



Master Thesis

Patterns in Domain Models

**A Methodology and its Application in the
Healthcare Management Domain**

Technische Universität Berlin

Institute of Database Systems and Information Management (DIMA)

In collaboration with

Universitat Politècnica de Catalunya

Faculty of Informatics of Barcelona (FIB)

Supervisors

Dr. Ralf-Detlef Kutsche

Dipl.-Inf. Henning Agt

Thesis by

Silvia Teresa Sandy-Martinez

Matr. Number: 341858

Head of Department

Prof. Dr. rer. nat. Volker Markl

Berlin, 22 July 2013

Declaration of authorship

Last name: Sandy-Martinez

First name: Silvia Teresa

I declare that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other University.

Formulations and ideas taken from other sources are cited as such. This work has not been published.

Berlin, 22 July 2013

Silvia T. Sandy-Martinez

Abstract

This master thesis was conceived in the context of domain analysis and reuse of domain knowledge. The main objective of this project was to provide a methodology for building domain patterns catalogs. With this motivation in mind, we developed a method for extracting patterns from models that covered the healthcare management domain. The resulting artifact enables software designers to obtain models of high quality through the reuse of abstracted knowledge from the domain.

In the process of method development, a number of approaches from the literature were studied and analyzed. The methodology proposed here was based on this study. In particular, the steps of our methodology were adapted from one or more existing methodologies with our purposes in mind. To make the developed methodology accessible, this work provides a step by step explanation of the method as well as accompanying examples. Furthermore, a metamodel to support the catalog elements representation and the classification schema was developed.

In order to validate the method presented here, we analyzed the healthcare management domain: we discovered and gathered recurring patterns from domain models collected from a number of sources ranging from research to industry. The resulting domain candidate patterns catalog is provided as an output of this work as well as the Eclipse projects that implement the catalog.

Abstract (in German)

Diese Masterarbeit wurde im Kontext von Domänenanalyse und der Wiederbenutzung von Domänenwissen erarbeitet. Das vorrangige Ziel dieses Projektes war die Entwicklung einer Vorgehensweise für die Ausarbeitung von Domain-Patterns Katalogen. Mit dieser Motivation im Hinterkopf haben wir eine Methode zur Extraktion von Patterns aus Modellen des Gesundheitsmanagementbereichs entwickelt.

Als Teil der Entwicklung der Methode wurden verschiedene Ansätze aus der Literatur analysiert. Die hier vorgeschlagene Methode basiert auf dieser Analyse. Eine oder mehrere existierende Methoden wurden, mit Hinblick auf das Ziel dieser Arbeit, angepasst und zu Schritten unserer Vorgehensweise zusammengefasst. Diese Arbeit stellt eine schrittweise Erklärung sowie unterstützende Beispiele zur Verfügung, um die Nutzung dieser Methode zu erleichtern. Des Weiteren wurde ein Metamodell entwickelt, um die Repräsentation der Elemente des Katalogs sowie das Klassifikationsschema zu unterstützen.

Um die hier vorgestellte Methode zu validieren, haben wir den Gesundheitsmanagementbereich analysiert. Hierzu wurden wiederkehrende Patterns in Domänenmodellen entdeckt und gesammelt. Die Modelle wurden aus einer Reihe von Quellen aus Wissenschaft und Industrie zusammengetragen. Der resultierende Domain-Candidate-Patterns Katalog wird hier als Ergebnis dieser Arbeit zusammen mit den Eclipse Projekten, die den Katalog implementieren, bereitgestellt.

Dedication

To my families, the ones that are far away and the ones that are closer, for being my source of strength, courage and wisdom.

To my friends, for being my family and for not letting me abandon my dreams.

To my best half, for being my rock, for everything and more that he has given to me in the past years.

To Valentina, my niece and my shortest best friend.

To the fear because without it, it wouldn't be possible to learn to get up and go on.

Acknowledgments

A special thanks and dedication of this work to all the excellent people that I met through my career that made a profound impact in the rest of my life:

To my thesis supervisors, Dr. Ralf-Detlef Kutsche and (soon to be Dr.) Henning Agt, for their constant support, the advices and for making this experience such a nice environment that we can share not only in the academic life.

To my coworkers from room EN702, for the funny moments, the lunches and for the great family they turned up to become each of them in a singular way.

To the BIZWARE project for all the skills I earned by working there.

Table of Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation	2
1.3	Objectives.....	2
1.4	Structure of this work.....	3
2	Modeling Foundations.....	4
2.1	Model	4
2.2	Modeling	6
2.3	The MOF Hierarchy	6
2.4	Model Driven Development.....	8
2.5	Domain	8
2.6	Prieto-Diaz's Domain Analysis.....	9
2.7	Feature-Oriented Domain Analysis (FODA)	11
2.8	Domain Specific Languages.....	13
2.9	Eclipse EMF.....	15
3	Patterns Foundations	17
3.1	Pattern	17
3.2	Pattern collections	19
3.2.1	Pattern Language.....	19
3.2.2	Pattern Catalog	19
3.3	Patterns and Software Development	20
3.3.1	Background History	21
3.3.2	Patterns in software engineering	22
3.3.3	Patterns in domain engineering	23
3.4	Pattern catalogs used in this work	29
3.4.1	Data model Patterns conventions of thought.....	29
3.4.2	Analysis patterns – reusable object models.....	29

3.4.3	The Data Model Resource Book	30
3.4.4	Object-Oriented modeling and design.....	31
3.4.5	Pattern Languages of Program Design.....	31
3.4.6	Other smaller pattern catalogs.....	31
3.5	Metadata standards.....	31
3.5.1	Health Level Seven, HL7	32
3.5.2	Trial Item Manager, TIM	33
3.5.3	Street - Address Meta Data Standard	34
4	Methodology Framework.....	37
4.1	Building a catalog of Patterns	37
4.1.1	Definition and representation of a pattern.....	37
4.1.2	Patterns Classification	38
4.1.3	Method	40
4.2	Pattern Identification	43
4.2.1	Analysis Patterns	43
4.2.2	Patterns in Database Modeling.....	44
4.2.3	Patterns in object-oriented design	45
4.2.4	Composite patterns	46
4.2.5	Cross-Domain design	49
4.3	Classification of patterns.....	49
4.3.1	Criteria.....	50
4.3.2	Domain Patterns Classification Schema.....	52
4.4	Metamodel for a Domain Patterns Catalog	55
4.4.1	Elements representation	57
4.4.2	Pattern Representation.....	58
4.4.3	Classification Representation	60
4.4.4	Vocabulary	60
5	Implementation.....	61
5.1	Method	61

5.2	Step 1 - Analysis of the Domain	63
5.2.1	Artifacts of Step 1	63
5.2.2	Sources	63
5.2.3	Model Management.....	64
5.2.4	Results and conclusions	71
5.3	Step 2 - Extract elements present in the Models	73
5.3.1	Artifacts of Step 2	73
5.3.2	What is an element?	73
5.3.3	Element Handling.....	74
5.3.4	Results and comments	76
5.4	Step 3 - Semantic analysis and refining the elements	79
5.4.1	Artifacts of Step 3	79
5.4.2	Elements handling into Elements Union	80
5.4.3	Architecture of the Elements Union	82
5.4.4	Results and comments	82
5.5	Step 4 - Candidate patterns.....	84
5.5.1	Artifacts of Step 4	84
5.5.2	Candidate patterns selection criteria.....	84
5.5.3	Candidate patterns in the patterns catalog	91
5.5.4	Results and Comments	95
5.6	Step 5 - Patterns catalog	98
5.6.1	Artifacts of Step 5	98
5.6.2	Validation suggestions	98
5.6.3	Maintenance and expansion suggestions.....	99
6	Summary	100
6.1	Conclusions	100
6.2	Issues	101
6.3	Recommendations and Future Work.....	102
7	Literature	104

Appendix A.	Metamodel of the Domain Patterns Catalog	111
I.	UML metamodel	111
II.	Ecore metamodel	112
Appendix B.	Elements Union	113
I.	Entities.....	113
II.	Relationships	135
Appendix C.	Domain Candidate Patterns Catalog.....	149
I.	Catalog Organization.....	151
A.	Catalog's Classification Schema	151
B.	Pattern template description	153
II.	The Candidate Patterns.....	154
A.	Domain-Specific Patterns.....	154
1.	Healthcare Management Domain.....	154
a)	Patterns for Early Design	154
b)	Patterns for Intermediate Design.....	163
c)	Patterns for Advanced Design.....	170
B.	Cross-Domain Patterns.....	173
a)	Patterns for Early Design	173
b)	Patterns for Intermediate Design.....	174

Index of Figures

<i>Illustration 1 - Original and Model criterion according to Stachowiak (1973) [5]</i>	4
<i>Illustration 2 Hospital Organization Domain Model – Patient, Hospital, Staff – Operations, Administrative and Technical. [6]</i>	5
<i>Illustration 3: The MOF Hierarchy</i>	7
<i>Illustration 4 Context View of Domain Analysis [13]</i>	9
<i>Illustration 5 Top-down-Bottom-up Domain Analysis process [14]</i>	10
<i>Illustration 6: Participants in the domain analysis process [15]</i>	12
<i>Illustration 7: Relationship between DSL Components [18]</i>	14
<i>Illustration 8 Basic parts of a pattern [26]</i>	18
<i>Illustration 9 GoF Pattern Catalog [25]</i>	20
<i>Illustration 10: Software development based on domain engineering [34]</i>	25
<i>Illustration 11: Cross-domain reuse of problem solutions, analysis patterns [38]</i>	27
<i>Illustration 12: Method for analysis patterns abstraction [38]</i>	28
<i>Illustration 13: The elements and structure of the LOM conceptual data schema [53]</i>	32
<i>Illustration 14 HL7 Message Segments example</i>	33
<i>Illustration 15 Creatinine Trial Example [54]</i>	34
<i>Illustration 16 Basic Address Elements in the Street, Address Metadata Standard [56]</i>	35
<i>Illustration 17 Address information according to ANZLIC Metadata Profile [57]</i>	36
<i>Illustration 18 Wellington City Council's Address and contact info using ANZLIC Metadata</i>	36
<i>Illustration 19: Steps for the construction of a SRP Patterns Catalog [58]</i>	42
<i>Illustration 20: Semantic Analysis Pattern (SAPs) generation method [46]</i>	43
<i>Illustration 21: Example of EIP and RIP [60]</i>	44
<i>Illustration 22: Meta-model for EIP and RIP [60]</i>	45
<i>Illustration 23: Design Clusters based on Keywords [61]</i>	46
<i>Illustration 24: Role diagram of the mediator pattern [62]</i>	46
<i>Illustration 25: Role diagram for the Observer Pattern [62]</i>	47
<i>Illustration 26: Role diagram of the Chain of Responsibility pattern [62]</i>	47
<i>Illustration 27: Role diagram of the Bureaucracy pattern [62]</i>	47
<i>Illustration 28: Prototype - Abstract Factory composite pattern [63]</i>	48
<i>Illustration 29: Cross-Domain Pattern example. Left: Solution to solve “Invoice” problem. Right: Solution to “Balance end of year close” problem [38]</i>	49
<i>Ilustración 30: Metamodel for a Domain Patterns Catalog</i>	56
<i>Ilustración 31: Elements Representation</i>	57
<i>Illustration 32: Patterns Representation</i>	59
<i>Illustration 33: Classification Representation</i>	60
<i>Illustration 34: Vocabulary Representation</i>	60
<i>Illustration 35: Method for developing a Domain Patterns Catalog</i>	62
<i>Illustration 36: Naming Standard for Model Management</i>	65

<i>Illustration 37: Storage Directory Standard for Model Management</i>	65
<i>Illustration 38: Example of Storage Directory Standard</i>	65
<i>Illustration 39: Model Management ER Diagram</i>	67
<i>Illustration 40: Steps to populate the Model Management Database</i>	67
<i>Illustration 41: Structure of the "Elements Listing" Artifact</i>	74
<i>Illustration 42: Example of data-types' heterogeneity</i>	78
<i>Illustration 43: Address according to D. Hay</i>	81
<i>Illustration 44: Example of entity design heterogeneity</i>	83
<i>Illustration 45: Clustering of attributes within an entity</i>	86
<i>Illustration 46: Contact Information Pattern Candidate</i>	87
<i>Illustration 47: Healthcare Party Pattern</i>	88
<i>Illustration 48: Test Pattern</i>	89
<i>Illustration 49: Patterns as views of a representation of the domain at a certain level of abstraction</i>	90
<i>Illustration 50: Sample Pattern in the Candidate Patterns Catalog</i>	92
<i>Illustration 51: The domain patterns catalog represented in the ecore metamodel</i>	93
<i>Illustration 52: Domain Patterns Catalog project</i>	94
<i>Illustration 53: Instantiation process 1 Left: insertion of the relevant terms. Right: instantiation of the entities, attributes and associations relevant to form the Sample pattern.</i>	95
<i>Illustration 54: Instantiation process 2. Insertion of the Sample pattern</i>	95
<i>Illustration 55: Patient Allergy pattern</i>	96

Index of Tables

<i>Table 1: Method for derivation of problem-context patterns by using general design patterns [39]</i>	<i>28</i>
<i>Table 2: Tabular Template corresponding to the Functional Patterns for ERP. [58] (translated)</i>	<i>38</i>
<i>Table 3: The ISO/IEC 9126-1 internal/external quality model [59]</i>	<i>39</i>
<i>Table 4 : A categorization of non-technical factors following the ISO/IEC 9126-1 style [59]</i>	<i>40</i>
<i>Table 5: GoF Classification Schema [25]</i>	<i>50</i>
<i>Table 6: Benefits of more specific and more generalized style of modeling [44]</i>	<i>54</i>
<i>Table 7: Domain Patterns Classification Schema</i>	<i>55</i>
<i>Table 8: Pattern Template</i>	<i>59</i>
<i>Table 9: Model Management Entities identification</i>	<i>66</i>
<i>Table 10: Model Management Relationships identification</i>	<i>66</i>
<i>Table 11: Store domain Hospital Management</i>	<i>68</i>
<i>Table 12: Store Sources (partial rows)</i>	<i>68</i>
<i>Table 13: Store Model (partial columns and partial rows)</i>	<i>70</i>
<i>Table 14: Results of Domain Analysis for the Hospital Management Domain</i>	<i>71</i>
<i>Table 15: Example of Entity element representation within the Element Listing document</i>	<i>75</i>
<i>Table 16: Example of Relationship element representation within the Element Listing document</i>	<i>76</i>
<i>Table 17: Results from the Element Extraction phase</i>	<i>77</i>
<i>Table 18: Example of Entity element representation within the Elements Union</i>	<i>82</i>
<i>Table 19: Example of Relationship element representation within the Elements Union</i>	<i>82</i>
<i>Table 20: Results from the Elements Refinement phase</i>	<i>83</i>
<i>Table 21: Relationships within the Patient Entity, from the Elements-Union</i>	<i>85</i>
<i>Table 22: Comparison between similar entities</i>	<i>87</i>
<i>Table 23: Candidate patterns classified</i>	<i>91</i>
<i>Table 24: Review of similar patterns found in other catalogs</i>	<i>97</i>

Index of Candidate Patterns

<i>Candidate Pattern 1: Patient's Allergy</i>	154
<i>Candidate Pattern 2: Antecedent Types</i>	155
<i>Candidate Pattern 3: Clinical Antecedent</i>	155
<i>Candidate Pattern 4: Patient's Habit</i>	156
<i>Candidate Pattern 5: Pediatric Environment Antecedent</i>	156
<i>Candidate Pattern 6: Familiar Antecedent</i>	157
<i>Candidate Pattern 7: Obstetric Antecedent</i>	158
<i>Candidate Pattern 8: Pediatric Birth Antecedent</i>	158
<i>Candidate Pattern 9: Hospital Types</i>	159
<i>Candidate Pattern 10: Vital Signs</i>	159
<i>Candidate Pattern 11: Laboratory Employee</i>	160
<i>Candidate Pattern 12: Medical Facility</i>	160
<i>Candidate Pattern 13: Ultrasound Types</i>	161
<i>Candidate Pattern 14: Movement Disorder Physical Examination</i>	161
<i>Candidate Pattern 15: Perception Disorder Physical Examination</i>	162
<i>Candidate Pattern 16: Organ System Physical Examination</i>	162
<i>Candidate Pattern 17: Department Types</i>	163
<i>Candidate Pattern 18: Familiar History</i>	163
<i>Candidate Pattern 19: Healthcare Party</i>	164
<i>Candidate Pattern 20: Healthcare Role</i>	165
<i>Candidate Pattern 21: Observation States</i>	165
<i>Candidate Pattern 22: Observation</i>	166
<i>Candidate Pattern 23: Supporting Unit Types</i>	167
<i>Candidate Pattern 24: Sample</i>	168
<i>Candidate Pattern 25: Test</i>	169
<i>Candidate Pattern 26: Healthcare Physical Examination</i>	170
<i>Candidate Pattern 27: Hospital Organization</i>	171
<i>Candidate Pattern 28: Medical Record</i>	172
<i>Candidate Pattern 29: Address</i>	173
<i>Candidate Pattern 30: Quantity</i>	173
<i>Candidate Pattern 31: Contact Information</i>	174
<i>Candidate Pattern 32: Party</i>	174
<i>Candidate Pattern 33: Employment</i>	175
<i>Candidate Pattern 34: Physical Observation</i>	175

1 Introduction

This master thesis aims to propose a methodology for gathering domain knowledge in the form of patterns. The method reflects the systematic process needed for the construction of a catalog of domain patterns in the field of healthcare management systems.

The construction of this catalog is intended to help improve the design process of healthcare management systems using patterns that facilitate reuse of knowledge within the domain.

In reaching this proposed procedure, we have analyzed the state of the art regarding the topic, we also provide a pattern definition by means of a metamodel of domain patterns catalog, and we made the extraction of patterns from a set of models of actual projects collected from several sources described later.

1.1 Background

This work was conceived under the BIZWARE project [1], dedicated to the research of Lifecycle Management for DSL development and to the development of knowledge-based services to support domain-specific modeling [2], among others.

In the modeling domain, there is a large variety of ways to model and model file formats. This heterogeneity along with the expertise gap of the modeler's knowledge in such formats and/or in the problem's domain may lead to some quality and reuse problems. Also, reuse of resources is a strategy for cost reduction and efficiency improvement within the software development process.

Patterns have lately become a popular means for communicating knowledge about proven solutions, and collections of patterns have been proposed for various fields. These collections are patterns catalogues, where the patterns are organized according to a structure that facilitates their identification for future application.

The most popular ways for making patterns available are physical books, digital media containing SQL schemas, and websites.

In any case, the design of a pattern's structure must be performed in order to make a consistent patterns catalog. This purpose can be achieved by means of designing a meta-model.

A meta-model in software engineering specifies the language and processes from which to form a model [3].

Domain-specific languages (DSLs) are languages tailored to describe a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-

purpose programming/modeling languages as well as provide tools for easy manipulation of the instances of the model in the domain of application.

1.2 Motivation

Due to the growing pressures for developing quality software in the shortest time, reuse techniques have been introduced as a key concept of software design.

Reuse involves making use of abstracted knowledge from any artifact produced during software design and construction. In our case we realize that models represent knowledge of a particular domain that can be abstracted in the form of patterns in order to be reused.

Research on the topic of patterns has been made but no methodology has been proposed in order to extract patterns from domain models such as the ones used as sources of knowledge in the context of this work, those are UML diagrams, ER diagrams and SQL schemas.

1.3 Objectives

The goals of this thesis are:

- Literature review
Perform a review of the existent research in the area of modeling and domain analysis and in the area of patterns and pattern catalogs; their methodologies for identification and classification.
- Construction of the methodology
This work aims to develop a methodology for building a domain patterns catalog. The approach should help getting quality conceptual data models based on reusable patterns.
- Construction of a metamodel for domain patterns catalog
Building a metamodel that is able to represent a domain patterns catalog that is general enough that could represent patterns from a domain other than the one analyzed in this thesis.
- Construction of a Domain Candidate Patterns catalog
In order to validate the method we aim to discover and gather recurring patterns from domain models collected from sources of research and industry. The accuracy of these patterns should be supported by patterns catalogs, whenever pertinent.
- Definition of the criteria for pattern classification
Extracted from several available domain model projects, such patterns should be classified in order to be better identified and used.

1.4 Structure of this work

After this brief introduction we present the contents of our research divided in six chapters.

Chapter 2 contains the prior research in the area of modeling. This chapter includes concepts as well as a summary of current research on the area.

Chapter 3 condenses the foundations about patterns and patterns catalogs that inspired our work as well as guided our methodology and helped us support our findings.

Chapter 4 summarizes how the methods found in the literature and current research inspired us and explains the adaptations we proposed to make over them in order to fulfill our purposes.

Chapter 5 introduces the implementation of the technique we developed: the method, the artifacts and a step-by-step description of the issues and results of each of the steps of the method.

Chapter 6 summarizes the conclusions, issues and aspiration for future work having this work as a foundation.

Chapter 7 contains the literature references used in the implementation of this work.

In the appendix section we attach some additional documents as well as the candidate patterns catalog produced in this work.

2 Modeling Foundations

This chapter introduces the terms that are important in the domain of the current work, most of which belong to modeling in the area of software engineering and business process management.

We start by describing the general concepts around modeling, domain as well as other relevant terms in the context of this work.

2.1 Model

Model is a very widely used word in several disciplines to describe an abstract representation of an aspect of a thing; May it be the structure, function, behavior, or others [4].

In order to distinguish models from other artifacts within the software development process, Stachowiak [5] states that any candidate must fulfill three criteria for being considered a model. These criteria are:

- Mapping criterion

There is an original object or phenomenon that is mapped to the model. This original object or phenomenon is referred to as “the original”.

- Reduction criterion

Not all the properties of the original are mapped on to the model, but the model is somehow reduced. On the other hand, the model must mirror at least some properties of the original.

- Pragmatic criterion

The model can replace the original for some purpose, i.e. the model is useful.

The representation of the resulting relations between an original and its associated model is visualized in Illustration 1:

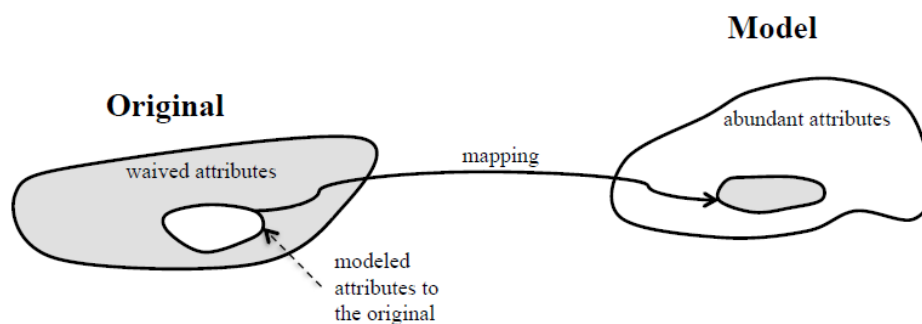


Illustration 1 - Original and Model criterion according to Stachowiak (1973) [5]

One should note that the mapping criterion does not imply the physical existence of the original; it may be planned, suspected, or fictitious. For example, the cost estimation of a software project is a speculative model of the future based on some known facts and calculations. Also, a model may act as the original of another model. For instance, a program design is a model of the code to be written, while the code is a model of the computation performed by the computer when the code is executed.

The advantage of the reduction criterion is that it simplifies the original in a way that makes it manageable and easier to understand. This is tightly coupled with the last criterion, the pragmatic criterion, describing the applicability of the model being used instead of the original.

In the context of this work a model is a representation of the elements and the relationships among these elements within the context of the healthcare management domain.

Some examples of models used are: UML diagrams, SQL Schema, .Net Code, Ontologies, and others.

The most relevant models used in this work are information models, as the one seen in Illustration 2. They describe how data is represented, either in a company, in an information system or management system database. These characteristics make the model particularly important to identify concepts and relationships within the domain.

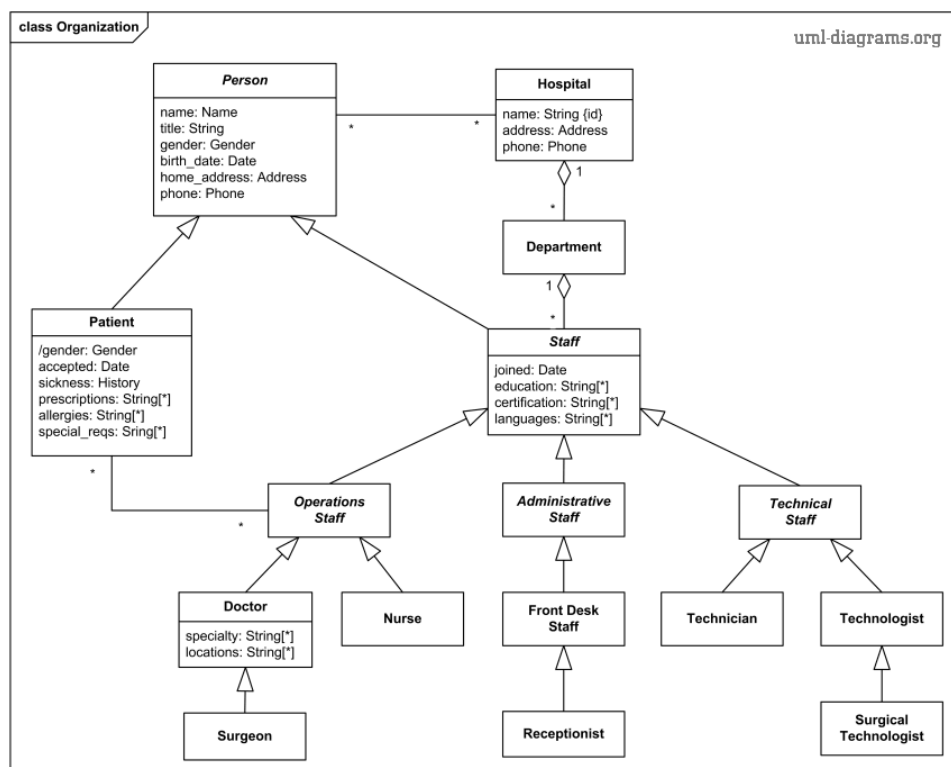


Illustration 2 Hospital Organization Domain Model – Patient, Hospital, Staff – Operations, Administrative and Technical. [6]

2.2 Modeling

Modeling is an important method of software engineering that gives the software specification and improves the communication between counterparts involved in the software development process; i.e.: between client and analyst, between software designer and developer, and so on [7].

In model driven development, on the other hand, models are created to express the structure or behavior of the software in an efficient and domain-specific way. The models are the base and after subsequent model transformations on this base we get the executable code [8].

Some examples of models in this context are database schemes, process models, design patterns, class diagrams, project plans, specifications and designs, metrics, and so on.

