Domain Independent Context Awareness Framework

By

Mira Vrbaski, M.Eng

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Applied Science
in
Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University
Ottawa, Ontario

December 2012

Your file Votre référence
ISBN: 978-0-494-94271-0

Our file Notre référence
ISBN: 978-0-494-94271-0

Your file Votre référence
ISBN: 978-0-494-94271-0

Our file Notre référence
ISBN: 978-0-494-94271-0

ABSTRACT

We are witnessing a significant expansion and penetration of wireless appliances, sensors, smart phones, and PDAs in a wide range of domains such as smart homes, hospitals, and elderly home care. This expansion of the highly dynamic pervasive computing paradigm is expected to continue over the decade. In this paradigm, context-awareness is the ability of a system to (i) adapt to an ever-changing landscape of users, sensors, devices, and information and (ii) proactively anticipate a user's needs without placing the burden on the user. Consequently, one highly important component of a context-aware system is context reasoning, because it provides system flexibility and automated run-time decision-making based on preconfigured criteria. The thesis proposes a domain-independent framework for managing context awareness in different applications. The framework, built from already available open-source components, is intended to be able to support the development of context-aware applications for different domains (health-care is an important example). In addition, the thesis proposes a new context-aware reasoning approach that uses goal-oriented models (CARGO) as run-time entities and provides more flexibility and configurability than the commonly used rule-based context reasoning.

Table of Contents

1 Introduction 12
1.1 Motivations and objectives 12
1.2 Proposed approach 14
1.3 Thesis contributions 16
1.4 Thesis contents 17

2 State of the Art 19
2.1 Context Awareness 19
2.2 Context Model 19
2.2.1 Context Information Ontology 20
2.3 Context-Aware Middleware 21
2.4 RDF Resource Descriptive Framework 22
2.4.1 RDF Graph data model 23
2.5 Web Services 24
2.5.1 REST web service 24
2.6 SOAP vs. REST web services 25
2.7 Notification Services 27
2.7.1 Notification in Mobile Networks 28
2.7.2 Notification in the Internet 29
2.7.3 Mobile Push scenarios 29
2.7.4 Benefits of PUSH over PULL services 30
2.7.5 Available Mobile Push implementations 30
2.8 Business Rule Engine 33
2.8.1 Business Rule Manifesto 34
2.8.2 Available Business Rule Engines 34
2.9 User Requirements Notation (URN) Modeling Language 38
2.9.1 Goal-Oriented Requirements Language (GRL) 39
2.9.2 Use Case Maps 41

3 Domain Independent Context Awareness Framework 44
3.1 Terminology 44
3.2 Context Awareness Framework Requirements 45
3.3 Framework Architecture 46
3.4 Context Awareness Framework Component selection 48
3.4.1 JBOSS Application server 50
3.4.2 MySQL Rational Data Base 51
3.4.3 Hibernate Relational Persistent Middleware 51
3.4.4 DROOLS Business Rule Engine 52
3.4.5 Web Services - RESTEasy JBOSS library 52
3.4.6 OPENFIRE Notification sever 53
3.5 Framework component integration 54
3.5.1 JBOSS AS - MYSQL integration 55
3.5.2 Hibernate and JBOSS integration 58
3.5.3 DROOLS Rule Engine and JBOSS integration 58
3.5.4 RESTEasy web service and JBOSS integration 60

3.5.5 XMPP Server and JBOSS 60
3.5.6 OPENFIRE installation and configuration 61
3.6 Framework use-cases 62

4 Health Care Application case study 63
4.1 Application Requirements 63
4.2 Application Use Cases 64
4.2.1 UC1 - Admission to the Emergency Department 65
4.2.2 UC2 - Locating the patient 66
4.2.3 UC3 - Checking ED patient's status realization 67
4.2.4 UC4 - Transferring the patient from ED to GM 68
4.2.5 UC5 – Lab Analysis result notification based on rule-based context reasoning 70
4.3 Use Cases linked with the framework 72
4.3.1 UC1 Admission to the Emergency 72
4.3.2 UC2 – Locating the patient 73
4.3.3 Checking ED patients status results remotely 74
4.3.4 Transferring the patient from ED to GM 75
4.3.5 Laboratory Analysis result notification 76
4.4 Domain Data Model 77
4.5 Context specification 78
4.6 Available REST APIs 80
4.7 Rules in Emergency Room Case Study 80
4.8 Context Notification 83

5 Context Awareness Framework extension: CARGO 85
5.1 CARGO - Context Aware Reasoning using Goal Orientation 85
5.1.1 CARGO workflow 86
5.1.2 CARGO Architecture 87
5.1.3 CARGO URN profile 90

6 CARGO Food service application for hospital case study 94
6.1 Goal Model 94
6.2 CARGO scenario model 96
6.3 CARGO URN profile – food service example 98
6.4 104

7 Conclusions and future work 104
7.1 Future Work 105

A. Appendix – REST web service APIs 106
Patient APIs 106
Medical Staff APIs 109
Department APIs 110
Sensors APIs 112
Devices APIs 113

References 115

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my Thesis Supervisor, Dr. Petriu. It was only through her patient guidance and flexibility that I was able to complete all the work in this thesis.

I would also like to express gratitude to Dr. G. Mussbacher for all the help and collaboration on defining context awareness reasoning with goal orientation.

A special thanks also goes out to my husband Srdjan Vrbaski who supported me to fulfill my dream, my children Anna and Marko, who were patient, my father, Dr. Zivota Sesic, who was my inspiration, my mother for believed in me and supported me, and all my friends who instilled me with confidence. Their support was instrumental in my journey towards the writing of this thesis.

Finally, I would like to extend my appreciation to Alcatel Lucent and all the other colleagues I've had the honor of working with over the years. Their flexibility and understanding allowed me to pursue my graduate studies while maintaining full-time employment.

ACRONYMS

APN – Apple Push Notification

AS – Application Server

CARGO – Context Awareness on Goal Orientation

C2DM – Google Cloud to Data Management

DBCP – Database Connection Pools

GPRS – General Packet Radio Service

GRL – Goal Oriented Requirement Language

GSM - Global System for Mobile Communications

3GPP – 3rd Generation Partnership Project is collaboration between groups of telecommunications associations, known as Organizational Partners

HTTP – Hypertext Transfer Protocol

IETF – Internet Engennering Task Force

IP – Internet Protocol

Java EE – Java Enterprise Edition

JMX – Java Management Extension

JNDI- Java Naming and Database Interface

LAN – Local Access Network

LTE – Long Term Evolution

MMS - Multimedia Messaging Service

NAT – Network Access Translaotion

OMA - Open Mobile Alliance

OTA – Over The Air

PDA – Personal Digital Assistant

PAP – Push Access Protocol

PPG – Push Proxy Gateway

PERVESIVE system – or ubiquitous system means existing everywhere

PUSH – a way for the user to subscribes for event and be notify once event happen

RDF – Resource Description Framework

RTC – Real Time Collaboration

RDBMS – Relational Database Management System

REST – Resource State transfer

SCTP – Stream Control Transfer Protocol

Smartphone –handheld computer integrated with mobile phones

SIP - Session Initiation Protocol

SMS - Short Message Service

TCP – Transfer Control Protocol

UDP – User Data Protocol

URN – User Requirements Notation

UCM – Use Case Maps

UMTS – Universal Mobile Telecommunication System

USSD- Unstructured Suplemenatry Service Data

UDP – User Datagram Protocol

WAP – Wireless Application Protocol

XMPP – Extensible Messaging and Presence Protocol

List of Figures

Figure 1.1 Context Awareness Framework Architecture 14
Figure 1.2 Context Awareness Framework with deployed multiple Context Awareness Domain Applications 15
Figure 1.3 Extended Context Awareness Framework extended with Goal Modeling 16
Figure 2.1– RDF triplets 23
Figure 2.2 Basic Element of the GRL notation [14] 41
Figure 2.3 Simple Use Case Maps [55] 42
Figure 2.4 Basic Call UCM with two dynamic stubs [55] 43
Figure 3.1 Context Awareness Framework Architecture build from open source components 49
Figure 3.2 Context Awareness Notification component diagram 53
Figure 3.3 Openfire administration web based console 54
Figure 3.4 JBOSS web data source 57
Figure 3.5 JBOSS MYSQL integration workflow 58
Figure 3.6 Jabber XMPP server integrated with JBOSS server stack 60
Figure 3.7 OPENFIRE installation with embedded database 61
Figure 3.8 OPENFIRE installation with MySQL database 61
Figure 3.9 Real time notification process 62
Figure 4.1 Admission to the emergency use case 66
Figure 4.2 Locating the patient use case 67
Figure 4.3 checking patients' statuses use case 68
Figure 4.4 Transfering the patient to GM department use case 70
Figure 4.5 Lab analysis result notification and running the rules to found who should be notified 71
Figure 4.6 UCM for managing the patient location in the emergency department 73
Figure 4.7 UCM locating the patient 74
Figure 4.8 UCM checking ED patient status remotely 75
Figure 4.9 UCM for patient transfer from Emergency to General medicine 76
Figure 4.10 UCM Laboratory result real time notification 77
Figure 4.11 Emergency room data model for the Emergency room case study 78
Figure 4.12 Laboratory Analysis Result Notification workflow an example 80
Figure 4.13 Monitoring Alarm Real time Notification workflow an example 81
Figure 4.14 OPENFIRE Administration console 84
Figure 5.5.1 CARGO reasoning paths 86
Figure 5.5.2 CARGO architecture 87
Figure 5.5.3 Interaction between the CARGO Manager, the rule engine, and the goal engine 89
Figure 6.1 Goal model for food service application 95
Figure 6.2 Scenario model for food service application 96
Figure 6.3 Start of the scenario 96
Figure 6.4 Food Service Application Input dialog 97
Figure 6.5 Output results 97
Figure 6.6 CARGO workflow model 98

List of Tables

Table 2.1 Available Rule Engine....38
Table 3.1 DROOLS libraries....59
Table 4.1 Human Context....79
Table 4.2 Physical Context....79
Table 4.3 Laboratory Result Rule Definition....82
Table A.1 Get Patients API....106
Table A.2 Get Patient API....106
Table A.3 Get All Patients in Department API....106
Table A.4 Get All Active Patients in Department API....107
Table A.5 Create a new Patient API....107
Table A.6 Update Patient API....107
Table A.7 Delete Patient API....107
Table A.8 Get Patient Info API....108
Table A.9 Update Patient Info API....108
Table A.10 Get Patient's Sensors API....108
Table A.11 Add Patient's Sensor API....108
Table A.12 Delete Patient's Sensor API....109
Table A.13 Get All Medical Staff API....109
Table A.14 Get All Active Doctors in a Department....109
Table A.15 Get All Active Nurses in a Department API....109
Table A.16 Get a Medical Staff API....110
Table A.17 Create a Medical Staff API....110
Table A.18 Update the Medical Staff API....110
Table A.19 Delete the Medical Staff API....110
Table A.20 Get All Departments API....111
Table A.21 Get a Department API....111
Table A.22 Create a Department API....111
Table A.23 Update a Department API....111
Table A.24 Delete a Department API....112
Table A.25 Get All Sensors API....112
Table A.26 Get Sensor API....112
Table A.27 Create a Sensor API....112
Table A.28 Update Sensor API....113
Table A.29 Delete a Sensor API....113
Table A.30 Get All Device API....113
Table A.31 Get a Device API....113
Table A.32 Create a Device API....114
Table A.33 Update a Device API....114
Table A.34 Delete a Device API....114

1 Introduction

1.1 Motivations and objectives

In the past few years we witnessed a huge expansion and penetration of wireless appliances, sensor, smart phones, PDAs in a wide range of domains, such as smart homes, hospitals [5] and elderly home care [4]. Such complex systems, known as ***pervasive*** or ***ubiquitous*** systems, share a computing paradigm in which information processing has been integrated into everyday objects and activities. The pervasive concept was introduced for the first time by Weiser (1991) and refers to the seamless integration of devices into the user's everyday life. There are different models of pervasive or ubiquitous computing, which are sharing the idea that small, cheap, smart processing devices are distributed throughout everyday life, and are able to communicate unobtrusively.

A pervasive computing environment is *highly dynamic*, for example the users can move rapidly and the position of many portable devices can be changed [6]. That means that the associations between users and devices are *runtime variables*. In addition, a pervasive computing environment *has to be extensible* because new sensors, devices and users should be able to participate in it seamlessly

Some examples of pervasive systems and their benefits can be found in elderly home care and hospital environments. Healthcare [1] is mission-critical, real-time, and is a complex socio-technical system that requires consideration, collaboration, communication and coordination

between actors in the system (doctors, patients, nurses, lab technicians, administration, etc.)

A subfield of pervasive computing is ***Context Aware systems*** [3]. Context aware systems *are highly adaptable* systems, where the system operations adapt during the runtime to the current context without explicit user intervention. The system takes environmental (outside) context information into consideration in order to increase the usability and effectiveness of the system at runtime. A good example of extremely dynamic environments is a mobile system, where the participants are highly mobile and execution context frequently changes.

The research objective of this thesis is to design a Domain-Independent Framework for Context Awareness that includes rule-based context reasoning. The proposed framework will serve as a basis for the development of context-aware applications in different domains. The proposed framework architecture is implemented with open source components. We used examples of emergency room use cases to build context-aware applications for the emergency-room domain and to validate the proposed framework architecture.

In addition, the research has been taken a step further and improved the context reasoning, which is one of the crucial components of a context aware system, by integrating the rule-based context reasoning with goal oriented requirements model. The extended context reasoning is named CARGO (Context-Aware Reasoning with Goal-Orientation). The proposed extended context awareness reasoning solution is illustrated with a hospital food services system.

1.2 Proposed approach

This thesis proposes a framework for context awareness based on context rule reasoning for supporting context-aware applications for different domains, which are built and deployed on top of the framework. The primary focus was to build the framework by reusing existing available open-source components.

Figure 1.1 presents domain independent context awareness framework architecture that consists of: (1) a relational database, (2) an application server for running context-aware applications, which also integrates other middleware functionalities, (3) persistence middleware, (4) a web service support (5) a rule based engine, and (6) notification component.

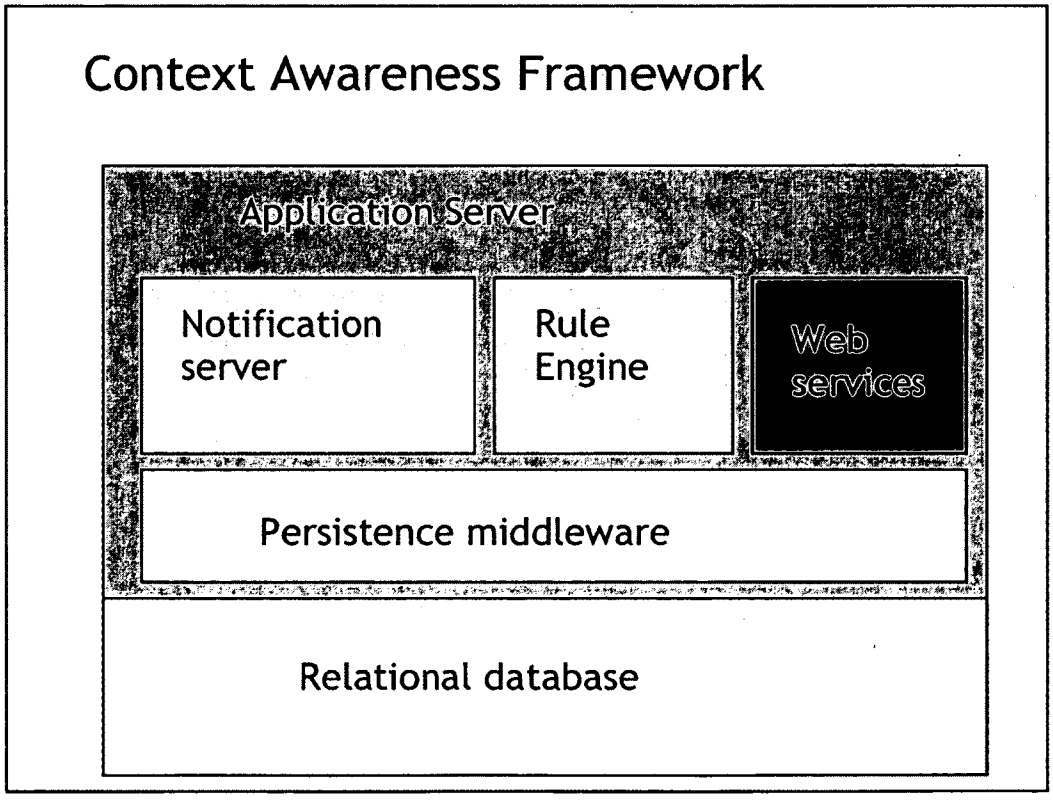

Figure 1.1 Context Awareness Framework Architecture

Figure 1.2 shows the context awareness framework that has deployed multiple context-aware domain applications. Each deployed application contains domain specific capabilities, but they all share common context aware capabilities. In chapter 3 it is explained how the generic parts of the framework will be configured for a specific application.

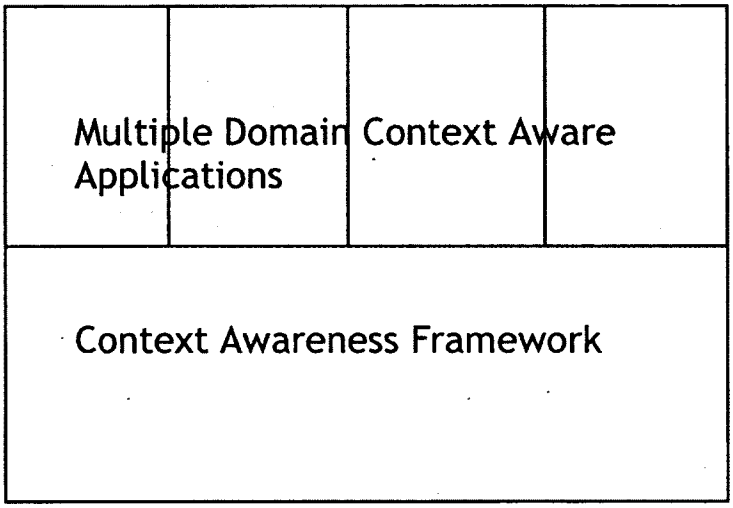

Figure 1.2 Context Awareness Framework with deployed multiple Context Awareness Domain Applications

Figure 1.3 shows the developed Extended Context Awareness Framework that has been extended with goal modeling, which is discussed in detail in Chapter 4.

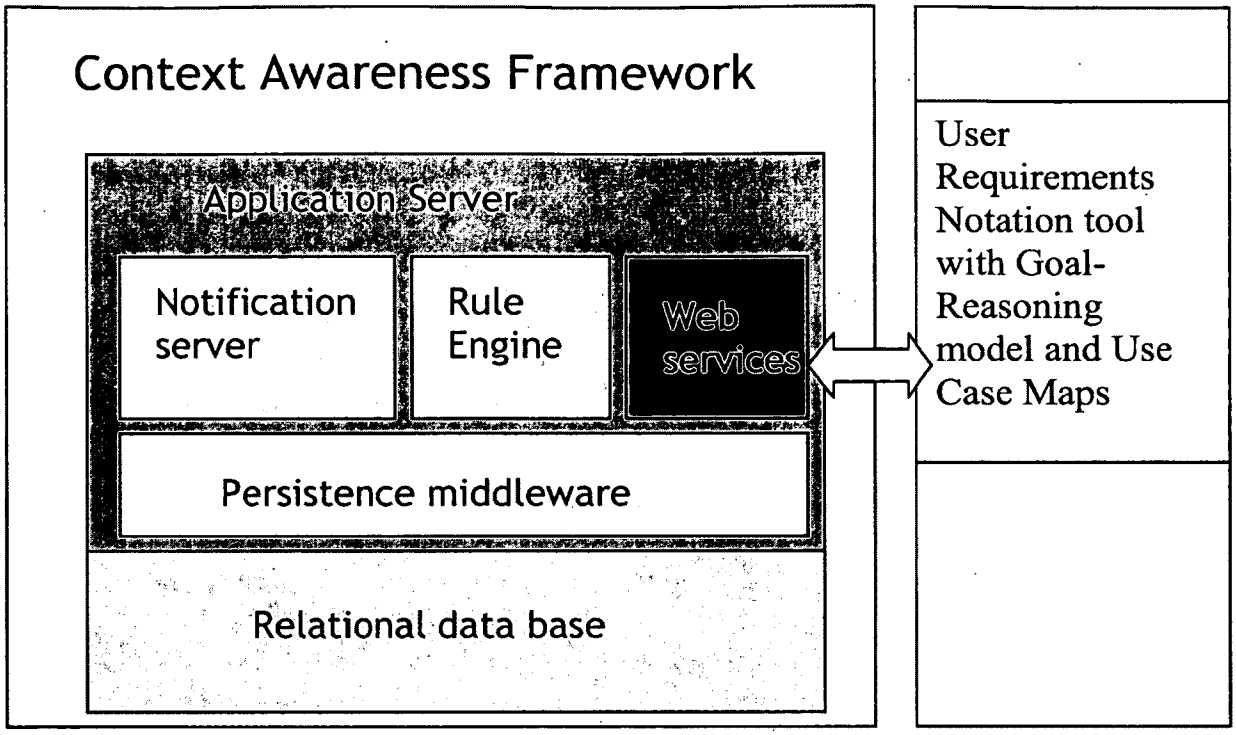

Figure 1.3 Extended Context Awareness Framework extended with Goal Modeling

1.3 Thesis contributions

Contribution to knowledge:

- Design of a domain-independent framework for context awareness using open-source components that provides the following features: context reasoning, storage, acquisition and notification to other systems or clients. The framework is designed based on service oriented concepts and can be used for developing context aware application for different domains.
- Extension of the reasoning capability of the framework by combining rule-based and goal-oriented reasoning (the latter complementing the former), providing more flexibility and configurability than the commonly used rule-based approach, and allowing for a more holistic assessment of the context as the goals of many stakeholders are taken into account.

Practical contributions:

- Implementation of the Context Awareness Framework by using the following open source components: RESTEasy web service library, Drools rule Engine, JBOSS application server, Hibernate Middleware, Smack Xmpp library, OpenFire Xmpp Server, MySQL server (database).
- Integration of the Context Awareness Framework with the User Requirement Notation tool JUCMNav for goal modeling. This integration required extensions of JUCMNav with the capability to invoke REST APIs.
- Application of the framework to the development of context-aware applications in the health care domain.

The outcome of the thesis was published in the following papers:

- Vrbaski, M., Petriu, D., and Amyot, A. (2012) Tool Support for Combined Rule-Based and Goal Based Reasoning in Context-Aware Systems. *20th IEEE Intl. Rqmnts. Eng. Conf.* (Chicago, USA, Sep. 2012). RE'12. IEEE CS. (***awarded for the best poster and demo***).
- Vrbaski, M., Mussbacher, G., Petriu, D., and Amyot, A. (2012) Goal model as Run Time Entities in Context-Aware Systems. *15th ACE/IEEEE Intl. MODELS* (Innsbruck, Austria, September 30).MRT'12 IEEE CS.

1.4 Thesis contents

The thesis consists of seven chapters.

