-aware systems*. 2003. Springer.
211. Preuveneers, D., et al., *Towards an extensible context ontology for ambient intelligence.* Ambient intelligence, 2004: p. 148-159.
212. Strang, T., C. Linnhoff-Popien, and K. Frank. *CoOL: A context ontology language to enable contextual interoperability*. 2003. Springer.
213. Gu, T., H. Pung, and D. Zhang, *Toward an OSGi-based infrastructure for context-aware applications.* Pervasive Computing, IEEE, 2005. **3**(4): p. 66-74.
214. Gaševic, D., et al. *Approaching OWL and MDA through technological spaces*. 2004. Citeseer.
215. www.eclipse.org. *ATL Use Case - ODM Implementation (Bridging UML and OWL)*. 11 October 2010]; Available from: http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/.
216. www.omg.org. *ODM 1.0*. May 2009 11 October 2010]; Available from: http://www.omg.org/spec/ODM/1.0/.
217. Thomas, D., *MDA: Revenge of the Modelers or UML Utopia?* IEEE software, 2004. **21**(3): p. 15-17.
218. Long, Q., et al., *Consistent code generation from UML models.* 2005.
219. Riehle, D., et al. *The architecture of a UML virtual machine*. 2001. ACM.
220. www.openrdf.org. *AliBaba 2.0-beta3 - Object Repository*. 3 November 2010]; Available from: http://www.openrdf.org/doc/alibaba/2.0-beta3/alibaba-repository-object/index.html.
221. www.openrdf.org. *openRDF.org: Home*. 5 January 2010]; Available from: http://www.openrdf.org/.
222. Baldauf, M., S. Dustdar, and F. Rosenberg, *A survey on context-aware systems.* International Journal of Ad Hoc and Ubiquitous Computing, 2007. **2**(4): p. 263-277.
223. ewh.ieee.org. *Pervasive Computing*. 2002 19 November 2010]; Available from: http://ewh.ieee.org/r10/bombay/news4/Pervasive_Computing.htm.
224. Garlan, D., et al., *Project aura: Toward distraction-free pervasive computing.* IEEE Pervasive Computing, 2002: p. 22-31.
225. www.hpl.hp.com. *HPL-2003-192.pdf (application/pdf Object)*. 2003 19 November 2010]; Available from: http://www.hpl.hp.com/techreports/2003/HPL-2003-192.pdf?jumpid=reg_R1002_USEN.
226. oxygen.lcs.mit.edu. *MIT Project Oxygen: Overview*. 21 June 2010]; Available from: http://oxygen.lcs.mit.edu/Overview.html.
227. Yao, W., et al. *Using Ontology to Support Context Awareness in Healthcare*. 2009.
228. Peffers, K., et al., *A design science research methodology for information systems research.* Journal of Management Information Systems, 2007. **24**(3): p. 45-77.
229. Simon, H.A., *The sciences of the arti cial*, 1969, Cambridge, MA: MIT Press.
230. Winter, R., *Design science research in Europe.* European Journal of Information Systems, 2008. **17**(5): p. 470-475.
231. *ISR. Editorial Statement and Policy* Information Systems Research, December 2002.
232. Hevner, A.R., et al., *Design science in information systems research.* Mis Quarterly, 2004. **28**(1): p. 75-105.
233. Zmud, R., *Editor's Comments.* MIS Quarterly, 1997(21:2): p. pp. xxi-xxii.

234. Lee. *Keynote - ICIM 2000 - Lee*. 2000 15 February 2011; Available from: http://www.people.vcu.edu/~aslee/ICIM-keynote-2000/ICIM-keynote-2000.htm.
235. Aboulafia, M., *Philosophy, social theory, and the thought of George Herbert Mead*1991: State Univ of New York Pr.
236. Weber, R., *Toward a theory of artifacts: a paradigmatic base for information systems research.* Journal of Information Systems, 1987. **1**(2): p. 3-19.
237. March, S.T. and G.F. Smith, *Design and natural science research on information technology.* Decision Support Systems, 1995. **15**(4): p. 251-266.
238. Nunamaker Jr, J.F., M. Chen, and D.M. Purdin. *Systems development in information systems research*. 1990. IEEE.
239. Franklin, J. and A. Daoud, *Proof in mathematics: an introduction*2011: Kew Books.
240. desrist.iwi.unisg.ch. *DESRIST 2010 Home*. 2010 14 February 2012; Available from: http://desrist2010.iwi.unisg.ch/home/.
241. Zhao, J.L., R. Winter, and S. Aier, *Global Perspectives on Design Science Research.* LECTURE NOTES IN COMPUTER SCIENCE, 2010. **6105**.
242. Yu, W., K. Kangas, and S. Brewster. *Web-based haptic applications for blind people to create virtual graphs*. 2003. IEEE.
243. Johnson, V., et al. *The San Francisco project: An object-oriented framework approach to building business applications*. 1997. IEEE.
244. www.businessreviewonline.com. *The 10 Best Open Source Rules Engines*. 2008 9 December 2008]; Available from: http://www.businessreviewonline.com/os/archives/2008/07/10_best_open_so.html.