Consequently, modeling acts as a starting point for understanding the common basis for developers and users. Modeling normally integrates domain experts, who are involved in a business process, and their knowledge, into the software development.

2.3 The MOF Hierarchy

The Meta-Object Facility (MOF) is an Object Management Group (OMG) standard for model-driven engineering, where models can be exported from one application, imported into another transported across a network, stored in a repository and then retrieved, rendered into different formats (including XMI, OMG's XML-based standard format for model transmission and storage), transformed, and used to generate application code [9].

The core concepts of this hierarchy are models, meta-models and meta-meta-model. Some important characteristics [10] we should know about these concepts are:

- A **model** is an abstraction of a physical system (software, hardware or people) with a certain purpose. In simple words models are representations of things that are (usually) complex.
- A **meta-model** is a model that defines the language for expressing a model i.e., it describes the structure of a model.
- A **meta-meta model** is a model that defines the language for expressing a meta-model. The relationship between a meta-meta model and a meta-model is analogous to the relationship between a meta-model and a model.

The MOF hierarchy is a four-layer modeling hierarchy defined by the OMG as: Layer 0: Instance/Information, Layer 1: Model, Layer 2: Meta-model and Layer 3: Meta-meta-model. For ease of designation, the layers are generally referred to as M0 to M3, being the level of

abstraction in ascent from bottom to top as seen in Illustration 3 and as explained subsequently [9] [10].

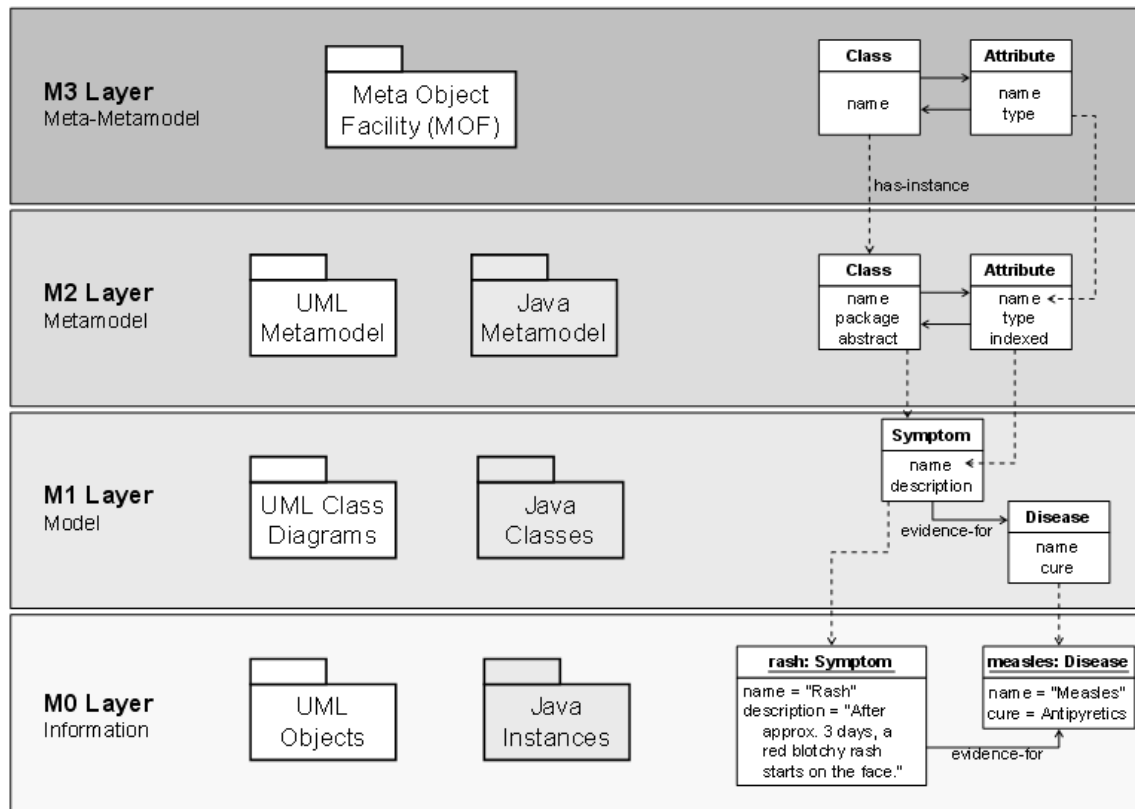


Illustration 3: The MOF Hierarchy¹

M0, Information Layer: provides an integral representation of reality. In connection with the modeling of technical systems, it usually denotes the overall system or certain aspects of interest at runtime. It has several names such as the Information Layer, Instance Layer, Original Layer or Data Layer; mainly because that is what we find in this layer, real life entities. I.e. Measles, the disease.

M1, Model Layer: The model layer abstracts different entities of the information layer M0. It generalizes (classifies) thereby some aspects from reality to model. In other words, M0 layer instantiates what is described by a model of M1 layer. I.e. the class Disease. Furthermore, a model of the M1 layer forms an instance of a modeling language at level M2. Object orientated software can for example be described by UML models.

M2, Metamodel Layer: this layer contains a metamodel, a model, that describes a specific modeling language. It generally defines:

- The elements of which a model can be constructed.

¹ From <http://protege.cim3.net/cgi-bin/wiki.pl?XMIBackendTechnicalBackground>

² From http://en.wikipedia.org/wiki/Model_Driven_Software_Development

³ <http://www.sei.cmu.edu/>

- What characteristics these elements have.
- How these elements are related to each other.

The metamodel only describes a part of the modeling language, namely the abstract syntax of the possible models, but not the used notation and the meaning (semantics) behind. A modeling language, specified by a metamodel at the level M2, can be instantiated by models at level M1. The metamodel itself forms another instance of a modeling language at the overlying level M3.

M3, meta-metamodel Layer: A meta-metamodel provides a description of the meta-models of the M2 layer. It represents therefore a modeling language for the abstract syntax of modeling languages. This modeling language can be instantiated by models of the M2 layer. The elements of a UML metamodel are, for example, referred to as meta-classes, meta-attributes and meta-associations.

2.4 Model Driven Development

The Model-driven development (MDD) approach to software development allows people from different expertise levels to work together on a project. This has gained popularity in the past years since it allows companies to maximize effective work on a project by improving communication, minimizing the overhead necessary to produce working software and to minimize the efforts of validation by end users [11].

MDD is also known as model-driven software development (MDSD) and a particular approach is model-driven architecture (MDA)².

The idea of model-driven software development is to work at a higher abstraction level than conventional programming, by describing the software as a model [12]. Thereby most of the development happens at the model level and a major part of the source code is generated automatically [10]. The changes in the models can be carried out also when new functions that are not present in the model are added to the implementation.

2.5 Domain

A domain refers to the context or universe of discourse being discussed.

In the context of software engineering it is most often understood as an application area, a field for which software systems are developed.

In their book Prieto-Diaz and Arango [13] discuss that domain analysis is the major factor of reusability in software development.

²From http://en.wikipedia.org/wiki/Model_Driven_Software_Development

For them domain analysis is a process by which information used in development of software systems is identified, captured and organized with the purpose of making it reusable when creating new systems.

The model of domain analysis presented by their work is summarized in Illustration 4.

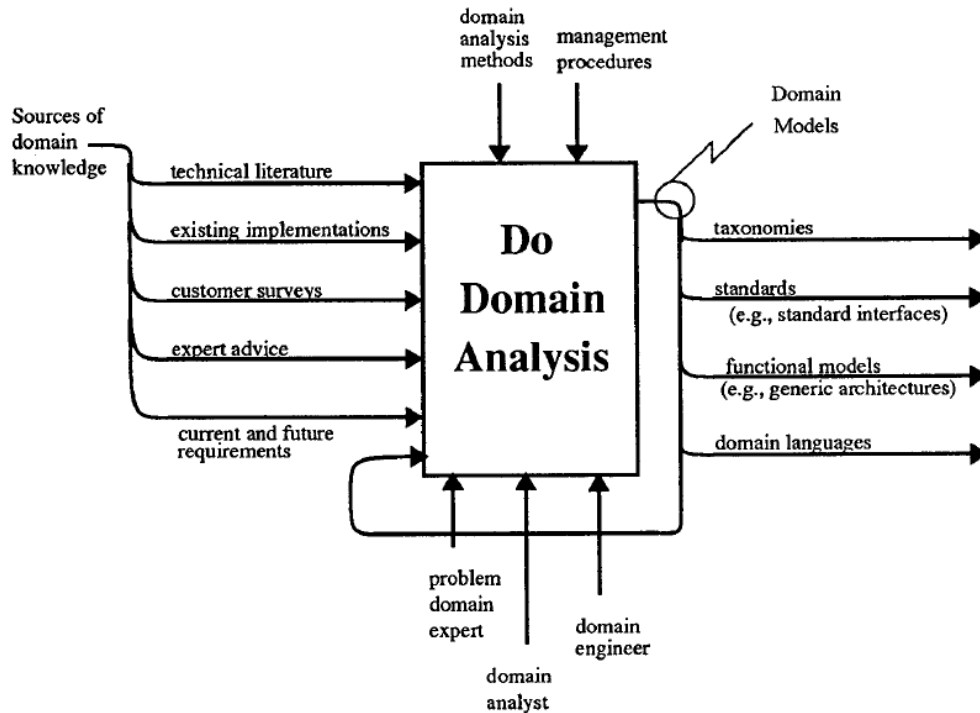


Illustration 4 Context View of Domain Analysis [13]

This model describes domain analysis as an activity that takes multiple sources of input, produces many different kinds of output, and is heavily parameterized. Raw domain knowledge from any relevant source is taken as input. Participants in the process can be, among others, domain experts and analysts. Outputs are (semi)formalized concepts, domain processes, standards, architectures, among others.

The scope of a domain investigation can vary widely.

In the context of this work the domain is healthcare management. We take the perspective of software modeling, database modeling, and business process management modeling coming from projects of real life and research projects and we use their terminology to describe the problem space.

2.6 Prieto-Diaz's Domain Analysis

The process for domain analysis that Prieto-Diaz proposes in [14] can be seen in Illustration 5. This approach proposes a framework to integrate domain analysis in a software development

process. In this framework the products of domain analysis are continually reviewed and refined as new systems in the domain are built.

The bottom-up approach describes the identification of objects and operations. The top-down approach is the systematic analysis to identify domain models; where high level designs and requirements of current and new systems are analyzed for commonality.

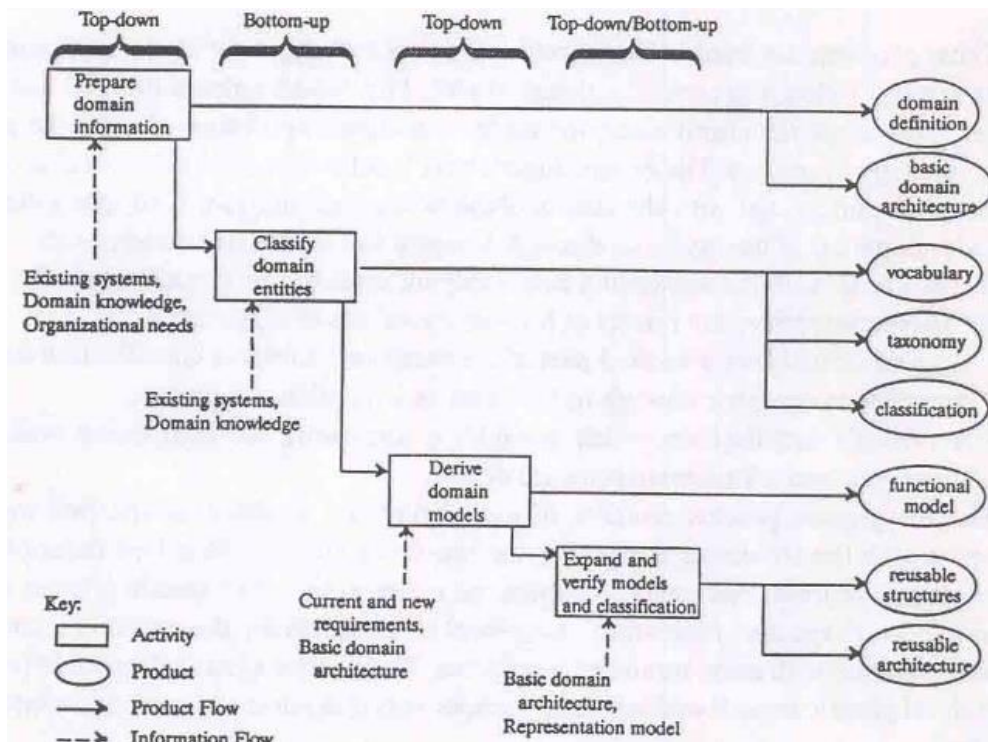


Illustration 5 Top-down-Bottom-up Domain Analysis process [14]

- **Prepare domain information:** is the first top-down activity in the domain analysis. It consists of the domain definition as an informal statement and the domain architecture as a high-level description of architectural properties shared by applications in the domain.
- **Classify domain entities:** a bottom-up analysis activity following domain information preparation. At this point the domain analysis examines low-level requirements, source code and documentation from existing systems. The resulting artifacts contain a preliminary vocabulary (glossary of terms), domain taxonomy, the classification structure and the standard descriptors. The result of this stage provides a framework to verify the basic domain architecture when deriving domain models or for defining reusable components.
- **Define domain models:** uses the products of the second activity in order to generate a generic functional model. This model is expressed as layers of groups of functions. It

supports design and development of new systems through composition of reusable components.

- Expand and verify models and classification: uses the resulting model from the previous activity, which helps domain analysts select the proper structural components and integrate them for standardizing designs during this activity.

In the context of this work we attempt to follow the approach proposed by Prieto-Diaz in order to develop a catalog of domain model patterns having as input domain models. That is to say, we intend to prepare the domain information by studying relevant sources, then we intend to discover elements of the domain by doing a bottom-up analysis then we will derive domain patterns in a top-down manner to finally classify and expand the patterns.

The steps and artifacts produced by the method will be described later in section 4 Implementation.

2.7 Feature-Oriented Domain Analysis (FODA)

FODA (Feature-oriented domain analysis) is a domain analysis method, which was introduced by the Software engineering institute³ (SEI) in 1990. The FODA method supports the reuse at architectural and functional levels meaning that feature oriented domain analysis produces domain products representing common functionality and architecture of applications in a domain [15].

The feature-oriented concept is based on identifying those features a user commonly expects in applications in a domain.

The method gathers and represents information on software systems that share a common set of capabilities and data. It aims to support the development and reuse of abstractions by gathering a set of the following modeling concepts:

- Aggregation/decomposition; aggregation being the abstraction of a collection of units and decomposition the refinement of an aggregation in the units that constitute it.
- Generalization/specialization; generalization being the abstraction of the commonalities among similar units and specialization the refinement containing the details of the individual units.
- Parameterization; is a component development technique in which components are adapted in many different ways by substituting the values of parameters which are embedded in the component.

³ <http://www.sei.cmu.edu/>

The FODA method applies the aggregation and generalization primitives to capture the commonalities of the applications in the domain in terms of abstractions. Differences between applications are captured in the refinements [15]. The parameters are defined to uniquely specify the context for each specific refinement. The result of this approach is a domain product consisting of a collection of abstractions and a series of refinements of each abstraction with parameterization.

The steps identified for the method are:

1. **Context analysis:** involves defining the extent (or bounds) of a domain for analysis.
2. **Domain modeling:** deals with describing the problems within the domain that are addressed by software.
3. **Architecture modeling:** consists in creating the software architecture(s) that implements a solution to the problems in the domain.

The three participant groups identified in the domain analysis process are sources, producers and consumers as seen in Illustration 6.

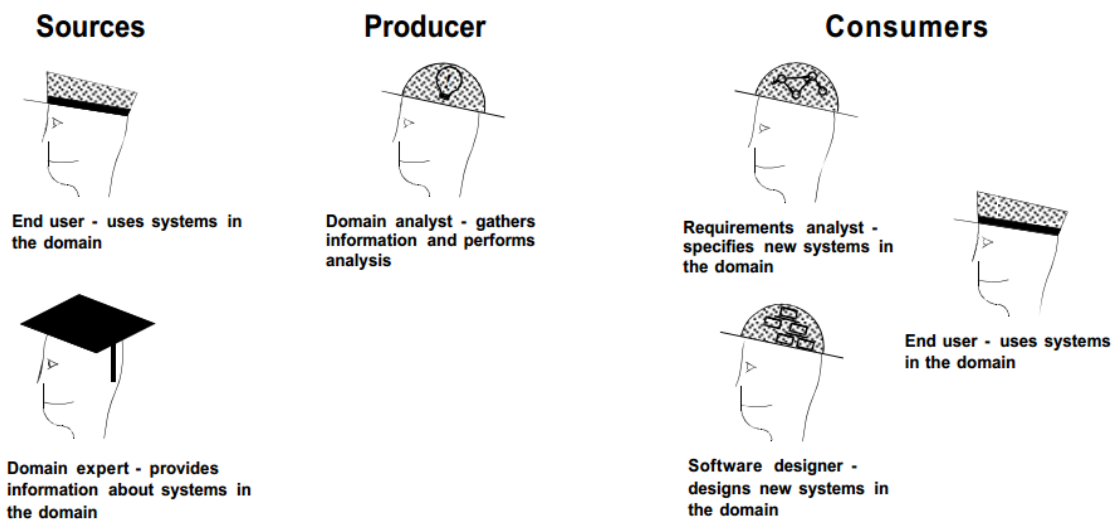


Illustration 6: Participants in the domain analysis process [15]

During the context analysis stage the domain analyst interacts with users and domain experts to establish the bounds of the domain and establish a proper scope for the analysis that is likely to yield exploitable domain products. The analyst also gathers sources of information for performing the analysis.

The results of this phase provide the context of the domain in a context model. This requires representing the primary inputs and outputs of software in the domain as well as identifying other software interfaces.

Within the domain modeling task, the domain analyst uses information sources and the other products of the context analysis to support the creation of a domain model by identifying the commonalities and differences of the problems that are addressed by the applications.

This model is reviewed by the system user, the domain expert, and the requirements analyst.

The products of this phase describe the problems addressed by software in the domain. They provide: features of software in the domain, standard vocabulary of domain experts, documentation of the entities embodied in software as well as generic software requirements via control flow, data flow, and other specification techniques.

The three major activities this phase consists of are: feature analysis, entity-relationship modeling, and functional analysis.

During the architecture modeling task the domain analyst uses the domain model to produce the architecture model, a software "solution" to the problems defined in the domain modeling phase. This model should be reviewed by the domain expert, the requirements analyst and the software engineer. The requirements analyst and software designer will use the products of a domain analysis to establish the structure of implementations of software in the domain. The representations generated provide developers with a set of architectural models for constructing applications and mappings from the domain model to the architectures. These architectures can also guide the development of libraries of reusable components.

2.8 Domain Specific Languages

Fowler [16] defines DSLs as:

“Domain-specific language (noun): a computer programming language of limited expressiveness focused on a particular domain”.

There are four key elements that lead to this definition:

- **Computer programming language:** A DSL is used by humans to instruct a computer to do something. Such as with any modern programming language, it is designed to make the task easy for humans and still it should result in an executable computer application.
- **Language nature:** A DSL is a programming language, and as such should have a sense of fluency where the expressiveness comes not just from individual expressions but also from the composition of them.
- **Limited expressiveness:** While a general-purpose programming language provides lots of capabilities and abstraction levels a DSL is more limited but is easier to learn and

use. Also, a DSL, being designed to describe a rather specific domain, it only needs to support a minimum of features needed in such domain.

- **Domain focus:** A limited language is only useful if it has a clear focus on a small domain. The domain focus is what makes a limited language meaningful.

Domain-specific languages (DSLs) can be graphical and textual and they offer significant gains in expressiveness and ease of use compared with general-purpose programming languages for a specific problem domain [17].

Domain-specific modeling languages (DSML) have been of great importance in the development of model-based languages. Every DSML is based on a domain-specific language (DSL). In the literature the terms DSML and DSL are used undifferentiated, especially because both are languages, which address problems in a particular domain and provide built-in abstractions and notations for that domain; with the difference that one provides programming abstractions and the other modeling abstractions.

Some authors argue that DSLs are languages in which the domain experts themselves could develop programs, without the help of computer science specialists [17] [18].

Programming language and modeling language development has been studied in order to improve quality and low cost for development. As part of the language development process we have to analyze some components of it. Such components are the metamodel or abstract syntax, the concrete syntax and the semantics as seen in Illustration 7.

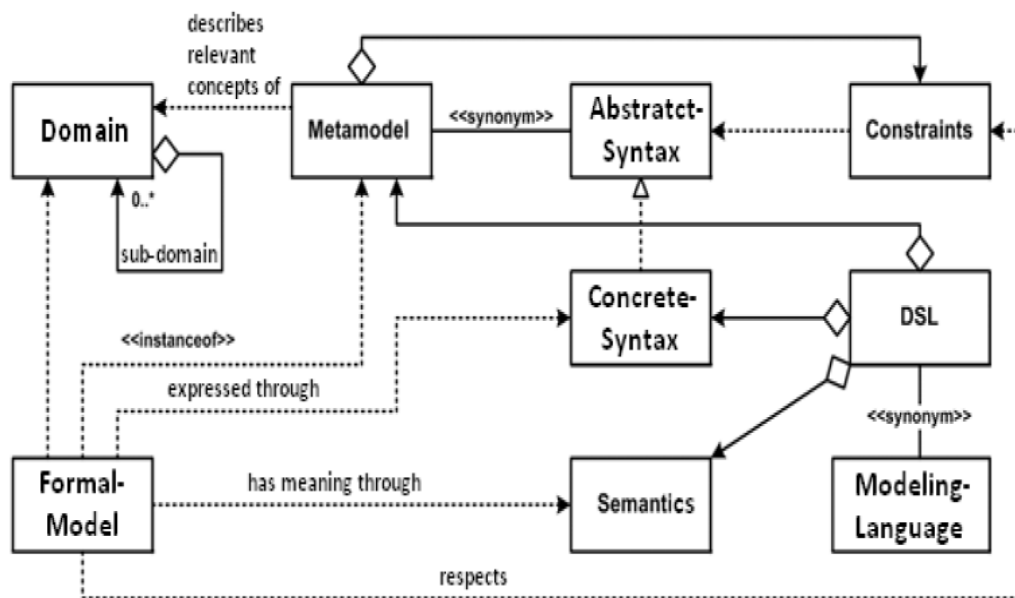


Illustration 7: Relationship between DSL Components [18]

The metamodel corresponds to a representation of the model, taking the relevant concepts of it. The abstract syntax therefore describes the grammar of the language by defining the allowed elements and the relationships among them within the language [19].

The concrete syntax is the actual representation of the language. The semantics give the meaning to the models developed with the DSL. They are mostly defined partly in the metamodel and partly by means of constraints.

Language constraints are used to guarantee the correctness of models designed using the DSL. This is important because not everything can be expressed and described by meta-models. For instance, values which can be only in a certain interval or names that can be only in character letters, etc.

Finally, we mention that there is no clear algorithm to create new languages but Kleppe [17] gives an understandable seven steps recipe to do so:

- 1) Create an abstract syntax model, taking into account references to other languages.
- 2) Generate a concrete syntax model, and experiment with some example programs.
- 3) Revise the abstract syntax model and reiterate.
- 4) Create semantics for the language, constraints and rules.
- 5) Revise the abstract syntax model and reiterate, possibly changing the concrete syntax model(s) as well.
- 6) Create the tools that support the language for user usage.
- 7) Devise a way to handle different versions of the language, because users might demand changes whereas some others would prefer to stick to older versions.

2.9 Eclipse EMF

The Eclipse Modeling Framework Project [20] (EMF) is an Eclipse-based modeling framework that offers not only modeling features, but also code generation for building tools and other applications based on a structured data model. EMF allows creating the meta-model via different means, e.g. XMI, Java annotations, UML or a XML Schema.

After creation of the model, Eclipse EMF gives the chance to generate instances from the model specification described in XMI, for instance. EMF provides tools and runtime support to produce the Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor [21].

The EMF project is part of the modeling project of Eclipse [22]. The modeling project is organized logically into projects that provide the following capabilities: abstract syntax

development, concrete syntax development, model-to-model transformation, and model-to-text transformation.

Within the abstract syntax development EMF's Ecore model serves as the metamodel for defining DSLs. We can further refine the structure and semantics of our DSL using Object Constraint Language (OCL), in addition to providing support for transactions, query, and validation.

Concrete syntax development; the abstract syntax for a DSL usually must be presented for use by humans, so one or more concrete syntaxes must be developed. EMF provides XMI serialization of model instances, but within Eclipse also GMF, that provides a graphical concrete syntax, and also TMF, for a textual concrete syntax, are available.

Model Transformation; arises from the need to produce some output from the instance models created by our DSL. The Modeling project provides both model-to-model and model-to-text transformation components. There are several components within the modeling project of Eclipse that support both types of transformations. One of the most popular ones is JET - Java Emitter Templates- component that is used by EMF.

3 Patterns Foundations

In this chapter we introduce the theory of patterns, the evolution of them in several areas of implementation as well as specific theory and research important for the development of our work.

3.1 Pattern

The usage of the term ‘pattern’ in software design was inspired by the conception of architectural patterns introduced by Alexander [23].

Patterns for the architectural design phase and the analysis phase, and a language for the implementation phase were developed and presented in the book “*A Pattern Language: Towns, Buildings, Construction*”.

The definition that Alexander gives declares as follows:

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution over and over again, without ever doing it the same way twice.” [23]

In the architecture domain patterns include ways of building things from big to small, in a way that has succeeded and has persisted providing accuracy, practicality and beauty.

Martin Fowler defines in reference to analysis patterns:

‘A pattern is an idea that has been useful in one practical context and will be probably useful in others’. [24]

In the software design domain the most representative book is the Design Patterns from Gamma et. al.; in their book they define patterns as follows:

“Patterns describe recurring solutions to common problems in software design” [25]

As a conclusion extracted from reviewing [26,23,27,25,24] we collect a set of characteristics that a pattern must have.

A pattern must:

- Solve a problem; presenting a core solution.
- Have a context; it must describe where the solution can be used.
- Recur; it must be relevant in other situations.
- Teach: it must provide sufficient understanding to tailor the solution.
- Have a name; it must be referred to consistently.

To sum up, patterns are core solutions for recurring problems; they are tested and accepted by experts in the domain, they are in everyday life and they should be referred to in a consistent manner.

In the software developing domain we can say that patterns represent best practice, proven solutions, and lessons learned that aid in evolving software engineering into a mature engineering discipline. Also, patterns support reuse of software architecture and design; they capture the static and dynamic structures and collaborations of successful solutions to problems that arise when building applications in a particular domain.

As a result we present in Illustration 8 Basic parts of a pattern taken from Tesanovic [26].