Chapter 2 "State of the Art" reviews the literature on context awareness, context models and

context-aware middleware. Chapter 3, "Domain Independent Context Awareness Framework", covers the requirements, design and implementation of the framework, while Chapter 4 "Health Care Application case study", applies the framework, showing how it can be used to build examples of emergency room context-aware applications. Chapter 5, "Context Awareness Framework extension: CARGO", presents the extension of rule-based reasoning with goal modeling. Chapter 6, "CARGO Food service application for hospital case study", presents the use of CARGO for a hospital food services case study. Chapter 7, "Conclusions and future work" presents the conclusions and future research directions.

2 State of the Art

2.1 Context Awareness

Context-aware systems are very complex systems consisting of various seamlessly connected distributed components such as sensors, actuators, context information stores, context information processors, etc [5] that acquire, interpret, process, transfer or use the context [6]. A context-aware system takes into consideration at runtime external (outside) context information making effective decisions without direct human intervention.

Context is a very broad concept. Most definitions agree that the context represents *situational information* of ***entities*** such as a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [6][8][3]. The word context is derived from the Latin words *con* (with or together) and *texere* (to weave), which leads to the idea that context is *an active process dealing the ways humans do to weave their experience with their whole environment to give it meaning* [7]. For example, in the smart home context-aware system temperature and light in the room can be a part of the context information that will be measured outside of system with sensors. Based on their values, the system will take at the runtime different actions according to previously defined logic rules, without direct human intervention at runtime.

2.2 Context Model

An important aspect of a context-aware system is the context model. Very often in the

literature authors try to model the context by defining an ontology for a particular domain [6]. Many disciplines now develop standardized ontologies that domain experts can use to share and annotate information in their fields. For example, the field of medicine has produces large, standardized, structured vocabularies such as SNOMED and semantic network of the Unified Medical Language System [10]. An ontology for the telecommunication domain is defined in the 3gpp specification [49]

To summarize, **ontology** defines a common vocabulary for researchers and domain field engineers who need to share information in a domain and between multiple integrated systems. Ontology includes machine-interpretable definitions of basic concepts in the domain and relations among them [10].

2.2.1 Context Information Ontology

Context information consists of a triplet (Entity, Context Type, Context Value) [6], where Entity represents an element of the context, such as a person, device, task etc.; Context Type describes an attribute of the Entity, such as location of the entity, current activity status etc.; and Context Value includes the real data value of the context type. For example, if we have an entity patient, his/her "health status" presents the context type and context value can be for example "good".

When we look closely into the context entities, three types are obvious: a) *place or location of the entity* (rooms, buildings, cities etc.), b) *people* or users (individuals and groups) and c) *things* (sensors, physical objects, computer devices and etc.) [3].

Each of these three entities can be described with various attributes that can be classified into four types: i) *identity* (each entity has a unique identifier), ii) *location* (the entity's position, sub-position, closeness to, etc.), iii) *status* (for example the light is on or off, the air pressure is low or high etc) and iv) *time* (timestamps of the event) [3].

2.3 Context-Aware Middleware

This chapter will discusses briefly the Context Awareness Middleware proposed by E. Kim and J. Choi [6], because some of its elements will be used in this thesis. The Context Awareness Middleware is realized as a set of functions (which are called services in [6]): is : (1) Context provider service, (2) Context aggregation service, (3) Context observation service, (4) Content ontology reasoning service, and (5) Context discovery service.

The purpose of **Context provider** service is to *collect raw context* information from various sources such as sensors and to *translate* it to low lever context information based on the context model. The service is registered with the Context discovery service.

The **Context aggregation** service receives a request for context information from a context-aware application. The context aggregation service asks the context discovery service what context provider service is able to offer the requested context information. After invoking the appropriate services and receiving all the necessary context information from different context provider services, the context aggregation service aggregates them into one aggregated response to the context aware application.

The **Context observation** service is a means for the application server to register to be notified when context values of interest are changing. The name context observation service was introduced by Kim et al. [5]. From now on, the thesis use the term "context notification service" instead of" context observation" since it describes more precisely the purpose of the service. The Context notification service works on publish-subscribe mechanisms.

Context ontology service consists of two modules: Context ontology reasoning module and Rule-based context reasoning module. Context ontology reasoning is based on context relationships, where rule based context reasoning is based on user predefined rules.

Context discovery service maintains a mapping between Context provider service and context information. It provides two interfaces: *register interface for Context provider services* and context *discovery interface* for the Context aggregation service and Context observer service.

2.4 RDF Resource Descriptive Framework

This section will present briefly the Resource Descriptive Framework RDF [24] that is a general-purpose language for representing information in the Web. The RDF has an abstract syntax, which corresponds to a simple graph-based data model, and well-defined formal semantics which allows for well-founded deductions on RDF data.

The RDF has been developed for the following uses:

- Web metadata: providing information about Web resources and the systems that use

them.

- Applications that require open rather than constrained information models (for example, scheduling activities, describing organizational processes, annotation of WEB resources etc).
- Allow data to be processed outside the particular environment in which it was created, in a way that can work at Internet scale.
- Interworking between applications: combining data from several applications to arrive at new information.
- Automated processing of Web information by software agents: the Web is moving from having just human-readable information to being a worldwide network of cooperating processes. RDF provides a worldwide language for that process.

2.4.1 RDF Graph data model

The RDF data are organized as ***collection of triplets***. Each triplet consists of a subject, a predicate (also called a property), and an object. A set of such triplets is called an ***RFD graph*** (see the Figure 2.1).

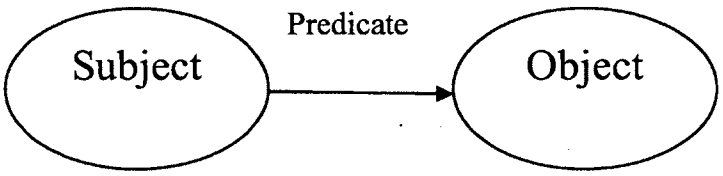

Figure 2.1– RDF triplets

An RDF graph consists of two nodes, subject and object, and one link, the predicate. The

direction of the arc always points toward the object. Each triple represents a statement of a relationship between the things denoted by the nodes that it links. More specifically, the predicate or property denotes a relationship between the subject and the object.

As already mentioned in the Chapter 2.2.1 Context Information Ontology, Context information consists of a triplet (Entity, Context Type, Context Value) as well as RDF graph data model, which indicates that RDF graph data model can be used for presenting context information ontology.

2.5 Web Services

Service-oriented architectures (SOA) is a distributed computing paradigm where large complex applications are composed of independent software components running on heterogeneous platforms that offer services to one another through well-defined interfaces. There are different supporting technologies (also known as service platforms) that support the implementation of service-based systems, each providing different features for publishing, discovering and invoking services. Two well known service platforms are web services based on standards such as SOAP, XML and WSDL (Earl, 2005) and Representational State Transfer (REST) services using HTTP verbs such as GET, POST, PUT, etc. [11].

In this thesis we decided to use REST web services and in the following section we will provide background on REST services and a comparison with SOAP-based Web Services.

2.5.1 REST web service

REST was introduced for the first time in 2000 by Roy Thomas Fielding in his PhD

dissertation [11]. REST is an abbreviation of Representation State Transfer, and represents a basic architectural pattern used in the web. Fielding observed in his dissertation how web pages are organized and how they are linked to each other. REST is modeled around a large number of "Resources" which are "linked" to each other [12].

In order to understand REST web services it is important to understand the following concepts: resource, state and representation. Resource can be anything. A resource can be a physical object or an abstract object. A representation is any useful information about the state of the resource. A resource may have multiple different representations. REST defines two types of state: resource state, which is information about the resource and application state, which is information about the path that client has taken through the application [1].

The reason why the REST web service was chosen as an architecture pattern was that a resource in the REST web service terminology and a context in the Context Awareness terminology present the same thing in two different theoretical domains.

2.6 SOAP vs. REST web services

SOA web services are based on the Remote Call Procedure RPC concept that realizes cross network object messaging in object oriented applications. In the RPC paradigm, the server provides a set of procedures and the client calls these procedures to fulfill a task. Procedures are verb-like, which makes traditional SOA web service focused on verbs (i.e., actions).

On the other hand, REST web services emphasize Resource and Linkage between the resources. Resources are noun-like, and this makes REST web services focused on nouns.

The REST web service architecture assumes that each resource will have a globally

identifiable Uniform Resource Identifier URI. A resource is represented by some Representation format, which is typically extracted by an idempotent HTTP GET (the representation may embed other URI which refers to other resources) this emulates an HTML link between web pages and provide a powerful way for the client to discover other services by traversing its links, and makes building Service Oriented Architecture SOA search engine possible.

The REST web service defines a small number of Actions based on five existing HTTP methods: GET, PUT, POST, DELETE and HEAD. For reach resource we can define maximum five procedures that correspond to existing five HTTP methods.

The HTTP GET method is used to get a representation of the resource. The PUT HTTP method is used to create a new representation of the resource, where the new representation of the resource is embedded inside the HTTP Body. The HTTP DELETE method is used to delete an existing representation of the resource. The HTTP HEAD method is used to get metadata of a resource. The HTTP POST method is used to update existing resource, where a new representation of existing resource will be embedded into the HTTP Body.

Research done by Feng et al. [1] where they compared traditional SOA web service based on the RPC and REST web service on scalability, coupling, security and performance show that REST web services are better in scalability, coupling and performance. Regarding security, the REST web services security model is simpler and more effective; the difficulties of implementing security model are reduced in comparison with SAO web service security model.

2.7 Notification Services

An important component of context aware systems is the *notification service* which provides runtime notifications when a change of state happens in the system to other collaborating systems, known as *system-to-system notification*, or to the end user, known as *system-to-client notification.*

System to system notification is not a new concept and has been researched and used a lot in practice. If both the notification sender and receiver) are static, this makes their integration easier. A bigger challenge is presented by a system-to-client notification where the end client is mobile. (For example, the end user is an emergency room doctor that has to be notified immediately when the patient results are completed in the laboratory; this can be realized by sending a notification message to the doctor's smartphone). From here on, this section will be focused on the notification for mobile users.

In the last couple of years we witnessed changes regarding the mobile phone usage and functionalities. Mobile phones **used to be only** electronic devices that allowed its user to make and receive telephone calls from the public telephone network, which included other mobiles and fixed phones across the world. Modern mobile phones also known as **Smartphones** (such as iOS iPhone, Android, Blackberry, etc.) offer advanced computing ability and connectivity; they are handheld computers integrated with mobile telephones that allow the user to install and run advanced applications that are very often connected to the

Internet.

This dual functionality of smartphones being at the same time a mobile phone that uses mobile network and handheld computer integrated with the Internet creates a need for an ecosystem, in other words a "marriage" between Mobile Network and Internet that allows mobile devices to receive common web transactions and allows Internet peers to be notified of mobile generated events [18].

2.7.1 Notification in Mobile Networks

Mobile (Teleco) networks are complex networks that are able to offer more customized services to their users. Once the mobile users are attached to the mobile network the users are automatically subscribed to network events and they are able to receive notifications from the network such as phone calls, Short Message Session (SMS) message and etc.[18] .

These ways of communication are known as a **PUSH** or **publish/subscribe** mechanism. The mobile user once attached to the network maintains connection with the connection access point and can be reached through the connection point at any time.

Mobile networks are using connection-oriented protocols, where before exchanging data the sender (mobile phone attached to the network) and receiver (network access point) first establish a connection (attachment) and possibly negotiate a protocol they will use. When they are done, they must release (terminate) the connection [9].

2.7.2 Notification in the Internet

On the other hand, the Internet network has a much simpler architecture than a Mobile network. It is based on the client request and the connection, known as a session that lasts until the client receives the requested information. If the client wants data from the Internet, the client search and receives the data. The concept is known as **PULL** mechanism.

The Internet Protocol (IP) is most widely used connectionless protocol, where no setup in advance is needed [9].

2.7.3 Mobile Push scenarios

There are three representative usage scenarios addressed by the mobile push architecture: stationary users, nomadic users and mobile users [20].

The stationary user presents a user with a fixed IP address. The user IP address is fixed and does not change over time. An example is a user that has an office desktop computer on a LAN (local access network).

The second scenario enables nomadic users to access the service from different networks using most often laptops or other portable computers via dial-up modem lines or wireless.

The third scenario is a mobile user that can frequently change its location in the network. The user accesses the network using a mobile phone and the connection to the network can change in the same session. The host IP address changes as well.

2.7.4 Benefits of PUSH over PULL services

Push services are the preferred way of communication in the mobile networks since they use network bandwidth effectively. The PUSH service communication only occurs when information is available. The push service than notifies subscriber (mobile client) about incoming changes.

On the other hand, in PULL based communication, the user must perform periodical calls to the server for available updates. Very often with the user will call when no real update happened at the server. The pull approach includes PDP context (PDP context specified access to an external packet-switching network) activation, which consumes network resources (air interface bandwidth, buffers, IP-addresses) and terminal resources (transmit power). In addition to that, the smart phone device has to wake up from the idle mode in order to make a call to the server, which is hard on the smart phone battery usage.

2.7.5 Available Mobile Push implementations

There are several available mobile push implementations in mobile networks and Internet. Some of them are as follows:

- **SMS** - Short Message Service
- **MMS** - Multimedia Messaging Service
- **WAP** – Wireless Application Protocol
- Mobile proprietary push
- **SIP** - Session Initiation Protocol

- **XMPP** – Extensible Messaging and Presence Protocol

SMS is an integrated element of GMS, UMTS and LTE networks. It is suitable for push delivery of short text message. The SMS architecture consists of a SMS User Agent that is integrated in a mobile phone and a SMS Center that stores and forwards messages. SMS is a proven system in all GMS/UMTS networks. It is scalable and roaming is available. SMS has a couple of disadvantages such as: interworking with internet is not possible, SMS message are limited to only 160 characters, application addressing is not possible and message delivery is not guaranteed.

Multimedia message service MMS is a way to send multimedia content messages to and from mobile phones. MMS is extended from the core Short Message Session protocol and allows sending and receiving message longer than 160 text characters. The most popular use is to send photographs from camera-equipped handsets, although it is also popular as a method of delivering news and entertainment content including videos, pictures, text pages and ringtones.

The Wireless Application Protocol WAP is a standard for accessing information over the mobile wireless network. The WAP architecture consists of a WAP Browser installed on the mobile phone, and a WAP Push Proxy Gateway (PPG) on the terminal. The PPG accepts push requests from a push initiator in the Internet, using Push Access Protocol (PAP). The PPG converts the request into the push Over The Air protocol (OTA) format and forward the request to the terminal.

Most of the smart phones currently do have some proprietary push mechanism. For example: BlackBerry got famous with e-mail push notification; IPhone recently announced Apple push notification service[27]; Android has Android Cloud to Device Messaging Framework C2DM [28] that is able to push content to an Android user.

A big disadvantage of the proprietary push mechanism is that any kind of server push implementation requires, for each version and a model of the smartphone, a separate code implementation on both client and server side.

Session Initiation Protocol (SIP) is an application layer control protocol developed in the MMUSIC working group of the IETF. SIP was invented to provide rendezvous for session establishment and negotiation on the Internet. SIP was initially developed for Internet voice/video communication and in the last couple of years by adding extensions such as SIMPLE started to support instant messages and presence [29] .

SIP uses the following signalling protocols:

- User Datagram Protocol (UDP) - connection-less protocol. By using UDP protocol the packets can be lost, which is not a huge problem if voice and video streaming is transported. The human brain is capable to make up for the lost sounds and image. This type of transport protocol is inadequate for transporting structured data.
- Transition Control Protocol (TCP) - connection-oriented protocol.
- Stream Content Transmission Protocol (SCTP) – connection-oriented protocol.

The Extensible Messaging and Presence Protocol (XMPP) is an open technology for real-time communication, which powers a wide range of applications including instant messaging,

presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data. The core technology behind XMPP was invented by Jeremie Miller in 1998, refined in the Jabber open-source community in 1999 and 2000, and formalized by the IETF in 2002 and 2003, resulting in the publication of the XMPP RFCs in 2004 [19].

XMPP was invented to do structured data exchange such as synchronous or active presence and text communication among a group of people. XMPP was initially developed for instant messaging communication and in the last couple of years XMPP added support for session negotiation using the Jingle extension [29]. XMPP uses Transition Control Protocol as signalling protocol, on ports 5222 an 5223. The context awareness framework used in this the XMPP for notification functionality.

2.8 Business Rule Engine

Rule-based reasoning was first introduced in the field of Artificial Intelligence. In knowledge-based systems, human knowledge and reasoning has been captured and stored in the form of complex rules. Typically, the rules are described using declarative languages. The rules are stored in a Rule Base and processed in a special component, called the Rule Engine or the Inference Engine. The Rule Engine evaluates the conditions of the rules and at any point in time determines which rules are eligible to be fired (executed) [36].

Business rules represent basic business knowledge in information systems that solve real-life problems. Business people and software developers have a different understanding of what business rules are. For business people business rules are directives that are intended to

influence or guide business behaviour. On the other hand, developers tend to see business rules as atomic pieces of program logic that are typically in "IF THEN" form.

2.8.1 Business Rule Manifesto

The Business Rule Manifesto was defined by the Business Rules Group [36].

Based on the business rule manifesto, business rules are primary requirements and they are essential for business models and technology models. Business rules should be separated from processes and procedures. Rules are basic business knowledge and they must be explicit.

Furthermore, rules are declarative in nature and they should be expressed by using natural language sentences that are easily understood and validated by the business audience.

The platform on which the business rule application runs should support and accommodate continuous change in business rules. Rules define what acceptable business activities are.

In addition, having fewer but well defined rules is better that having a lot of rules.

Knowledgeable business people should define rules and tools that allow the business people to formulate, validate and manage business rules.

2.8.2 Available Business Rule Engines

The Table 2.1 presents 17 different business rule engines. The first three rule engines: MS-BRE (developed by Microsoft), DROOLS (belonging to JBOSS product family), and JESS are well known; they implement RETE rule engine algorithm [53] and provide great performance. The document "Microsoft's Rule Engine Scalability Results – A comparison

with Jess and Drools" [42], provides comparison study for three (MS-BRE, DROOLS and JESS) widely used rule engine.

The document [43] published by Java-source.net provides the list of java open source rule engines.

Name	Description
MS-BRE [Microsoft]	Microsoft Business Rule Engine implements the RETE rule engine algorithm [53], which is the best-known and most widely used matching algorithm in rule engines.
DROOLS [JBOSS/Java]	DROOLS is a java based rule engine is that implements the RETE rule engine algorithm [53] and is a part of the JBOSS product family.
JESS [open source/java]	JESS is small, light weight, java based, rule engine build on the top of the eclipse development platform. Jess uses the RETE rule engine algorithm [53].
OPEN RULES [open source /java]	OpenRules is a full-scale open source Business Rules Management Framework that efficiently uses the power of MS Excel, the Eclipse IDE and open source Java libraries.
MANDARAX [open source /java]	The new version of MANDARAX is a rule compiler that (instead of interpreting rules) compiles rules into standard Java code. It also uses a domain specific language (DSL) to define rules. The DSL supports the smooth integration of rule definitions into Java.
SWEETRULES [open source / java]	SweetRules is a uniquely powerful integrated set of tools for semantic web rules and ontologies, revolving around the

	RuleML (Rule Markup/Modeling Language) emerging standard for semantic web rules, and also supporting the closely related SWRL (Semantic Web Rule Language), along with the OWL standard for semantic web ontologies, which in turn use XML and, optionally, RDF.
TAKE	Take uses the Mandarax Compiler and consists of a scripting language that can be used to define derivation rules, and a compiler that creates executable Java code and deploys it into running systems.
TERMWARE	TermWare is a rule-processing engine intended for embedding into Java applications.
JRuleEngine	JRuleEngine is a Java rule engine, based on Java Specification Request 94, release 1.1, i.e. Rules can be loaded by an XML file or by means of JRuleEngine APIs, so rules can be stored externally into a database too. The distribution consists of a library that can be embedded into a Java application, so it can be used in any kind of application (web based or not).
JLISA	JLisa is a powerful framework for building business rules accessible to Java and it is compatible with JSR94 V, the JavaTM Rule Engine API.
JEOPS –Java Embeddable Object Production System	JEOPS is a Java based forward chaining RULE ENGINE. This Rule Engine is used to power up the business process by rules in Java Application Servers, Client Applications, and Servlets.

PROVA LANGUAGE	Prova (from Prolog+Java) is a rule-based system for Java and agent scripting and information integration extending the Mandarax engine with proper language syntax and enhanced semantics.
OPEN LEXICON	Open Lexicon is a business rules and business process management tool that rapidly develops applications for transaction and process-based applications.
ZILONIS	Zilonis provides a Multithreaded Rules Engine platform and a scripting environment for Java based applications. The core of the Zilonis platform is based on a variation of the forward chaining RETE algorithm.
HAMMURAPI RULE	Hammurapi Rules is a JSR-94 compliant rules engine. With Hammurapi Rules, Java developers don't need to learn a new rules language - rules are written in Java. Hammurapi Rules leverages Java language type system and naming conventions to plug rules into its inference engine.
OPEN TABLET	OpenL Tablets is full-fledged open source Business Rules Management System (LGPL license). Its business friendly approach to rules authoring and management allows it to keep executable business rules as close as humanly possible to the original source that Business Analysts can easily create and

	maintain them.
DTRULES	DTRules is a Decision Table based Rules Engine. Decision Tables are maintained in spreadsheets, and the actions and conditions are specified in a Domain Specific Language. DTRules is a deterministic rules engine (the Decision Tables define exactly the order in which Rules are applied), which makes DTRules very simple and very fast.

Table 2.1 Available Rule Engine

DROOLS JBOSS rule engine has been selected in this thesis work for the next three reasons: DROOLS rule engine is built on the JAVA programing language, belong to the JBOSS product family and has been widely used and tested, and on the end uses RETE rule engine algorithm.

2.9 User Requirements Notation (URN) Modeling Language

The User Requirements Notation (URN) is an international standard [13] intended for elicitation, analysis, specification and validation of requirements. URN is unified language that combines and presents in graphical way requirement goals and scenarios, and shows link between both. In this thesis, we use URN to extend rule-based context reasoning with goal modeling.

URN is a suitable language for specifying and analyzing dynamic systems and business processes. URN has concepts for the specification of behavior, structure, goals, and non-functional requirements, which are all relevant for business process modeling.

Software and Requirements Engineers together with Business Analysts can use URN for specifying software system or processes, as well as to use URN to analyze requirements for correctness and completeness.

The User Requirements Notation standard is combination of two sub-languages: the Goal-oriented Requirements Language (GRL) and Use Case Maps (UCM) [13] that will be presented in more detail in the following sections.

2.9.1 Goal-Oriented Requirements Language (GRL)

The Goal oriented requirements language (GRL) is a connected goal graph that connects an actor and intentional elements with intentional links.

An actor represents system or a stakeholder of the system or its environment. Actors are active entities in the system or in its environment that do have intentional elements: goals, tasks, resources, softgoals and beliefs.

Goals are functional quantifiable requirements; they are conditions or situations that can be achieved or not to be achieved. On the other side softgoals are non-functional requirements, where non-objective satisfactory measures are defined. Tasks represent solutions for goal or softgoals and can require recourses for execution.

Links are connectors between isolated elements in the requirement model, and they are decompositions, contributions and dependencies. Decomposition shows sub-components of a task. A contribution shows how one element influents other element. A dependency shows

interdependencies between agents.