245. Müller, T. *Conceptualization of a generic context model in OWL*. 2010 19 January 2010; Available from: http://on.cs.unibas.ch/owl/1.0/Context.owl.
246. Colombo, E., J. Mylopoulos, and P. Spoletini, *Modeling and Analyzing Context-Aware Composition of Services.* LECTURE NOTES IN COMPUTER SCIENCE, 2005. **3826**: p. 198.
247. Chaari, T., F. Laforest, and A. Celentano, *Design of context-aware applications based on web services.* Universita Ca'Foscari di Venezia Reference: Technical Report CS, 2004.
248. Autili, M., et al. *A conceptual model for adaptable context-aware services*.
249. openjena.org. *Jena Semantic Web Framework*. 2 January 2010]; Available from: http://openjena.org/.
250. Vaishnavi, V., V.K. Vaishnavi, and W. Kuechler, *Design science research methods and patterns: innovating information and communication technology*2007: Auerbach Pub.
251. Fensel, D., et al., *OIL: An ontology infrastructure for the semantic web.* Intelligent Systems, IEEE, 2005. **16**(2): p. 38-45.
252. Rubin, D.L., et al. *Protégé-OWL: Creating ontology-driven reasoning applications with the web ontology language*. 2005. American Medical Informatics Association.
253. www.microsoft.com. *Active Directory Server | Identity | Credential | Protection Overview*. 2012 24 April 2012; Available from: http://www.microsoft.com/en-us/server-cloud/windows-server/active-directory-overview.aspx.
254. www.openldap.org. *OpenLDAP, Main Page*. 2012 24 April 2012; Available from: http://www.openldap.org/.
255. Taboada, M., D. Martínez, and J. Mira, *Experiences in reusing knowledge sources using Protégé and PROMPT.* International Journal of Human-Computer Studies, 2005. **62**(5): p. 597-618.
256. Smith, B., et al. *Towards a reference terminology for ontology research and development in the biomedical domain*. in *Proceedings of KR-MED*. 2006.

257. protegewiki.stanford.edu. *Protege Ontology Library - Protege Wiki*. 20 December 2009]; Available from: http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library#OWL_ontologies.
258. swoogle.umbc.edu. *Swoogle Semantic Web Search Engine*. 2012 22 June 2012; Available from: http://swoogle.umbc.edu/.
259. jade.tilab.com. *Jade - Java Agent DEvelopment Framework*. 2010 3 November 2010]; Available from: http://jade.tilab.com/.
260. xstream.codehaus.org. *XStream - About XStream*. 2011 29 January 2012; Available from: http://xstream.codehaus.org/.
261. www.semfinder.com. *Semfinder: Home*. 2011 1 November 2011; Available from: http://www.semfinder.com/.
262. www.semfinder.com. *Semfinder: Code*. 2012 13 April 2012; Available from: http://www.semfinder.com/en/method/semantic-interpreter/code.html.
263. apps.who.int. *ICD-10 Version:2010*. 2013 13 February 2013; Available from: http://apps.who.int/classifications/icd10/browse/2010/en.
264. www.openehr.org. *openEHR - What is openEHR?* 2013 13 February 2013; Available from: http://www.openehr.org/what_is_openehr.
265. Annas, G.J., *HIPAA regulations—a new era of medical-record privacy?* New England Journal of Medicine, 2003. **348**(15): p. 1486-1490.
266. www.hhs.gov. *HIPAA Administrative Simplification Statute and Rules*. 2012 19 July 2012; Available from: http://www.hhs.gov/ocr/privacy/hipaa/administrative/index.html.
267. www.privacy.gov.au. *Privacy Act*. 2012 19 July 2012; Available from: http://www.privacy.gov.au/law/act.
268. www.legislation.gov.uk. *Data Protection Act 1998*. 2012 19 July 2012; Available from: http://www.legislation.gov.uk/ukpga/1998/29/contents.
269. www.health.gov.au. *Person-controlled Electronic Health Records.pdf (application/pdf Object)*. 2009 19 July 2012; Available from: http://www.health.gov.au/internet/nhhrc/publishing.nsf/Content/BA7D3EF4EC7A1F2BCA25755B001817EC/$File/Person-controlled%20Electronic%20Health%20Records.pdf.
270. jade.tilab.com. *JADE_Security.pdf (application/pdf Object)*. 2010 22 June 2012; Available from: http://jade.tilab.com/doc/tutorials/JADE_Security.pdf.
271. Vila, X., A. Schuster, and A. Riera, *Security for a Multi-Agent System based on JADE.* Computers & Security, 2007. **26**(5): p. 391-400.
272. Szpryngier, P. and M. Matuszek. *Selected security aspects of agent-based computing*. 2010. IEEE.
273. shiro.apache.org. *Apache Shiro | Java Security Framework*. 2012 22 June 2012; Available from: http://shiro.apache.org/.
274. www.ontotext.com. *OWLIM | Ontotext*. 2009 17 June 2009; Available from: http://www.ontotext.com/owlim.