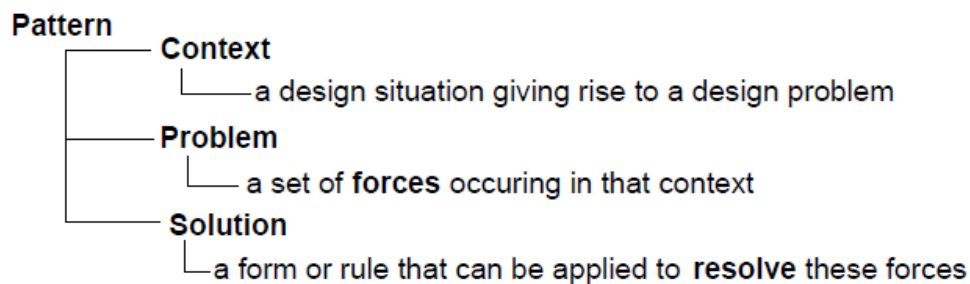


Illustration 8 Basic parts of a pattern [26]

Every pattern has these three essential elements, which are: a context, a problem, and a solution. The context describes a recurring set of situations in which the pattern can be applied. The problem refers to a set of forces, often describing when to apply the pattern i.e., goals and constraints, which occur in the context. The solution refers to a design form or a design rule that can be applied to resolve the forces. Solution describes the elements that constitute a pattern, relationships among these elements, as well as responsibilities and collaboration.

For the context of our work, we take the recommendations of Tesanovic, although, since the context for us is defined by the domain being modeled, we consider problem, solution and consequences as well as a name and the reference to related patterns as described in section 4.4.2 of this thesis.

3.2 Pattern collections

The different types of pattern collections have been identified in [28]: pattern language, and pattern catalog. These pattern collections have different degrees of structure and interaction described as follows.

3.2.1 Pattern Language

One commonly used definition of a pattern language can be found in [23].

“A pattern language defines a collection of patterns and the rules to combine them into an architectural style”.

In the software design domain, for instance, pattern languages describe software frameworks or families of related systems.

According to Coplein a slightly different definition of the pattern language can be found in [29].

“A pattern language is a structured collection of patterns that build on each other to transform needs and constraints into an architecture”.

It is important to note that a pattern language is not a programming language. It might be found as a prose document created with the purpose to guide and inform the designer. A pattern language should include rules and guidelines to explain in a way how and when to apply the patterns.

A pattern language could also be viewed both as a lexicon of patterns and a grammar.

3.2.2 Pattern Catalog

Pattern catalog is a collection of related patterns, where patterns are organized according to a structure of broad categories that facilitates their identification and application, which usually include some amount of cross referencing between patterns [28].

As a way of exemplification the pattern catalog introduced by Gamma et al. [25], the Gang of Four or GoF, is discussed in more detail as follows.

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight (195) Observer State Strategy Visitor

Illustration 9 GoF Pattern Catalog [25]

They classified patterns based on two criteria, as shown in Illustration 9: purpose and scope. Purpose reflects what a pattern does. Patterns can have either creational, structural, or behavioral purpose. Creational patterns are concerned with the process of object creation. Structural patterns deal with the composition of classes or objects. Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility.

Scope specifies whether the pattern applies primarily on classes or on objects. Class patterns deal with relationships between classes and their sub-classes. These relationships are established through inheritance. Object patterns deal with object relationships. Patterns labeled as class patterns are those that focus on class relationships.

In the context of this work, we intend to give the baseline for the creation of a patterns catalog, the candidate patterns catalog for domain patterns.

3.3 Patterns and Software Development

As previously stated, a pattern corresponds to a solution of a problem in a context. It helps create a common language to communicate ideas and experiences about the problem and its solution.

Patterns have been used in various areas of software engineering, they were created in the first place to resolve problems identified during the design of software packages [25].

We will now describe some background history of the emergence of patterns, then we expose patterns in software engineering and finally we refer to patterns in domain engineering.

3.3.1 Background History

In this section we present some representative authors that influenced the concept of patterns to lead to the use and understanding of them as we know it today.

3.3.1.1 *Christopher Alexander*

The term pattern was first used by the architect Christopher Alexander in 1979, who in his book *The Timeless Way of Building* [30] proposed the use of patterns to increase the quality in construction of buildings.

Alexander aimed to create structures that were good for people across history. He thought that those structures have a positive influence on society by improving their comfort and quality of life. To this end architects should constantly strive to construct products that conform and adapt to the needs of all its inhabitants. To do this, he described his ideas for achieving these goals, as means of architectural patterns [23].

3.3.1.2 *Ward and Kent*

Later, in 1987, Ward and Kent, consultants at Tektronix's Semiconductor Test Systems Group, decided to use some of the ideas of Alexander, by letting representatives of the users finish the design. This exercise resulted in the development of a small language of five patterns that helped the novice designers take advantage of the Smalltalk's language strengths [31].

3.3.1.3 *Peter Coad*

During the 1990s Coad co-authored several books on the analysis, design, and programming of object-oriented software. Coad became renowned through his work on Object-oriented analysis (OOA) as well for his exploration of patterns [32].

3.3.1.4 *Jim Coplien*

Jim Coplien began compiling a catalog of idioms C++ used to teach objects in this language within AT&T [29], getting his work to be a source for the later work of Gamma et al [25]. Latter he was a conference organizer and a coauthor of the proceedings of the first PLoP - Pattern Languages of Programming - conference compiled in the book *Pattern Languages of Program Design* [33].

3.3.1.5 *Erich Gamma et al*

The term Gang of Four refers to the authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides.

On the one hand, Erich Gamma worked on object-oriented design in ET++ as part of his PhD thesis. He had realized that recurring design structures or patterns were important but the task of capturing and communicating them was not yet tackled. On the other hand, the other authors

were also discovering the importance of reusing and transmitting the pieces of solutions used by experts to aid software designers produce better designs.

They got to know each other around 1990 and realized they shared common ideas about the key ideas behind writing reusable Object Oriented software. That is how they joined forces to write Design Patterns, where they identified many patterns of software OO design [25].

3.3.1.6 *Other authors*

Other important authors⁴ for the development of Patterns and patterns theory are:

- Ralph Johnson
- Bruce Anderson
- Ron Casselman
- Desmond De Souza
- Norm Kerth
- Doug Lea
- Wolfgang Pree
- Frank Buschmann

These authors were contemporaneous of the Gang of Four and influenced patterns theories in conferences and workshops such as OOPSLA⁵, Thornwood Workshop⁶ and PLoP⁷.

3.3.2 *Patterns in software engineering*

Patterns in software engineering became popular especially with the acceptance of the book Design Patterns: Elements of Reusable Object-Oriented Software [25]. Their definition of design patterns focuses on patterns of object-oriented design; however small changes can be adjusted to describe software design patterns in general [25], given that:

- A design pattern names, summarizes, and identifies the key aspects of a common design structure and that fact makes it especially useful for creating reusable object-oriented design.
- The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities.
- Each design pattern focuses on a particular problem of object-oriented design.

⁴ Taken from <http://c2.com/cgi/wiki?HistoryOfPatterns>

⁵ Object-Oriented Programming, Systems, Languages and Applications - <http://en.wikipedia.org/wiki/OOPSLA> and <http://splashcon.org/>

⁶ <http://c2.com/cgi/wiki?ThornwoodWorkshop>

⁷ <http://hillside.net/conferences/plop>

- A pattern describes when it may or may not be applied when considering design constraints, and the consequences and trade-offs of its use.
- Usually gives examples of its application. For example, code that illustrates its application. Although design patterns describe object-oriented designs, they rely on practical solutions that have been implemented in major object-oriented programming languages.

There are many other types of software patterns, and design patterns, such as those used for organizational modeling, project planning, requirements engineering, and software configuration management, to name a few.

Currently, the most used patterns in the software community are the architectural patterns, design patterns and, more recently, organizational patterns:

Architectural patterns: An architectural pattern expresses a structure organization or scheme of software systems. Provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

- Design Patterns: provides a possible refinement of subsystems or components of a software system, and the relationship between them. Within them is commonly described the structure of components that can solve a general recurring design problem in a particular context.
- Organizational Patterns: These patterns can be placed at the most abstract level of requirements engineering. An organizational pattern presents a proposal for modeling an organization. These standards are based on the theory of organizations.

Quite often we talk about design patterns referring to any kind of pattern.

3.3.3 Patterns in domain engineering

Domain engineering is the process of reusing domain knowledge in the production of new software systems. It has been studied, being it a key concept in topics such as software reuse. A key idea in systematic software reuse is the “application domain”, which can be seen as a software area that analyzes various software systems sharing commonalities [34].

Most organizations work in only a few domains and since several organizations might work in similar domains, the construction of similar systems is a recurrent task. Of course there are variations within systems because the construction must meet different customer needs.

The discipline states that rather than building each new system variant from scratch, significant savings may be achieved by reusing portions of previous systems in the domain to build new ones.

The process of identifying domains, bounding them, and discovering commonalities and variability among the systems in the domain is called domain analysis (as seen in section 2.6 and section 2.7).

As part of domain engineering we should note two types of domains [34]:

- Vertical domains; they represent the system domain; they define the structure of the application areas, e.g. domain of medical record systems, domain of portfolio management systems, logistics, etc.
- Horizontal domains; they represent domains of parts of systems; they outline the structure of the software architecture used, e.g. Databases, GUI, Process scheduling, ...

Domain engineering encompasses three main process components: domain analysis, domain design and domain implementation.

- Domain Analysis; the purpose of this process is defining a set of reusable requirements for the systems in the domain
- Domain Design; whose purpose is establishing a common architecture for the systems in the domain
- Domain Implementation; that consist in implementing the reusable assets, e.g. reusable components, domain-specific languages, generators, and a reuse infrastructure.

With this in mind we can conclude that while conventional software engineering concentrates on satisfying the requirements for a single system, domain engineering concentrates on providing reusable solutions for families of systems.

However by using both disciplines designers can guarantee improvement on product quality by merging them and developing software based on domain engineering as seen in Illustration 10.

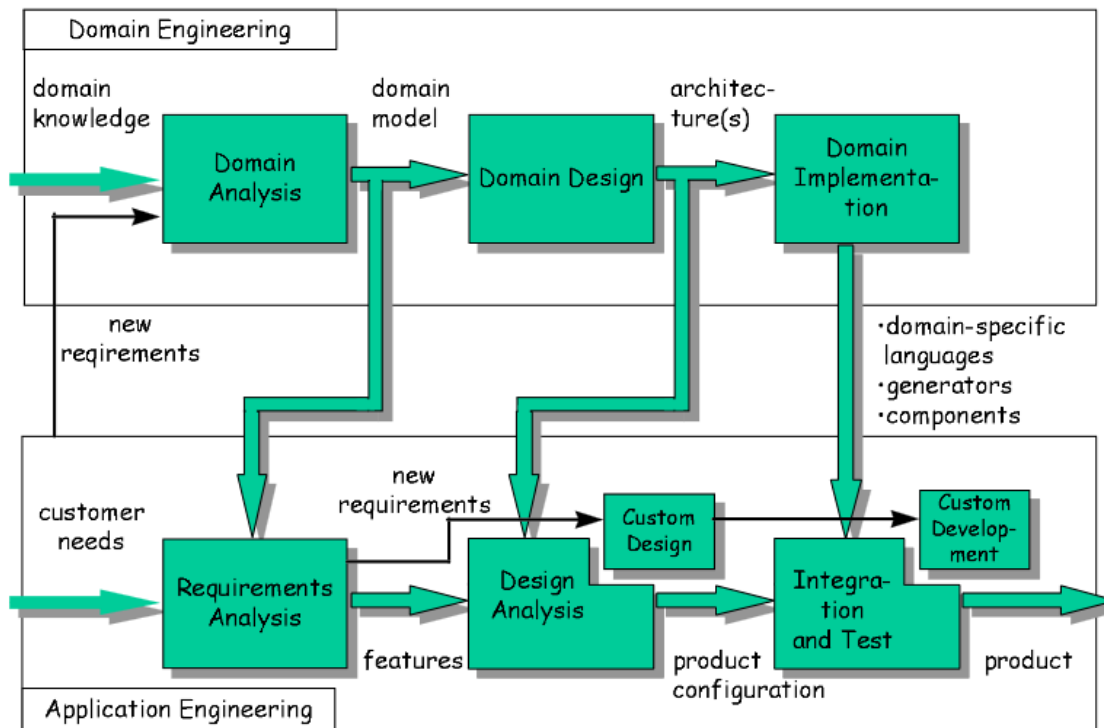


Illustration 10: Software development based on domain engineering [34]

By collecting domain knowledge and customer needs, we start with the development of our product. Requirements analysis is enriched by the domain model product of the domain analysis, which might lead to the generation of new requirements. Meanwhile we design the architectures within domain analysis and design also software configurations enhanced with it. As result of the domain engineering we obtain components or other reusable designs that will aid in the development partially or totally of the product. At any point during the application engineering process, new requirements coming from the environment, customer or development constraints may arise and expand the domain engineering process.

Domain engineering can be applied to a variety of problems, such as development of domain-specific frameworks, component libraries, domain-specific languages, and generators. Whereas the main resulting products are components – or reusable pieces of software-, they are not the only product of domain engineering. Some examples of such products include reusable requirements, analysis and design models, architectures, patterns, generators, domain-specific languages, frameworks, etc.

As we have previously stated, the purpose of domain design is to develop a “domain architecture” for the systems in the domain. It has been proven that certain recurring arrangements of elements have appeared and have been particularly useful in many designs, these arrangements are architectural patterns.

Each architectural pattern aims at satisfying a different set of user requirements as explained by Buschmann et al. [28]. He states that architectural patterns help us specify the fundamental structure of an application. Every development activity that follows is governed by this structure.

He describes some architectural and design patterns and their categorization, as well as the idioms introduced by those patterns. Finally, he also discusses that patterns do not exist in isolation, but that there are many interdependencies between them. Therefore he proposes some ideas to reflect the relationships namely pattern systems. These systems describe how the patterns are connected and how they complement each other [35].

He states there are two fundamental ways of integrating patterns:

- Refinement: One pattern refines the structure and behavior of another pattern to address a specific sub-problem or implementation detail.
- Combination: Two or more patterns arranged to form a larger structure that addresses a more complex problem.

Also, he discovered two main relationships regarding choice:

- Alternatives: Some patterns describe alternatives to one another. They address the same or a similar problem, but each pattern considers a slightly different set of forces. Thus, the patterns provide different solutions and have different consequences.
- Cooperation: Some patterns nicely complement one another, mutually reinforcing their structural and behavioral properties.

A pattern system therefore supports the effective use of patterns in software development.

Rumbaugh et al. [36] identify domain classes by analyzing the problem's domain, namely the problem statement, and the context, namely the application domain. They analyze the three upfront activities within object oriented development and design, to be precise, analysis, design and implementation. They state that during analysis, a model of the application domain is constructed without regard of eventual implementation. Later, during design, solution-domain constructs are added to the model. Finally, at implementation, both application-domain and solution domain constructs are made.

Fraser [37] analyses the separate approaches of domain analysis and pattern analysis. He takes a similar approach to the one in this project, meaning, the process of identifying and evaluating patterns of numerous solution variants and invariants is based on the induction through

exploration of solution instances, and he recognizes the importance of having experienced domain experts.

He concludes that the approach of performing in parallel but in a disjoint manner both patterns analysis and domain analysis provide an effective mechanism to develop context and domain models which help achieve reusable and highly functional designs.

He also supports the idea that the techniques of FODA would facilitate the use of existing design patterns, and would accelerate the discovery and evaluation of new design patterns, leading to an approach of “pattern-by-intent”.

Fülleborn and Heisel [38] presented some methods to support cross-domain reuse of analysis patterns. With the help of a case study, from the business domain, they illustrate their method. They discover analysis patterns in analysis models that not only reflect the knowledge of the domain, but also can be applied to other domains.

In Illustration 11 domain A is the solution-seeking domain and within it there is no solution for requirement A1. However in domain B, a solution exists for the requirement B2 that is also suitable to fulfill the requirement A1 of domain A. The challenge is to make this solution available for A1 in despite the different vocabulary.

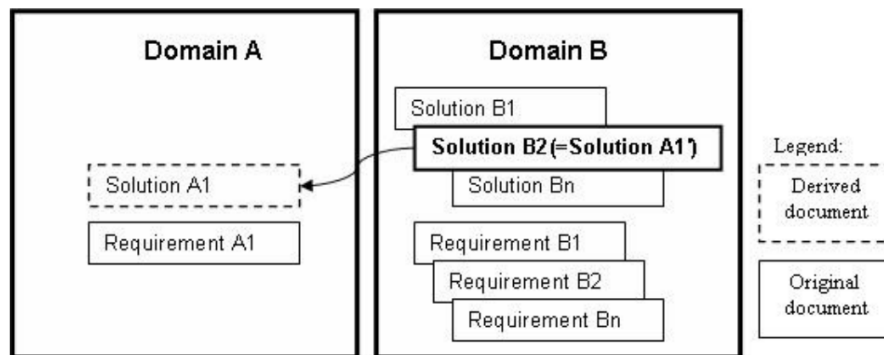


Illustration 11: Cross-domain reuse of problem solutions, analysis patterns [38]

Their method suggests first describing and modeling the domain-specific problems and their solutions. They suggest going through the problem statement or requirements document sentence by sentence and collecting sentences, classes identified and problem or context in an aligned manner. He proceeds to model each problem using UML class diagrams, and then he refines the model by abstracting in a way that it does not contain a problem statement anymore, only the solution for its domain. In order to create the analysis patterns he abstracts from the solutions for specific domains into a generalized model as seen in Illustration 12.

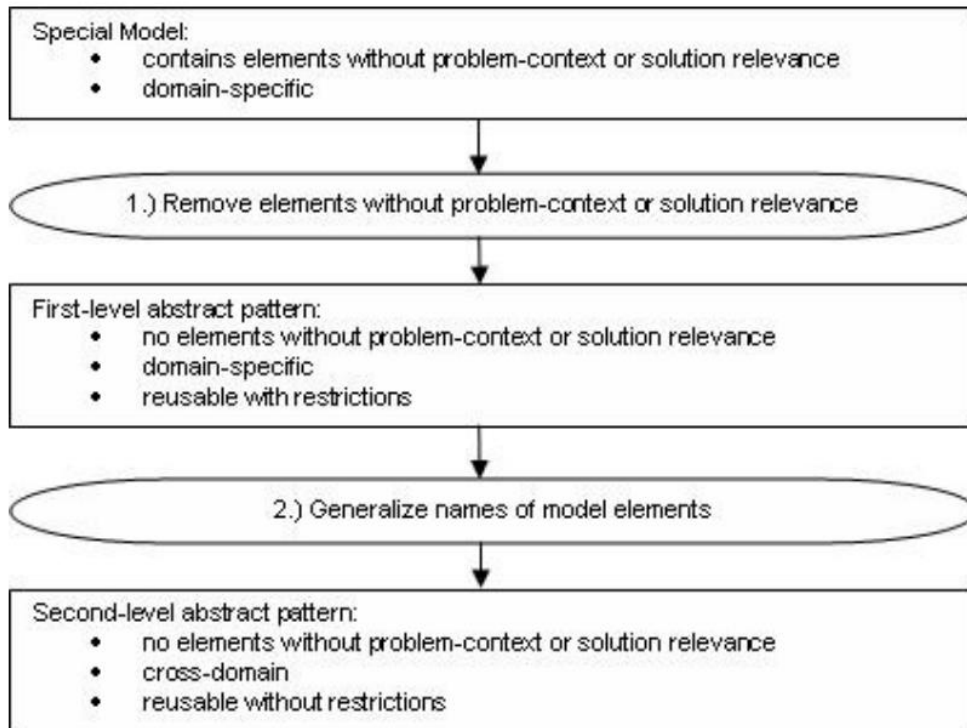


Illustration 12: Method for analysis patterns abstraction [38]

In a later paper and in collaboration with the technical university of Ilmenau, Fülleborn, Meffert and Heisel [39] propose a method for selecting and applying general design patterns, in particular they study the Gamma design patterns [25]. Also a key idea within the method is that designers should complement the patterns with the knowledge of the problems they solved in the context.

Step	Description
1	Choose a problem-bearing, domain-specific source code or UML model example
2	Annotate the chosen problem-bearing source code and UML models with <i>problem motives</i>
3	Perform transformations by applying design pattern under consideration to the chosen source code and UML models
4	Annotate the resulting source code and UML models with <i>solution motives</i>
5	Annotate the UML solution model of the cross-domain design pattern with the same solution motives as in Step 4
6	Perform <i>inverse</i> design pattern transformations to the existing design pattern UML solution models that are annotated according to Step 5

Table 1: Method for derivation of problem-context patterns by using general design patterns [39]

The method as seen in Table 1 takes as input domain-specific application examples and after some analysis annotates motives of the problem which is the result of steps 1 and 2. Later happens the application of design patterns and the respective annotation of the motives for the solution. As part of the complementation task of the designer making use of these patterns, he

should create a UML model of a general- cross-domain pattern plus the motives stated earlier. Later, the designer should also perform inverse design transformations in order to obtain the problem-context pattern that fits to the chosen design pattern.

To summarize, the author presents two methods to discover domain patterns to improve reuse of domain-specific knowledge at an abstract level.

In his work Tepandi et al [40] explore domain models and domain analysis methodologies applied to projects of real life. The work is developed within the framework of Archetypes Based Development and the domain of business. Their method bases on the use of archetypes and archetypes patterns to development models to finally develop applications. They discuss the importance to look for ways to minimize (better to completely avoid) changes in the domain logic and in the data source layers as these changes are risky and time consuming. They proved with a case study that the use of patterns improved the possibilities to fulfill user requirements only by making changes in the presentation or in the communication layers. As an aspiration for future work they propose that these changes could be made by end users or even at run-time.

For the purpose of this work we mention the authors above as a way of exemplification of the vast study of patterns in domain engineering; it is important to note that the literature is not limited to only them.

Not only they, but also other authors will be presented in other topics, such as representation of patterns, classification of patterns and methods for building catalogs.

3.4 Pattern catalogs used in this work

In this section we describe the main pattern catalogs that we used in order to discover our methodology, as well as the goals of our study.

3.4.1 Data model Patterns conventions of thought

In the book Data Model Patterns conventions of thought, Hay [41] implements a catalog for the domain “enterprise” that he extracted from his years of experience in industry data modeling.

He argues that a resulting system using his patterns would produce a robust and flexible design.

It is important to note that the models of the book are expected to be produced during strategic planning and requirements analysis phases of application development. Moreover, he uses a relational approach to present the patterns proposed.

3.4.2 Analysis patterns – reusable object models

In his book Analysis Patterns, Fowler [24] emphasizes in the boundary between analysis and design, saying that at analysis level the models produced are a reflection of our understanding of

the problem as well as “human artifacts”, whereas at design level it should reflect also the constraints and advantages of the solution proposed.

He also argues that in order to build effective models the domain experts should be involved in the process.

The patterns found in the book are groups of concepts that represent a common construction within the business modeling stage. Some of them are relevant to only one domain, whereas others can be abstracted to many domains. He also proposes supporting patterns that can be used alone, but the main reason he gives for presenting them is that they describe how to apply the analysis patterns in a real design.

3.4.3 The Data Model Resource Book

In the pattern catalogs of Silverston, namely “The data model resource book” volumes 1 to 3 we found a great amount of patterns in the several domains, general as well as specific for the healthcare domain.

We consider Volume 1 [42] since it includes specific patterns for the domains of products management, shipments, accounting and budgeting, as well as universal patterns for all enterprises.

Volume 2 [43], adds to the universal models from Volume 1, and shows industry-specific data constructs.

The scope of domains studied in this book is in the range of manufacturing, telecommunications, healthcare, insurance, financial services, travel, and eCommerce.

Volume 3 [44] on the other hand, focuses on the fundamental, underlying patterns that affect over 50 percent of most data modeling designs⁸. The author argues that the patterns of this book can be used to considerably reduce modeling time and cost, especially of designs where the designer is novice in the domain. They can be considered as standards and guidelines to increase data model consistency and quality.

We use these books to evaluate our source models as well as our pattern designs. The structure of Vol 3 is particularly interesting for us, since he provides for each pattern various alternatives ranging from very specific to very generalized ways of modeling.

⁸ <http://eu.wiley.com/WileyCDA/Section/id-352027.html>

3.4.4 Object-Oriented modeling and design

This book of Rumbaugh [36] promotes a better understanding of requirements, cleaner designs and more maintainable systems. It emphasizes that object-oriented technology is more than just a way of programming by applying techniques to the entire software development cycle.

The book contains case studies of industrial object-oriented applications developed by the authors, as well as examples and exercises that we use as patterns to our purpose.

3.4.5 Pattern Languages of Program Design

We use two volumes of the book Pattern Languages of Program Design. Volume 1 [33] summarizes the first conference on Pattern Languages of Program Design (PLoP) and Volume 2 [45] corresponding to the second conference of 1996. The conference represents a turning-point event that gave a public voice to the software design pattern movement.

These books comprise the work of several software professionals from around the world working together to capture and refine software experience that exemplified "good design."

By capturing these expert practices as problem-solution pairs supported with a discussion of the forces that shape alternative solution choices, and rationales that clarify the architects' intents, these patterns convey the essence to guarantee good software designs.

3.4.6 Other smaller pattern catalogs

In this category we mention other smaller pattern catalogs that we found as part of the current advances and studies in the area of patterns.

- Semantic Analysis Patterns [46]
- Analysis patterns for Patient Treatment Records [47]
- The SOAP Pattern for Medical Charts [48]
- A Pattern Language for Business Resource Management [49]
- A Confederation of Patterns for Resource Management [50]
- Service Provider: A Domain Pattern and its Business Framework Implementation [51]

3.5 Metadata standards

Metadata is often defined as “data about data” or as the Oxford English Dictionary [52] defines it: *“A set of data that describes and gives information about other data”*.

The purpose of metadata is “to facilitate search, evaluation, acquisition, and use” of resources [53]. Moreover, in the case of educational resources, the purpose is also “to facilitate the sharing and exchange of learning objects, by enabling the development of catalogs and inventories

while taking into account the diversity of cultural and lingual contexts in which the learning objects and their metadata will be exploited” [53].