GRL modeling tries to present "why" certain choices for behaviors or/and structures are made or constraint are introduced. By focusing on "why" rather than on "what", which is common in modeling, the modeler can omit operational details or component interactions and focus more on higher levels of modeling that is very often called "strategic stance".

GRL supports strategies. Strategies are user-defined sets of intentional evaluations on a GRL graph [11]. These evaluations have satisfaction levels assigned to some intentional element (mostly leaves of the graph) in the model. Those values are (during evaluation) propagated from leaves to the top-level intentional elements through the various links. Evaluation shows how well goals in a model are achieved in a given context.

GRL supports reasoning about goals and requirements. GRL establishes correspondence between intentional elements and functional components and responsibilities in scenario.

Figure 2.2 shows the Basic Elements of GRL Notation [14].

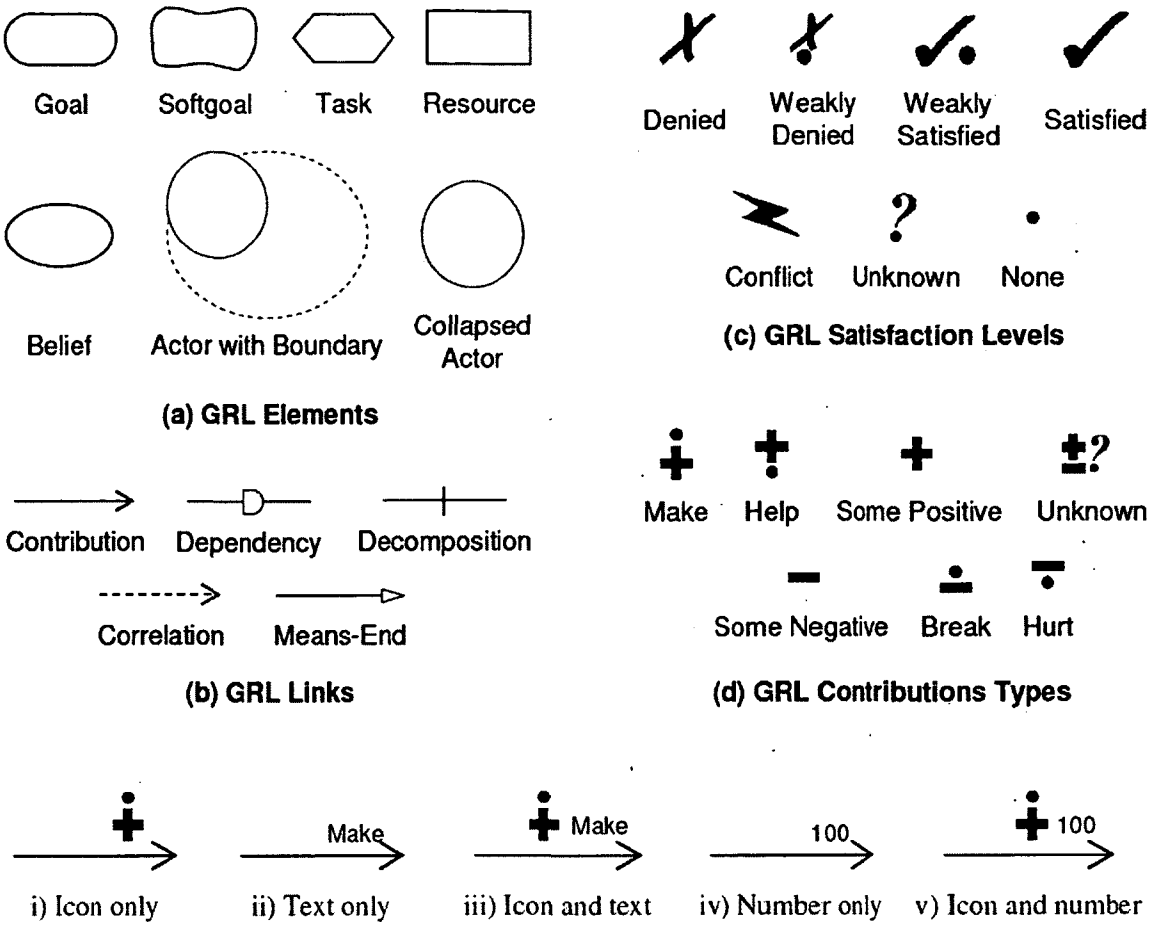

Figure 2.2 Basic Element of the GRL notation [14]

2.9.2 Use Case Maps

The Use Case Maps notation was developed at Carleton University by Professor Buhr and his team. Since 1992 the notation and has been used for the description and the understanding of a wide range of complex applications.

Use Case Maps (UCM) is subset of URN language that addresses language requirements. UCM aims to link behavior and structure explicitly and visually. The basic elements of UCM

are: map, path, start and end point, responsibility, OR-fork, OR-join, AND-fork, AND-join, loops, waiting place and times.

A map is consisted of many path and components. Each path starts with start point and ends with end point. Responsibility describes required actions or steps to fulfill the scenario. Waiting places and timers denote locations on the path where the scenario stops until a condition is satisfied. If an endpoint is connected to a waiting place or a timer, the stopped scenario continues when this end point is reached (synchronous interaction).

Figure 2.3 shows Simple Use Case Maps and basic Use Case Maps components.

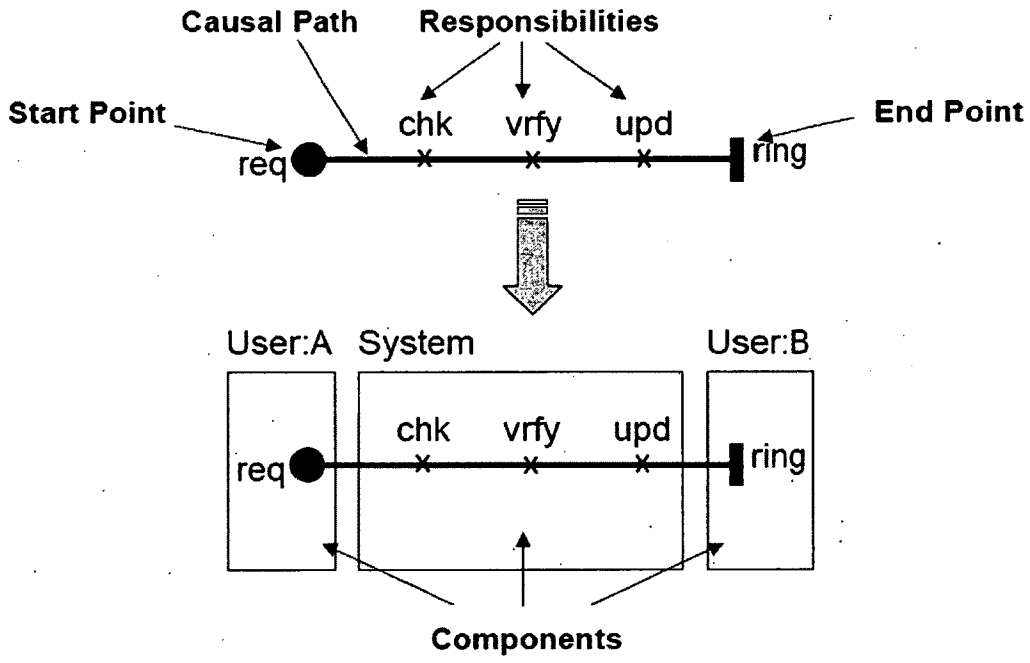

Figure 2.3 Simple Use Case Maps **[55]**

Figure 2.4 shows Basic Call UCM with two dynamic stubs.

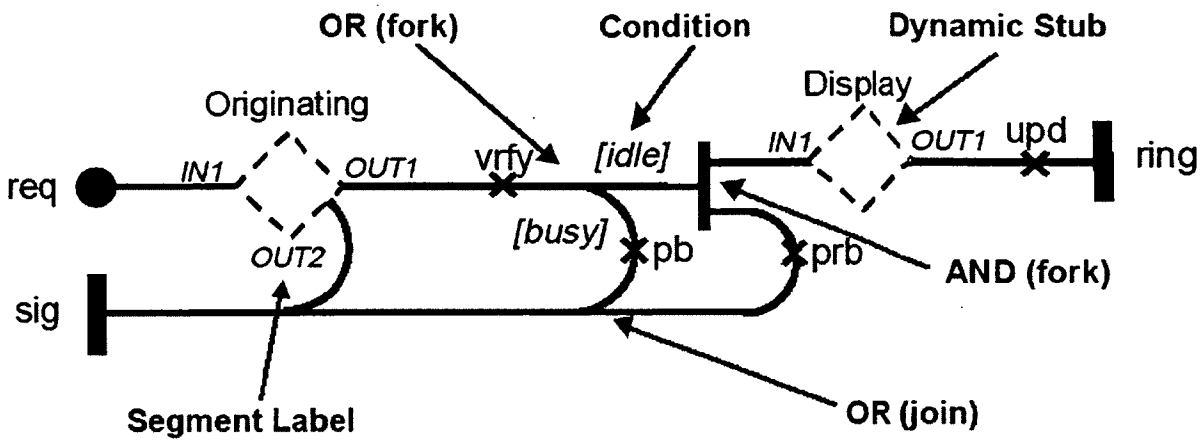

Figure 2.4 Basic Call UCM with two dynamic stubs [55]

3 Domain Independent Context Awareness Framework

This chapter presents the proposed architecture for a Domain Independent Context Awareness Framework, which is built out of existing reusable open source components. One of the main challenges was to select the appropriate open source components and to integrate them together in order to build a domain independent framework for context awareness that can be configured to be used for different applications in multiple domains. The chapter starts with a brief explanation of the terms used in the thesis. After that the requirements for the context awareness framework are presented, the required components for the framework are identified, and the general use cases that such of framework must execute are described. In the end the list of selected open source components that are used for building the proposed framework and explain their integration points is presented.

The next chapter illustrates how an application can be built on top of the proposed framework, by using healthcare application examples. More specifically, it is shown how different components of the generic framework can be configured and specialized to support domain-specific applications.

3.1 Terminology

A Server is a computer program that provides services to other computer programs. The server and the clients can be deployed on the same or on different computers. In the client/server-programming model, a server is a running program that waits for and responds to client requests. A *client* is also a program that sends service requests to the server and waits for the

replies.

A Service represents encapsulated application business logic or individual functionality presented and exposed to external clients through well-defined interfaces.

System to system notification is notification between two physical servers, where both servers have fixed IP addresses.

System to client notification is notification from a physical server with fixed IP address to a mobile device that does not have a fixed IP address. This type of notification is also known as *push notification.*

Application software (or shortly *application*) is a program that provides specific tasks for the end user. For example, a health care application provides specific tasks for the health care users.

A Web application is a software application that can be accessed over the Internet Network.

3.2 Context Awareness Framework Requirements

The six requirements for the domain-independent context awareness framework have been identified. Intension was to build the framework out of open source components, which should be able to support the development of different context-aware applications:

1. The domain-independent context awareness framework shall provide capabilities that facilitate the development of context aware applications in different domains. In other words, the framework is domain independent.

2. The domain-independent context aware framework shall provide support for data persistency on behalf of the applications to be built on top of it.

3. The domain-independent context awareness framework shall provide support for defining and invoking web services in order to allow for easy system-to-system and system-to-client integration based on the service-oriented paradigm.

4. The domain-independent context awareness framework shall provide Context Reasoning capabilities that can be incorporated with the help of a Business Rule Engine.

5. The domain-independent context awareness framework shall provide real time notifications to the end mobile users, as well as system-to-system notifications.

6. Software reuse is important, so the domain-independent context awareness framework shall be built with generally available open source components.

3.3 Framework Architecture

By analyzing the requirements, the following six types of components have been concluded as required (reasons are discussed below): (1) a relational database, (2) persistence middleware, (3) a web application server, (4) web services, (5) context reasoning with rule engine, and (6) notification server. Figure 1.1 shows the proposed framework architecture.

The framework is generic, yet has to offer support for application data persistency, where the data is application and/or domain specific. A **relational database**, which is one of the most widely used types of databases for structured data, has been selected. In a relational database,

the data is organized into tables and can be easily created, accessed and/or reassembled without the need to reorganize the database tables. A relational database can be easily extended: that means that after the original database creation, a new domain and new tables can be added by adding an new set of tables to the existing database schema. This fact is especially important in this thesis concept where a domain-independent context awareness framework has been built to support multiple applications in different domains. Each application will have a set of tables that can be added to the relational database without modifying the existing database schema of other applications.

A persistence or database middleware connects and translates data between two systems: application on the one side and database on the other. The middleware allows direct access to data and provides direct interaction with the databases.

An application server is a server that runs application software, providing framework services such as notification, data transactions, web services and etc.

A business rule engine is a software component that executes preconfigured business rules at runtime. The business rules system enables separation of the business rule policies and application code. That means by changing the policy rules, their definition, testing and execution does not require application code change. Some example of business rules are "if patient has a family doctor, notify the doctor by e-mail", or "remove meals that the patient is allergic to".

A notification server is a server that provides notifications to registered clients, such as smart phones, PDA, Tablets and etc.

3.4 Context Awareness Framework Component selection

The context awareness framework is implemented using the platform independent Java language. Java is a widely used general-purpose, object-oriented language that is designed to have a number of implementation dependencies as low as possible. In addition to that the context awareness framework is build of open source java based components. The **Figure 3.1** presents Context Awareness Framework selection.

There are many **relational database** that were available for this project, some of them open source and some vendor specific. Here is the list of well-known widely used relational databases: Oracle (developed and own by Oracle Inc), DB2 (IBM)) , SQLServer (Microsoft), TimesTen (Oracle) in memory database, MySQL, SQLLite and many more. For this project decision is made to use the free MySQL database. The database can be easily replaced in at a later time if it is needed. More information about the **MySQL** database can be found further in this document.

For the **database middleware** selection is made to use **Hibernate.** More information on hibernate can be found in further in this chapter. Some other possible Java-based persistence middleware options are: jdbc , jdbc-odbc connection, and EJB 3 database persistence.

Some of the available Java-based **application servers** are Geronimo, Tomcat, IBM WebSphere, Glassfish (open source), Bea Weblogic application server, and JBoss. For this

project we decided to use JBOSS. More information on the jboss can be founded further in the chapter.

For system to system and system to mobile client integration decision is made to use **REST web services**, which have a number of advantages compared with SOAP web services: REST services are gaining popularity, are allowing for easier development, and they are preferred especially for smart phone applications. There are a few possible Java REST web service libraries such as Apache Wink, Jersey, Jena, and RESTEasy. For this project RESTEasy is used, because it is well integrated with JBOSS. More information on the RESTEasy can be found further in the chapter.

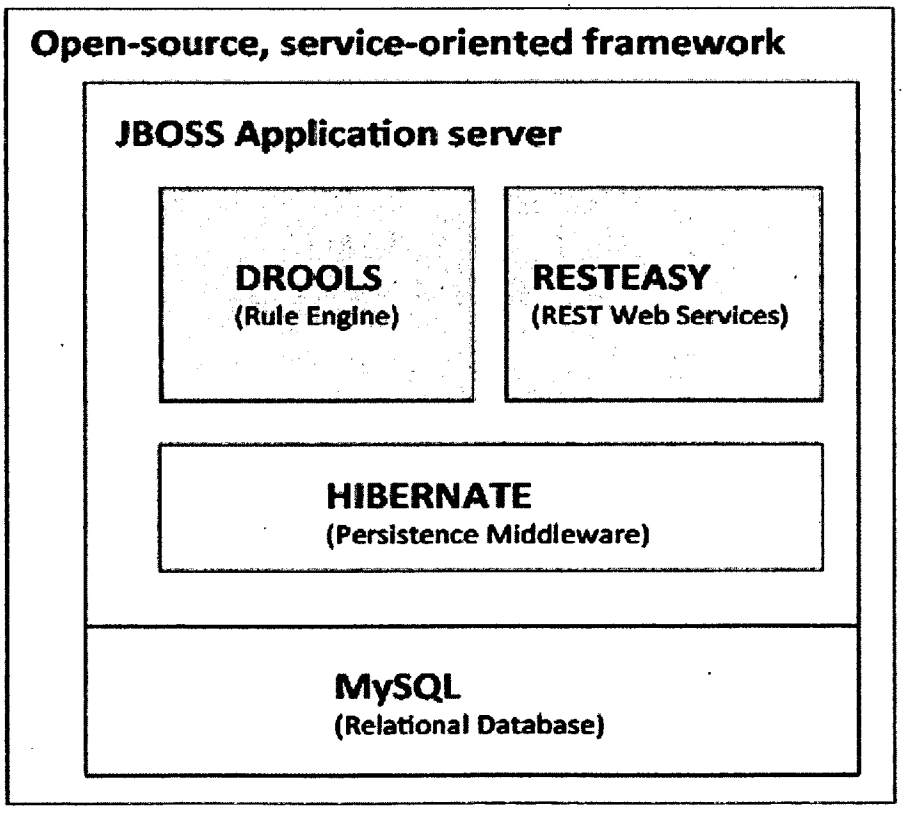

Figure 3.1 Context Awareness Framework Architecture built from open source components

There are a lot of Java-based **business rule engines** on the market, such as DROOLS, BRMS (Business Rule Management System by IBM), and JESS. For this project DROOLS is used, because it is well integrated with JBOSS AS.

There are plenty valuable **notification solutions** some of them are: Google C2DM, APNS (Apple Notification server), XMPP server (OPENFIRE), SIP Server , and Urban Airship. For this project OPENFIRE XMPP server is used. More information on the Open Fire server can be found further in the document.

3.4.1 JBOSS Application server

The JBOSS Application Server is the widely used Java application server on the market [37] that is developed by hundreds of professional open source developers, member of JBOSS wide community, over the years.

The JBOSS application server is a Java EE certified platform for developing and deploying enterprise Java applications, Web applications, and Portals. JBOSS Application Server AS 6, which is selected version of the JBOSS in this project, provides the full range of Java EE 6 features as well as extended enterprise services including clustering, caching, and persistence.

In addition to the JBOSS application characteristics mentioned above, the next an additional great benefit have been found: the JBOSS AS has been integrated with a large line of the JBOSS community products. Additional products chosen for this project from the JBOSS community are: DROOLS rule engine, RESTeasy REST web service library and framework,

and Hibernate middleware.

One of important and very useful JBOSS features is JBOSS JMX console that provides easy management of the JBOSS server. After successful installation and start of the JBOSS AS, JMX console can be accessed at http://localhost:8080/jmx-console in the default JBOSS distribution.

JMX console provides a raw view into the microkennel of the application server and a list of all registered services (MBeans) that are active in the application server. The services can be accessed programmatically or through the JMX console. In addition to the list of all registered services, the JMX console also displays the JNDI tree, generates a thread dump, displays the memory pool usage, and manages the deployment scanner, redeploys an application and allows shout down of the JBOSS.

3.4.2 MySQL Rational Data Base

The thesis relational database choice is the MySQL database [38]. The MySQL database is a highly popular open source database because of its high performance, high reliability and ease of use. It is also the database of choice for a new generation of applications built on the LAMP stack (Linux, Apache, MySQL, PHP / Perl / Python.) MySQL runs on more than 20 platforms including Linux, Windows, Mac OS, Solaris, HP-UX, IBM AIX.

3.4.3 Hibernate Relational Persistent Middleware

Hibernate, which is one component of JBOSS community products, is a high-performance

Object/Relational persistence and query service [40]. Hibernate is known as very flexible and powerful Object/Relational solution that takes care of the mapping from Java classes to database tables and from Java data types to SQL data types. It provides data query and retrieval facilities that significantly reduce development time. Hibernate is designed to relieve the developer from 95% of common data persistence-related programming tasks by eliminating the need for manual, hand-crafted data processing using SQL and JDBC.

3.4.4 DROOLS Business Rule Engine

The DROOLS Rule Engine is one more product from the JBOSS community palette of products. It is a business rule management system (BRMS) with a forward chaining rule based engine, more exactly known as a production rule system, using and enhanced implementation of the RETE algorithm.

The RETE algorithm, developed by Charles L. Forgy in 1982, is an efficient method for comparing a large collection of patterns with a large collection of objects. The algorithm finds all objects that match each pattern. A detailed algorithm description can be found in Forgy's paper [53].

DROOLS Rule Engine supports the JSR-94 standard for its business rule engine and enterprise framework for the construction, maintenance, and enforcement of business polices in the organization, application or service [41].

3.4.5 Web Services - RESTEasy JBOSS library

RESTEasy is also a product from the JBOSS community palette of products and allows their

users to build RESTful Web Services and RESTful Java applications. It is a fully certified and portable implementation of the JAX-RS specification, which is a new JCP specification that provides a Java API for RESTful Web Services over the HTTP protocol [44].

3.4.6 OPENFIRE Notification sever

OPENFIRE [34] is a real time collaboration (RTC) server licensed under the Open Source Apache License. It uses the widely adopted open protocol for instant messaging, XMPP (also called Jabber) [35]. OPENFIRE server is easy to set up and administer. In addition to that, OPENFIRE server is a good security and performance instant messaging system.

OPENFIRE server can be configured to use an external database or can use a built-in database, which is done in this work.

OPENFIRE server comes with the nice an administration console that can be used for configuring the server, administering the users and monitoring of activities. The user can access the administration page over HTTP on port 9090 or HTTPS on port 9091.

Figure 3.3 shows the Context Awareness Notification component diagram, while Figure 3.4 shows the OPENFIRE administration console.

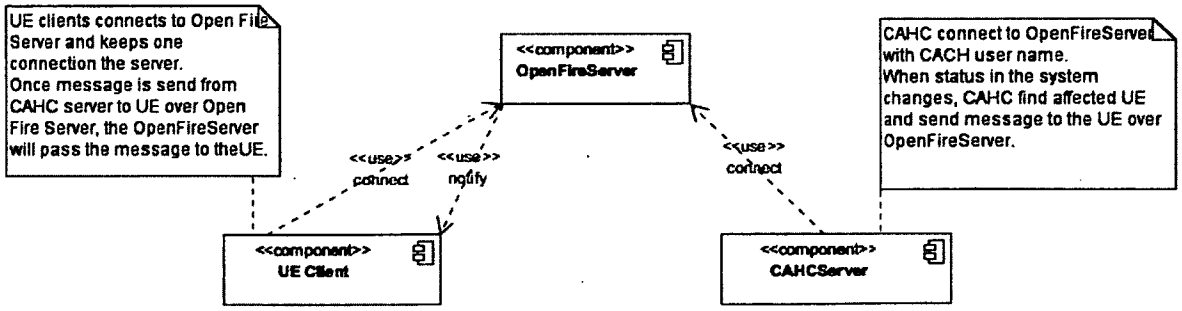

Figure 3.2 Context Awareness Notification component diagram

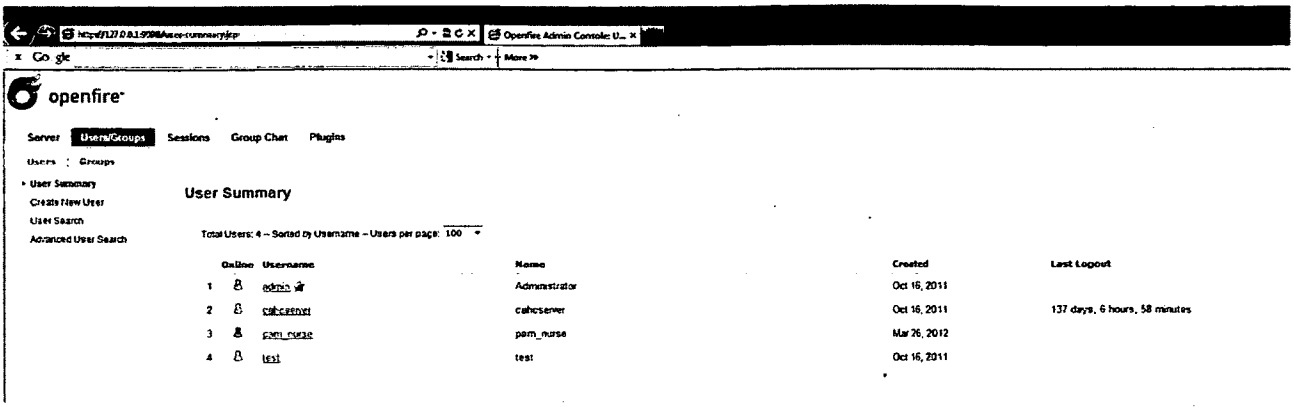

Figure 3.3 Openfire administration web based console

3.5 Framework component integration

After selecting the open source components for building the context awareness framework it is important to properly integrate them to work with each other. This section will cover component integration issues.

The framework consists of three servers: a MySQL database server, a JBOSS application server and OPENFIRE notification server.

An installation prerequisite of the Context Awareness Framework is that all three servers have to be installed in advance before proceeding with the part integration.

This section will cover the next integration issues:

- JBOSS AS – MySQL integration
- Hibernate middleware integration with JBOSS AS and MySQL
- RESTEasy integration with JBOSS AS
- OpenFire – JBOSS AS integration

- DROOLS Rule Engine integration with JBOSS AS

3.5.1 JBOSS AS - MYSQL integration

The JBOSS Application Server and MySQL Server integration requires that both servers are previously installed. The MySQL server has to be up and running, and a database has to be created on the MySQL server. For example, CAHC database that has been used for Health Care Application covered in the Chapter 4.

Application deployed on the JBOSS AS server can connect to the MySQL database in two ways:

- **Direct data base connection** - the application is managing their own database connection or
- **Shared database connection pool** -the database connection pool is managed by the JBOSS AS.

The second is the recommended approach. Because connections are pooled and shared among applications, configuration and maintenance is simplified and connection with the database is faster. In addition, shared connections produce better utilization of the connections. Applications are portable: they do not depend on internal integration of the external environment.

Furthermore a connection directly managed by the application requires more testing, adds unnecessary lines of code that will manage the database connections, and complicates deployment since it requires a separate configuration for each application.

Based on the previously stated shared data base connection pool over the direct data base connection the health care context awareness application developed in this work will use shared database connection pool.

In order to create shared connection pools in the JBOSS server the next steps have to be performed [45]:

- Installation of a database driver (in this case JDBC driver for MySQL database is used).
- Defining a MySQL database connection pool (RDBMS DBCP)
- Map JBoss –managed RDBMS DBCP to the application's resource reference

3.5.1.1 Installation of a JDBC driver

Each database vendor provides a set of libraries for the most commonly used development languages called database drivers. Database drivers are used by applications to create and manage database connections.

JDBC stands for Java database connectivity and is developed by using the generic development language Java.

3.5.1.2 Defining RDBMS DBCP Resource

JBOSS AS comes with RDBMS DBCP resource templates for the most commonly used relational databases.

Some of the important datasource elements that should be configured are:

- Username – the user name required for database authentication
- password – the password required for database authentication.

- min_pool_size –the minimum number of pooled database connection. The value will be initialized when the pool is first accessed. Default value is 0.
- max_pool_size – the maximum number of pooled database connections. Once the limit is reached, client will be blocked. The default value is 20.
- blocking_timeout_millis – the maximum blocking time (in milliseconds) that will the connection will wait before timing out by throwing an exception. The default value is 5000 ms (5 seconds).
- Track_statements – boolean value that defines if system has to monitor and report for non closed statement and result sets. If the value is set to true the system throwns a warning message. Default value is false.
- Idle_timeout_minutes the maximum time before idle connections are closed.

3.5.1.3 Map Resources

This step maps the application resource (<resource-ref>) to the real database resource provided by the JBOSS AS database pool. The web application has to express this mapping in XML, as shown in Figure 3.5.

```
<jboss-web>
        <resource-ref>
                <res-ref-name>jdbc/CACHDB</res-ref-name>
                <res-type>java.sql.DataSource</res-type>
                <jndi-name>java:/CAHCDB</jndi-name>
</resource-ref>
</jbos-web>
```

This effectively maps java:comp/jdbc/CAHCDB to java:/CAHCDB

Figure 3.4 JBOSS web data source

The Figure 3.5 shows JBOSS MYSQL integration workflow.

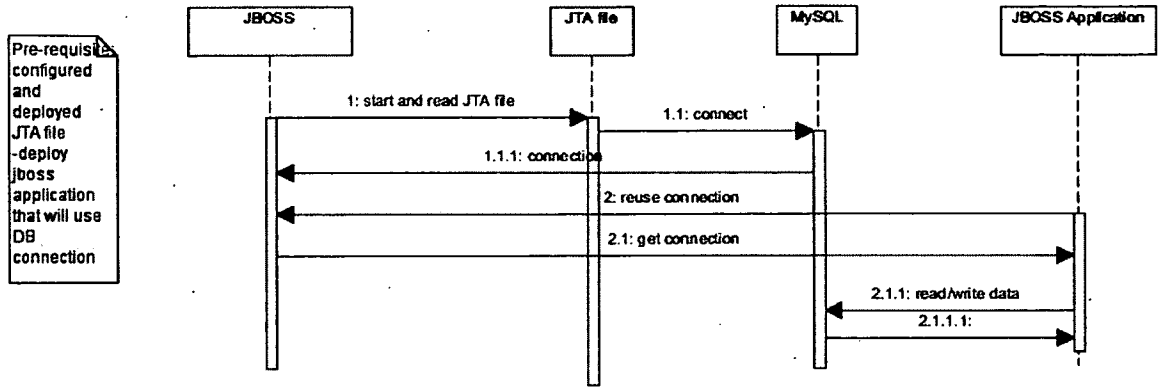

Figure 3.5 JBOSS MYSQL integration workflow

3.5.2 Hibernate and JBOSS integration

Hibernate middleware 3.6.0 has been already integrated into the JBOSS AS 6. No additional integration of hibernate APIs with the JBOSS is needed.

3.5.3 DROOLS Rule Engine and JBOSS integration

DROOLS software libraries can be divided in two groups: the first group is required during the rule definition and compilation time and the second during the run time [41] . The run time DROOLS engine is a very compact and included in two jar files with only 100 kilobytes. Developers can choose two options: to include all drools libraries at run time, which will bring flexibility, or to deploy the rules in binary form and include only run time libraries.

Table 3.1 shows the important JBOSS DROOLS libraries.

Library name	Description
Knowledge-api.jar	Provides factories, the user APIs and rule engine APIs.
Drools-core.jar	Contains the core engine and runtime component. The library contains both the RETE and the LEAPS engine. The LEAPS (Lasy Evaluation Algorithm for Production Systems) developed by Don Batory [54], is production system compiler that produces fastest sequential executable on the rule set.
Drools-compiler.ar	Contains the compiler/builder component that takes rule source and builds executable rule bases.
Drools-jsr94.jar	This is a layer on the top of DROOLS compiler that provides JSR-94 compliant implementation.
Drools-decisiontables.jar	This is the decision tables "compiler" component, which uses the drools-compiler component. The library supports excel and CSV formats for input.

Table 3.1 DROOLS libraries

3.5.4 RESTEasy web service and JBOSS integration

RESTEasy installation and configuration depends on the running environment. In this work where theJBOSS AS 6 has been chosen, where RESTEasy is already bundled and integrated completely, that means no additional framework integration work is required [44].

The developer of the context awareness system can focus on building the application that will provide the RESTEasy services. Example of services will be covered in section 4.6.

3.5.5 XMPP Server and JBOSS

There are two possible ways for integrating the notification component functionality into the framework. One option is to use open source XMPP server library Jabber and integrate it into the JBOSS server. In this work the second option has been taken; it has been used open source OPENFIRE server as the notification server. In the second approach, the JBOSS server and OPENFIRE server are two separate servers. The second approach simplifies and shortens development time. Figure 3.6 shows the solution where open source XMPP server Jabber is integrated with JBOSS stack and uses the same MySQL database.

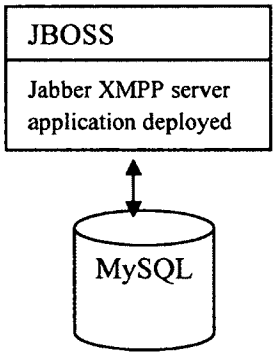

Figure 3.6 Jabber XMPP server integrated with JBOSS server stac

3.5.6 OPENFIRE installation and configuration

The OPENFIRE server comes with two possible installations. By default, OPENFIRE will installed to use embedded OPENFIRE database for keeping information about subscribers, sessions, configuration and administration. Figure 3.7 shows the default OPENFIRE installation that will use a default embedded database.

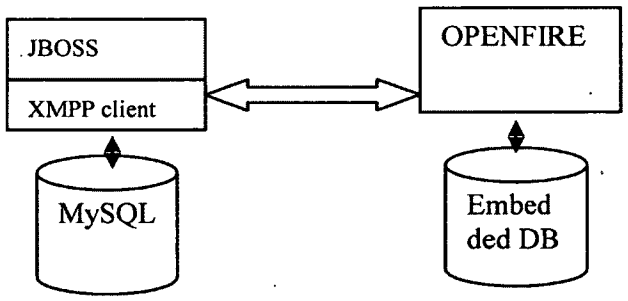

Figure 3.7 OPENFIRE installation with embedded database

The second approach, showed in Figure 3.8, is to install OPENFIRE and use an external database, which is the MySQL database used by the context-aware application. The database will be shared between JBOSS and OPENFIRE servers.

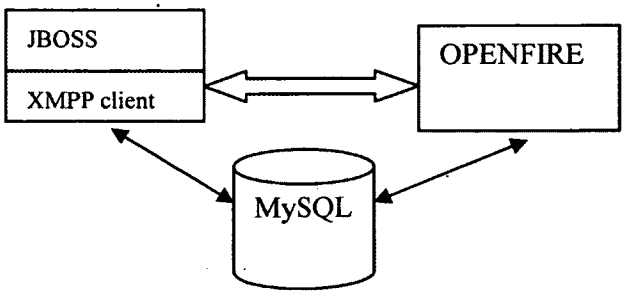

Figure 3.8 OPENFIRE installation with MySQL database

3.6 Framework use-cases

There are two generic use cases that framework has to fulfill:

- The first use case requires that context awareness framework has to be able to **notify** other systems or mobile users (for example, smart phones) in real time when a certain change of state occurs in the system. Figure 3.9 shows the real time notification process.

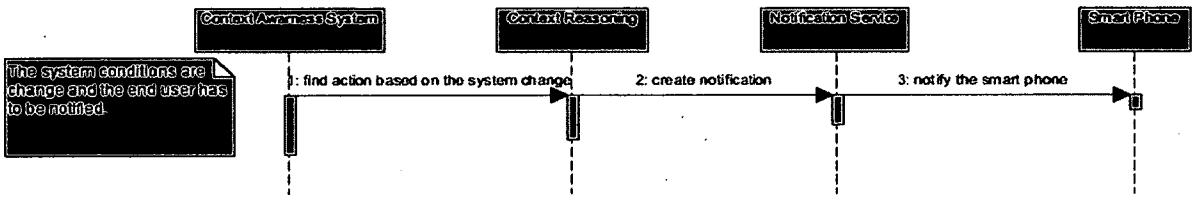

Figure 3.9 Real time notification process

- The second use case requires that the context awareness framework should be able to provide **automatic context reasoning** results based on a preconfigured set of rules taking into consideration the current state of the system and context values.

4 Health Care Application case study

This chapter presents how the context awareness framework designed in the previous chapter can be used for building domain specific applications on top of it. More specifically, the thesis designs and implements emergency room case studies that were identified by Yu et al. in [1]. According to [1], the healthcare domain has the following characteristics: (1) *None-routine:* emergency, exceptions, and interactions occur frequently in hospitals, which makes pre-scheduling of activities difficult or even impossible; (2) *Mobile*: Physicians are frequently moving between patients and medical facilities, so they access the system from different locations. (3) *Context-driven*: Physicians are frequently switching from one patient to another, one device to another, or one software application to another, so the interaction with the system may be driven by the context; (4) *Highly collaborative*: Multiple clinicians provide care for each patient, therefore information needs to be communicated between people in different shifts, roles, and locations. (5) *Multi-tasking*: Each clinician provides care for multiple patients in parallel. (6) *Time-critical*: There is a need to complete certain tasks in a temporal sequence to process patients according to the workflow (i.e., the tasks need to be performed in a certain order). (7) *Information reach*: Physicians are faced with a constant need for interpreting rich data, as they deal with information loads of various kinds that change with time.

4.1 Application Requirements

The context-aware application built in this chapter considers five emergency room use cases described in the next section. The application considers at least three types of departments

(emergency, laboratory and general medicine) and the following types of users: patient, doctor, nurse, and administrator. The application requirements are:

1. Different types of users can be added to the system seamlessly.
2. The application should support adding a new department and its members easily.
3. The system should allow adding new devices and sensors easily. Each sensor will acquire measured values that will be stored in the database.
4. Each sensor could be linked to a patient for a certain time.
5. A patient could have one or more different sensors linked to him/her. Radio Frequency Identification RFID identification bracelets are used for identifying each patient.
6. The system can add mobile devices without stopping and starting the application server.
7. Each doctor, nurse or administrator, can use one or more mobile devices.
8. A doctor, nurse or administrator will be notified when the system state changes and they need to know about the change. For example, a doctor has to be notified when the laboratory results requested for a patient are ready.
9. The administrator is able to configure rules that will define certain conditions in the system. This should be done without stopping and starting the server.

4.2 Application Use Cases

The application covers five use cases listed below:

1. Admission to the Emergency Department (ED)
2. Locating a patient

3. Checking an ED patient's status
4. Transferring a patient from ED to GM
5. Laboratory Analysis result notification based on rule-based context reasoning

4.2.1 UC1 - Admission to the Emergency Department

This use case covers the patient admission to the Emergency Department. When a patient is admitted to the triage, if the patient 's profile does not exist already it the system the profile will be created. otherwise the profile will be updated if needed. The patient profile contains the patient address, patient health status, such as food and medication allergies, the patient's family doctor and family members that should be contacted.

At the end of the admission process, the nurse will put an RFID identification bracelet on the patient's wrist. The RFID identification bracelet will be used to locate the patient for fast patient identification and to retrieve the patient profile by simply scanning of the patient bracelet. The patient profile will be linked with the RFID bracelet id.

Figure 4.1shows an admission to the emergency department use case workflow.

This use case does not differ a lot from the current admission procedure. The only difference to the standard way of doing admission is that a bracelet put on the patients wrist is RFID sensor that will provide the patient location and will be use to identified the patient and retrieve the patient info by an authorized doctor or a nurse.

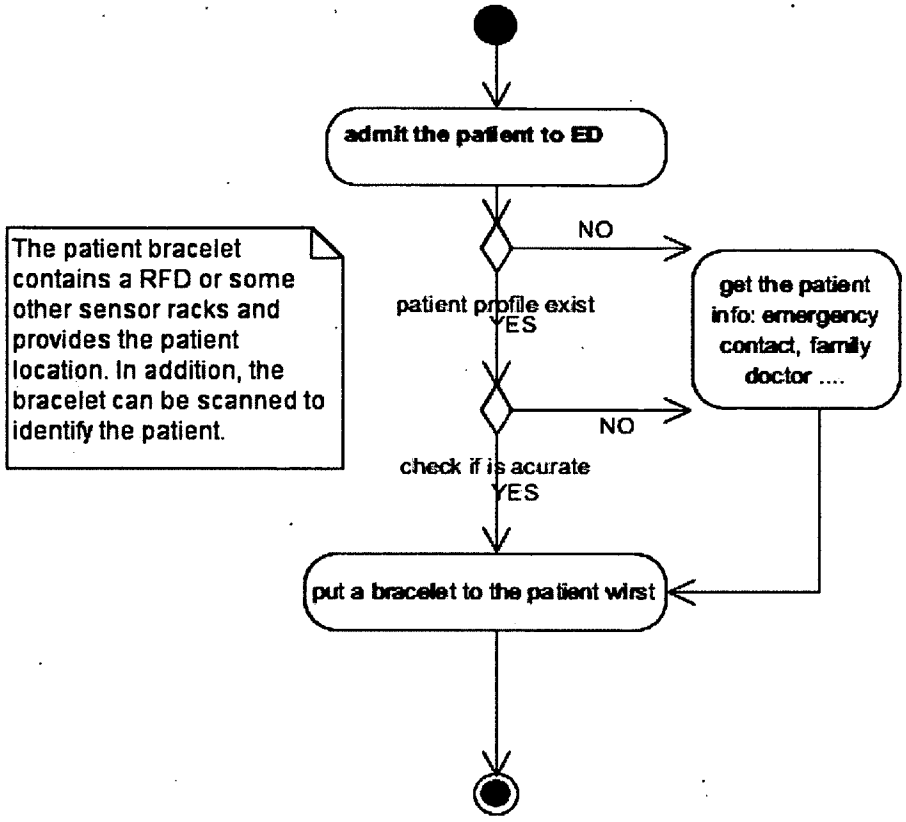

Figure 4.1 Admission to the emergency use case

4.2.2 UC2 - Locating the patient

Emergency studies done by Ye [1] et al. show that doctors and nurses spend a lot of time trying to locate a patient. For instance, very often after sending the patient to the laboratory and receiving the patient's results a doctor has difficulty locating the patient, because the patient has been moved to another room in hospital. This use case tries to solve this problem. Prerequisite for this use case is the previous use case – admission to emergency where RFID bracelet was placed on the patient and linked to the patient profile.

The RFID sensor provides the location of the patient: the sensor sends location updates to the

database when the location is changed, and it is linked to the patient profile. Figure 4.2 shows the workflow for locating the patient use case.

This use case can be used, for example, when the doctor receives notification from the context awareness framework that patient's laboratory result is ready; the doctor will view the results and retrieve from the system the patient location.

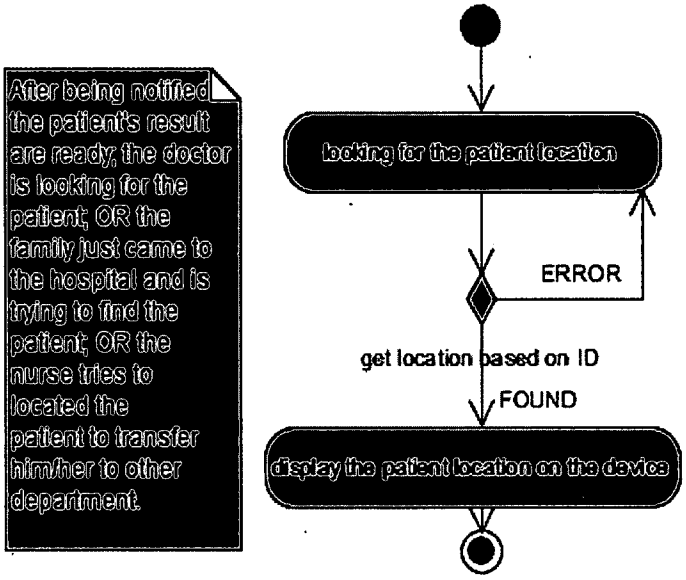

Figure 4.2 Locating the patient use case

4.2.3 UC3 - Checking ED patient's status realization

The emergency room study done by Ye [1] et al. suggested that it would be beneficial for the doctor in the emergency room to be able to look up the current list of patients in emergency prior to the doctor's shift.

The doctor should login to the context-aware application via smartphone or portal by

providing credential information. The doctor requires from the application a list of patients currently admitted to the emergency. The doctor is able to sort result based on the different sorting criteria such as emergency level, time spent in triage, etc. The doctor is able to select a single patient and read the patient profile and the current status. In addition, the doctor can even view the laboratory tests remotely. Figure 4.3 shows the workflow for checking ED patient status.

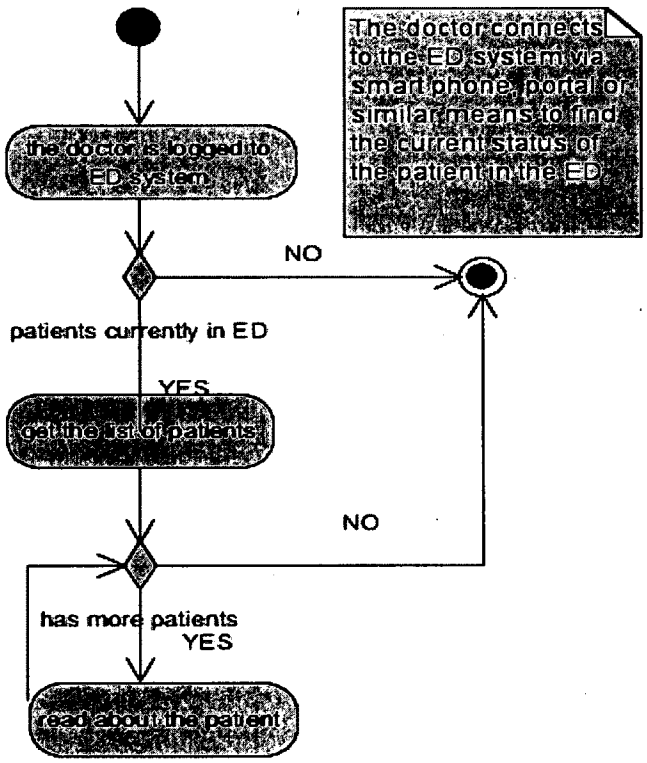

Figure 4.3 checking patients' statuses use case

4.2.4 UC4 - Transferring the patient from ED to GM

The emergency room study [1] shows that after making the diagnosis for a patient, the doctor in the emergency department (ED) makes the decision to transfer the patient to some other department in the hospital, for example general medicine (GM). This use case tries to make this transfer smoothly and efficiently.

When the ED doctor decides to transfer a patient to an other department, for example GM, the ED doctor will use the context-aware application to find currently available doctors in the GM department. The ED doctor will notify the GM doctor about the patient. The GM doctor will request the patient profile and review comments and results. If needed, the two doctors may have a phone or conference call where each doctor will look at the same results on their computers, portals or smart phones. (As part of the context-awareness feature, the results may be displayed differently, depending on the device used). In parallel, the ED doctor or ED nurse will send a request to the nurse in the general medicine department that the patient with the respective patient id has to be moved to the GM department. The GM nurse will be able to locate the patient easily and quickly, based on the patient RFID sensor (UC2). Figure 4.4 shows the workflow for transferring the patient use case.