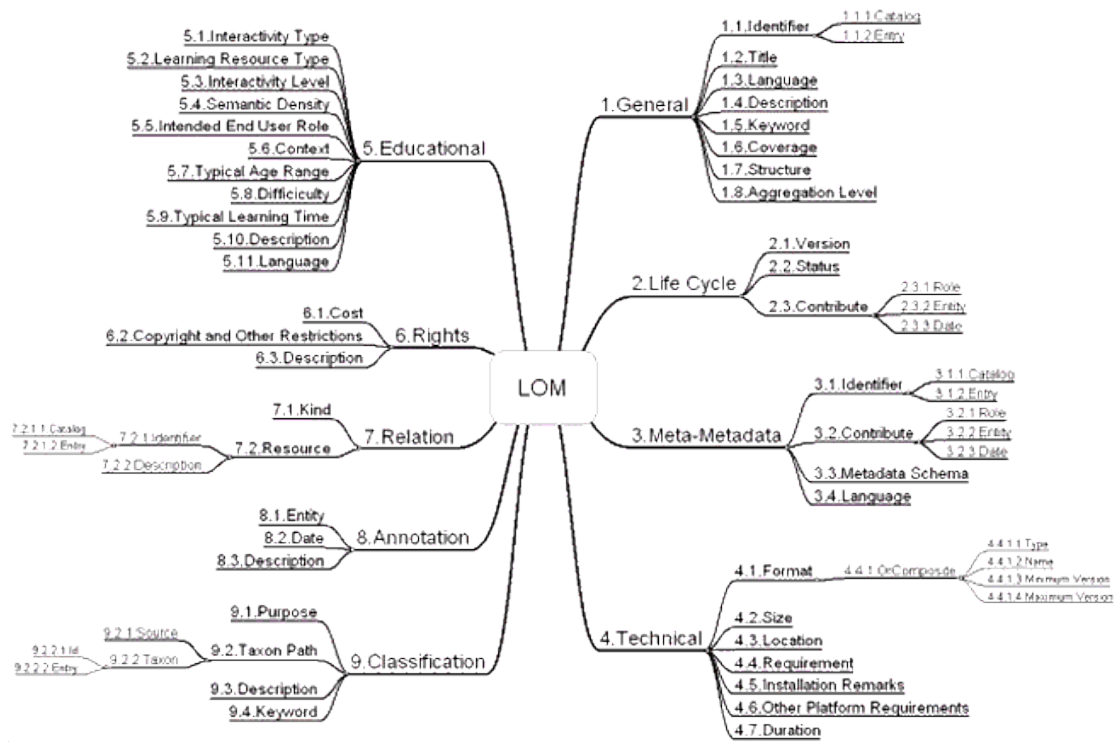


Illustration 13: The elements and structure of the LOM conceptual data schema [53]

Illustration 13 presents a graphical illustration of the elements in the data schema, which shows how the elements are divided into nine top level categories: General, Life Cycle, Meta-Metadata, Technical, Educational, Rights, Relation, Annotation, and Classification. Each of these branches comprises several elements, some of which are leaves; others are sub-branches which lead to leaves.

In the context of healthcare Löbe et. al and HL7 International have developed metadata standards for specifying metadata items in clinical research and life sciences [54] and for electronic health information [55] respectively.

3.5.1 Health Level Seven, HL7

Health Level Seven International (HL7) is a non-profit organization involved in the development of international healthcare informatics interoperability standards (e.g., HL7 v2.x, v3.0, HL7 RIM).

HL7 and its members provide a framework (and related standards) for the exchange, integration, sharing, and retrieval of electronic health information. It aims to enable the electronic communication in healthcare facilities (especially in hospitals).

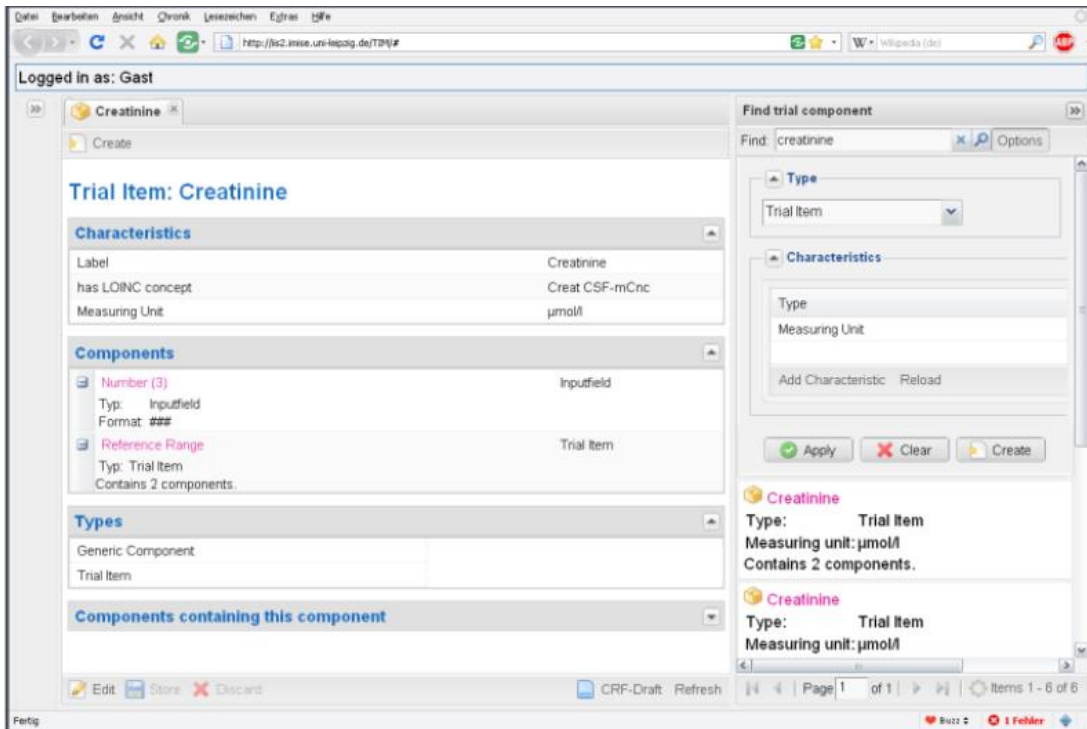


Illustration 15 Creatinine Trial Example [54]

As a way of illustration we include a screenshot of the software application showing the applied metadata standard. We find in Illustration 15: the right side corresponds to the search for items labeled “creatinine” using a type (trial item) and a characteristic (measuring unit) restriction. In the left side we find a detailed view of the first item of the result set including characteristics and subcomponents as part of the metadata exemplified.

3.5.3 Street - Address Meta Data Standard

Street addresses are the location identifiers most widely used by state and local government and the public. Street addresses are critical information for administrative, research, marketing, mapping, GIS, routing and navigation, and many other purposes.

In the context of this work we consider important to analyze metadata standards referring to addresses and make a comparison of the common terms within geographical regions. Therefore we analyze: Street Address Data Standard, from USA and ANZLIC Metadata, from New Zealand.

3.5.3.1 *Street Address Data Standard*

This standard was established by the Federal Geographic Data Committee (FGDC) of the United States of America, which promotes the coordinated development, use, sharing, and dissemination of geographic data [56].

The Street Address Data Standard provides, in four separate parts, data content, classification, quality, and exchange standards for street, landmark, and postal addresses:

- ***f* Data Content** provides semantic definitions of a set of objects. This part specifies and defines the data elements that may appear in or describe street, landmark, and postal addresses.
- ***f* Data Classification** provides groups or categories of data that serve an application. Classification data are the attributes common to elements of a group. This part defines classes of addresses according to their syntax, that is, their data elements and the order in which the elements are arranged.
- ***f* Data Quality** describes how to express the applicability or essence of a data set or data element and include data quality, assessment, accuracy, and reporting or documentation standards.
- ***f* Data Exchange** describes how to produce or consume packages of data, independent of technology and applications that will facilitate moving data between agencies and systems.

Among the contents of the Address elements this standard proposes we consider the ones seen in Illustration 16:

- Address Elements**
- Address Number
 - Street Name
 - Building, Floor, and Unit
 - Landmark Name
 - Larger Areas (Place, State, ZIP)
 - USPS Postal Address Elements (PO Box, etc.)
 - USPS Address Lines
- Attribute Elements**
- Address ID
 - Address Coordinates
 - Descriptive Attributes
 - Quality Testing Attributes
 - Address Lineage Attributes

Illustration 16 Basic Address Elements in the Street, Address Metadata Standard [56]

3.5.3.2 ANZLIC Metadata Profile

ANZLIC — the Spatial Information Council is the peak intergovernmental organization providing leadership in the collection, management and use of spatial information in Australia and New Zealand [57].

As an example we present the metadata for address (see Illustration 17) and concrete instantiation of the metadata corresponding to the Wellington City Council (see Illustration 18).

Address	
Name (Number)	CI_Address (380) [UML]
Short Name	Address
Definition	location of the responsible individual or organization
Obligation/Condition	Use obligation/condition from referencing object
Maximum Occurrence	Use Maximum Occurrence from referencing object
Data Type	Class <<DataType>>
Domain	Lines 381-386

Illustration 17 Address information according to ANZLIC Metadata Profile [57]

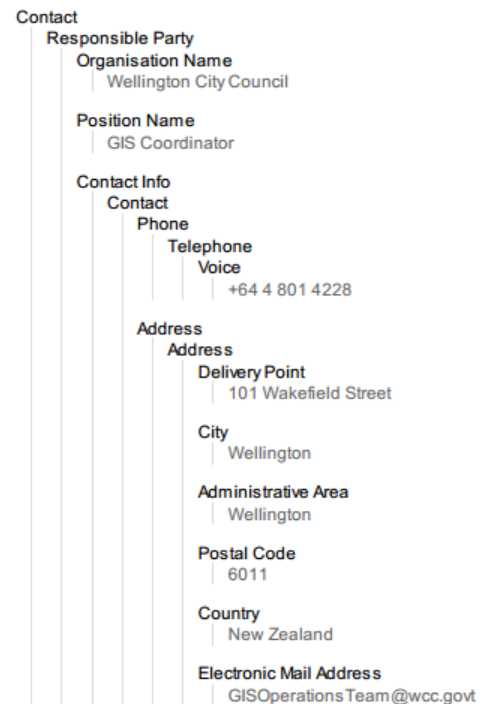


Illustration 18 Wellington City Council's Address and contact info using ANZLIC Metadata¹⁰

¹⁰ Example taken from the Wellington's City Council Contact information Site:
<http://koordinates.com/layer/2222-wellington-city-parks-and-gardens-tracks-and-walkways/metadata/#>

4 Methodology Framework

In this section we study and summarize the relevant methodologies within the context of this work. We analyze some authors and we then explain how they motivated the decisions taken within the development of our own methodologies. We explain our adaptations we produced as well as our own methodology designs.

We first explain the core methodology that helped us design the patterns catalog, then we analyze the methodology for pattern identification, later we explain the pattern classification methodology and finally we describe the design of the metamodel that will be used as design for the pattern catalog.

4.1 Building a catalog of Patterns

In this chapter we explore the method proposed by Carreon [58] for building a catalog of patterns of functional requirements for ERP (Enterprise Resource Planning).

4.1.1 Definition and representation of a pattern

According to Carreon a pattern of functional requirements is a reusable artifact found during the various activities of the requirements engineering process and represents in a structured and abstract way one or more functional needs of an organization.

The notation used in his work for the construction of patterns is as follows:

*“Pattern **goal** <must> property-to-fulfill {variables}”*

where:

- The **goal** of the pattern describes the objective to be achieved by pattern.
- The **property to fulfill** describes what must be met to achieve the goal.
- The **variables** describe the part of the pattern will depend on the characteristics of each project when used. They might not always appear in the patterns.

In Table 2 we can find the tabular structure, that the author uses as a template to represent patterns.

The table illustrates the definition of a pattern of functional requirements for ERP, where the fixed part corresponds to the example above.

Identifier: <pattern number> Name: <Name of the pattern>			
ERP Module	<ERP Module>		
Keywords	<List of keywords>		
Version	<Version number> (<Date>)		
Author	<Author of the version> (<Organization>)		
Source	<Source of the current version>		
Purpose	<General purpose of the pattern>		
Description	<Description of the pattern>		
Fixed Part	<i>Purpose <must> property to fulfill {variables}</i>		
	Parameter's name	Metric's name	Metric's Type
	{parameter}	{metric}	{Type of the metric}
Extension	Identifier: < extension's number> Name: <Name of the extension>		
	<i>Purpose of the extension <must> property to fulfill {variables}</i>		
	Parameter's name	Metric's name	Metric's Type
	{parameter}	{metric}	{Type of the metric}
	...		
Comment	<comments>		

Table 2: Tabular Template corresponding to the Functional Patterns for ERP. [58] (translated)

The pattern is composed of two main parts:

- Fixed part. Corresponds to the definition of the pattern itself, it's a phrase that expresses the objective literally, in other words, the general purpose of the pattern and properties.
- Extension/s. One or more extensions of the general part. They are optional, especially in cases where the general part is applied. They could also be included in the catalog as multiple patterns.

The requirements templates help to express and identify the requirements easily. In this way and in order to formalize functional requirements, they created the template that shows this structure.

4.1.2 Patterns Classification

In order to classify the patterns, and thus have a catalog structure, the author proposes to use the model quality characteristics of ISO / IEC 9126-1.

Since in this work we don't aim to describe the whole quality standard nor the process to get the final classification, we present below a partial summary of the classification established by the ISO/IEC extracted from a document of Botella in [59], document in which the author analyzes the quality standard defined by the International Organization for Standardization (ISO).

The main idea behind this standard is the definition of a quality model and its use as a framework for software evaluation. An ISO/IEC 9126-1 quality model is defined by means of general "characteristics" of software, which are further refined into sub characteristics, which in turn are decomposed into attributes, yielding to a multilevel hierarchy; intermediate hierarchies of sub characteristics and attributes may appear.

Also, at the bottom of the hierarchy appear the measurable software attributes, whose values are computed by using some metric.

The quality model introduced in the standard is common for external and internal quality aspects, whilst another model for quality in use is introduced. Table 3 enumerates the six quality characteristics defined in the ISO/IEC 9126-1 internal/external quality model and their decomposition into sub characteristics.

The attributes that can be measured during the development process are referred to as internal. The external behavior can be measured during the testing process, and finally the user's view of quality is shown measuring the quality-in-use attributes.

Characteristics	Subcharacteristics
Functionality	suitability
	accuracy
	interoperability
	security
	functionality compliance
Reliability	maturity
	fault tolerance
	recoverability
	reliability compliance
Usability	understandability
	learnability
	operability
	attractiveness
	usability compliance
Efficiency	time behaviour
	resource utilisation
	efficiency compliance
Maintainability	analysability
	changeability
	stability
	testability
	maintainability compliance

Table 3: The ISO/IEC 9126-1 internal/external quality model [59]

Botella mentions that the standard doesn't consider non-technical factors (e.g. from management, economics or politics). Considering the importance of them, the author strengthens the classification of the ISO/IEC document by using a convenient and coherent way to handle non-technical factors.

It consisted in structuring the non-technical factors in the same way as the technical ones, in other words, by adding a set of high level characteristics – vendor vs. product characteristics - and the respective sub characteristics. As a result, they produced an enlarged quality model including both types of factors as shown in Table 4.

Characteristic	Subcharacteristic	Attribute
Vendor	Economy	Market share
		R+D budget
	Reputation	Years in the market
		Certification of the process
		Directory of clients
	Support	Distribution channel
		Offered services
		Location
Product	Distribution	Commercialisation strategy
		Licence cost
	Stability	Time in the market
		History of versions

Table 4 : A categorization of non-technical factors following the ISO/IEC 9126-1 style [59]

The great advantage of using this classification is that the quality features expressed in the document - features, attributes and sub-features – is that they are general for any enterprise software application.

Carreon uses a subset of the classification above in his work.

The classification aims to facilitate search and identification and structuring patterns during requirements definition within a particular project.

4.1.3 Method

This section outlines the steps followed by Carreon for creating the first catalog of requirements for ERP.

First of all, the author proceeded to study the domain of ERP, ERP modules and dependencies, and collecting the literature base for collecting the patterns: requirement books from real projects.

Then he performed the pattern extraction process by following a bottom-up process. That is, he used the requirements books of real projects for building the patterns.

The steps followed for the construction of the catalog of patterns of functional requirements for ERP are the following:

1. An analysis of the ERP domain.
2. Extract the functional requirements of the books of requirements.
3. Perform semantic analysis and refine the functional requirements.
4. Insert in the catalog of “candidate patterns” those requirements that might become patterns after coincidences search.
5. Create and / or refine the pattern and insert or update the catalog of patterns.

The tasks and the information flow that comprise the method can be found in detail in Illustration 19. Note that the first step: Analysis of the Domain is not included in the diagram.

1. Extraction of functional requirements.
2. Semantic analysis and refinement.
3. Insertion in the catalog pattern candidates.
 - a. Matching candidates catalog.
 - b. Validation of coincidences and update or insert into the catalog of candidates.
4. Insertion in the pattern catalog.
 - a. Refinements of the pattern.
 - b. Formalization and storage in the pattern catalog.

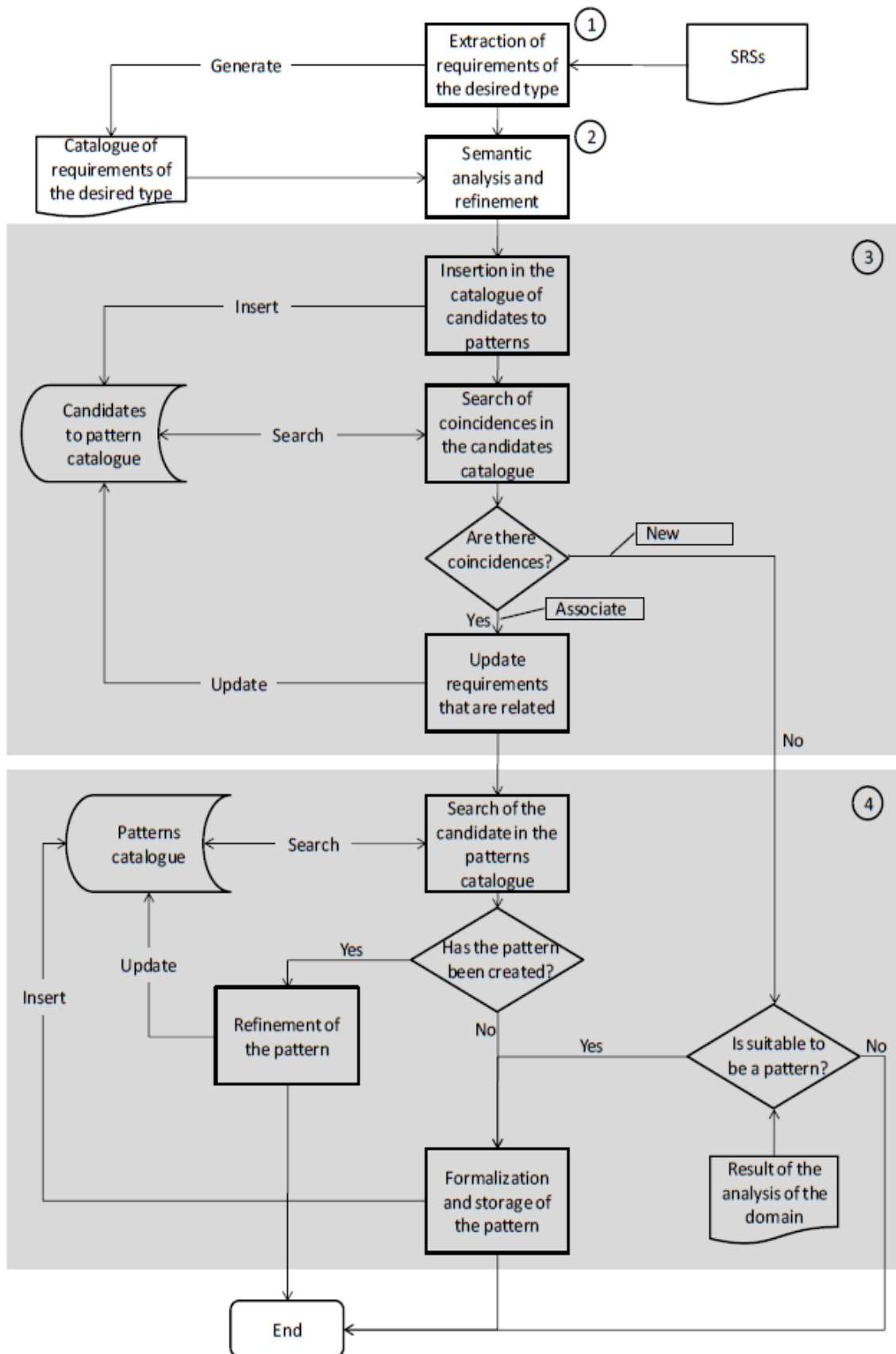


Illustration 19: Steps for the construction of a SRP Patterns Catalog [58]

4.2 Pattern Identification

In order to discover patterns from the domain and extract them we analyzed the methods that some authors successfully used to discover patterns in areas such as database design, object-oriented design and domain design.

4.2.1 Analysis Patterns

In these area we take as main work the one of Fowler [24], where he discusses patterns as being the main entities, as they are a key technique to discussing and capturing what makes good design. The importance of his work is described in section 3.4.2 Analysis patterns – reusable object models.

Another work is the one on Semantic Analysis Patterns from Fernandez [46], where he describes Semantic Analysis patterns as a pattern that describes a small set of coherent Use Cases that together describe a basic generic application.

The Use Cases used for his research are selected in such a way that the application can fit a variety of situations.

Semantic Analysis Patterns (SAPs) focus on typical Design patterns are closer to implementation, they focus on typical design aspects, i.e., user interfaces, creation of objects, basic structural properties. They don't necessarily apply to any application.

His method for producing instances of SAPs consists in selecting Use Cases to then generalize the original pattern by abstracting its components and later deriving new patterns from the abstract pattern by specializing it or by using analogy to directly apply the original pattern to a different situation as seen in Illustration 20.

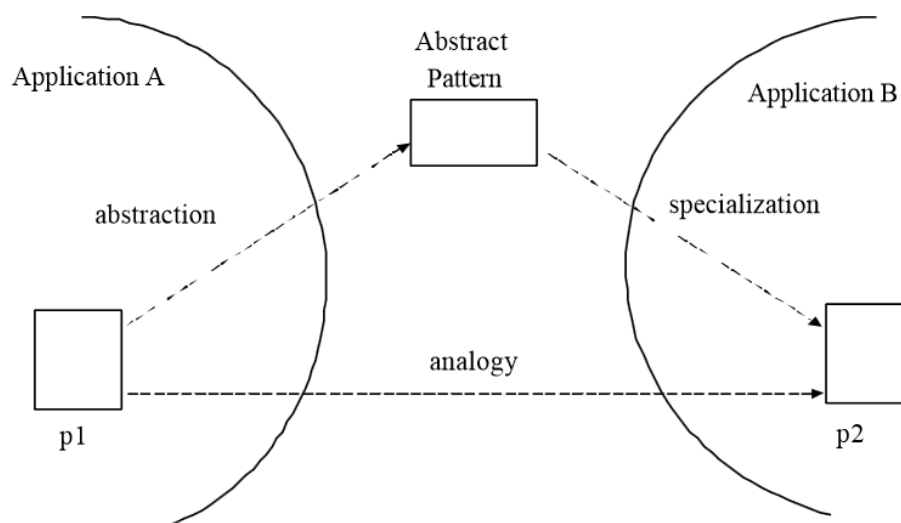


Illustration 20: Semantic Analysis Pattern (SAPs) generation method [46]

We use the idea of producing patterns by abstracting the solutions found in one application and although we don't produce solutions to prove that our patterns can be applicable, we compare them to other solutions as well as to pattern catalogs available.

4.2.2 Patterns in Database Modeling

Thonggoom [60] worked on trying to reuse domain knowledge contained in data base schemas to aid in database modeling within the information system development process.

His methodology includes database reverse engineering in order to discover reusable instance patterns containing knowledge about application domain. The reusable elements proposed by the results of his work are Entity Instance Patterns (EIP) and Relationship Instance Patterns (RIP) which are stored in a repository.

As an example we show Illustration 21 where we can see that:

- An EIP is a pattern of a single entity and its properties. i.e. Order.
- A RIP is a binary relationship with cardinality constraints between two entities.

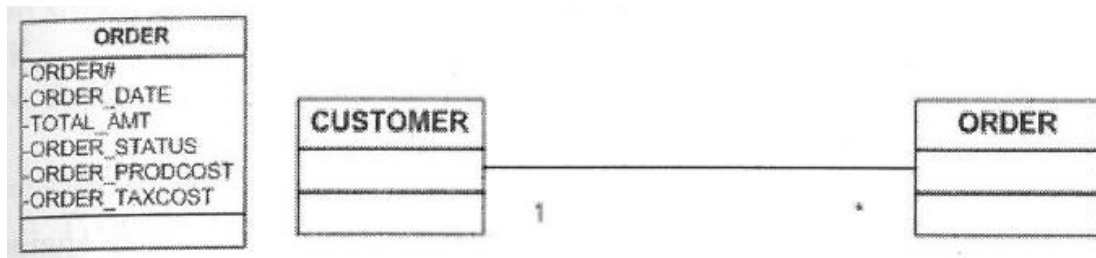


Illustration 21: Example of EIP and RIP [60]

Entity-Relationship (ER) model is a well-known conceptual modeling formalism, easy to understand, powerful to model real-world problems and easy translation to a database schema.

Thonggoom considers these strengths of the ER model to make it the base meta-model to store his patterns as seen in Illustration 22 and also for making easier the practical use of them.

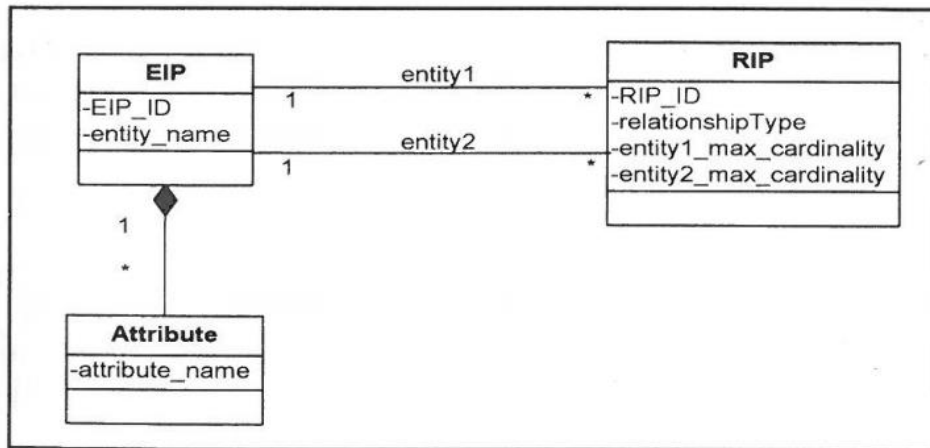


Illustration 22: Meta-model for EIP and RIP [60]

The benefits encountered by the use of the patterns found after application of his method proved to:

- Improve the process of conceptual modeling design.
- A novice designer can easily use the knowledge about application domain contained in the patterns, therefore improving the quality of the model and the performance of the designer.
- EIR and RIR simplify the work of experienced designers; they are domain-specific and therefore easier to understand and reuse.