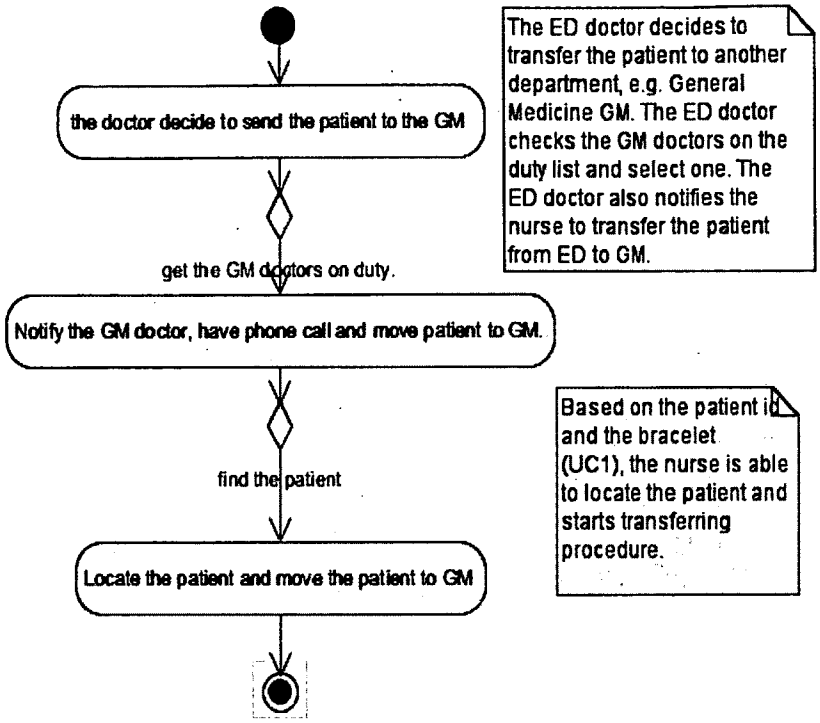

Figure 4.4 Transfering the patient to GM department use case

4.2.5 UC5 – Lab Analysis result notification based on rule-based context reasoning

The emergency room study [1] showed that a lot of time and human energy is wasted when the doctor sends a patient to the laboratory for tests. The doctor has to go periodically to a room where the results are delivered. The second problem that was already mentioned is that once the results are ready, the doctor may have a hard time locating the patient.

This use case uses the context awareness notification mechanism. Once a laboratory test is done and the results are saved in the database, the system will run a preconfigured rule to determine who should be notified about the respective results and which notification media should be used. This is an example of rule-based context reasoning. The system will take into

consideration result values and patient profile. For example, the emergency doctor who requested the test will be notified in real time when the results of his/her patient are ready. On the other hand the family doctor may receive an e-mail notification about the patient results if that was preconfigured in the rule. Figure 4.5 shows the workflow for the lab analysis result notification use case.

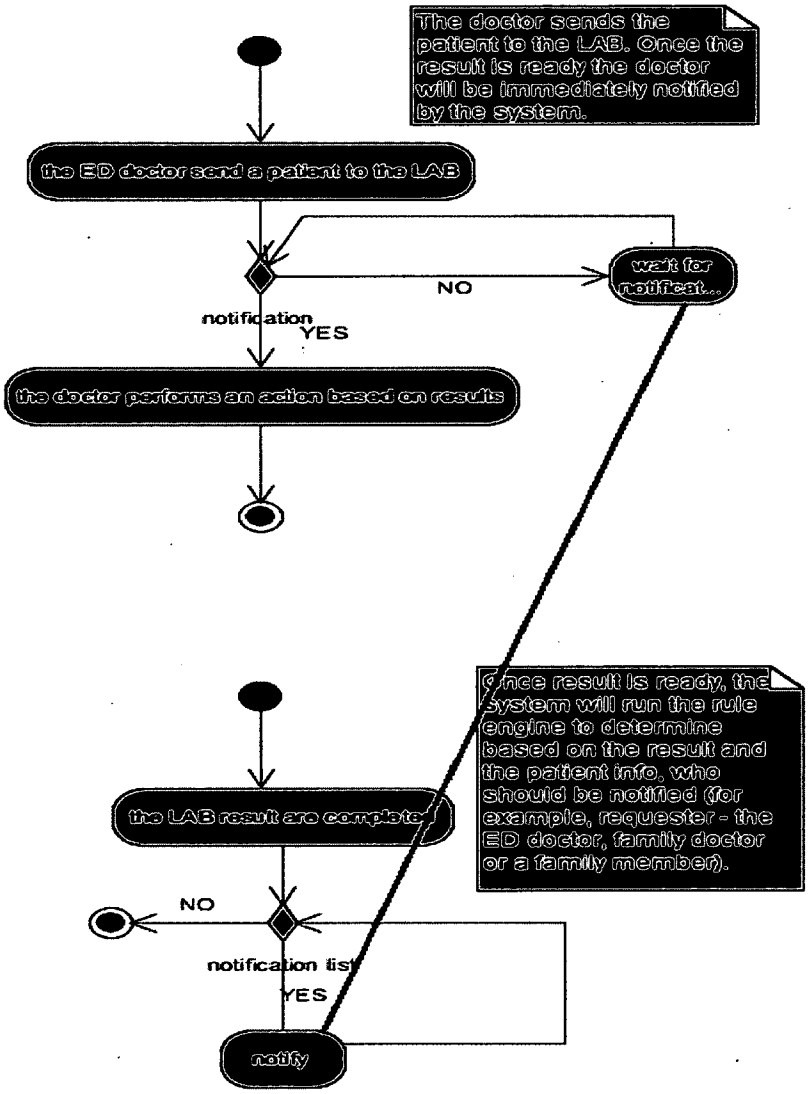

Figure 4.5 Lab analysis result notification and running the rules to found who should be notified

4.3 Use Cases linked with the framework

In this section of the thesis will show the link between application use cases and framework components.

The previously explained five use cases will be covered.

4.3.1 UC1 Admission to the Emergency

In this use case a nurse or a doctor will start two consecutive actions. In first action, the nurse or the doctor will use a web application or a mobile application to create a new patient profile, if the profile does not exist, otherwise they will update the patient profile. The context aware application will invoke a REST web service that updates the patient profile. The REST web service will delegate the patient profile to Hibernate middleware component, which will save it in the patient data base table.

In the second consecutive action, the nurse or the doctor will put a RFID bracelet on the patient wrist and update the patient profile with the RFID bracelet id.

Every time the patient goes from room to room, the RFID sensor will update its location in the database. This way the application will be able to provide the patient's current location at any time. Figure 4.6 uses Use Case Maps to present this use case.

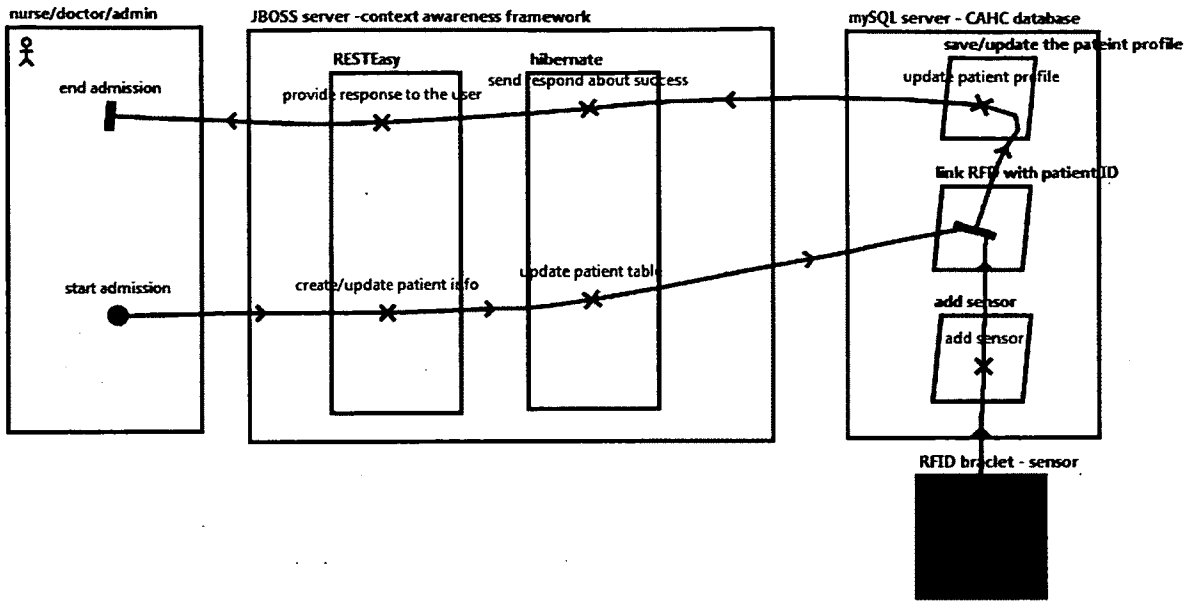

Figure 4.6 UCM for managing the patient location in the emergency department

4.3.2 UC2 – Locating the patient

Locating the patient can have two possible paths. In the first path, a nurse or a doctor would like to find the patient's location. The nurse or the doctor will connect to the system using a web browser, a smartphone or a portal that will connect to the context aware application, and will request the current location of the patient based on the patient id. The request will go to the REST API that provides the patient location based on the patient's unique id. The request will use Hibernate middleware to retrieve data from the database.

In the second path, the patient RFID bracelet will update the location in the database every time the patient changes room. Figure 4.7 uses Use Case Maps to present the scenario for locating the patient.

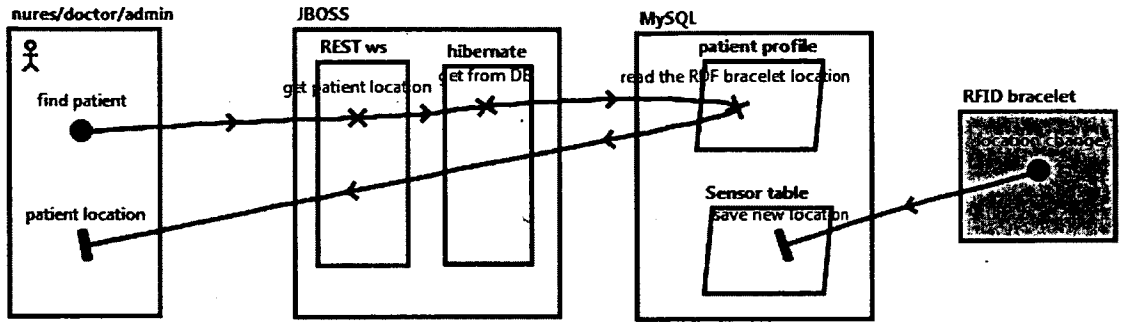

Figure 4.7 UCM locating the patient

4.3.3 Checking ED patients status results remotely

The doctor is connected remotely to the ED system using a smartphone, portal or laptop, after the context aware application checks the authentication information provided by the doctor. The doctor requests the list of all patients in the emergency department via a RESTweb service that will ask the Hibernate middleware to fetch the respective data from the database. After displaying and reviewing the list of patients, the doctor can request to view a patient profile by clicking on the patient listed. The click will invoke the RESTweb service that will provide the patient profile of the selected patient. The RESTweb service will delegate to Hibernate middleware the fetching of the patient profile from the database. Figure 4.8 shows the UCM scenario for checking remotely the ED patient status.

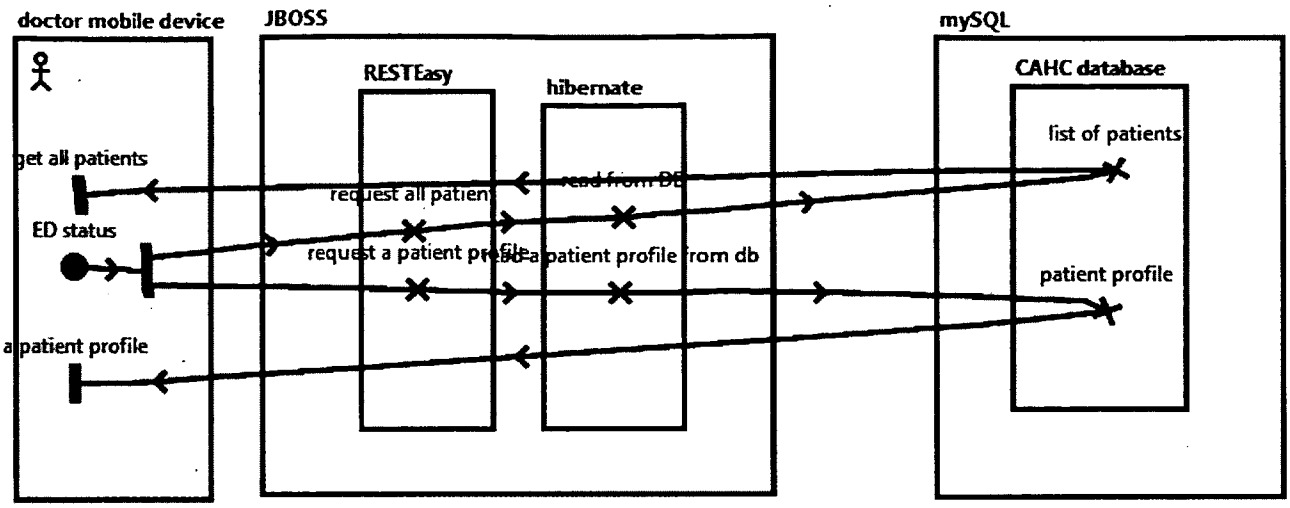

Figure 4.8 UCM checking ED patient status remotely

4.3.4 Transferring the patient from ED to GM

In this use case after diagnosis in the Emergency Department the patient will be transferred from ED to the General Medicine Department (GM). The ED doctor will use a web application or mobile device application to find an available GM doctor. The application will request the REST API that provides a list of currently available doctors in the GM department. The REST web service will delegate the task to Hibernate middleware component that will fetch data from the database. Once the list of available doctors in GM is presented to the ED doctor, the doctor will select one of the doctors from the list and the system will send a push notification message to the selected GM doctor. The GM doctor will receive notification on one of his portable devices. The application will establish a connection between the two doctors and they will be able to exchange information on the patient and even to look at the lab results together, by viewing the patient profile, even if they are in two different locations. Figure 4.9 shows UCM scenario for transferring the patient from ED to GM.

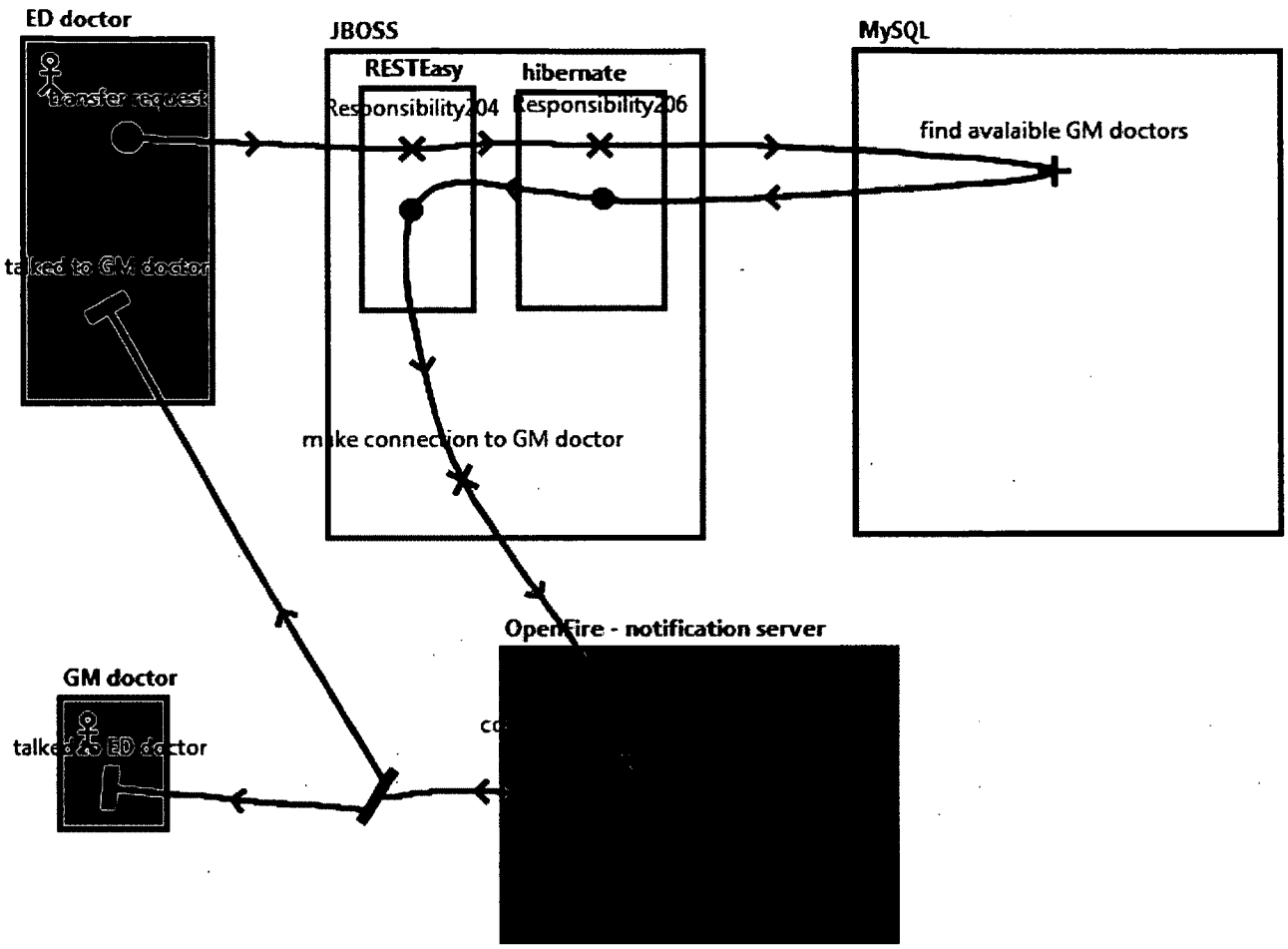

Figure 4.9 UCM for patient transfer from Emergency to General medicine

4.3.5 Laboratory Analysis result notification

In this use case the ED doctor request laboratory tests for his/her patient. The doctor is using a web application or smartphone application to request the laboratory test; the test linked to the patient profile. The patient is sent to the lab. After this step, the path will wait for the results. Once the results are ready, the system will update the database, and the patient profile will be fetched from the database. Once fetched, the patient profile will be used by the rule engine to determine who should be notified about the patient results and how. For example, the ED

doctor that requested the lab tests will be notified by push message, and the patient family doctor will be notified by e-mail. Figure 4.10 shows the UCM result notification.

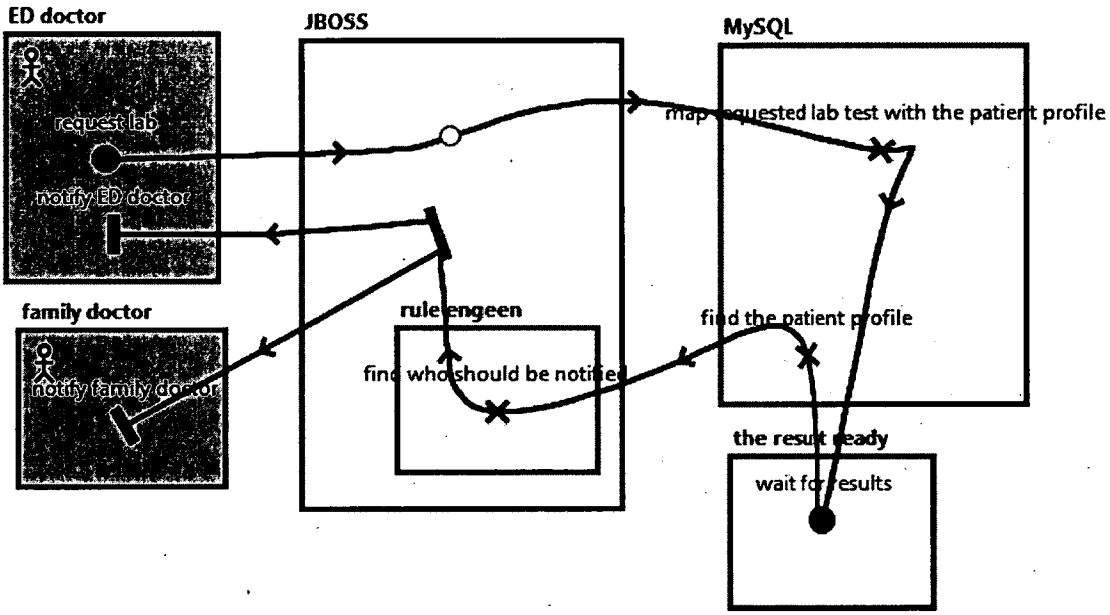

Figure 4.10 UCM Laboratory result real time notification

4.4 Domain Data Model

Figure 4.11 presents the data model for the Emergency Room applications in the form of a class diagram.

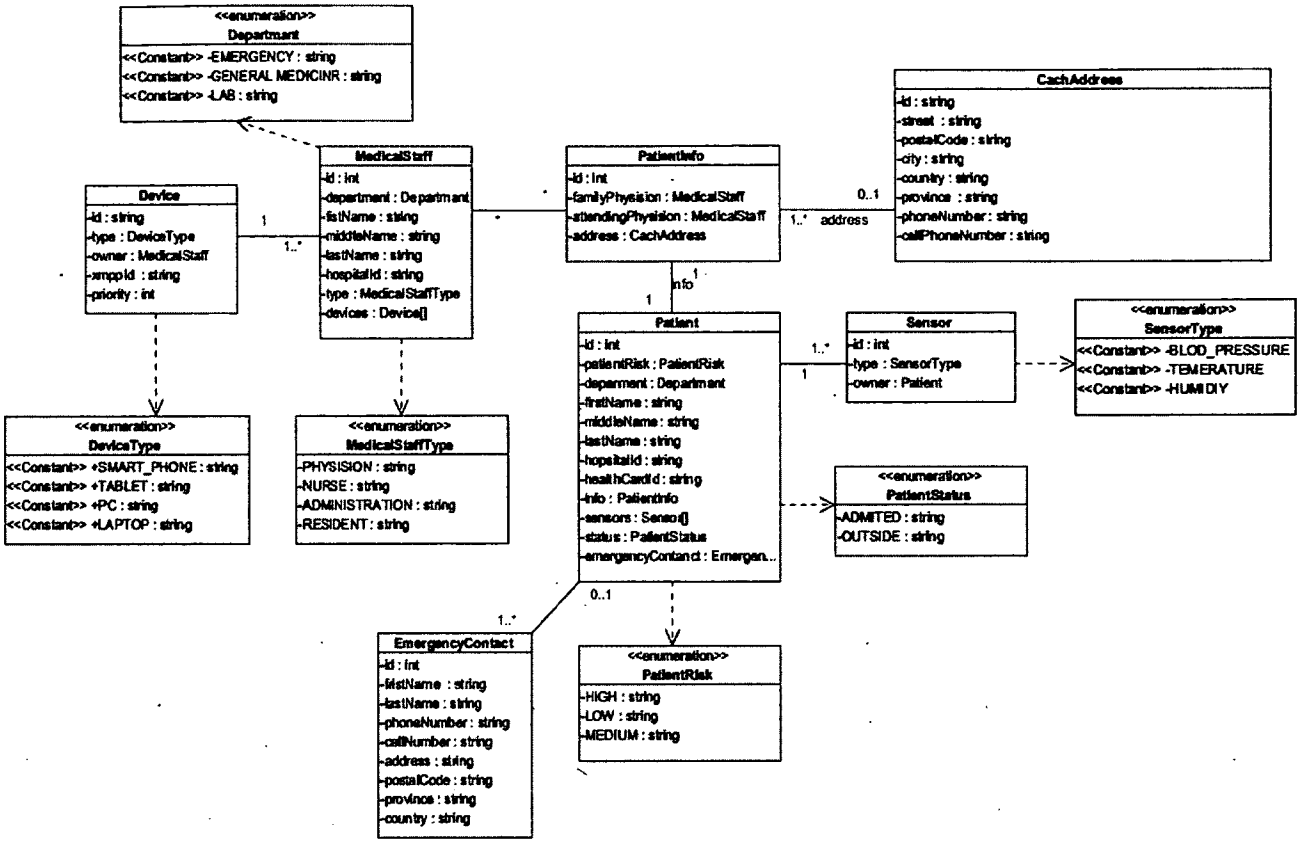

Figure 4.11 Emergency room data model for the Emergency room case study

4.5 Context specification

During the course of the study has been observed that context can be divided into two groups: (1) Human Context, and (2) Physical Context. The Human context (Table 4.1) provide information about the humans in the application, such as get the list of currently available doctors in General Medicine department (UC3), or get the list of patients currently admitted in the Emergency department (UC2).