In the context of our work, we perform reverse engineering as well in order to extract and map the information from database schemas. This approach is mainly used to discover patterns of single elements as well as patterns of three elements, as is the case of patterns abstracted from a binary relationship.

4.2.3 Patterns in object-oriented design

Han et al. [61] propose a methodology for building object-oriented design fragments. Design fragments are a new kind of artifact that is composed of several element-clusters, is a whole design that is developed and indexed based on common patterns.

Based on previous designs taken from the enterprise domain, the method arranges clusters of elements based on combinations of keywords as seen in Illustration 23.

Keyword-based clusters	Designs in the cluster
R ₁ (employee, agreement, specific item)	D ₁ , D ₅ , D ₆ , D ₇
R ₂ (company, employee, agreement line item)	D ₂ , D ₄
R ₃ (employee, agreement, agreement line item)	D ₂ , D ₅
R ₄ (employee, agreement line item)	D ₄ , D ₈
R ₅ (company, employee, agreement)	D ₁ , D ₂
R ₆ (employee, specific item, step)	D ₇ , D ₁₀
R ₇ (agreement, specific item, agreement line item)	D ₃ , D ₅
R ₈ (agreement, specific item, performance)	D ₃ , D ₆
R ₉ (employee, specific item, performance)	D ₆ , D ₁₀
R ₁₀ – Remainder	D ₉

Illustration 23: Design Clusters based on Keywords [61]

These clusters are found in one or more source designs, and are then indexed according to similarities to store them later in a repository that will aid efficient reuse.

Although the final implementation of the method is not yet available, we find the clustering of keywords important and thus applicable to our work. We decided to use this approach in order to discover structures and substructures, meaning groups of more than one element, appearing in the models and across them in order to identify patterns and composite patterns.

4.2.4 Composite patterns

Another important work we analyzed to build our search method is the work of Riehle [62], where he analyzes the issues while documenting how patterns work together.

He takes as input the Design Patterns book [25] and further refine the definition of the patterns in there. He afterwards discovers small groups of patterns that often work together. He proposes an analysis method for building the pattern compositions by means of grouping atomic or composite patterns.

As an example, we analyze the following patterns described by Riehe as:

The **Mediator pattern**; it serves to decouple, manage and integrate several Colleague objects by means of a coordinating Mediator.



Illustration 24: Role diagram of the mediator pattern [62]

The **Observer pattern**; it serves to decouple Observer objects from a Subject object while maintaining state dependencies. The maintenance is achieved by using events for inter-object communication.

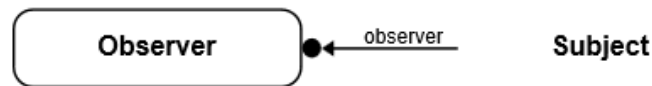


Illustration 25: Role diagram for the Observer Pattern [62]

The **Chain of Responsibility pattern**; it serves to define an object chain along which requests are passed until they are handled. Thus, by configuring the chain, the receiver of a request can be defined dynamically.

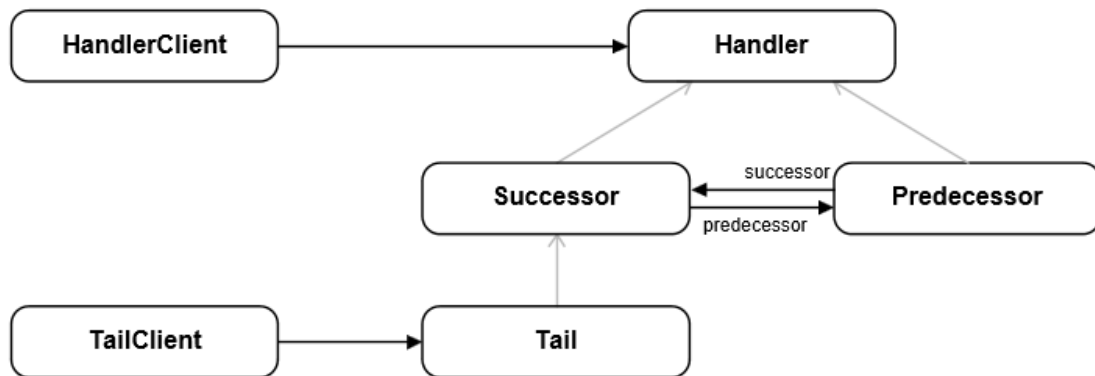


Illustration 26: Role diagram of the Chain of Responsibility pattern [62]

He built a matrix of relationships based on the participant classes of each pattern. He then united them in the Bureaucracy pattern as an example of a complex composite pattern as seen in Illustration 27, where the structure of the roles applies the chain of responsibility pattern, whereas the manager-subordinates relationship obeys the observer pattern and the mediator pattern to manage the children of the manager role.

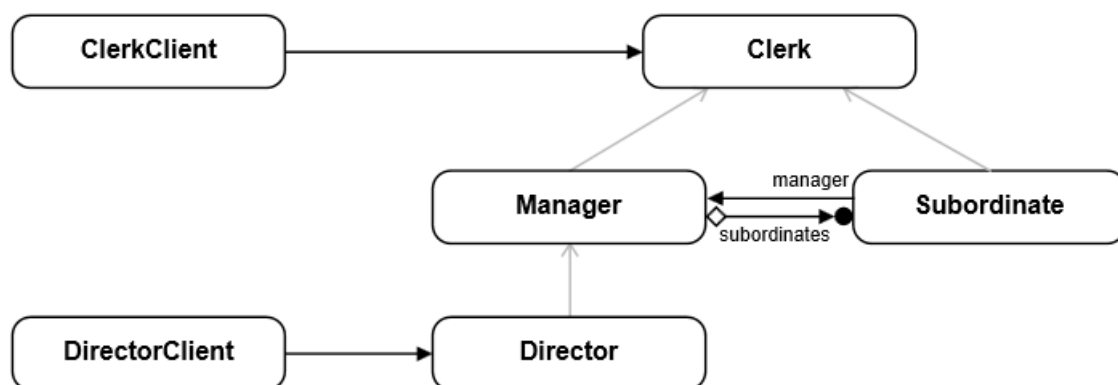


Illustration 27: Role diagram of the Bureaucracy pattern [62]

Vlissides published a report [63] about Riehle's work citing some other composite patterns extracted from the Design Patterns book [25] and, of course, his own experience.

Some examples of such composite patterns are:

Template Method – Factory Method; this pattern takes the capabilities of both patterns in order to merge behavioral description and responsibility. Whereas the template method separates the variant and invariant parts of an operation, it isn't specific about the behavior, only about responsibility. On the other hand, the pattern factory method concedes behavior to the subclasses.

Factory method can therefore serve as primitive operations to template methods within this composite pattern.

Prototype – Abstract Factory; normally, prototype can be used by abstract factory for the creation of products. This can add flexibility to the design and moreover reducing the number of classes that Abstract Factory introduces as seen in Illustration 28.

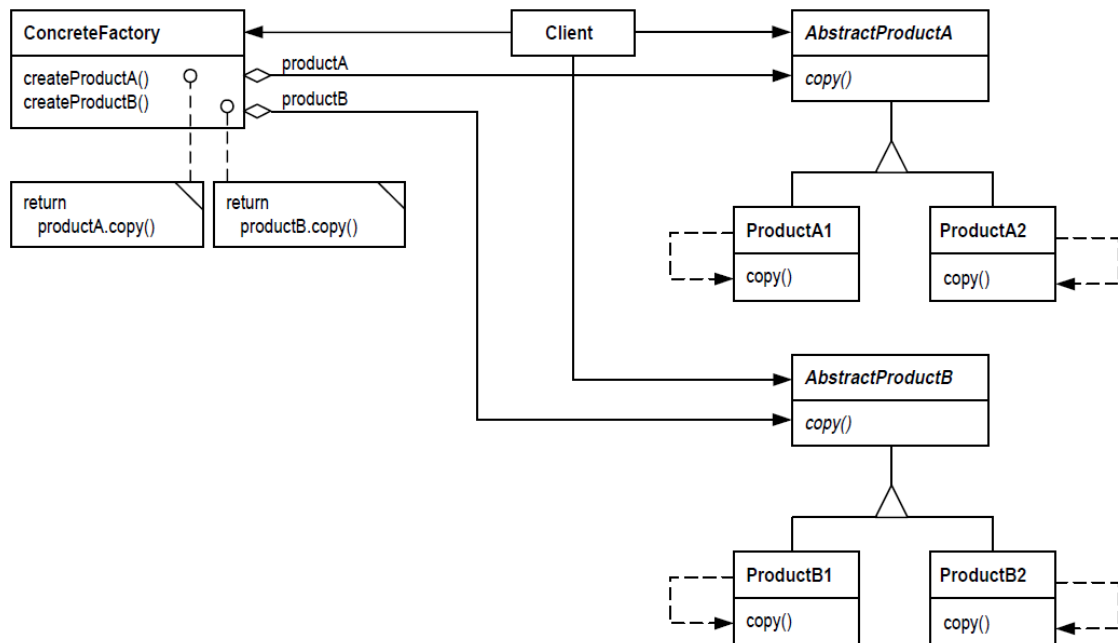


Illustration 28: Prototype - Abstract Factory composite pattern [63]

Some other composite patterns include: Composite-Decorator, Composite – Flyweight, Composite – Iterator – Visitor, among others.

With these works we discovered the importance of being able to describe both atomic and composite patterns within the context of our work. We used this idea as baseline to design the metamodel to represent the patterns catalog as well as to guide the discovery of possible complex patterns.

4.2.5 Cross-Domain design

Fülleborn presents in [38] a method for creating cross-domain analysis patterns with the purpose of reuse as explained in section 3.3.3 Patterns in domain engineering.

He uses the method to discover domain patterns to improve reuse of domain-specific knowledge by performing abstractions and forming domain-free or cross-domain patterns as shown in Illustration 29.

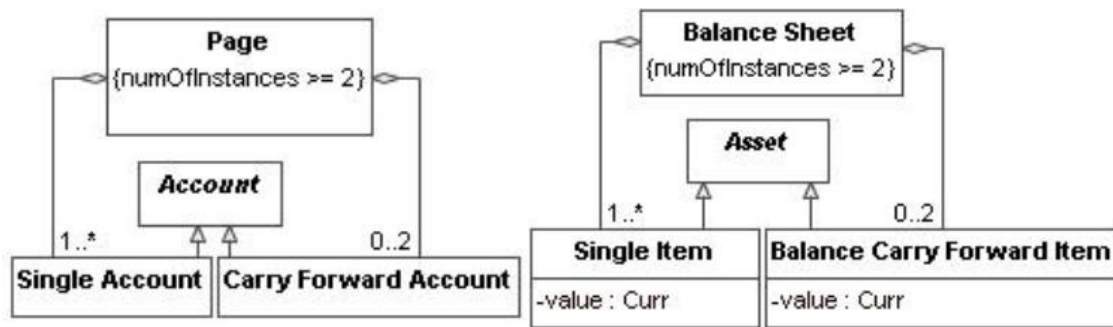


Illustration 29: Cross-Domain Pattern example. Left: Solution to solve “Invoice” problem. Right: Solution to “Balance end of year close” problem [38]

We consider important the fact that we can also identify structures or patterns that can be reused in different domains. Therefore we use pattern catalogs not only of the healthcare domain, but also of the business domain and of other domains in order to identify structures that are repeated in those domains.

4.3 Classification of patterns

According to our study we discovered that domain designs and therefore domain patterns vary in their granularity and level of abstraction.

Because there are many design patterns, we need to develop proper methodologies and techniques how to organize them according to common properties.

As analyzed within the literature, we know that the type of properties is not fixed and may include criteria such as structure, intent, or applicability.

Different classification schemas can have different dimensions. A two-dimensional schema, for example, uses two criteria in the classification process as we see in the patterns catalog of GoF [25]. Their design patterns catalogue is one of the most widely known pattern classification schema, it is two dimensional and classify the patterns according to purpose and scope as seen in Table 5.

By Purpose				
		Creational	Structural	Behavioral
By Scope	Class	<ul style="list-style-type: none"> Factory Method 	<ul style="list-style-type: none"> Adapter (class) 	<ul style="list-style-type: none"> Interpreter Template Method
	Object	<ul style="list-style-type: none"> Abstract Factory Builder Prototype Singleton 	<ul style="list-style-type: none"> Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy 	<ul style="list-style-type: none"> Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Table 5: GoF Classification Schema [25]

Buschmann [28] the classification schema should be simple and easy to learn, in this manner, the catalog complies with its purpose of helping the designer, it should be easy to use, to classify and select patterns. The schema should also have only few classification criteria in order to reduce complexity and ambiguity.

Finally the schema should reflect pattern properties, representing the main properties of the patterns to be classified as well as becoming a roadmap to aid selection of the appropriate pattern.

4.3.1 Criteria

A criterion should reflect a natural property of a pattern. It should divide patterns into different categories each reflecting a specific property. We present in the following sections some of the most common criteria found within our research.

4.3.1.1 Granularity

Patterns in a software system can also be found categorized depending on the level at which they address the system or the system development process.

It is sometimes difficult to draw the border line between different pattern categories. Therefore the same pattern could be classified in different granularity categories.

As an example we present the Buschmann [28] categorization according to granularity:

- **Architectural patterns:** refer to a software system at high level.
- **Design patterns:** focus on subsystems and components.
- **Idioms** (coding patterns): address the lowest level of a software system.

In the same manner, Silverstone [44] creates a categorization of data model patterns in levels 1 to 4, where:

- **Specific pattern** are the ones that illustrate and communicate information requirements, show specific attributes within entities. Correspond to the L1 or L2 patterns.
- **Generalized pattern** are the ones that provide a sound foundation for database design, incorporate flexibility and the application of patterns for a better design. Correspond to the L3 or L4 patterns.

Also, within the Multi-View modeling of Kutsche [10], he describes the levels in which a UML design, for us a UML Class diagram, can be classified. Those are the phases 1, 2 and 3.

- **Phase 1:** Gives a general overview to the modeling intention and contains only the most important concepts and ideas.
- **Phase 2:** Displays much more detail, finally containing all analysis and design phase refinements.
- **Phase 3:** Has all the detail to serve as a basis for implementation

We consider this criterion quite important, that is why we choose to implement it as part of our catalog taking inspiration from the works of the cited authors.

4.3.1.2 *Domain*

A natural way to classify patterns is to categorize them according to the domain where they are applied. For example, the oldest patterns were originated from the field of urban construction by Alexander [23].

As an example from software engineering, patterns are mostly known for, but not restricted to, solving problems of software design and architecture.

Different patterns have been proposed to solve design problems in various areas of software engineering. Patterns have been applied in the domain of real-time systems, communications, distributed systems, user interface, embedded systems, etc. [42] [29]

Thonggoom [60] uses the classification according to entity categories, that are domain knowledge and used as a tip for identifying candidate entities, they are taken from the WordNet hierarchy of terms.

It is unlikely that a pattern applied in one discipline becomes relevant to another, but for them to do, normally an abstraction is necessary as we see within the universal patterns catalog [44].

For the stated above we may conclude that according to the domain a pattern can be:

- **Domain-Specific** pattern.
- **Cross-domain**, Universal, or general pattern.

4.3.1.3 *Some other criteria*

In this section we present other criteria that we discovered within our research, but that we are not using directly in our design.

4.3.1.3.1 Purpose

The purpose represents the kind of problems the pattern solves. This criterion is the most useful since it describes the concrete situation where the pattern applies [28] [25].

4.3.1.3.2 Paradigm

Patterns are a good practice within software development; therefore they were created not only in object-oriented software systems design, but also in imperative programming.

We focus in object oriented paradigm patterns, which are described in terms of concepts like objects, classes, object composition, and class inheritance [28] [32] [25].

In imperative programming, patterns are mainly expressed in terms of other concepts such as functions and procedures.

4.3.1.3.3 Scope

Patterns can then be classified according to the features responsible of implementing them.

For example, in the object-oriented paradigm, design patterns can be implemented using either object composition or class inheritance [25] [36].

4.3.2 **Domain Patterns Classification Schema**

Based on the findings above, we needed to define the characteristics of the criterion we would later choose to represent the patterns classification.

In this section we first describe the criteria characteristics and then we proceed to describe the criteria chosen to design the classification schema.

4.3.2.1 *Criterion characteristics*

Three main characteristics were ideal to perform the decision. Those are:

- **Conceptual:**
A conceptual criterion defines categories that are conceptual subsets of the criterion.
- **Universal:**
A universal criterion defines categories for all patterns. This enforces that it should not only contain existing patterns but also unwritten ones.

- **Specific:**

A specific criterion defines specific categories. Specific categories provide more detailed information to the user of the classification scheme.

4.3.2.2 *Criterion Classifiers*

In this section we describe the classification of domain patterns so that we can refer to families of related patterns.

We identified the advantages of designing a two-dimensional classification schema, which facilitates the finding and reusing of patterns.

The main criteria we use as classifiers within the schema are Level of Design and Domain as described next.

4.3.2.2.1 *Level of Design*

Since the sources used for this work came from multiple stages of the software development process we identified some patterns that correspond to Business modeling at early stage, as well as advanced. Also some database design at design stage as well as full implementations. Finally, we discovered class diagrams, some used to illustrate the advantages of the modeling technique and some others to be used in real life developments.

This heterogeneity within sources has originated a need of classification of patterns according to Level of Design.

The level of design of the patterns found within the sources can be organized according to:

- **Early Design**

Gives a general overview of the problem domain that is simple. Contains patterns that illustrate and communicate information requirements, show specific entities and attributes within entities.

- **Intermediate Design**

Gives an overview of the problem domain that is more advanced than the previous, the application of abstraction of some concepts can be seen. It is a hybrid approach between early and advanced design.

- **Advanced Design**

Gives a solution that can be found when the design is ready for implementation. Containing patterns that are foundation for database design, they incorporate flexibility and the application of design patterns, for example.

We guide the instantiation of patterns according to the analysis made by Silverston (see Table 6) about the benefits of specific, early design, and generalized, implementation design, styles of modeling.

BENEFITS OF A MORE SPECIFIC STYLE OF MODELING	BENEFITS OF A MORE GENERALIZED STYLE OF MODELING
Easier to understand model.	More flexible. Can more easily accommodate additional data and/or changes to data, without needing to change the data model. Provides the ability to meet more current and future needs.
Easier to use as a way to communicate with nontechnical audiences, validate and gather requirements, and define scope of the data requirements.	More consistency. Higher-level patterns tend to result in data models that have the same type of structures and can promote consistency and standardization, either within the same data model or across data models.
Good way to start in order to understand the data requirements before generalizing the model.	Basing the physical database on more generalized data models allows more use of common routines to manage and access data, because the data structures tend to be more similar.
Can specify and enforce more business rules directly via the data model.	Can sometimes provide more power and capabilities by combining various types of data within the same data model structure. For example, the model allows for powerful analysis capabilities when maintaining all classifications of products together in the same entity.
Easier to implement prototypes.	Provides much more solid and stable foundation when used as a basis for a physical database design, especially when using to develop a robust, production quality database design.

Table 6: Benefits of more specific and more generalized style of modeling [44]

4.3.2.2.2 Domain

Domain is the area of application of the pattern, where it can be found an applied.

- **Domain-Specific**

Domain specific patterns are patterns that cannot be applied in more domains than the one stated.

- **Cross-Domain**

Cross-domain patterns are the ones that are universally applicable; they can be found and applied in multiple domains.

The visual representation of the classification schema proposed by this work can be summarized in Table 7.

			Level of Design		
			Early Design	Intermediate Design	Advanced Design
Domain	Domain-Specific			
		Healthcare			
	Cross-Domain				

Table 7: Domain Patterns Classification Schema

Note that the design of the schema includes the healthcare as domain-specific sub-classification in order to facilitate other domain patterns to be attached in the future.

4.4 Metamodel for a Domain Patterns Catalog

In order to hold the patterns discovered in the process, we developed a meta-model represent the structure and details of a Patterns Catalog, as well as the classification schema for the patterns.

For readability problems we omit the attributes in the design we present in Ilustración 30. Therefore we offer the possibility to see the attributes in the next sections, where we explore the parts we intended to represent within the metamodel including the attributes of their main classes. A complete version of the schema can be found in Appendix A.

As a conclusion of our analysis of diverse pattern catalogs in the literature review, we discovered that in a patterns catalog one can normally distinguish three independent but interrelated parts:

- The **core of the catalog** representing the type of patterns that it holds. i.e. design patterns, requirement patterns, data model patterns, etc. Within this part the **elements** that compose a pattern solution are described.
- The **vocabulary** sometimes represented as a patterns language or some others as a glossary of terms. For simplicity reasons, we chose the glossary type of representation.
- The **classification schema** is the way the patterns are grouped in families of patterns.

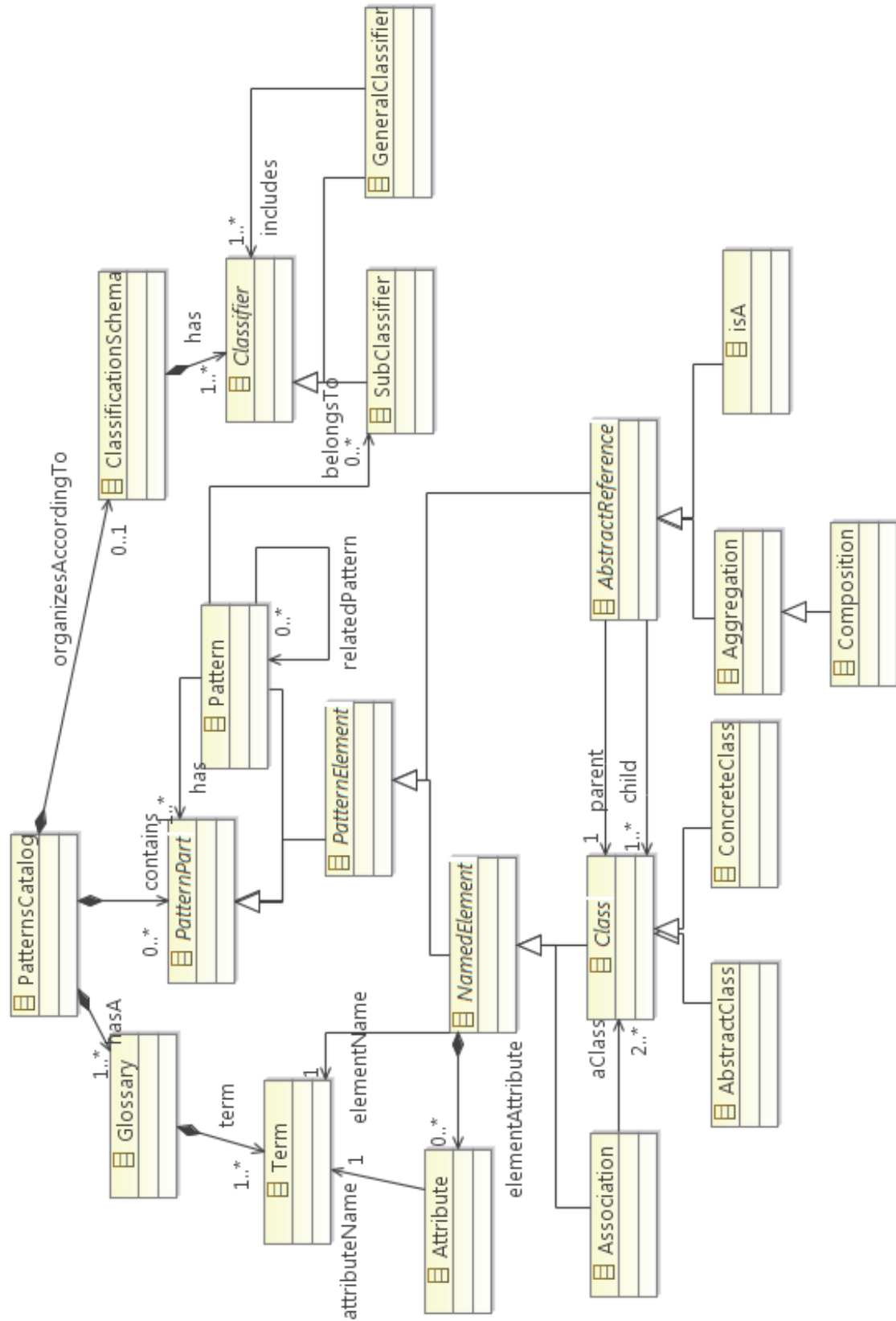


Ilustración 30: Metamodel for a Domain Patterns Catalog

4.4.1 Elements representation

For the purpose of our work, we decided to represent the domain elements in a way that was homogeneous with the ones we found in our source models.

As main elements of our design we observe: classes, associations and attributes. As internal representation we adopt a simplified version of the meta-models for ER Diagrams and UML Class diagrams as seen in Ilustración 31. The description of the elements can be found later.

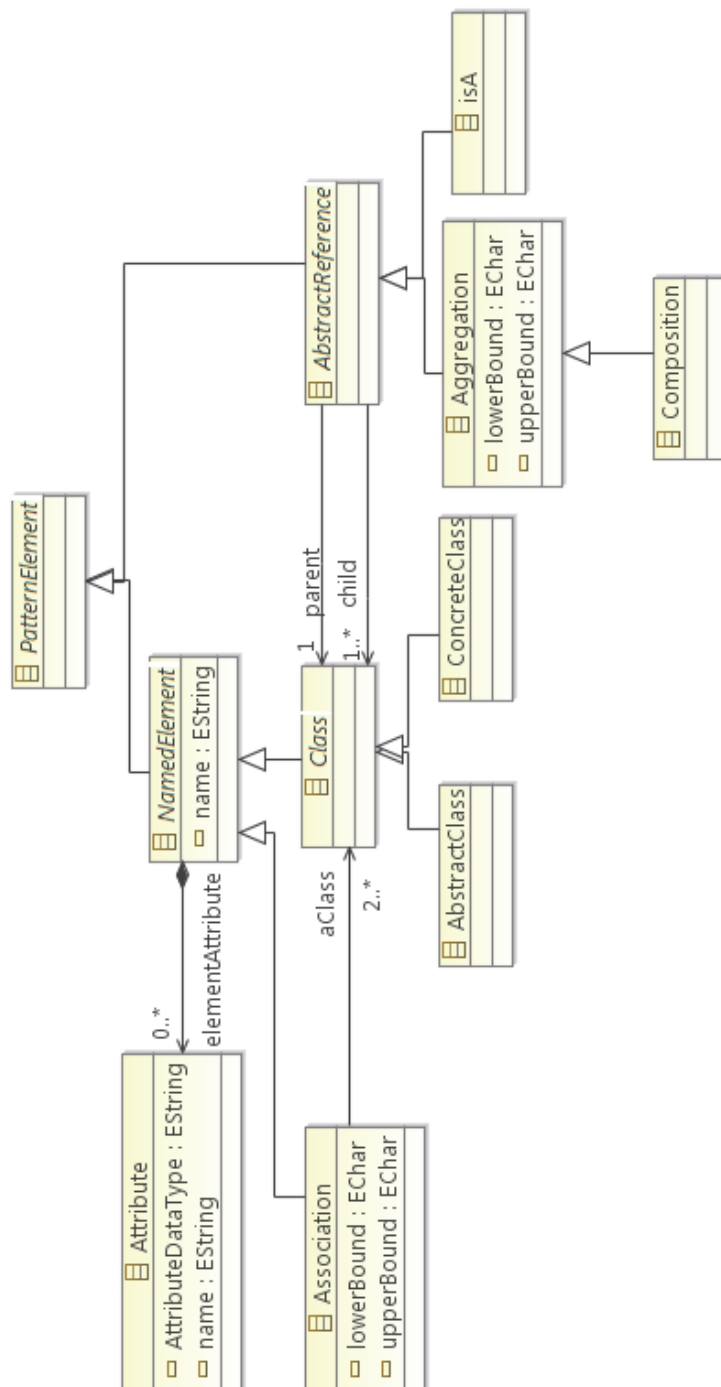


Ilustración 31: Elements Representation

4.4.1.1 *Class*

A class is an entity type from ER diagrams, a class from UML conceptual diagrams, a table from SQL database schemas; these are elements that can be all mapped to a class in our metamodel. A class can be abstract or concrete and is a named element.

4.4.1.2 *Association*

Associations represent the relationships between entities and associations between classes that were found as part of the domain models. They can be simple associations, called “Association” or can be an abstraction type form of association, called “Abstract Reference”.

Association represents both a binary association and an n-ary association, via the `aClass` relationship to class with cardinality `2..*`.

Abstract Reference represents Aggregations, composition and generalization/specialization or IsA associations. Since our model aims to be simple, support to multiple inheritance was discarded. Therefore, it is only possible to have only one parent class for multiple children classes.

Associations, aggregations and compositions can define their cardinality via the `lowerBound` and `upperBound` attributes.

4.4.1.3 *Attributes*

Attributes represent the characterization of class elements and associations. We consider as important attributes for both elements are: the name that will identify the element and the attributes with name and `dataType`. These attributes are part of the glossary of the catalog.

4.4.2 *Pattern Representation*

Each domain pattern systematically should name, explain, and evaluate an important and recurring design in domain models.

One issue presented in the design of this part is that pattern elements are not isolated units of knowledge, thus they are the core carriers of the recurring design.

In order to represent the patterns we needed the possibility to represent patterns as a composition of elements as well as elements and other patterns. That’s why we took as inspiration the “Composite” design pattern [25] that allows us to develop a flexible design as it can be seen in Illustration 32.

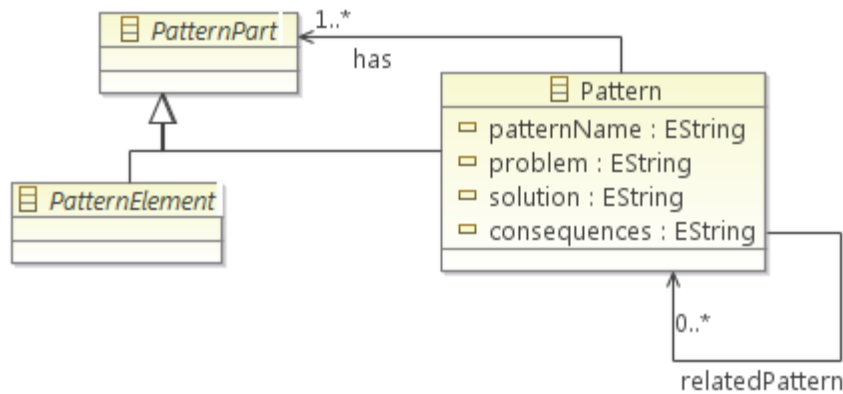


Illustration 32: Patterns Representation

As parts of a pattern we identified (see Table 8) as important the following:

4.4.2.1 Pattern Name

The name of a pattern describes a design problem, its solutions, and consequences in a few words. Naming a pattern makes it easier to think about design and improve communication with the designers that will made use of it.

4.4.2.2 Problem

The problem describes when to apply the pattern by explaining the problem and its context. It addresses the question: What particular design issue or problem does this pattern address?

4.4.2.3 Solution

The solution describes the elements that make up the design, their relationships, and attributes. It includes the pattern arrangement per se.

4.4.2.4 Consequences

The consequences are the results of applying the pattern. It aims to aid the designer evaluate design alternatives by understanding the benefits and trade-offs of using the pattern.

4.4.2.5 Related patterns

It is the relationship between patterns within the catalog. It aids the designer to know which patterns are closely related or may be also used in combination with the one being observed.

Pattern Name	
Problem	
Solution	
Consequences	
Related Patterns	

Table 8: Pattern Template

4.4.3 Classification Representation

The classification is a part that appears normally in catalogs that contain several patterns. It is a way of grouping the patterns in order to help communicate with the designer as well as to aid his/her search.

For this matter we decided to support a tree structure of classification that is flexible enough to support one-dimensional as well as multi-dimensional classification schemas (see Illustration 33). A pattern belongs to the smallest sub classification within the schema.

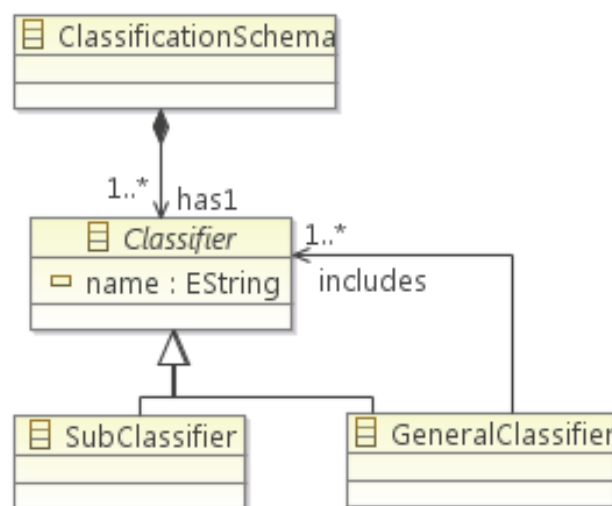


Illustration 33: Classification Representation

4.4.4 Vocabulary

The vocabulary allowed within the context of the catalog is contained as a glossary of terms as seen in Illustration 34.

The terms are bound to their synonyms to aid the use of the pattern at regional vs. universal level.

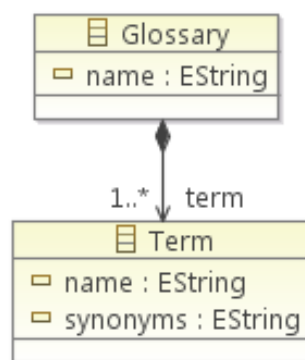


Illustration 34: Vocabulary Representation

5 Implementation

In this chapter we will first describe the method used to perform the implementation as a result of the methods analyzed in the literature and then we explain the artifacts related to the process of the discovery and classification of patterns in domain models.

The detailed information of the steps that will guide the construction of a preliminary domain patterns catalog will be described in detail in the remaining part of this section.

5.1 Method

First of all an initial documentation research about the thesis domain topic, Patterns in Domain Models, was performed. As a result of this research, we decided to use the work of Carreon, summarized in section 4.1 Building a catalog of Patterns, as baseline for the implementation methodology for construction of a preliminary domain patterns catalog. We then refine the steps and adapt them for our purpose, leading to the following steps:

1. Analyze and select relevant Models of the Domain.
2. Extract elements present in the Models.
3. Perform semantic analysis and refine the elements to form patterns.
4. Insert in the catalog of candidate patterns.
5. Create and/or refine the pattern and insert or update in the patterns catalogue

It is important to note that for the purpose of this work an “Element” is any of the following: class, entity, attribute, relationship and other model elements.

We present in Illustration 35 a summary of the activities and artifacts used and produced during the development of this work. Note that Step 5 is not included in our development, yet we include it in the method for the sake of completeness.

The activities within the steps will be further detailed in the rest of the section, after a brief explanation of the artifacts used and produced by this work.

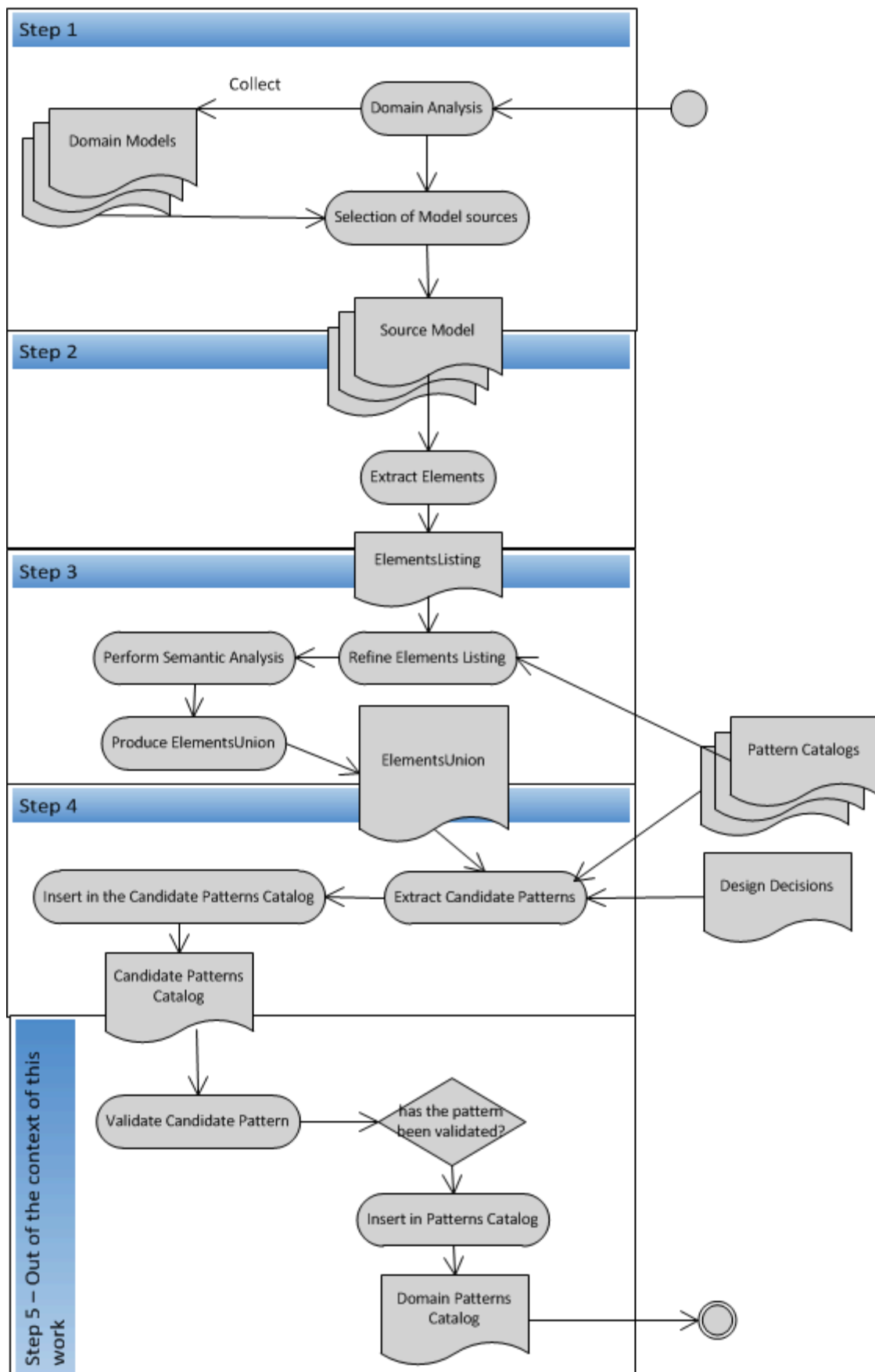


Illustration 35: Method for developing a Domain Patterns Catalog

5.2 Step 1 - Analysis of the Domain

The task consisted in collecting and identifying documentation relevant to identify and classify patterns as well as identifying sources for collecting models that are relevant to identify the common elements – patterns - composing the domain.

In this section are described the artifacts used and produces in the step, later the description of the sources used to get domain models are found. Later follows the introduction of the pattern catalogs identified for verification and semantic analysis. Finally the illustration of the implementation of the first phase of the method, the issues and results are described.

5.2.1 Artifacts of Step 1

The **input** artifacts of the step are:

- Domain analysis documents
They are a collection of papers and works of other authors that have worked in the topic of model driven development, patterns, reuse and quality of software and pattern catalogs construction. The more important works are listed in the literature chapter of this work in section 7.
- Domain Models
Correspond to projects of the real world containing SQL schemas, UML Diagrams among others. The information about the sources and the documents can be found in section 5.2.2.

The **output** artifacts are:

- Source Domain Models
Due to some problems later described, a filtering of some models had to be performed over the total of models found in the sources. The resulting models are the ones that were the base for element extraction in Step 2.

5.2.2 Sources

The models used to build the proposed patterns were extracted from projects of industry and research taken from the internet.

We were looking for repositories of models containing models of the domain of healthcare. The models should contain model elements such as entities, relationships, attributes among others.

The criterion they had to fulfill in order to be considered a source is that they have to contain valid and relevant models; that is, the element is not only a result of the search but can be downloaded and open, the model has enough elements and they make sense within our domain.

We explain further the results of the elimination by using these criteria in the results section (Section 5.2.4) explaining Table 14.

The main sources used in this work are the following:

- a) www.uml-diagrams.org; Is a website containing several UML diagrams, such as domain model, use case model and others by Kirill Fakhroutdinov.
The author is a Senior Software Engineer that has worked in the domain of healthcare for the past ten years. His website aims to provide information and examples of UML and UML diagrams of real world projects [64].
- b) Kross, Knowledge Repository of Schemas and Semantics; is a repository that contains classified and indexed schemas in several categories, as well as the search tool to use them [65]. The project is developed and maintained by the iSchool at Drexel.
- c) www.databaseanswers.org; Is a library of over 1,000 Data Models in several domains resulting from the work as consultant and as Enterprise Data Manager of Barry Williams. The library contains works of industry of the over 25 years' experience, starting with IBM, in Enterprise Data Management [66].
- d) www.bvbssoft.com ; Is a repository of source code, of software development projects, made available by a community of developers across the globe [67].
- e) DIMA SVN repository; is the repository of all works of the Database and Information Management department of the Technical University of Berlin. In this repository one diploma thesis “Metamodelle und Ontologien für domänenspezifische Sprachen” – in English: Metamodels and Ontologies for Domain Specific Languages – where the author, Wandelt, presents a reference model for the domain Hospital as a result of a case study [68].
- f) SIIH System Technical Reference; is the document containing information about the complete development of the “Sistema Integrado de Información Hospitalaria” – in English Integrated System of Hospital Information – that was produced as part of a bachelor's theses. From this work we use the database schema and Class diagrams produced [69].

5.2.3 Model Management

In order to record the initial set of models we decided to set some standards and then to design a relational database to save and manage the information present in them.

5.2.3.1 Standards

We decided to use as naming standard the format starting with lowercase. In the case of compound names all words after the first one should start with uppercase, but not separated by any other symbol.

As an example we present Illustration 36:

attributeName

Illustration 36: Naming Standard for Model Management

On the other hand we decided to make a standard for the storage of the models. This standard is based on the domain, source and model names as directory names as well as a root directory.

In other words the directory hierarchy is as seen in Illustration 37:

modelsRoot->domainName->sourceName->modelName.extension

Illustration 37: Storage Directory Standard for Model Management

As we see in the example of Illustration 38, the model class-example-hospital-organization.png is stored in the source directory uml-diagrams of the domain healthcare.




healthcare	Nombre	Fecha	Tipo
databaseanswers.org	 class-example-hospital-organization	26/11/2012 12:27	Imagen PNG
DIMA SVN	 class-example-hospital-treatment	26/11/2012 17:03	Imagen PNG
KROSS	 class-example-hospital-wards	26/11/2012 16:10	Imagen PNG
SIH			
uml-diagrams			

Illustration 38: Example of Storage Directory Standard

5.2.3.2 Database design

In order to design the database we analyzed the elements we wanted to store, namely Domains, Sources, Models and their relationships. We also analyzed according to the structure that we found in the models collected, the attributes that would sufficiently describe them.

A description of the entities, part of the Database design, and the attributes of each of them can be found in Table 9. The relationships corresponding to these entities can be seen in Table 10.

EntityType	Attributes	Description
Domain	<u>domId</u> ; corresponds to the identifier of the domain.	A domain is the context or universe of discourse being discussed in the content of the models.
	<u>domName</u> ; is the name of the domain.	
	<u>description</u> ; is a brief description of the domain.	
Source	<u>srcId</u> ; corresponds to the identifier of the source.	Is the origin where we found the models.
	<u>website</u> ; is the URL of the Website where the model was found.	
	<u>project</u> ; is a brief description of the project that developed the model used.	
Model	<u>mdlId</u> ; corresponds to the identifier of the model.	Model is any file containing elements and their relationships that describe the domain. Some examples are: Database Schemas, Class diagrams, source code.
	<u>modelUrl</u> ; is the exact URL where the model was found.	
	<u>modelName</u> ; is the name the author gave the model.	
	<u>extension</u> ; is the filetype of the model.	
	<u>status</u> ; is the status of the model. The value contained can be valid, invalid, irrelevant and undownloadable.	
	<u>keywords</u> ; is a list of the main elements found on the model.	
	<u>licence</u> ; is the name and version of the license if any.	
	<u>nElements</u> ; is the number of elements found in the model.	
	<u>version</u> ; is the version of the model if any.	
	<u>hashcode</u> ; is the hashcode of the model found.	
	<u>observations</u> ; any other relevant information.	

Table 9: Model Management Entities identification

EntityType	RelationshipType	EntityType	Cardinality	Description
Domain	Has	Model	1-n	Every domain has several models.
Model	FoundIn	Source	n-1	A model can be found in a Source.

Table 10: Model Management Relationships identification

As a result we designed the Entity Relationship diagram shown in Illustration 39 that was later used in order to implement a Relational Database.

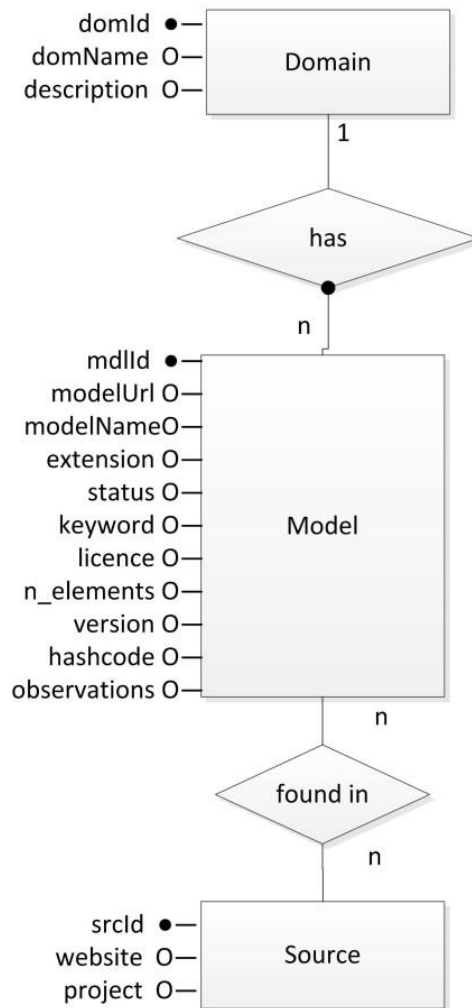


Illustration 39: Model Management ER Diagram

This model, thus the database, can be reused when studying further domains, sources and models in future work.

5.2.3.3 Database population

Having the database ready the steps followed to populate it is described in Illustration 40 as follows:



Illustration 40: Steps to populate the Model Management Database

The population has to go in this direction to fulfill the cardinality constraints at the moment of storing the models.

It is important to note that all identifiers are auto incremental integers.

5.2.3.3.1 Insert Domain

In this step we inserted the information related to the Hospital Management domain. The description used was found in Wikipedia under the entry “Health Administration”¹¹.

dom_id_int	dom_description_txt	dom_name_chr
1	Health management or healthcare management is the field relating to leadership, management, and administration of hospitals, hospital networks, health care systems, and public health systems.	Hospital Management

Table 11: Store domain Hospital Management

5.2.3.3.2 Insert Source

We then proceed to store the information about the identified sources.

Since the sources come either from projects the internet or from thesis documents we could easily find the description of the project where the models belong to.

src_id_int	src_website_vch	src_project_txt
1	http://www.uml-diagrams.org/class-diagrams-examples.html	UML Modeling and Design
2	http://www.databaseanswers.org/data_models/hospital_management	Subject Area Models have been created for each Area in this top-level Management Model:- Hospital Admissions Hospital Outpatients Hospital Patients Hospital Staff Hospital Supplies Barry Williams Principal Consultant Database Answers Ltd. London, England June 17th. 2012
3	http://cluster.ischool.drexel.edu:8080/kross/queryGoogle.jsp?query=hospital	KROSS - Knowledge Repository of Schemas and Semantics 2008-2011 Category: hospital

Table 12: Store Sources (partial rows)

¹¹ http://en.wikipedia.org/wiki/Health_administration

5.2.3.3.3 Insert Model

Table 13 shows only some of the columns due to space limitation and to aid readability.

Some of the design decisions that we had to consider while storing the models are the following:

- Keywords; since in some occasions there are too many elements, we decided to include up to the six more relevant entities encountered.
- Version; we could identify that, as general case, there was no information related to the versioning of the models, therefore we decided to give a default of version 1 to each model that didn't explicitly had one.
- Hash code; at the moment of saving the hash code we decided to save the one corresponding to the model file and not to the whole HTML in the case when they are part of a Website.
- License; we discovered that, as general case, the information related to the license - that we had considered important - is not available. We decided to allow leaving the attribute empty.
- NumberOfElements; In order to store the number of elements we have to process first the model. We decided to derive this attribute after the analysis of the model and to consider all relevant elements in the count.
- Observations; We decided that this field would be mandatory in the cases where the model was found invalid, to have a clear argument of why we declared it as such.

mdl_id_int	mdl_extension_vch	mdl_status_vch	mdl_keywords_txt	src_modelurl_vch	src_modelname_vch	src_id_int	dom_id_int
1	png	valid	person, patient, hospital, department.	http://www.uml-diagrams.org/class-diagrams-examples.html#hospital-domain	class-example-hospital-organization	1	1
2	png	valid	ward, doctor, patient, team.	http://www.uml-diagrams.org/class-diagrams-examples.html#hospital-domain	class-example-hospital-wards	1	1
3	png	valid	treatment, doctor, patient, diagnosis, examination.	http://www.uml-diagrams.org/class-diagrams-examples.html#hospital-domain	class-example-hospital-treatment	1	1
5	sql	irrelevant	osm2pgsql, amenity, admin_level, geometry, buildings, landuse, leisure	https://github.com/mapserver/basemaps/blob/master/contrib/osm2pgsql-to-imposm-schema.sql	osm2pgsql-to-imposm-schema	3	1
4	sql	valid	patient, physician, insurance.	http://cs.ecs.baylor.edu/~donahoo/practical/sql/patient.sql	patient	3	1
6	sql	irrelevant	rooms	http://infosmart-habbocms.googlecode.com/files/fix_rooms.sql	fix_rooms	3	1

Table 13: Store Model (partial columns and partial rows)

5.2.4 Results and conclusions

First of all, we performed a search within the repositories of models in order to identify the models that we could use for our work.

We decided to perform the search by using terms that we considered relevant for the domain. Some examples are: Hospital, Doctor, Physician, Patient, Nurse, among others. It is important to note that out of this search some models that contain more than one relevant term appear again so we had to discard these duplicate results to obtain Total Individual Models (as seen in Table 14).

In order to illustrate the results generated by the search in the Domain Analysis step of our method we produce Table 14 that exemplifies the search for the single terms Hospital, Doctor and Patient.

DOMAIN 1 - HOSPITAL MANAGEMENT SYSTEM						
SEARCH KEYWORD \ SOURCE	uml-diagrams.org	databaseanswers.org	Kross, Knowledge Repository	bvbsoft.com	DIMA SVN	USFX
Hospital	1	0	16	13	1	1
Doctor	2	1	25	1	1	1
Patient	3	2	32	4	1	1
Total Results Count	6	3	73	18	3	3
Repeated Models	3	1	18	3	2	2
Found at blog or forum	0	0	41	0	0	0
Undownloadable	0	0	10	11	0	0
Total individual Models	3	2	14	15	1	1
Valid models	3	2	2	1	1	1

Table 14: Results of Domain Analysis for the Hospital Management Domain

Then we further analyzed the contents of each model and in this process we discovered some problems that made us believe that it was important to identify the models that are valid within the previous result of Total individual Models.

The main problems that we discovered for filtering the results are:

- The model was already inserted (repeated models filter).
- The model is contained in a Blog as exemplification using very few attributes in an inaccurate way (Found at blog or forum filter).

- The model File is not found on server (undownloadable filter).

As a result we discarded as input models the models that have status: Undownloadable, Irrelevant and Invalid. Although, we decided to store them in our database so that in future searches we won't analyze them again, but ignore them.

Originally more sources were considered. Some examples are:

- **Moogle**; a metamodel based Search Engine. Were we found several projects for technical domains [70].
- **ReMoDD**; a repository of models to aid model driven development. We found that they offer several projects from research but mainly in a pdf format as part of papers or conference proceedings. Also only other domains are available [71].
- **Zoos**; The Metamodel Zoos are a collaborative open source research effort intended to produce experimental material that may be used by all in the domain of Model Driven Engineering. Basically it is a repository of models and modeling standards for software development [72].

Unfortunately we could only discover that they didn't contain models for our domain while crawling within the models.

5.3 Step 2 - Extract elements present in the Models

In this step we use the models identified in Step 1 and then perform the element extraction. Then we group them according to similarity and align the common elements found across models, being this listing the resulting artifact –Elements Listing - produced in this step.

We first described the artifacts used and produced in the step, to then start with the description of it.

Before proceeding with the extraction of elements it is necessary to define what an element is. We then proceed to explain about the extraction and alignment process. Finally we report the results of the phase.

5.3.1 Artifacts of Step 2

The **input** artifact is:

- Source Domain Models, described in Step 1.

The **output** artifact is:

- Elements Listing
Elements listing; is a list of the elements found in the models ordered per project and aligned in parallel according to similarity. It is the output Artifact from step 2 and the input for step 3.