Human Context	Value	Description
A doctor	In Hospital, out of Hospital	Shows if a doctor is currently in the hospital or not
A patient	Admitted in ED, outside	Shows if a patient is currently

		admitted in the Emergency Department
A patient risk	High, low, medium	Shows a patient health risk level
A patient's family member	Family member	Provides a patient's family member info if it is applicable
A patient's family doctor	Family doctor	Provides a patient's family doctor info if it is applicable

Table 4.1 Human Context

Physical Contexts (Table 4.2) are all values that have been monitored by sensor, devices or lab results

Physical Context	Value	Description
A patient's location	Room	Provides a patient's current room.
A patient's hearth rate	Hearth rate value	Provides a patient's current hearth rate monitoring value
A patient's lab result	Ready or not ready	Provides information if a patient's lab results are available
A doctor's device	Connected or not connected	Provides information if a doctor's device is connected to the Context Awareness Framework and is available to receive a notification from the system

Table 4.2 Physical Context

4.6 Available REST APIs

This section describes the interfaces for the REST web services that were designed for the emergency room case study. The following five API subsections for accessing data has been developed: patients, medical staff, departments, sensors and devices. More details about patients, medical staff, departments, sensors and devices API can be found in Appendix A.

4.7 Rules in Emergency Room Case Study

In the Emergency Room Case Study two scenarios, where context reasoning was required, were implemented: Laboratory Analysis Result Notification (

Figure **4.12**), and Monitoring Alarm Notification (Figure 4.13).

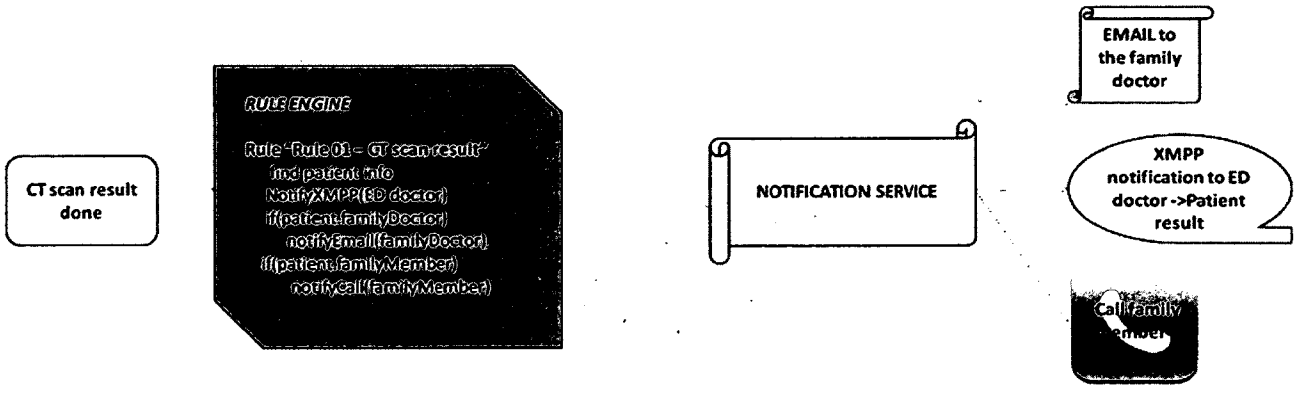

Figure 4.12 Laboratory Analysis Result Notification workflow an example

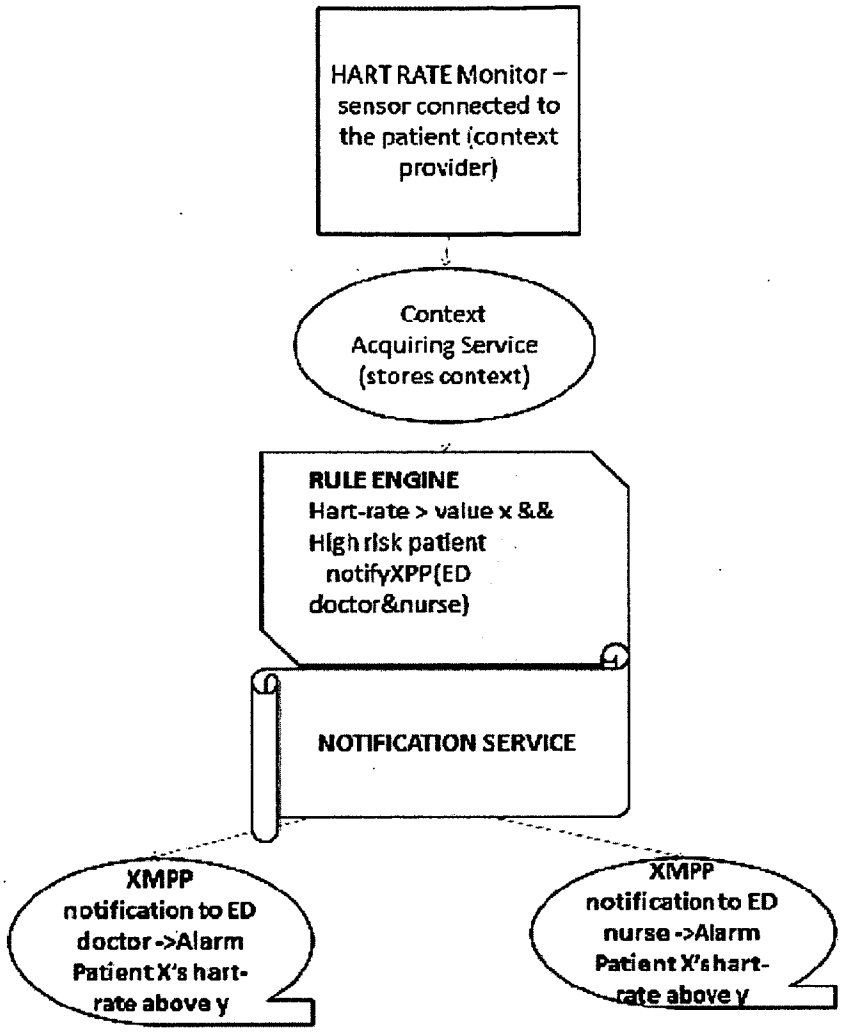

Figure 4.13 Monitoring Alarm Real time Notification workflow an example

The Laboratory Analysis Result scenario, when a laboratory result is ready and saved in the database, the database save process will invoke Laboratory Analysis Result Rule. The Rule consists of a set of conditions and actions that have to be considered during the rules execution. Table 4.3 shows the definition of the Laboratory Result Rule.

Conditions	Actions	Description
Result ready	Notify the ED doctor	When results are ready the ED doctor will be notified by xmpp notification.
Result ready and patient has family doctor	Notify the family doctor	Find the preferred notification from the family doctor record and use the same to notify.
Result ready and patient has family member	Notify the family member	Find the preferred notification from the family member record and use the same to notify.

Table 4.3 Laboratory Result Rule Definition

In the Monitoring Alarm Notification scenario, when the patient's sensor that measures some value goes outside of some range and alarm notification is send. For example, the patient's hearth rate went above the alarm value the alarm notification will occur. Table 4.4 shows the rule definition for the Monitoring Alarm Notification.

Conditions	Actions	Description
If patient status is high and value is above alarm value	Notify the ED doctor	When monitoring condition is above the allowed value notify the ED doctor with XMPP notification.
If patient status is high and	Notify the family doctor	Find the preferred notification

value is above alarm value and patient has family doctor		from the family doctor record and use the same to notify.
If patient status is high and value is above alarm value and patient has family member	Notify the family member	Find the preferred notification from the family member record and use the same to notify.

Table 4.4 Monitoring Alarm Notification Rule Definition

4.8 Context Notification

The notification system consists of three components: OPENFIRE Notification server, a Device, and a Context Awareness Framework XMPP component.

The OPENFIRE server comes with an administration console that can be used to create uses, view sessions, to configure server properties, view group chats and more. Figure 4.14 shows User Summary page of the OPENFIRE Administration console. The page shows three types of users: the console admistrator, CAHCserver user and other users. The Context-Aware for Health Care server (CAHCserver) user will be used by the Domain Independent Context Awareness Framework to connect to the OPENFIRE server and send the notification to the end user (a doctor, or a nurse). Each user has unique XMPP user id and password. The unique XMPP user Id and password are used by a device to connect to the OPENFIRE Notification server. The XMPP user Id is used by the Domain Independent Context Awareness Framework to send notification message to the device.

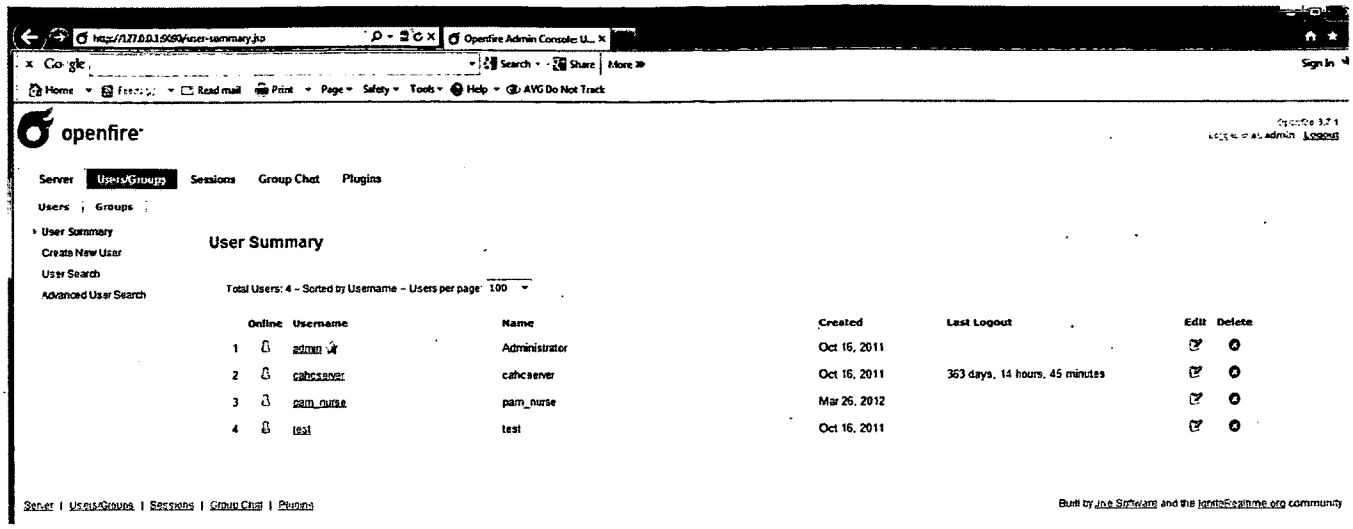

Figure 4.14 OPENFIRE Administration console

Each Device, which can be a smartphone or a tablet, or simply instant messaging running on a PC, should have installed a library that can connect to the OPENFIRE server and is able to send and receive messages from other users configured on the OPENFIRE server. For the thesis Emergency room study the open source Smack library [35] has been used.

The Domain Independent Context Awareness Framework contains a notification component that connects to the OPENFIRE Notification server after starting the application server and keeps open the connection with the OPENFIRE server. The Domain Independent Context Awareness Framework uses CAHCserver credential to connect to the server and to send notification to devices by using the device unique XMPP user id when needed.

5 Context Awareness Framework extension: CARGO

A highly important component of a context-aware system is *context reasoning*, which provides system flexibility and automated, run-time decision making based on preconfigured criteria and the current context without direct human intervention. Typically, as we already mentioned, context reasoning is based on predefined rules, often logic-based and described using declarative languages. Rules are stored in a *rule base* and processed by a rule (inference) engine. The *rule engine* evaluates the rule conditions based on the current contextual information and determines which rules are eligible to be executed.

While rules excel in filtering out unsuitable solutions based on clear criteria, it is difficult to rank the remaining suitable solutions based on vague, qualitative criteria for several stakeholders with a rule-based approach. To overcome this problem the thesis propose, a new context-aware reasoning approach called CARGO (Context-Aware Reasoning using Goal-Orientation).

5.1 CARGO - Context Aware Reasoning using Goal Orientation

Context-Aware Reasoning using Goal Orientation CARGO uses goal-oriented and scenario-based models as run-time entities, combines rule-based and goal-oriented reasoning (the latter complementing the former), provides more flexibility and configurability than the commonly used logic-based, rule-based approach, and allows for a more holistic assessment of the context as the goals of many stakeholders are taken into assessment of the context as the goals of many stakeholders are taken into account at run-time.

5.1.1 CARGO workflow

Figure 5.5.1 shows two possible reasoning paths. The first path, which ends with 2A.1 presents context reasoning without goal orientation extension. The second path, which ends with 2B.2 presents reasoning with CARGO. Change contextual information (1) in the context awareness system will trigger preconfigured rules (2) in the rule engine that will perform an action. Very often, the rules are used to performed filtering and produce a list of possible options. The list result is random order (2A.1). In the second path, after step 2 the list of results will go to a goal engine, where results will be evaluated in the goal model that will produce a ranked list of result. This chapter will present how the context awareness framework introduced in chapter 3 is inter-connected with the JUCMNav tool[50] in order to use the goal modeling capabilities of the latter for extending the rule-based context reasoning of the former.

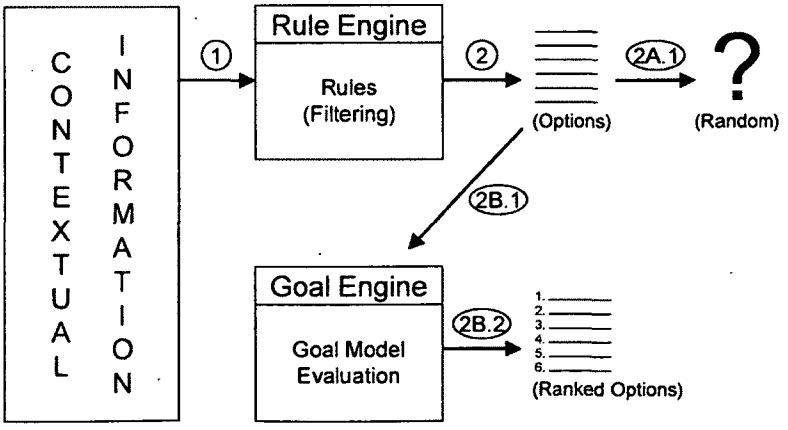

Figure 5.5.1 CARGO reasoning paths

5.1.2 CARGO Architecture

With a clearer understanding of the complementing goal-oriented reasoning approach, an architectural decision needs to be made in terms of how to integrate goal-oriented reasoning with the existing context awareness framework introduced in chapter 3. Figure 5.5.2 illustrates (i) the open-source architectural components of the service-oriented framework that are relevant to this task and (ii) the main components of the jUCMNav tool. The service-oriented framework is responsible for capturing the context in the database and any changes to the context. The rule engine reacts to these changes and the effects of the resulting actions are communicated over a Web service infrastructure. jUCMNav, on the other hand, is an Eclipse-based plugin that is capable of traversing workflow models and evaluating goal models.

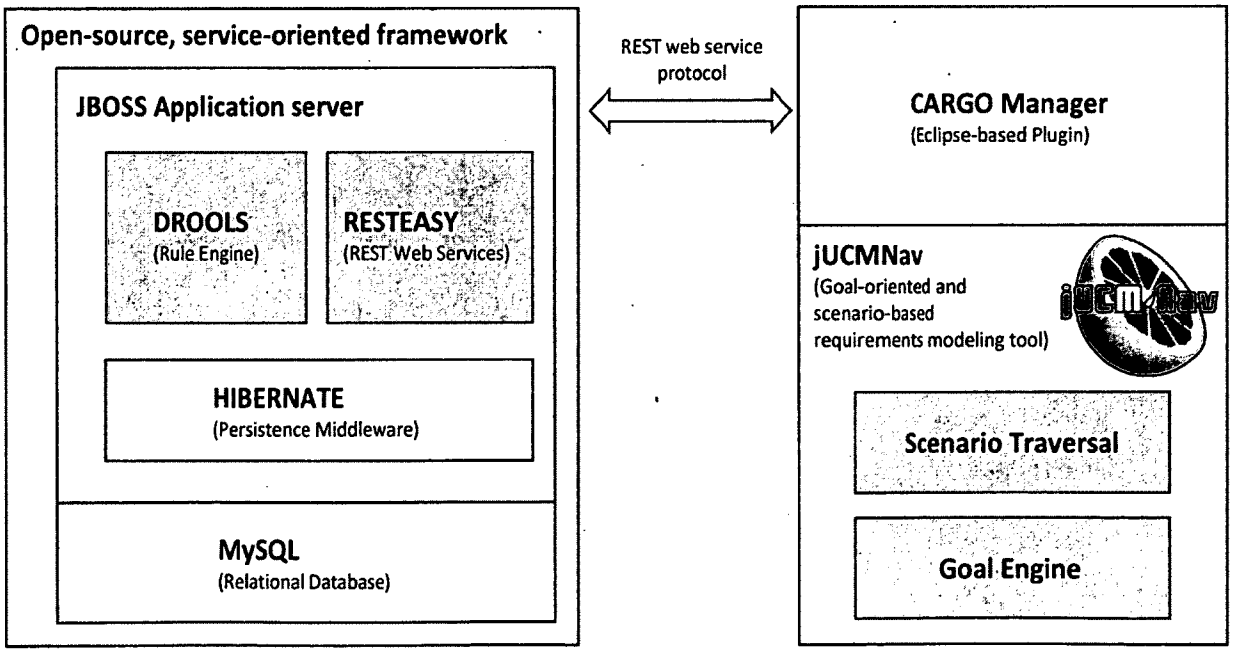

Figure 5.5.2 CARGO architecture

There are three possible alternative architectures. The fist architectural alternative extracts the goal-oriented reasoning engine from jUCMNav and incorporates it directly into the rule engine. In this alternative, goal-oriented reasoning is just another kind of condition to be used by a rule. All rules are handled internally by the rule engine. However, a goal model editor has to be provided in the service-oriented framework in addition to the reasoning mechanism to be able to specify and maintain goal models, making this a rather unappealing alternative.

The second architectural alternative is rule engine-driven. In this case, goal-oriented reasoning is again just another kind of condition that can be used by a rule. However, the rule engine delegates the evaluation of the condition (i.e., the goal model) to jUCMNav's reasoning engine. This alternative requires jUCMNav to be Web service-enabled and does not make any use of URN's workflow notation to describe the functionality of the context-aware system or to use this description as the basis for simulation and execution. Hence, this alternative is also not further pursued.

The third architectural alternative is URN-driven – it is the one shown in Figure 5.5.3. The CARGO Manager is added to the jUCMNav environment. It is capable of executing URN workflows of a context-aware system and interacts with the rule engine and the URN goal engine as defined by the URN workflows. This alternative requires the rule engine to be Web service-enabled, which it already is. The complete infrastructure for URN modeling supported by jUCMNav is directly available to the CARGO Manager. From the rule engine's point of view, goal-oriented reasoning is again just another kind of condition a rule can use, but the combination with the goal-oriented reasoning mechanism requires a specific interaction protocol to be followed by the CARGO Manager, the rule engine, and the goal engine as

depicted in Figure 5.5.3 . The protocol ensures that the CARGO Manager, the rule engine, and the goal engine can exchange all contextual information required for combined reasoning.

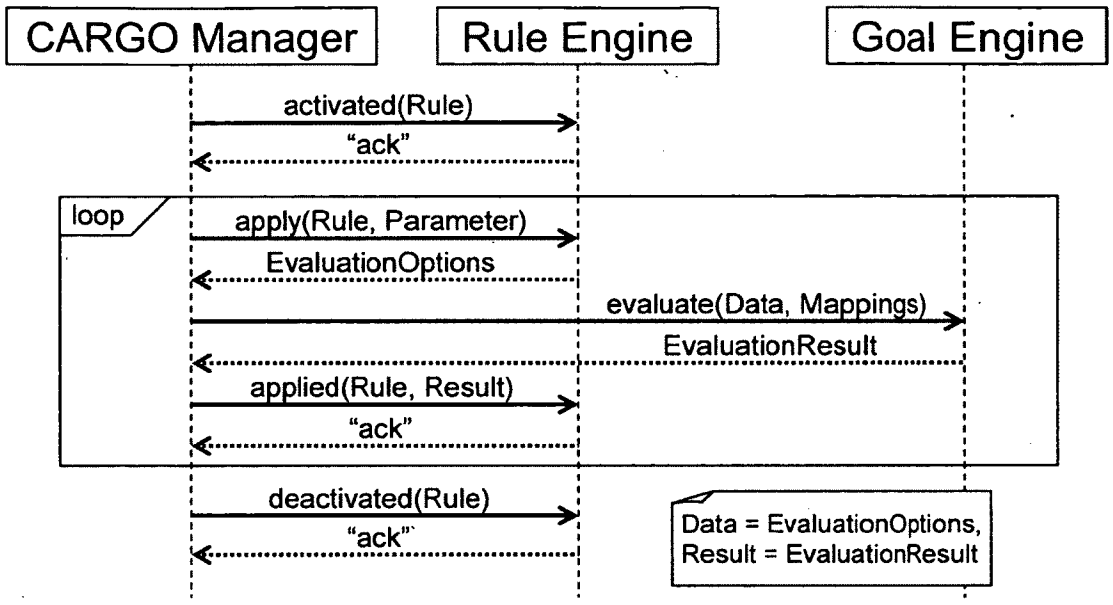

Figure 5.5.3 Interaction between the CARGO Manager, the rule engine, and the goal engine

The rule engine still requires all the rules to reach the list of options in Figure 5.5.1. In addition, one rule must state that if goal-oriented reasoning is activated, path 2B is followed. A second rule must state that if goal-oriented reasoning is not deactivated, an option is chosen randomly (2A.1). The messages in Figure 5 between the CARGO Manager and the rule engine are Web service calls and Java method calls in the case of the goal engine. An application that wants to use combined rule-based and goal-oriented reasoning to process contextual information starts by telling the rule engine that goal-oriented reasoning is **activated** for a particular **Rule**. This sets a flag to ensure that path 2B is chosen in Figure 2. Whenever a list of options becomes available, it now needs to be handled by the goal engine. Since many lists may become available at any given point in time, they are queued by the rule engine. At some point, the CARGO Manager informs the rule engine to **apply** a **Rule**. The

first queued list of options that corresponds to this rule is now made available to the goal engine as a list of evaluation options. The goal engine evaluates the options (**evaluate**) and the CARGO Manager sends another message to the rule engine informing it of the results of the evaluation (**applied**). This continues in a loop until the application is terminated by the user, at which point the rule engine is told that goal-oriented reasoning is **deactivated** for a particular **Rule**. As an alternative to evaluating the next list of options in the queue, the CARGO Manager may specify a **Parameter** when informing the rule engine to **apply** a rule. In this case, the list of options for a particular entity (e.g., the list of meals for a particular patient) is provided to the goal engine.