5.3.2 What is an element?

Alexander [23] tells us that the pieces that compose a pattern are “the lowest-level elements and relationships between them, they form building blocks”.

Coad [32] tells us that “Classes and Objects correspond to Alexander’s lowest-level elements”.

Thonggoom [60] has identified that the elements for his work in Knowledge-Based Database modeling were entities and relationships taken from database schemas.

With that in mind, and after analyzing our domain models, we discovered that our input domain models are described as means of:

- Entities / classes and
- relationships / associations

both with their corresponding attributes as main lowest-level elements.

5.3.3 Element Handling

As previously mentioned, in this step we use the models identified in Step 1 - Analysis of the Domain.

We used a Spreadsheet as tool to help us perform the alignment of common elements, and thus obtain the artifact Elements Listing at the end of the Step.

First of all we took one model and systematically started to extract the elements one by one by adding a row in the table and filling in all the characteristics for the element.

Once we finished with the model we took the next model and proceeded similarly, with the difference that before adding a new row we checked whether or not we had the element already in the table. If the element was already in the table, we aligned the coincidence in the same row, if not we added a new row as in the previous case.

As a way to illustrate the structure of the “Elements Listing” document we present an abbreviated example of it in Illustration 41.

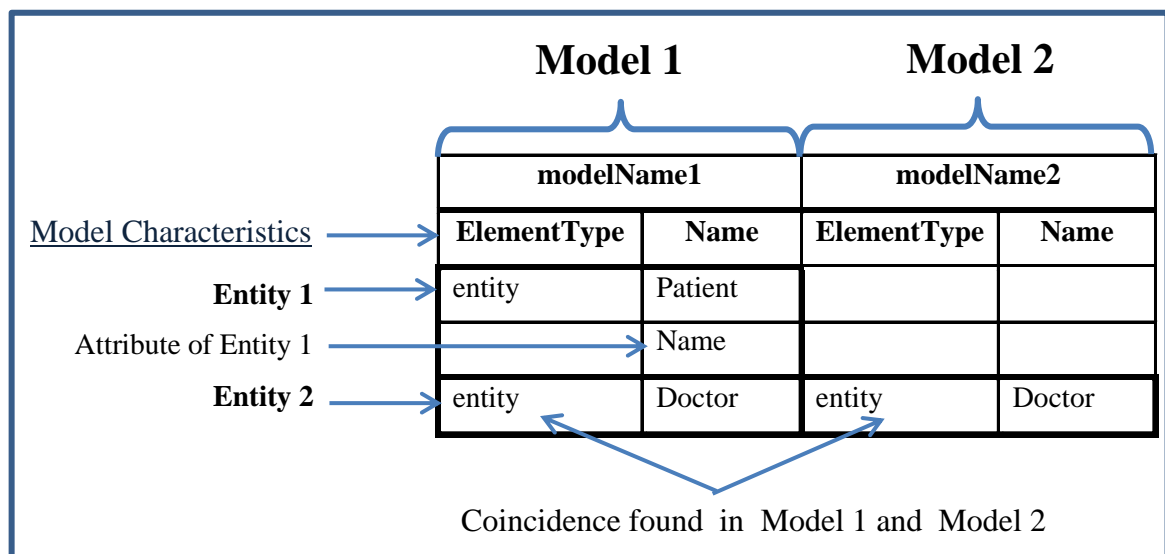


Illustration 41: Structure of the "Elements Listing" Artifact

It should be noted that we discovered the final representation in the process of analyzing the models, having started only with a representation of: ElementType and Name and determining the important attributes as we found them.

The characteristics that we identified important to describe the model and its elements, thus being part of the final Element Listing Artifact, are:

- modelName; contains the name of the model where the element comes from as stored in the database. We find it important because in this way we maintain a reference to the origin of the element.
- Id; the element identifier is a sequential code that we assign to one whole row. A common element across models. Only entities and relationships are assigned a code, not their attributes.
- modelType; refers to the type of model where the elements are extracted from. Some examples are: class diagrams, database schemas (SQL), ER diagrams.
- elementType; describes whether it is an entity or a relationship.
- As; describes the physical characteristic of the element, its physical type. For instance, if the element is an entity it can be abstract or concrete; if it is a relationship it can be binary, isA (meaning a generalization/specialization relationship), aggregation or composition.
- Name; contains the name of the element, of all entities, relationships and attributes. In the case of relationships the name is the name of the classes involved.
- Type; contains the value of the primitive or complex type of the attributes.
- Cardinality; contains the value of the cardinality of the relationship, it can be: 1-1, 1-n and n-n.
- Parent; in the case when the element is a child of another element, it contains the name of the parent class.

Below we present a small illustrative example of the representation of both entities (see Table 15) and relationships (see Table 16) within the elements listing artifact.

modelName						
Id	modelType	elementType	as	Value	type	Parent
E1	Class-diagram	entity	concrete	Patient		
				name	String	
				birthdate	Date	
				gender	String	
E2	Class-diagram	entity	concrete	Doctor		
				name	String	

Table 15: Example of Entity element representation within the Element Listing document

modelName						
Id	modelType	elementType	as	Value	cardinality	
R1	Class-diagram	relationship	binary	Doctor-Patient	n-n	
				date	Date	

Table 16: Example of Relationship element representation within the Element Listing document

We discovered that for a later automation of the process we should propose an initial characterization of the elementType for both entities and relationships. Therefore we present our findings as follows.

5.3.3.1 Entities elementType

The entities that can be identified within our source models are:

1. Entity
2. Class
3. Table

5.3.3.2 Relationships elementType

The binary relationships that can be identified across models are:

1. 1:N for relationships identified by Foreign Key.
2. 1:N for relationships identified by partial keys.
3. N:M for relationships identified by relationship relations.
4. IsA relationships.
5. Recursive relationships.
6. Aggregation relationships.

5.3.4 Results and comments

In this step the models were compared and the common elements present in them were aligned and grouped according to similarity.

The result of this phase consists of a set of common elements, and groups of them, found in models of the healthcare management domain as seen in Table 17 :

Element	Amount
Entities	97
Relationship	121
Attributes	591
Primitive Types	5
Complex Types	4
Enumerations	3
Constraints	8

Table 17: Results from the Element Extraction phase

Since the patterns should contain the abstraction of a solution and not the instance of it we decided to omit for the refinement stage the elements that could be specific from the design. Such elements are the definitions of user defined types, enumerations and constraints.

The challenge in this step was to try to express all models in the same way. This was fairly simple for entities, classes and tables of SQL schemas but a bit more complicated for the case of relationships, especially with the SQL Schemas, since we had to perform reverse engineering in order to map the Foreign Key references between tables and the relationship tables to relationships, for instance.

We discovered a heterogeneous way of representing attributes. For example, some authors preferred to use Enumerations when possible, while others preferred to use primitive types. i.e. Gender: GenderEnum vs. Gender: String being GenderEnum{‘Masculine’, ‘Feminine’}

We also detected this heterogeneity in the use of composite attributes and simple attributes for the same concepts. An example of it is the attribute Address sometimes modeled as Address: Address and some other times as Address: String. Furthermore, this example also presented to us the issue of having the use of single valued versus multivalued attributes: PatientAddresses: Address [1..n].

As a way of exemplification we present Illustration 42, where we can see some representations for the attribute ‘Address’ found in the models.

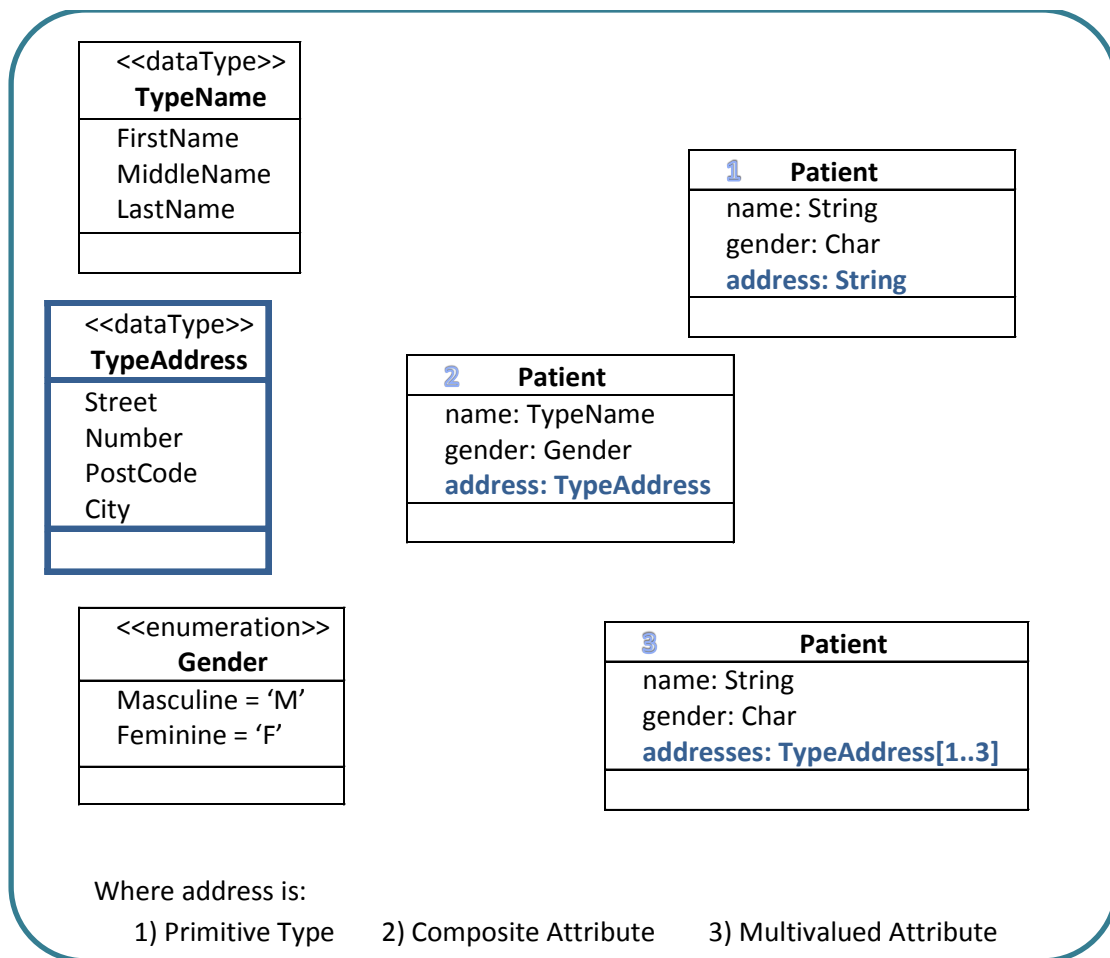


Illustration 42: Example of data-types' heterogeneity

5.4 Step 3 - Semantic analysis and refining the elements

Once we have the elements listing, it is necessary to do a semantic analysis and refinement of the elements found in it. These elements are originally expressed as found in their corresponding models, so the first task for us to do was to find convention rules that will help us unite them in order to produce the elements union. This union will, in a later step, be the basis for the candidate patterns.

The same element may be expressed differently in each of the models or one element can be expressed as several elements in another, which is the case of hierarchical structures involving elements such as MedicalStaff, Doctor, and Nurse.

Because of this, it is necessary to study each element alone and the element across models, that is to say elements of one same row, by performing a semantic analysis. This semantic analysis was performed in a manual but systematical way, thus it can later be automated. This analysis allows us to identify and understand the elements per se, as well as the relationships among them.

Another reason is that it helps us identify the rules of building models and rules to validate the relevance of certain concepts within models of the domain.

In the following sections we present as usual the elements used and produced in this step. Later in this section we explain the rules identified as well as the books and catalogs used to validate our decisions and we also explain how we use them in the creation of an Element Union artifact.

5.4.1 Artifacts of Step 3

The **input** artifacts for this step are:

- Elements Listing, described in Step 2.
- Pattern Catalogs
They are books or other formats of catalogs of patterns in the domain of healthcare and of software design (i.e. online, paper publications, etc.)

An intermediate artifact is the refined listing, which is a list taken from the element-listing, all elements are refined after semantic analysis with appropriate names, synonyms and attributes coming from the analysis of pattern catalogs and dictionaries.

The **output** artifact is:

- Elements Union

It is the union of all refined elements from the refined listing. It is afterwards used as input for the construction of the domain candidate patterns catalog. It is an output from step 3 and an input for step 4.

5.4.2 Elements handling into Elements Union

Due to the heterogeneity of the source models and the diverse purposes of their authors at the moment of designing them, we discovered that an intersection of the elements gave as a result only few elements, not enough to form candidate patterns, that also as a whole did not represent the domain as accurately as a union of elements did.

In the rest of the section we describe the main problems and opportunities discovered while processing the elements listing that afterwards lead us to get the elements union as result of Step 3.

5.4.2.1 Names and synonyms

The first issue to tackle was the fact that one of the sources was in Spanish [69] and another in German [68]. The first step was to translate these models and therefore their elements into English, the language of convention for this thesis. For this matter we used different online tools, such as translators¹² and dictionaries¹³.

While working on this subject we detected the importance of synonyms, since one same concept was named differently between models. We discovered as well the regional usage of some name over another between English speaking countries, and decided to treat these terms as synonyms. As examples we mention the use of the term “attending physician”, in USA, versus “consultant doctor”, in UK; or the term “Hospital” over “Clinic” or over “Institution” across models.

As a result of these observations we decided to add a special characterization to the elements union to collect possible synonyms of the terms. For this matter we made use not only of the synonyms found within the models, but also of some tools such as an online thesaurus¹⁴, from Lexico Publishing, LLC, and WordNet¹⁵, a glossary of Princeton University.

¹² Google translator : <http://translate.google.com>

¹³ Spanish – English: Medical online dictionary - <http://www.merriam-webster.com/medical>
 German – English: Medical online dictionary - <http://www.tk.de/rochelexikon/>
 And for both: <http://thesaurus.com/> , <http://www.wordreference.com/> and <http://www.dict.cc>.

¹⁴ Online Thesaurus: <http://thesaurus.com/> , <http://dictionary.reference.com/>

¹⁵ Wordnet Online Glossary: <http://wordnet.princeton.edu/>

5.4.2.2 Types

As mentioned in section 5.3.4, the results of Step 2, we discovered a heterogeneous way of representing attribute types within the models. Among the problems described we found the use of enumerations, primitive types, composite attributes, simple attributes and multivalued attributes for the same element attribute.

For dealing with this matter, we used the aid of pattern catalogs [24] [33] [36] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] to help us know what the commonly employed ways of describing attribute types are.

As an example we present Illustration 43, the ‘Address’ representation as found in [41], where Address is the location of a party, a person or an organization.

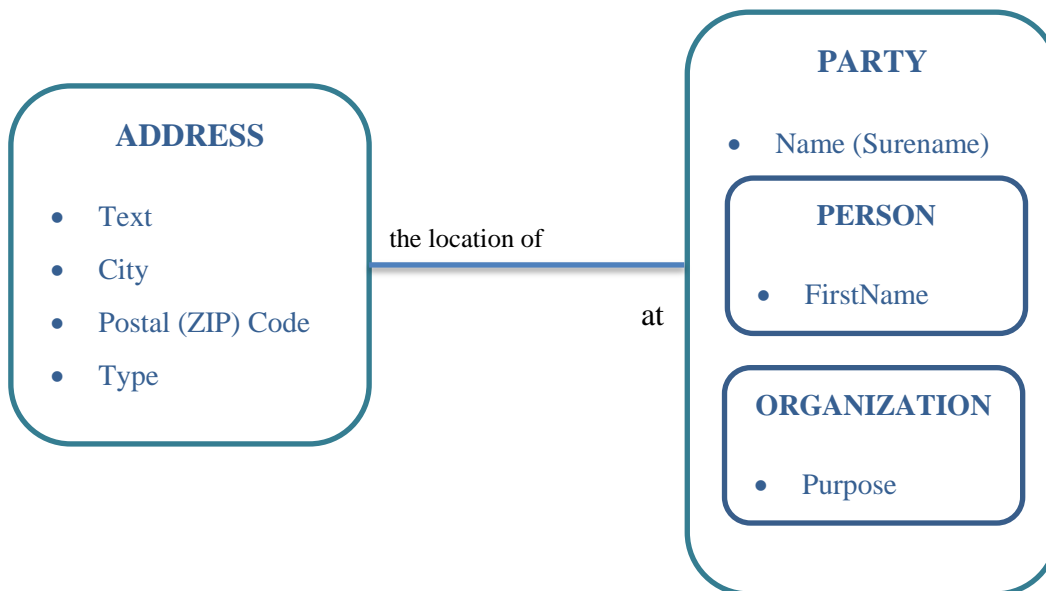


Illustration 43: Address according to D. Hay

Attributes for Address include:

- the "text" of the address I.e. Einsteinufer 17,
- "city" i.e. Berlin,
- "postal (ZIP) code" i.e. 10587 and finally it should also include
- "type," that could be "billing address," "shipping address," "home address," and so forth.

In the same manner we used the metadata standards presented in section 3.3 and after analyzing the similarities and deciding to preserve the simplicity of the models; we refined the attributes and their types.

It is important to note that we use the pattern catalogues not only to perform the semantic analysis, but also as support or validation for the patterns identification process explained later in chapter 5.

Finally, as part of the semantic analysis we discovered some elements that were incomprehensible for the author, that even after deep analysis of books, catalogs and thesaurus it was impossible to identify their meaning within the context. The decision taken implied filtering out such kind of attributes from the Elements Union. To mention an example, we excluded the attribute “hospitalNumber” as part of the entity “Staff”.

5.4.2.3 Naming conventions

It is worthwhile mentioning that as a design decision and as a way to preserve consistency we choose to use as syntax for naming the first word in lowercase and every next word starting with uppercase with no spaces or hyphens or underscores in between. i.e. elementName.

5.4.3 Architecture of the Elements Union

Below we present a small illustrative example of the architecture of both entities (see Table 18) and relationships (see Table 19) of the Elements Union artifact.

elementsUnion						
Id	elementType	as	Value	Synonym	type	Parent
E1	Entity	concrete	patient			person
			name		TypeName	person
			birthdate		Date	person
			gender		Gender	person
E2	entity	concrete	doctor	physician, medic		person
			name		TypeName	person

Table 18: Example of Entity element representation within the Elements Union

elementsUnion						
Id	elementType	as	Value	-	cardinality	-
R1	relationship	binary	Doctor-Patient		n-n	
			Date		Date	

Table 19: Example of Relationship element representation within the Elements Union

A complete outline of this artifact can be found in Appendix B “Elements Union”.

5.4.4 Results and comments

As a result of this step we get the artifact Elements Union where we find a summary of the elements that exposes (see 5.4.3 “Architecture of the Elements Union”):

- The separation of elements and their attributes
- The identification of Types
- The assignment of synonyms

The result of this phase consists of a set of refined elements, and groups of them, found in models of the healthcare management domain as seen in Table 20:

Element	Amount
Entities	231
Relationship	307
Attributes	615

Table 20: Results from the Elements Refinement phase

The challenge in this step was to try to abstract all elements as part of one same model. This required the application of some abstraction methods in order to improve the design. The heterogeneity within designs gave us the chance to choose between designs that represented the same, but were designed with more or less attributes. An example of this fact is the way ward and room were modeled, for which we choose to keep as main name Room and merge all relevant attributes as seen in Illustration 44.

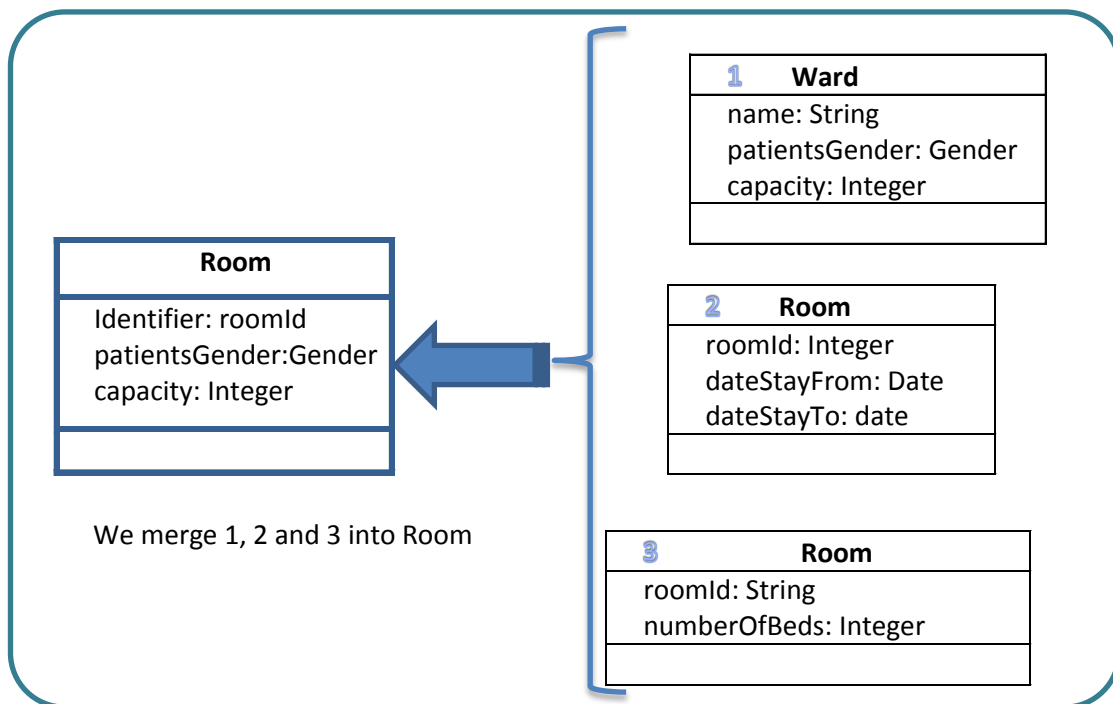


Illustration 44: Example of entity design heterogeneity

5.5 Step 4 - Candidate patterns

Taking as an input the elements union document and therefore the elements listing refined, we identified structures that recurrently appeared within models as well as in other pattern catalogs in order to select them as candidate patterns.

To form those clusters of elements we considered some selection criteria that are described in the section 5.5.2. The inspiration for selecting those criteria is described in section 4.2 Methods for pattern identification.

In the following sections are described the artifacts used and produced in the last step of our implementation, then the selection criteria applied over the elements union in order to form pattern candidates and examples of those. Later we present the process for inserting the patterns into the patterns catalog and we present the metamodel used for the implementation of it. Finally we summarize the results and comments of the step.

5.5.1 Artifacts of Step 4

The **input** artifacts are:

- Elements Union, described in Step 3.
- Patterns Metamodel is a model representing the architecture of a patterns catalog. It is described in section 4.4.

The **output** artifact is:

- Domain Candidate Patterns; are the grouping of elements that appears in the models with more frequency will be nominated as candidate patterns and then entered as instances of the patterns catalog.
- Domain Candidate Patterns Catalog
Domain Candidate Patterns Catalog; is a collection of all Domain Candidate Patterns identified in the step and classified according to certain criteria to aim understandability and ease of use. It is the resulting artifact from step 4.

5.5.2 Candidate patterns selection criteria

In this section we describe the main relationships used for clustering the elements to form the pattern candidates. Note that all of these criteria were accompanied with a further investigation about the elements forming the patterns in order to make sure that we are actually representing the domain and not only the result of random coincidences.

5.5.2.1 Relationships of attributes within an entity

First of all we went through the elements union artifact and we analyzed the relevant attributes one by one and identified the relationships within them.

In order to illustrate the process of using this criterion we take as an example the entity patient, as it looks in the Elements Union as seen in Table 21.

Id	Element Type	as	Value	Synonym	type	Parent
E5	Entity	concrete	Patient	inpatient, outpatient, referral,sick person		Person
			patientId	Id, Patient_Id, patientNo,	String	
			admissionDate	inPatientDate	Date	
			sickness	illness	Text	
			prescriptions	drugs	Text	
			allergies		Text	
			name	name	Name	Person
			gender	Sex	Gender	Person
			birthdate		Date	Person
			homeAddress	address	Address	Person
			phone	telephone, landline, homePhone	Phone	Person
			workPhone		Phone	
			cellPhone		Phone	
			Height		Double	
			Weight		Double	

Table 21: Relationships within the Patient Entity, from the Elements-Union

Taking a look at the attributes we can identify and then group the attributes according to the relationships that we describe next and we show in Illustration 45.

- **Attributes that are assigned / saved only once** and at the moment the person is admitted as patient in the healthcare institution.
- **Attributes found in other entities**, is the category that will help us discover hierarchies of entities that are similar or abstractions of their similar attributes.
- **Attributes found as entities**; this category of attributes helps us know that there are attributes that are not only characteristics of the entity, but also that they contain important information on the matter they represent.
- **Contact Information**; this category was represented very differently among entities, yet the idea of contact information was persistent. For some entities like medical employees emergency contact information was relevant. On the other hand the representation of patient within models differed in the amount of contact information

they required, for example some models had address and phone for home and address and phone for work, sometimes even the cellphone as well.

- **Measurements**; we identify a type of attributes that are updated probably every time the patient goes to visit the doctor, which means that the patient entity would be subject to update when only some attributes are volatile.

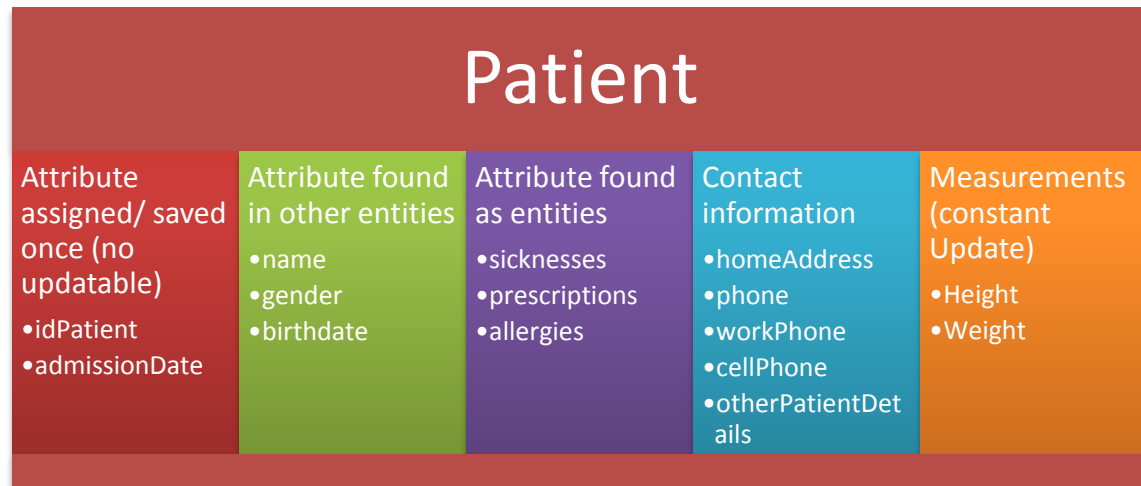


Illustration 45: Clustering of attributes within an entity

For this particular example, we assign the first category as attributes to the patient entity. The second category is part of the abstraction of this and other entities that have a name, a gender and a birthdate, that is to say a “Person” entity, abstraction of employees, patients among others. In the case of the attributes found as individual entities, we choose the entity representation, although for this particular case it is important to note that the entities of the category mean two different things. They represented both notes in the medical records and antecedents of the patient. For this case, we abstract the entities in an “Antecedents” entity. Contact information is considered to be important and can be extracted into a new entity that is more flexible for the designer’s needs, as well as measurements.