5.1.3 CARGO URN profile

Context-aware systems are a novel application domain for URN. To enable the specification, simulation, and execution of context-aware systems based on the third architectural alternative, CARGO requires specific extensions to URN, which are defined in a *URN profile* for CARGO. The URN profile constitutes a domain-specific language that provides the ability to specify actions in a URN workflow model required by a context-aware system, i.e., these actions correspond to the messages in Figure 5.5.3. The actions cover activation and deactivation, input, application of a rule, evaluation of a goal model including trade-off analysis, and output of evaluation results.

«activated: Rule» – The keyword *activated* indicates that the rule engine is contacted and informed that goal-oriented reasoning has been activated for the *Rule*. The resulting ErrorCode is set to OK if "ack" is received from the rule engine and to CANCEL if not.

«input: "Prompt" Variable» – The keyword *input* indicates that a dialogue with the specified *Prompt* is presented to the user. An input of type String is stored in the *Variable*. The variable is optional. If it is not specified, the user is presented the prompt and only allowed to select OK or CANCEL. In any case, the resulting ErrorCode is set to OK if the user clicks on OK and to CANCEL if the user clicks on CANCEL.

«applyRule: Result=Rule(Parameter)» – The keyword *applyRule* indicates that the rule engine is contacted and the specified *Rule* is applied. As a *Parameter*, a variable of type String is optionally specified. If a parameter is specified, then the list of options is gathered by the rule engine for the entity identified by the parameter. If no parameter is specified, then the next list of options in the queue of the rule engine is gathered by the rule engine. The *Result* of this action is a variable of type EvaluationOptions. This type contains a String for an identifier and a Set<EvaluationOption>. For each EvalutionOption, one or more name/value pairs can be specified that correspond to an attribute of the entity to be evaluated and the value of this attribute. The resulting ErrorCode is set to NOTFOUND if the identifier null is returned by the rule engine, to NOOPTIONS if the identifier is specified but Set<EvaluationOption> is empty, and to OK if the identifier is specified and Set<EvaluationOption> is not empty.

«evaluate: Result=Data(Indicator=Attribute;)» – The keyword *evaluate* indicates that the goal model is to be evaluated. The *Data* variable is an input of type EvaluationOptions, hence the result of the applyRule action can be used. One or more mappings between a goal model element (i.e., *Indicator*) and an *Attribute* of an EvaluationOption are specified. These mappings are used to initialize the goal model before evaluating it. For example, in the Food service case study that will be covered in the chapter 6, *Cost=cost* means that the indicator *Cost* in the goal model (see Figure 6.1) is initialized with the value of the *cost* attribute of an

EvaluationOption. Note that the indicator converts the dollar value (e.g., $5) during initialization into a goal model value in the range from -100 to +100 (e.g., 71). The indicators (denoted by ⟺) specify what context-related information is required for goal-based reasoning. The variable *Result* is of type EvaluationResult, contains a String for an identifier, and captures for each EvaluationOption the satisfaction values of the stakeholders in the goal model (e.g., HospitalKitchen/-20, HospitalAdminstration/41, and Patient/100) and the overall satisfaction value of the goal model (e.g., 53). The evaluation options in *Result* are sorted in descending order by the overall satisfaction value.

«applied: Rule(Result)» – The keyword *applied* indicates that the rule engine is contacted with the *Result* of the goal model evaluation for the specified *Rule*. The *Result* is of type EvaluationResult, hence the result from the evaluate action can be used. The resulting ErrorCode is set to OK if "ack" is received from the rule engine and to CANCEL if not.

«output: "Prompt" Result» – The keyword *output* indicates that the user is shown the *Prompt* and the reasoning *Result* from the evaluate action. The *Result* is of type EvaluationResult. *Result* is optional. If *Result* is not specified, the output action is typically used for warning or error messages.

«deactivated: Rule» – Finally, the keyword *deactivated* indicates that the rule engine is contacted and informed that goal-oriented reasoning has been deactivated for the *Rule*. The resulting ErrorCode is set to OK if "ack" is received from the rule engine and to CANCEL if not.

For the reaserch presented in this thesis, the jUCMNav tool [50] has been extended with a

proof of concept implementation of the URN profile for CARGO [15]. The extension is capable of executing workflows tagged with actions from the CARGO profile, demonstrating that (i) run-time interaction between the CARGO Manager, the rule engine, and the goal engine based on Architectural Alternative C is feasible, (ii) the workflow of a context-aware system can be specified with URN workflow models, and (iii) rule-based reasoning can be complemented with goal-oriented reasoning based on URN goal models. The thesis do not advocate to model all contextual information in a goal model, but rather focus on those areas that can benefit the most from goal-oriented reasoning and leave other areas to established rule-based reasoning techniques. As soon as more than one stakeholder and their vague, qualitative goals need to be considered; goal-oriented reasoning is likely a more appropriate choice than rule-based reasoning. The number of potential solutions, however, is irrelevant for the reasoning choice

6 CARGO Food service application for hospital case study

The example used for the CARGO case study revolves around a food service application for a hospital that chooses appropriate meals for all patients based on contextual information about patients and meals. A patient's allergies and scheduled lab tests provide clear criteria for the inclusion or exclusion of a particular meal for a patient on a particular day that can be expressed with filtering rules for the rule engine. On the other hand, the meals' nutritional value, cost, and preparation time are more difficult to evaluate because three stakeholders (i.e., Patient, HospitalKitchen, HospitalAdministrator) with differing objectives and intentions have to be taken into account. The *nutritional value* is of interest to the patient (and the doctor but this is not considered further by this example), the *cost* is taken into account by the hospital administration and also the patient (who does not want to pay hospital user fees or higher insurance fees), and finally the *preparation time* is a concern of the hospital kitchen.

6.1 Goal Model

The goal model, presented in Figure 6.1, indicates that the hospital kitchen wants to maximize employee retention and can achieve that by either reducing the amount of time it takes to prepare meals or by providing more resources. The latter, however, has an impact on the overall cost goal of the hospital administration. In general, the hospital administration minimizes cost by minimizing on-going and infrastructure cost. Finally, a patient wants to get

better as soon as possible while not paying any user fees. The goal model includes indicators (⬭ Preparation Time, Cost, and Nutritional Value) that further quantify some of the goals of the three stakeholders. These indicators describe the contextual information of a meal relevant in this situation. The goal model in Figure 6.1 also shows the result of an evaluation with color-coding and satisfaction values. The contextual values of a particular meal determine the initial satisfaction values of the indicators (indicated by a *), which are then propagated in the goal model resulting in an assessment of the satisfaction of the three stakeholders (-20, 41, and 100, in this case). Based on the stakeholder satisfactions, an overall satisfaction value for the whole model is calculated by the evaluation algorithm (53 in this case). A different meal option would have different initial satisfaction values and hence would yield a different result for each stakeholder and consequently the whole model. The overall satisfaction value is then used to rank the meal options in the goal-oriented approach, resulting in a more appropriate solution than a meal option being picked randomly by the rule-based approach.

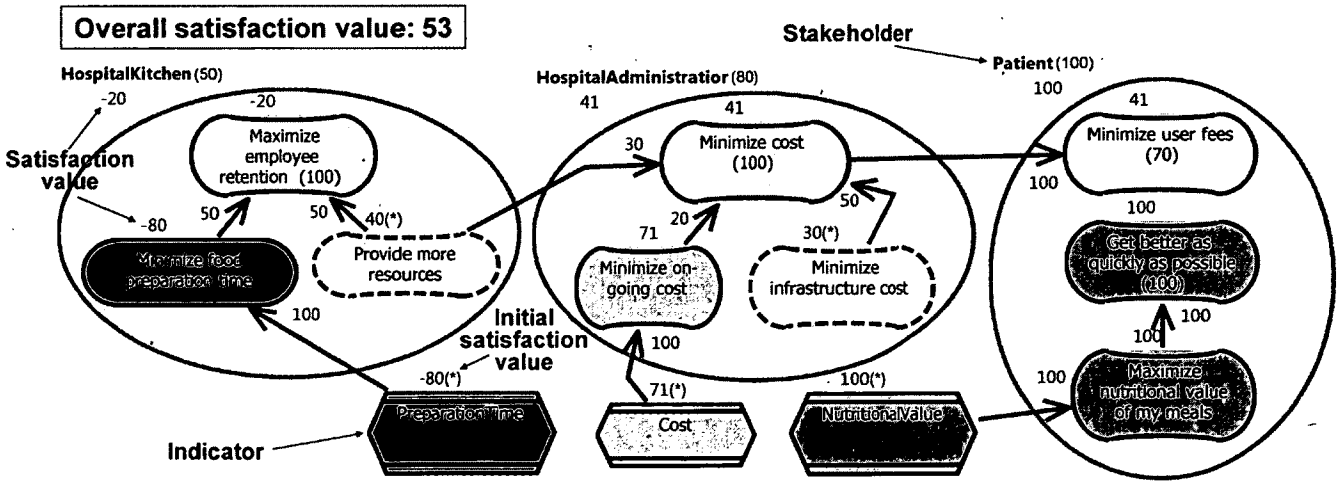

Figure 6.1 Goal model for food service application

6.2 CARGO scenario model

An example of a scenario model specified with CARGO and describing the food service application introduced in Chapter 5 is shown in Figure 6.2. The food service application interacts with the user, the rule engine, and the goal engine.

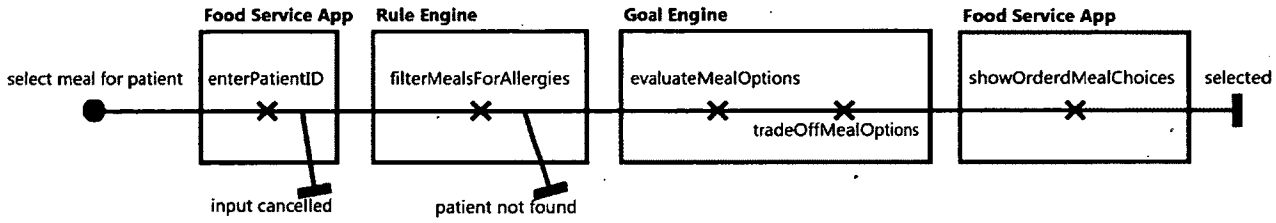

Figure 6.2 Scenario model for food service application

The scenario is consisted of six steps. In step 1, scenario is started. Figure 6.3 shows URL start scenario tool.

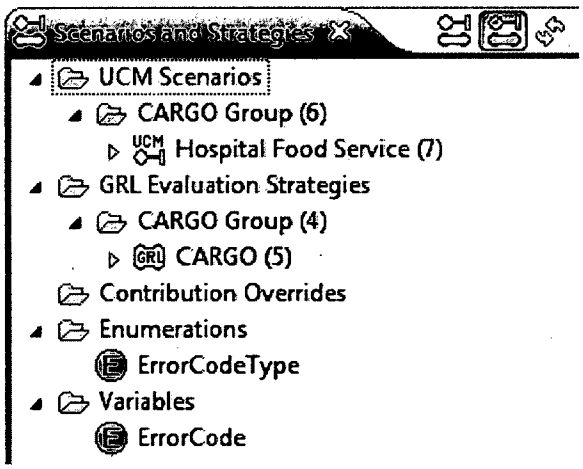

Figure 6.3 Start of the scenario

In step 2, the URN tools open input widows where patient Id value is required. Figure 6.4 shows Food Service Application input dialog that expects patient Id value to be entered. In this step the input can be cancelled which will terminate the scenario execution.

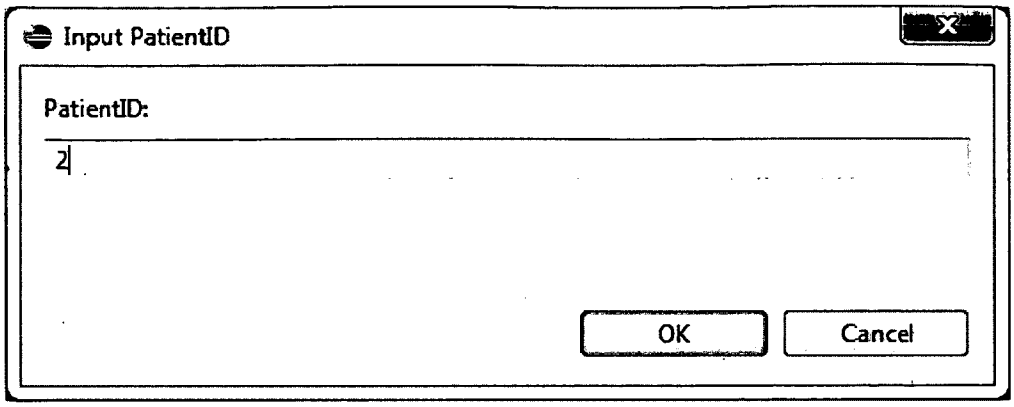

Figure 6.4 Food Service Application Input dialog

In step 3, the Rule Engine will be invoked to provide a filtered list of meals for the patient with the patient Id provided in the step3. If patient with provided patient Id is not found in the system database, this will stop further scenario execution.

In step 4, the Goal Engine will be invoked, which will evaluate meal options based on three indicators: Cost, Nutritional Value and Preparation Time.

In step 5, evaluation results will be presented as an ordered list of meals.

In step 6, output results presented. Figure 6.5 shows output results.

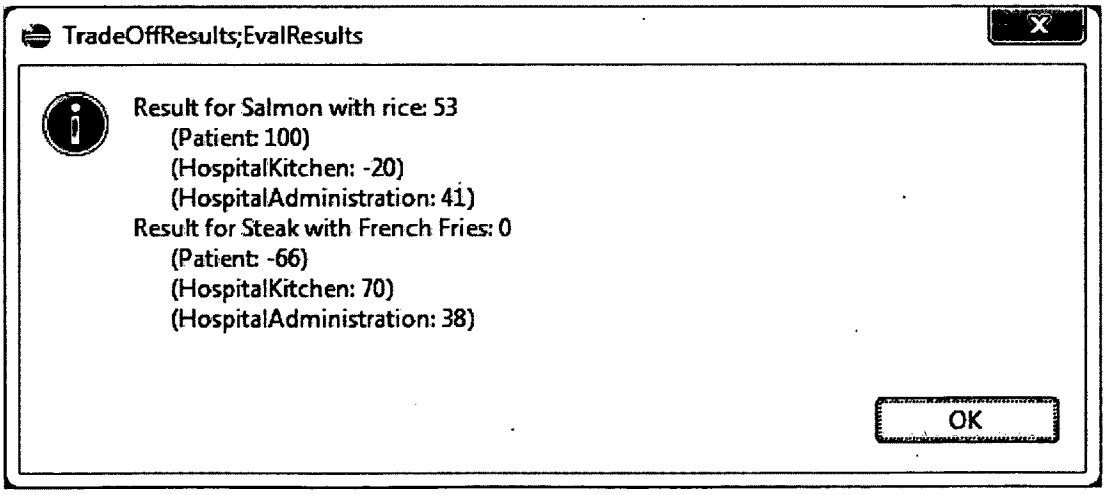

Figure 6.5 Output results

6.3 CARGO URN profile – food service example

An example of a workflow model specified with CARGO and describing the food service application introduced in Chapter 5 is shown in Figure 6.6.

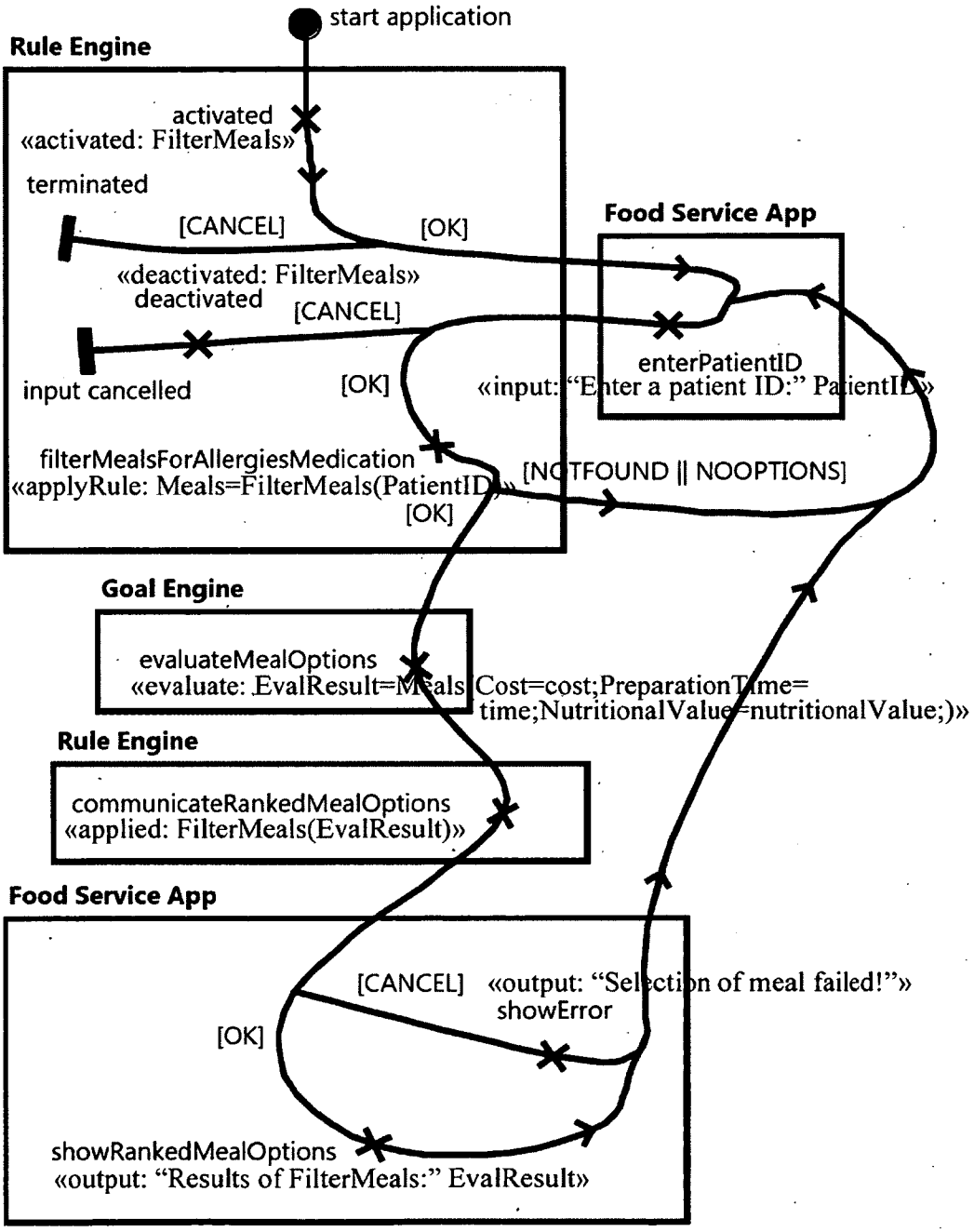

Figure 6.6 CARGO workflow model

As already mentioned in Chapter 5 CARGO requires specific extensions to URN, which are

defined in a *URN profile* for CARGO. The URN profile constitutes a domain-specific language that provides the ability to specify actions in a URN workflow model required by a context-aware system, i.e., these actions correspond to the messages in Figure 5.5.3. The actions cover activation and deactivation, input, application of a rule, evaluation of a goal model including trade-off analysis, and output of evaluation results will be further analyzed and explained on the Food Service example.

«activated: Rule» – The keyword *activated* indicates that the rule engine is contacted and informed that goal-oriented reasoning has been activated for the *Rule*. The resulting ErrorCode is set to OK if "ack" is received from the rule engine and to CANCEL if not.

In the workflow, the step *activated «activated: FilterMeals»* happens right after starting the application. The rule engine now knows that goal reasoning for the FilterMeals rule is available. If there is an error during activation, the rule engine returns *CANCEL* instead of *OK* and the application terminates.

«input: "Prompt" Variable» – The keyword *input* indicates that a dialogue with the specified *Prompt* is presented to the user. An input of type String is stored in the *Variable*. The variable is optional. If it is not specified, the user is presented the prompt and only allowed to select OK or CANCEL. In any case, the resulting ErrorCode is set to OK if the user clicks on OK and to CANCEL if the user clicks on CANCEL.

In the workflow, the step *enterPatientID «input: "Enter a patient ID:" PatientID»* solicits an input from the user. If the user enters a patient ID and clicks *OK*, the workflow continues. If the input is cancelled, then goal reasoning is deactivated (see last action in this section) and the application ends.

«applyRule: Result=Rule(Parameter)» – The keyword *applyRule* indicates that the rule engine is contacted and the specified *Rule* is applied. As a *Parameter*, a variable of type String is optionally specified. If a parameter is specified, then the list of options is gathered by the rule engine for the entity identified by the parameter. For example in Figure 6, the result of the input, i.e., PatientID, is used again in the applyRule action of filterMealsForAllergiesMedication. If no parameter is specified, then the next list of options in the queue of the rule engine is gathered by the rule engine. The *Result* of this action is a variable of type EvaluationOptions. This type contains a String for an identifier and a Set<EvaluationOption>. For each EvalutionOption, one or more name/value pairs can be specified that correspond to an attribute of the entity to be evaluated and the value of this attribute. For example, when evaluating a meal option for patient #7, then the identifier is "7" and cost/5 means that the cost of the meal option is $5. The resulting ErrorCode is set to NOTFOUND if the identifier null is returned by the rule engine, to NOOPTIONS if the identifier is specified but Set<EvaluationOption> is empty, and to OK if the identifier is specified and Set<EvaluationOption> is not empty.

In the workflow, the step *filterMealsForAllergiesMedication «applyRule: Meals=FilterMeals(PatientID)»* requests the list of meal options from the rule engine for the patient with the patient ID entered earlier. If the rule engine can provide a proper list in return (*OK*), the workflow continues. In the case of *NOTFOUND* and *NOOPTIONS*, the workflow loops back to the input of the patient ID.

«evaluate: Result=Data(Indicator=Attribute;)» – The keyword *evaluate* indicates that the goal model is to be evaluated. The *Data* variable is an input of type EvaluationOptions, hence the result of the applyRule action can be used. One or more mappings between a goal model

element (i.e., *Indicator*) and an *Attribute* of an EvaluationOption are specified. These mappings are used to initialize the goal model before evaluating it. For example, Cost=cost means that the indicator Cost in the goal model (see Figure 3) is initialized with the value of the cost attribute of an EvaluationOption. Note that the indicator converts the dollar value (e.g., $5) during initialization into a goal model value in the range from -100 to +100 (e.g., 71). Indicators (⟾) specify what context-related information is required for goal-based reasoning. The variable *Result* is of type EvaluationResult, contains a String for an identifier, and captures for each EvaluationOption the satisfaction values of the stakeholders in the goal model (e.g., HospitalKitchen/-20, HospitalAdminstration/41, and Patient/100) and the overall satisfaction value of the goal model (e.g., 53). The evaluation options in *Result* are sorted in descending order by the overall satisfaction value.

In the workflow, the step *evaluateMealOptions «evaluate: EvalResult = Meals(Cost=cost;PreparationTime=time;Nutritional-Value=nutritionalValue;)»* sets the initial satisfaction values of Cost, PreparationTime, and NutritionalValue in the goal model to the values of attributes cost, time, and nutritionalValue, respectively, for each meal in Meals (the list of meal options returned by filterMealsForAllergiesMedication). The results of the evaluation are recorded, i.e., the satisfaction values of the stakeholders as well as the overall satisfaction value.

«applied: Rule(Result)» – The keyword *applied* indicates that the rule engine is contacted with the *Result* of the goal model evaluation for the specified *Rule*. The *Result* is of type EvaluationResult, hence the result from the evaluate action can be used. The resulting ErrorCode is set to OK if "ack" is received from the rule engine and to CANCEL if not.

In the workflow, the step *communicateRankedMealOptions «applied:*

FilterMeals(EvalResult)» reports the results of the evaluation to the rule engine. If there is a problem with the results, an error message is shown to the user (see next action). If there is no problem, the results are shown to the user (see next action). In both cases, the workflow loops back to the input of the patient ID after the output.

«output: "Prompt" Result» – The keyword *output* indicates that the user is shown the *Prompt* and the reasoning *Result* from the evaluate action. The *Result* is of type EvaluationResult. *Result* is optional. If *Result* is not specified, the output action is typically used for warning or error messages.

In the workflow, the steps *showError «output: "Selection of meal failed!"»* as well as *showRankedMealOptions «output: "Results of FilterMeals:" EvalResult»* reports an error message or the results of the evaluation to the user, respectively.

«deactivated: Rule» – Finally, the keyword *deactivated* indicates that the rule engine is contacted and informed that goal-oriented reasoning has been deactivated for the *Rule*. The resulting ErrorCode is set to OK if "ack" is received from the rule engine and to CANCEL if not.

In the workflow, this action is used in the step *deactivated «deactivated: FilterMeals»*.

The jUCMNav tool [50] has been extended with a proof of concept implementation of the URN profile for CARGO [15]. The extension is capable of executing workflows tagged with actions from the CARGO profile, demonstrating that (i) run-time interaction between the CARGO Manager, the rule engine, and the goal engine based on Architectural Alternative C is feasible, (ii) the workflow of a context-aware system can be specified with URN workflow models, and (iii) rule-based reasoning can be complemented with goal-oriented reasoning

based on URN goal models. The thesis does not advocate modeling all contextual information in a goal model, but rather focus on those areas that can benefit the most from goal-oriented reasoning and leave other areas to established rule-based reasoning techniques. As soon as more than one stakeholder and their vague, qualitative goals need to be considered, goal-oriented reasoning is likely a more appropriate choice than rule-based reasoning. The number of potential solutions, however, is irrelevant for the reasoning choice.

6.4

7 Conclusions and future work

This thesis describes a Domain Independent Context Awareness Framework designed to reuse open-source components and to provide features such as context reasoning and notification to other systems and clients. The framework is designed based on service oriented concepts and can be used for developing context aware applications for different domain. In addition, the original rule-based reasoning capabilities of the framework have been extended with goal-oriented reasoning, known as CARGO that provides more flexibility and configurability than commonly used logic-based reasoning, allowing for a more holistic assessment of the context as the goals of many stakeholders are taken into account.

CARGO enables modeling, simulation, and execution of context-aware systems as well as combined rule/goal-based reasoning about contextual information. CARGO extends the standard language URN with its own profile, i.e., a language enabling the specification of typical actions for a context-aware system. The actions clearly define an interface for rule engines and goal model evaluation from URN workflows. In future work, the plan is to investigate whether there is a need to support a larger set of actions. Furthermore, the interactive execution mode is also of interest to the standard URN traversal mechanism to support a more advanced simulation environment for all models, i.e., not limited to context-aware systems.

7.1 Future Work

There are several research directions for enhancing the proposed framework.-

- Incorporate a goal engine in the framework. Currently we are using the goal processing included in the JUCM nav tool for the goal model evaluation. In order to increase the performance of the reasoning step, it would be better to move the component in charge with goal model evaluation to the domain-independent context-awareness framework.

- Adding ontology reasoning to the framework. Currently only rule-based reasoning is covered by the proposed framework. For some well-established domains, ontology reasoning may provide certain advantages: reuse of well defined specification, generality, understandability, performance.

- Performance evaluation. It would be useful to estimate the overheads introduced by the framework for reasoning in order to find out if they can satisfy real-time systems performance requirements. Also, investigate techniques for optimizing the rule configuration process in order to provide a faster resolution time.

A. Appendix – REST web service APIs

Patient APIs

Get Patents API returns the list of all registered Patients in the system.

Name of the API	getPatients
URL	/patients
Return Value	List of Patients
Argument	None
HTTP method	GET

Table A.1 Get Patients API

Get Patient API returns the Patient if patient with the same patientId exist, or null.

Name of the API	getPatient
URL	/patients/{patientId}
Return Value	Patient is patient exist or null otherwise
Argument	String – patientId
HTTP method	GET

Table A.2 Get Patient API

Get All Patients in Department API returns the list of all patients in department.

Name of the API	getAllPatientsInDepartmant
URL	/patients/{departmantId}
Return Value	List of Patients
Argument	String – departmantId
HTTP method	GET

Table A.3 Get All Patients in Department API

Get All Active Patients in Department API returns the list of all active patients in department.

Name of the API	getAllActivePatientsInDepartmant
URL	/patients/{departmantId}/active
Return Value	List of Patients
Argument	String – departmantId
HTTP method	GET

Table A.4 Get All Active Patients in Department API

Create a new Patient API create a new patient record in the database, returns 200 OK success otherwise error message.

Name of the API	createPatient
URL	/patients
Return Value	Void , 200 OK
Argument	Patient
HTTP method	PUT

Table A.5 Create a new Patient API

Update Patient API updates the patient record in the database. It returns 200 OK success otherwise error message.

Name of the API	updatePatient
URL	/patients/{hospitaId}
Return Value	200 OK
Argument	Patient and String –hospitalID
HTTP method	POST

Table A.6 Update Patient API

Delete Patient API deletes patient from the database, returns OK 200.

Name of the API	deletePatient
URL	/patients/{hospitalID}
Return Value	200 OK
Argument	String – hpsitalID
HTTP method	GET

Table A.7 Delete Patient API

Get Patient Info API returns the patient info.

Name of the API	getPatientInfo
URL	/patients/{hospitalID}/patientInfo
Return Value	Patient Info
Argument	String – hospitalID
HTTP method	GET

Table A.8 Get Patient Info API

Update Patient Info API returns the patient info.

Name of the API	updatePatientInfo
URL	/patients/{hospitalID}/patientInfo
Return Value	Patient Info
Argument	String – hospitalID, PatientInof
HTTP method	POST

Table A.9 Update Patient Info API

Get Patient 's Sensors API returns the list of Patient's Sensors.

Name of the API	getPatientSensors
URL	/patients/{hospitalID}/sensors
Return Value	Sensors
Argument	String – hospitalID
HTTP method	GET

Table A.10 Get Patient's Sensors API

Add Patient's Sensor API adds the sensor to the database.

Name of the API	addPatientSensor
URL	/patients/{hospitalID}/sensors/{sensorId}
Return Value	200 OK
Argument	String – hospitalID, String – sensorID, Sensor
HTTP method	POST

Table A.11 Add Patient's Sensor API

Delete Patient's Sensor API deletes patient's sensor from the database.

Name of the API	deletePateintSensor
URL	/patients/{hospitalID}/sensors/{sensorId}
Return Value	OK 200

Argument	String – hospitalID, String sensorID
HTTP method	DELETE

Table A.12 Delete Patient's Sensor API

Medical Staff APIs

Get All Medical Staff API returns the list of All Medical Staff in the system.

Name of the API	getAllMedicalStaff
URL	/medicalstaff
Return Value	List of Medical Staff
Argument	None
HTTP method	GET

Table A.13 Get All Medical Staff API

Get All Active Doctors in a Department API returns the list of all active doctors in the department.

Name of the API	getAllMedicalStaff
URL	/medicalstaff/doctors/{departmantID}
Return Value	List of Medical Staff
Argument	None
HTTP method	GET

Table A.14 Get All Active Doctors in a Department

Get All Active Nurses in a Department API returns the list of all active nurses in a department.

Name of the API	getAllMedicalStaff
URL	/medicalstaff/nurses/{departmantID}
Return Value	List of Medical Staff
Argument	None
HTTP method	GET

Table A.15 Get All Active Nurses in a Department API

Get a Medical Staff API returns the Medical Staff for the given medical staff Id.

Name of the API	getAllMedicalStaff
URL	/medicalstaff/{medicalstaffId}
Return Value	The Medical Staff
Argument	Medical staff id
HTTP method	GET

Table A.16 Get a Medical Staff API

Create a Medical Staff API creates a medical staff in the system.

Name of the API	getAllMedicalStaff
URL	/medicalstaff/{medicalstaffId}
Return Value	200 OK
Argument	Medical staff id, Medical Staff
HTTP method	PUT

Table A.17 Create a Medical Staff API

Update the Medical Staff API updates the Medical Staff for the given medical staff ID in the system.

Name of the API	getAllMedicalStaff
URL	/medicalstaff/{medicalstaffId}
Return Value	200 OK
Argument	Medical staff id, Meidcal Staff
HTTP method	POST

Table A.18 Update the Medical Staff API

Delete the Medical Staff API deletes the Medical Staff for the given medical staff ID in the system.

Name of the API	getAllMedicalStaff
URL	/medicalstaff/{medicalstaffId}
Return Value	200 OK
Argument	Medical staff id
HTTP method	DELETE

Table A.19 Delete the Medical Staff API

Department APIs

Get All Departments API returns the list of departments in the system.

Name of the API	getDepartments
URL	/departmants/
Return Value	List of Departments
Argument	None
HTTP method	GET

Table A.20 Get All Departments API

Get a Department API returns the department for the given department id.

Name of the API	getDepartment
URL	/departmants/{departmentId}
Return Value	Department
Argument	String – department Id
HTTP method	GET

Table A.21 Get a Department API

Create a Department API creates the department with the given department id.

Name of the API	createDepartment
URL	/departmants/{departmentId}
Return Value	200 OK
Argument	String – department Id, Department
HTTP method	PUT

Table A.22 Create a Department API

Update a Department API updates the department with the given department id.

Name of the API	updateDepartment
URL	/departmants/{departmentId}
Return Value	200 OK
Argument	String – department Id, Department
HTTP method	POST

Table A.23 Update a Department API

Delete a Department API deletes the department with the given department id.

Name of the API	deleteDepartment
URL	/departmants/{departmentId}
Return Value	200 OK

Argument	String – department Id
HTTP method	DELETE

Table A.24 Delete a Department API

Sensors APIs

Get All Sensors API returns the list of sensors in the system.

Name of the API	getSensors
URL	/sensors/
Return Value	List of Sensors
Argument	None
HTTP method	GET

Table A.25 Get All Sensors API

Get a Sensor API returns the sensor for the given sensor id in the system.

Name of the API	getSensor
URL	/sensorss/{sensorsId}
Return Value	Sensor
Argument	String – sensor Id
HTTP method	GET

Table A.26 Get Sensor API

Create a Sensor API creates the sensor with the given sensor id.

Name of the API	createSensor
URL	/sensors/{sensorId}
Return Value	200 OK
Argument	String – sensor Id, Sensor
HTTP method	PUT

Table A.27 Create a Sensor API

Update a Sensor API updates the sensor with the given sensor id.

Name of the API	updateSensor
URL	/sensors/{sensorId}

Return Value	200 OK
Argument	String – sensor Id, Sensor
HTTP method	POST

Table A.28 Update Sensor API

Delete a Sensor API deletes the sensor with the given sensor id

Name of the API	**deleteSensor**
URL	/sensors/{sensorId}
Return Value	200 OK
Argument	String – sensor Id
HTTP method	DELETE

Table A.29 Delete a Sensor API

Devices APIs

Get All Devices API returns the list of devices in the system.

Name of the API	**getDevices**
URL	/devices/
Return Value	List of Devices
Argument	None
HTTP method	GET

Table A.30 Get All Device API

Get a Device API returns the devices for the given device id .

Name of the API	**getDevice**
URL	/devices/{deviceId}
Return Value	Device
Argument	String – device Id
HTTP method	GET

Table A.31 Get a Device API

Create a Device API creates the device with the given device id.

Name of the API	**createDevice**

URL	/devices/{deviceId}
Return Value	200 OK
Argument	String – device Id, Device
HTTP method	PUT

Table A.32 Create a Device API

Update a Device API updates the device with the given device id in the system.

Name of the API	updateDevice
URL	/devices/{deviceId}
Return Value	200 OK
Argument	String – device Id, Device
HTTP method	POST

Table A.33 Update a Device API

Delete a Device API deletes the device with the given device id in the system.

Name of the API	deleteDevice
URL	/devices/{deviceId}
Return Value	200 OK
Argument	String – device Id
HTTP method	DELETE

Table A.34 Delete a Device API

References

[1] Erin Yu, Ryan Kealey, Mark Chignell, Joanna Ng, and Jimmy Lo, (2010), "Smarter Healthcare: An Emergency Physician View of the Problem", *The Smart Internet, Springer Lecture Notes in Computer Science, Vol. 6400/2010, pp. 9-26.*

[2] G. Ross Baker & Peter Norton, (2004), "Patient Safety and Healthcare Error in the Canadian Healthcare System: A Systematic Review and Analysis of Leading Practices in Canada with Reference to Key Initiatives Elsewhere", *Health Canada, http://www.hc-sc.gc.ca/hcs-sss/pubs/qual/2001-patient-securit-rev-exam/index-eng.php* (accessed November 2012).

[3] Mathias Baldaulf, Schahram Dustdar and Florian Rosenberg , (2007), "A survey on context-aware systems", *Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, 2007.*

[4] Hung Keng Pung, Tao Gu, Wenwei Xue, Paulito P. Palmes, Jian Zhu, Wen Long Ng, Chee Weng Tang, Nguyen Hoang Chung, "Context Aware Middleware for Pervasive Elderly Homecare", *Selected Areas in Communications, IEEE Journal, Volume 27, Issue 4, pp 510-524*

[5] Aamna Saeed, Tabinda Waheed , (2010), "An Extensive Survey of Context – Aware Middleware Architectures", *Electro/Information Technology (EIT), 2010 IEEE, Conference Publication, pp 1-6*

[6] Eunhoe Kim, Jaeyoung Choi, (2007), "A context-Awareness Middleware Based on Service-Oriented Architecture", *Ubiquitous intelligence and Computing, Lecture Notes in Computer Science, 2007, Volume 4611/2007, 953-962, DOI:1007/978-3-540-73549-6-93*

[7] Christina Bolchini, Carlo A. Curino, Elisa Quintareli, Fable A.Schreiber, Letizia Tanca, (2007), "A Data-oriented Survay of Context Models", *SIGMOD Record, December 2007 (Vol. 36, No. 4)*

[8] Herma van Kranenburg, Mortaza S. Bargh, Sorin Jacob and Arjan Peddemors , (2006), "A Context Management Framework for Supporting Context-Aware Distributed Applications", *Communications Magazine, IEEE, Volume 44, Issue 8, pp 67-74*

[9] *Andrew S. Tanenbaum, Maarten Van Steen, (2006),* "Distributed Systems Principles and Paradigms", *book, second edition*

[10] Natalia F. Noy, and Deborah L. McGuinnes, (2001), "Ontology Development 101: A Guide for Creating Your First Ontology", *Stanford University, Stanford, CA, 94305*

[11] Roy Thomas Feilding, (2000), "Architectural Styles and the Design of Network-based Software Architectures", *University of California, Irvine, PhD dissertation 2000.*

[12] Rick Ho "Common REST design pattern", http://architects.dzone.com/news/common-rest-design-pattern", *Architect design* (last time accessed October 2012)

[13] D. Amyot and G. Mussbacher, "User Requirements Notation: The First Ten Years, The Next Ten Years", *Journal of Software (JSW), Vol. 6, No. 5, 2011, pp. 747–768, doi:10.4304/jsw.6.5.747-768.*

[14] D. Amyot, S.Ghanavati, J. Horkoff, G.Mussbacher, L.Peyton, and E.Yu, (2010), "Evaluating Goal Models within the Goal-oriented Requirment Language", *International Journal of Intelligent Systems (IJIS), Wiley, 25(8):841-877.*

[15] Vrbaski, M., Petriu, D., and Amyot, A. (2012) Tool Support for Combined Rule-Based and Goal Based Reasoning in Context-Aware Systems. *20th IEEE Intl. Requirements. Engineering Conf.* (Chicago, USA, Sep. 2012). RE'12. (Best poster/demo award).

[16] Vrbaski, M., Mussbacher, G., Petriu, D., and Amoyot, A. (2012) Goal model as Run Time Entities in Context-Aware Systems. *15th ACE/IEEEE Intl. MODELS* (Innsbruck, Austria, September 30).MRT'12

[17] Jean-Fran,cois Roy, Jason Kealey, and Daniel Amyot "Towards Integrated Tool Support for the User Requirements Notation", http://jucmnav.softwareengineering.ca/ucm/pub/UCM/VirLibSam06jUCMNav/SAM06-Roy-Kealey-Amyot.pdf (last time accessed on October 2012)

[18] Angel Machin, Curro Dominquez, (2008), "Integration mobile services and content with the Internet", *WWW 2008, April 21--25, 2008, Beijing, China*

[19] Wael Labidi, Jean-Ferdy Susini, Pierre Paradinas, and Michael Setton, (2008)," XMPP based Health Care Integrated Ambient Systems Middleware", *Developing Ambient Intelligence, 2008, Part 1, 92-102, DOI:10.1007/978-2-787-78544-3_9*

[20] Ivana Podnar, Manfred Hauswirth, Mehdi Jazayeri , (2002), "Mobile Push: Delivering Content to Mobile Users", *Distributed Computing Systems Workshops, 2002, Proceedings, 22nd International conference*

[21] Gunther Pospischil, Johannes Stadler, Igor Miladinovic , (2010), "A Location-based Push Architecture using SIP", *XP-002235856, International Symposium Wireless Personal Multimedia Communications, September 2001, pp 295-300*

[22] Davide Tosi, (2003), "An advanced architecture for push services", *Fourth International Conference on Web Information Systems Engineering, Workshops (WISEW'03), 2003 wisew, pp 193-200*

[23] Peter Saint-Andre, Kevin Smith, Remko Tronçon,(2009) "XMPP: The Definitive Guide Building Real-Time Applications with Jabber Technologies", *O'Reilly book.*

[24] RDF - Resource Description Framework, http://www.w3.org/TR/rdf-syntax-grammar/ (last time accessed October 2012)

[25] MMS- Multimedia Messaging Service, http://www.openmobilealliance.org/Technical/release_program/mms_v1_3.aspx (last time accessed on October 2012)

[26] WAP -Wireless Application Protocol, http://en.wikipedia.org/wiki/Wireless_Application_Protocol (last time accessed on October 2012)

[27] ANS - IPhone push notification, http://support.apple.com/kb/HT3576 (last time accessed on October 2012)

[28] C2DM, Android Cloud Messaging Framework http://code.google.com/android/c2dm/index.html (last time accessed on October 2012)

[29] *Kindan Singh's Blog* ,"SIP vs XMPP or SIP and XMPP?" http://p2p-sip.blogspot.com/2009/11/sip-vs-xmpp-or-sip-and-xmpp.html (last time accessed on October 2012)

[30] SIP Center http://www.sipcenter.com/sip.nsf/html/Architecture (last time accessed on October 2012)

[31] XMPP org http://xmpp.org/about/ (last time accessed on October 2012)

[32] *XMPP over BOSH* http://xmpp.org/extensions/xep-0206.html (last time accessed on October 2012)

[33] Mobicents http://www.mobicents.org/products_sip_servlets.html

[34] OPENFIRE XMPP server - http://www.igniterealtime.org/projects/openfire/ (last time accessed on October 2012)

[35] Smack XMPP java based APIs http://www.igniterealtime.org/projects/smack (last time accessed on October 2012)

[36] Business Rule Manifesto, http://www.businessrulesgroup.org/brmanifesto.htm (last time accessed on October 2012)

[37] JBOSS Application server overview, http://www.jboss.org/jbossas (last time accessed

on October 2012)

[38] MySQL Rational Data Base, http://www.mysql.com/why-mysql/ (last time accessed on October 2012)

[39] Windows based interface for MySQL, http://www.heidisql.com/ (last time accessed on October 2012)

[40] Hibernate relational persistence, http://www.hibernate.org/about (last time accessed on October 2012)

[41] Drools – Business logic and interactional platform, http://www.jboss.org/drools (last time accessed on October 2012)

[42] Young, C. "Microsoft's Rule Engine Scalability Results – A comparison with DROOLS and JESS", http://geekswithblogs.net/cyoung/articles/54022.aspx, (last time accessed on October 2012)

[43] Published by Java-source .net, "Open Source Rule Engines in Java", http://java-source.net/open-source/rule-engines ,(last time accessed on October 2012)

[44] RESTEasy – JBOSS REST web service library, http://www.jboss.org/resteasy (last time accessed on October 2012)

[45] "JBOSS Admin tutorial - database integration" http://marakana.com/bookshelf/jboss_admin_tutorial/database_integration.html (last time accessed on October 2012)

[46] JBOSS and DROOLS integration, http://docs.jboss.org/drools/release/5.4.0.CR1/droolsjbpm-integration-docs/html/ch03.html, (last time accessed on October 2012)

[47] DROOLS and JBoss Integration, http://docs.jboss.org/drools/release/5.2.0.Final/droolsjbpm-introduction-docs/html/ch03.html, (last time accessed on October 2012)

[48] RESTEasy and JBOSS installation and configuration, http://docs.jboss.org/resteasy/2.0.0.GA/userguide/html/Installation_Configuration.html, (last time accessed on October 2012)

[49] 3GPP specification, http://www.3gpp.org/specifications, (last time accessed on October 2012)

[50] jUCMNav website, http://softwareengineering.ca/jucmnav, (last time accessed on October 2012)

[51] Use Case Maps Quick Tutorial, http://cserg.site.uottawa.ca/ucm/pub/UCM/VirLibTutorial99/UCMtutorial.pdf, (last time accessed on October 2012)

[52] Xinyian Feng, Jianjing Shen,Jing Fan, "REST: An Alternative to RCP web Service architecture", *First International conference on Future Information Networks 2009*

[53] Charles L. Forgy, "Rete: A Fast Algorithm for Many Pattern/ Many Object Pattern Match Problem", *Artificial Intelligence, Nort Holland 1982.*

[54] Don Batory, "The LEAPS Algorithms", *Department of Computer Science, The university of Texas, Austin Texas 78712.*

[55] Amoyt D., "Use Case Maps as a Feature Notation", available from UCM Virtual library http://www.UseCaseMaps.org/pub/ (last time accessed October 2012)