The result of the application of this criterion over the complete Elements-Union artifact lead to the creation of an intermediate model, from where we extracted candidate patterns that would be later analyzed and further improved before inserting them in the candidate patterns catalog. An example of candidate pattern resulting from this process is the Contact Information Pattern that represents phone, address and electronicAddress. It can be related to people as well as to organization; therefore in the example it is related to a “Party” entity as it is seen in Illustration 46.

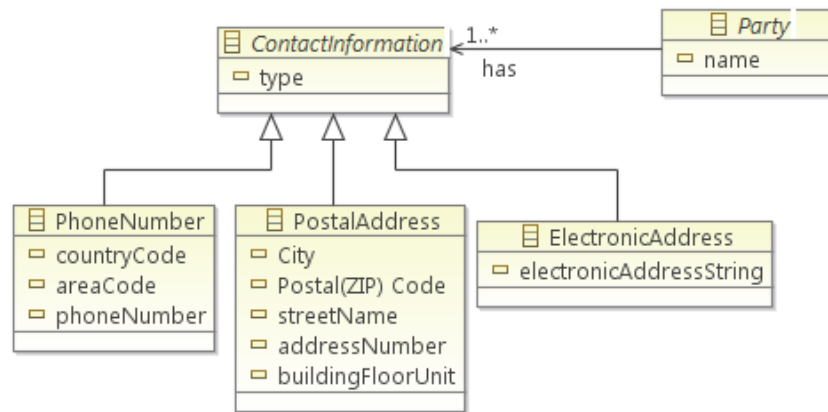


Illustration 46: Contact Information Pattern Candidate

5.5.2.2 Similarity between entities

This criterion helped us discover taxonomical relationships within the domain by comparing entities with one another and discovering similarities, attributes or relationships, between them.

As an example we present Table 22, where we summarize a view of some entities and the alignment of some similar attributes encountered among them, at the right the possible abstraction element that was later analyzed in order to form candidate patterns.

Entity					Abstraction
Person	Doctor	Nurse	Patient	Hospital	
name	name	name	name	name	Party
address	homeAddress	homeAddress	homeAddress	address	
phone	phone	phone	phone	phone	
				purpose	Organization
				email	
				website	
gender	gender	gender	gender		Person
birthdate	birthdate	birthdate	birthdate		
	education	education			Employee
	certification	certification			

Table 22: Comparison between similar entities

After performing the analysis over the Elements Union artifact, candidate patterns such as the Healthcare Party Pattern were discovered as seen in Illustration 47.

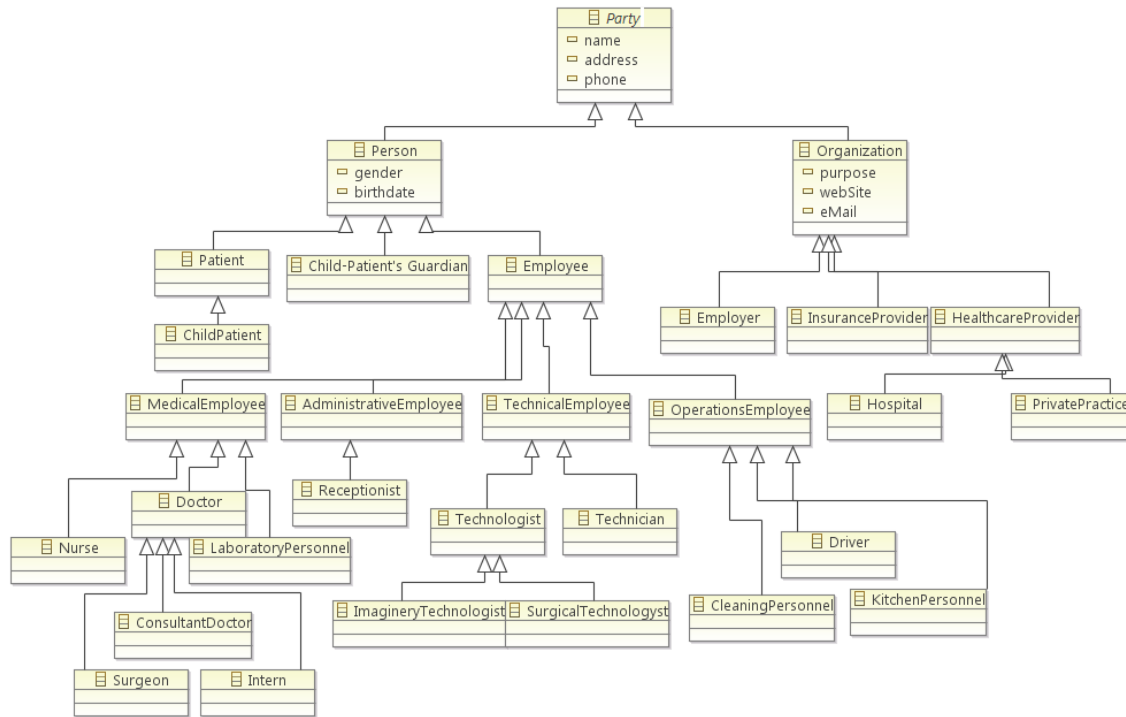


Illustration 47: Healthcare Party Pattern

5.5.2.3 Important entities and their relationships

In order to identify an entity as important we considered the appearances of the entities within models as well as the number of relationships they have with other elements of the Elements Union.

From this analysis we discovered that among other entities, the important entities are: Hospital, employee, department, patient, doctor, medicalRecord, nurse, surgeon, test, diagnosis, prescription, treatment, bed, bill, etc.

As part of the process of applying the criterion we took as core of the cluster each entity and then selected their relationships as part of a pattern.

As an example we can cite the Test Pattern seen in Illustration 48 where we take the element “Test” as center of the cluster and we analyze the relationships that give an insight of what the element is and how it behaves within the domain.

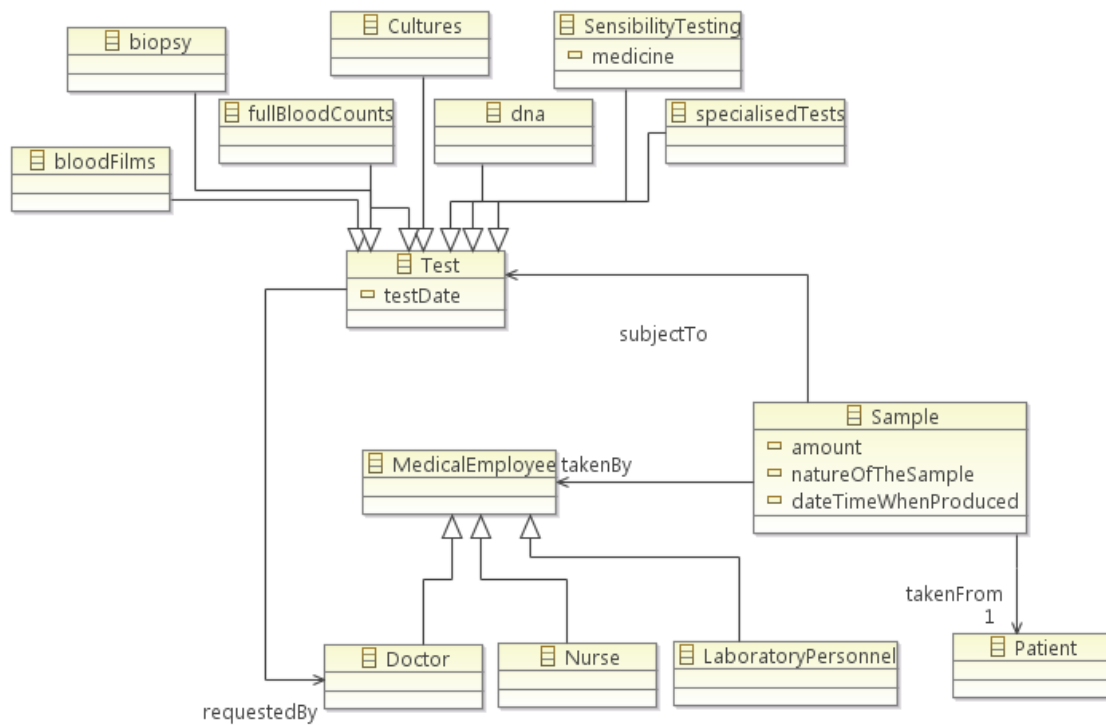


Illustration 48: Test Pattern

This means, tests are of a defined type and are requested by a doctor at a date. In order to perform a test a sample must be taken from a patient by either a doctor, a nurse or at the laboratory.

Another way of seeing how this criterion works is to see the clusters that form patterns as “views” of a more complete model that represents the domain at a certain level of abstraction. This means, if we create complete models of the domain homogeneous according to the level of abstraction, i.e. a hospital model, a private practice model, a laboratory model, etc., we could identify the patterns as reusable blocks among them. An illustration of this can be found in Illustration 49, where we take a piece of a model at an intermediate level of design to form two patterns: Test Pattern and Sample Pattern. They contain some elements that are common, but they are independent due to the possibility of adding particular constraints in an actual implementation. For example: in a specialized laboratory, only some tests are performed over specific samples.

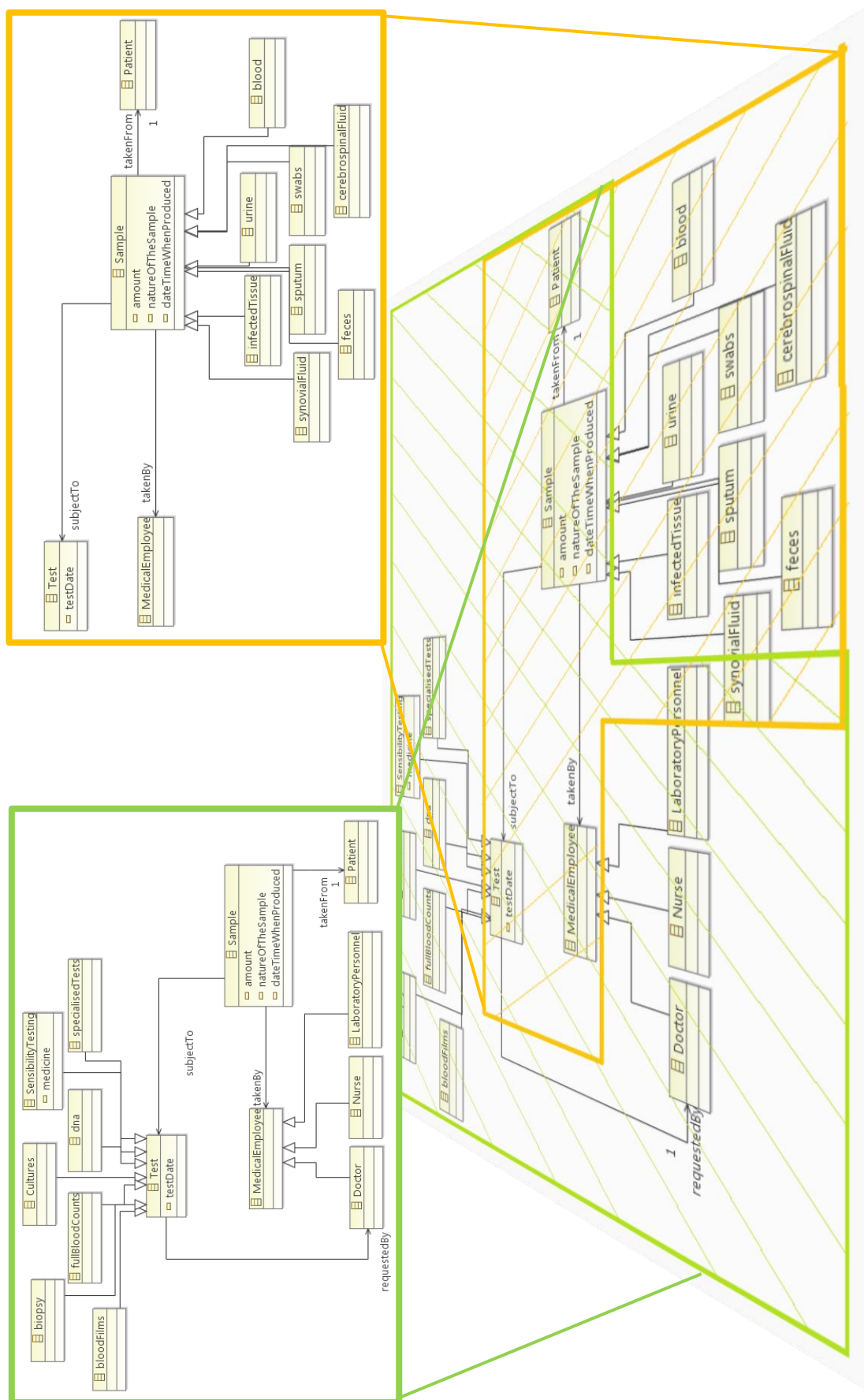


Illustration 49: Patterns as views of a representation of the domain at a certain level of abstraction

5.5.3 Candidate patterns in the patterns catalog

Before proceeding with the insertion of patterns we need to complete the information related to it. Among the information needed we consider name, problem, solution and consequences. Later we perform the classification of the pattern within the classification schema found in section 4.3.2, where we consider Domain and Level of Design as criteria for the classification. Finally the insertion of patterns was performed in the DSL for patterns catalogs that we provide as result of the development based on the metamodel explained in section 4.4. As a way of documentation, we prepare a document containing the patterns in a format where the structure of the patterns is explicit and therefore easier to read and understand.

5.5.3.1 Classify the pattern

The insertion of candidate patterns in the catalog depends on the type of candidate being inserted; therefore it is necessary to assign the patterns to the correspondent classification within the schema described in section 4.3.2 as seen in Table 23.

			Level of Design		
			Early Design	Intermediate Design	Advanced Design
Domain	Domain-Specific	Healthcare Management	Patient's Allergy Antecedent Types Clinical Antecedent Patient's Habit Pediatric Environment Anteced. Familiar Antecedent Obstetric Antecedent Pediatric Birth Antecedent Hospital Types Vital Signs Laboratory Employee Medical Facility Ultrasound Types Movement Disorder Physical Ex. Perception Disorder Physical Ex. Organ System Physical Ex.	Department Types Familiar History Healthcare Party Healthcare Role Observation States Observation Supporting Unit Types Sample Test	Healthcare Physical Examination Hospital Organization Medical Record
		...			
	Cross-Domain		Address Quantity	Contact Information Party Employmen Physical Observation	

Table 23: Candidate patterns classified

As a reminder, we mention that the level of design of the patterns can be:

- **Early Design;** if it gives a general overview of the problem domain that is simple.
- **Intermediate Design;** when the model contains an overview of the problem domain that is more advanced, presenting the application of abstraction of some concepts.
- **Advanced Design;** when advanced relationships and constraints are given in the pattern.

Domain is the area of application of the pattern. For that reason we found in the category:

- **Domain-Specific**

Patterns that cannot be applied in more domains than the one stated in the sub-classification. We analyzed only the healthcare domain.

- **Cross-Domain**

Patterns that are applicable in many domains.

5.5.3.2 Complete pattern information

Once a candidate pattern has been identified and improved by applying all identification criteria, the candidate pattern has to be formalized following the pattern structure presented in Section 4.4.2 Pattern Representation as seen in Illustration 50.

Pattern Name	Sample	Level of design Intermediate
Problem	This pattern addresses the problem of designing the management of samples within the healthcare domain.	
Solution	<p>Abstract the possible samples and the common attributes in a Sample class and identify the interacting elements in the domain.</p> <pre> classDiagram class Sample { amount natureOfTheSample dateTimeWhenProduced } class Test { testDate } class MedicalEmployee class Patient class infectedTissue class urine class blood class synovialFluid class sputum class swabs class feces class cerebrospinalFluid Sample < -- infectedTissue Sample < -- urine Sample < -- blood Sample < -- synovialFluid Sample < -- sputum Sample < -- swabs Sample < -- feces Sample < -- cerebrospinalFluid Sample --> Test : subjectTo Sample --> MedicalEmployee : takenBy Sample --> "1" Patient : takenFrom </pre>	
Consequences	The model using Sample has a clear representation of the sample types and the elements related to it within the domain.	
Related patterns	<u>Test</u>	

Illustration 50: Sample Pattern in the Candidate Patterns Catalog

This ecore model can be used then to create Java implementation of the Domain Patterns Catalog. The generated code, which can be found on the CD attached to this thesis, consists of the following:

- Model, containing the Interfaces and the Factory to create the Java classes.
- model.impl, containing the concrete implementation of the interfaces defined in the metamodel.
- model.util, containing the AdapterFactory

After some adaptations to the generated code that were necessary to polish the presentation of the pattern instances, we use the implementation to generate the plug-ins: the edit plug-in, that provide a wizard for creating new model instances and the plug-in editor which allows us to enter model information.

Finally the contents of the project's development look as shown in Illustration 52, which makes us ready to insert patterns in the catalog.

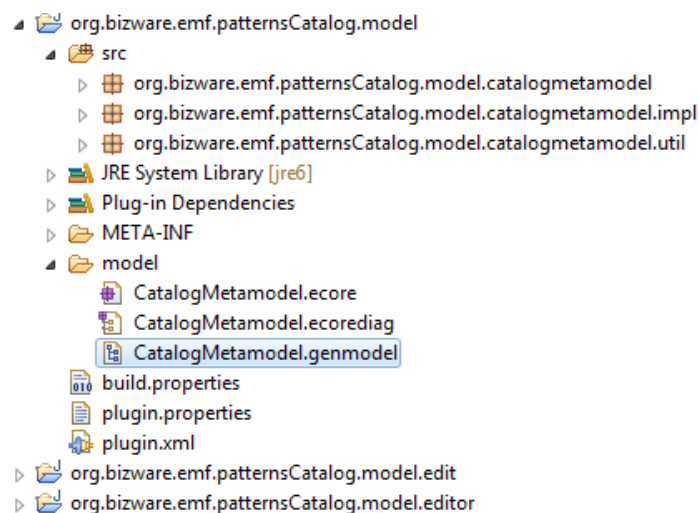


Illustration 52: Domain Patterns Catalog project

5.5.3.4 Insert domain patterns in the catalog

Following the example of the Sample pattern, we show next the process of instantiation of if within the domain patterns catalog.

First of all we have to make sure that the classification schema is instantiated and ready to use, next we add all relevant terminology in order to create the entities, attributes and associations. Then we create the entities and attributes with their respective associations as seen in.

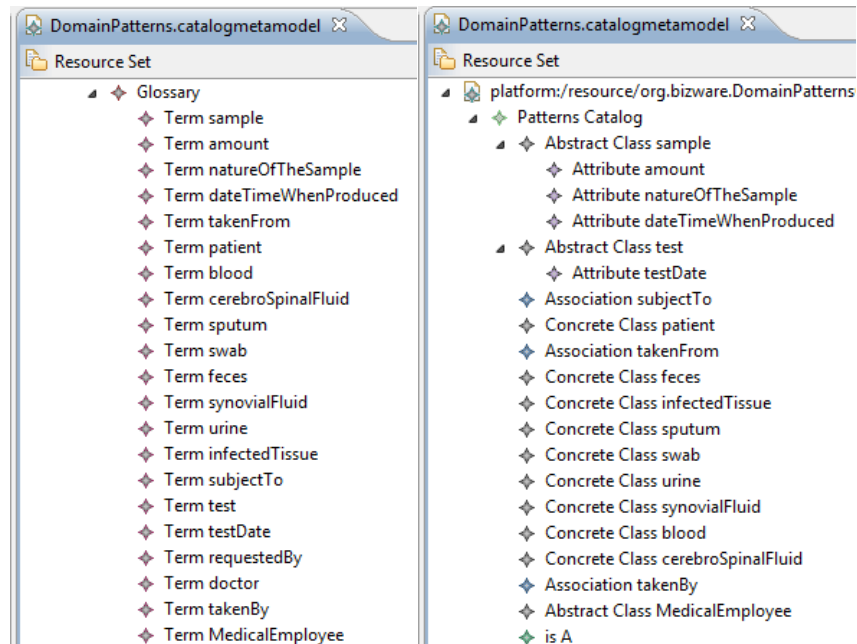


Illustration 53: Instantiation process 1 Left: insertion of the relevant terms. Right: instantiation of the entities, attributes and associations relevant to form the Sample pattern.

Finally, we instantiate the Sample pattern as seen in Illustration 54.

Property	Value
Belongs To	◆ Sub Classifier IntermediateDesign, Sub Classifier HealthcareManagement
Consequences	☰ The model using Sample has a clear representation of the sample types and the elements related to it within the domain
Has	◆ Abstract Class MedicalEmployee, Abstract Class sample, Abstract Class test, Association subjectTo, Association takenBy,
Pattern Name	☰ Sample
Problem	☰ This pattern addresses the problem of designing the management of samples within the healthcare domain.
Related Pattern	◆ Pattern Test
Solution	☰ Abstract the possible samples and the common attributes in a Sample class and identify the interacting elements in the

Illustration 54: Instantiation process 2. Insertion of the Sample pattern

It is important to note that all entities, attributes and relationships are reusable across patterns. Therefore, one could see the patterns as pieces of a complete model of the domain.

5.5.4 Results and Comments

As a result of this step we get the candidate patterns and the Candidate Patterns Catalog in two formats: as implementation and as documentation.

The challenge in this step was to identify the boundaries between one pattern and the next, that is to say how big or small a pattern should be. We realized that the size of the elements forming a pattern is not relevant, as long as the pattern expresses some key aspect of the domain and is able to transmit it to the designer for later reuse.

We discovered that the application of identification criteria helps extremely to speed up the process and to preserve the discipline making the process systematical instead of chaotic or random.

We learned that the use of Wikipedia [73] articles was particularly helpful in order to understand the domain and its particularities. This has lead us to improve the level of design of our models and therefore to get some patterns of advanced level.

The heterogeneity within designs gave us the chance to choose between designs that represented the same, but were designed with more or less entities. In this respect we realized that the entities transmit valuable information on the domain and that we should explore them further. An example of this fact is the concept of allergy that after further exploration lead to the Patient Allergies pattern seen in Illustration 55.

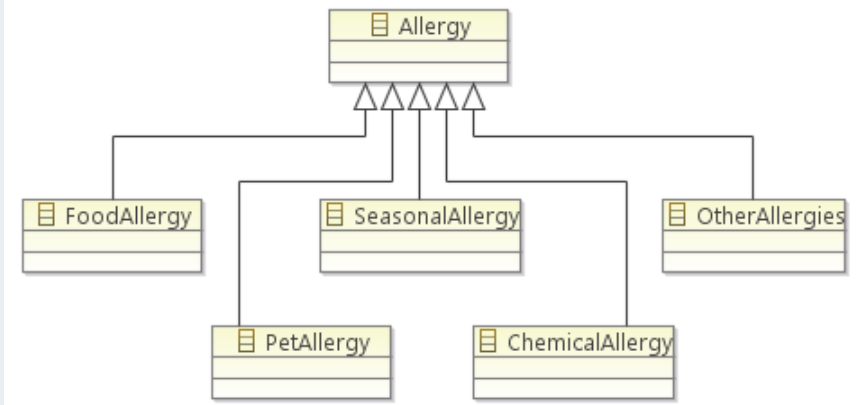
<i>Pattern Name</i>	Patient Allergy	Level of design Early
Problem	This pattern addresses the problem of identifying the possible allergies of a patient.	
Solution	Abstract the possible allergies in an Allergy class.  <pre> classDiagram class Allergy class FoodAllergy class SeasonalAllergy class OtherAllergies class PetAllergy class ChemicalAllergy Allergy < -- FoodAllergy Allergy < -- SeasonalAllergy Allergy < -- OtherAllergies Allergy < -- PetAllergy Allergy < -- ChemicalAllergy </pre>	
Consequences	The model using Patient Allergy has a clear representation of the possible allergy types within the domain.	
Related patterns	<u>Antecedent</u>	

Illustration 55: Patient Allergy pattern

It is important to note that since the patterns identified in this section are only candidate patterns, further discussion is welcome and should be considered, therefore refinements could still be done.

With that in mind we performed a review comparing our resulting patterns to the pattern catalogs described in section 3.4. The result of this review can be seen in Table 24.

Patterns	Amount
Patterns identified	34
Patterns found in other catalogs	16

Table 24: Review of similar patterns found in other catalogs

We feel that this revision may be a sign that the set of candidate patterns is a good result for this work and therefore a good starting point for further future work in the direction of validation with experts, of reusing the patterns and later creating tools to automatically perform the process.

It is important to note that we do not believe the candidate patterns result of this work is definitive. Also we are convinced that with a larger set of source models more patterns can be identified.

The complete set of patterns developed can be found in Appendix C.

5.6 Step 5 - Patterns catalog

Although the scope of the project does not include this step, we consider appropriate to suggest some guidelines in order to finalize the development of the catalog.

In the rest of this section we introduce the suggestion of the artifacts of the step, then a suggestion of the activities that should be performed in this step.

5.6.1 Artifacts of Step 5

The **input** artifacts are:

- Domain Candidate Patterns, described in Step 4.
- Domain Candidate Patterns Catalog, described in Step 4.

The **output** artifact is:

- Domain Patterns Catalog
Domain Patterns Catalog; is a collection of all Domain Patterns, that is to say Domain Candidate Patterns validated according to certain criteria and metrics. It is the resulting artifact from step 5.

5.6.2 Validation suggestions

First of all a validation step has to be performed over the candidate patterns catalog in order to know for sure which candidate patterns are valid patterns.

The validation could be performed by means of surveys to domain experts and domain engineers.

Second of all a validation of the impact of the patterns in novice designers should be performed.

This validation could be performed by means of asking novice users to design sample problems in the domain with and without the aid of the patterns catalog. The designers should later fill in a survey to know how the use of patterns impacts the ease of design. From this experiment, the quality of the resulting designs could be measured and compared.

As other alternative for validation we also suggest the automation of an algorithm that, taking as input meta-models, should perform the following sub steps:

- Search for coincidences in the candidate patterns catalog by using a counter of elements or counter of patterns (that a similar element is already in the catalog).
- Validation of coincidences and update in case the pattern needs changes (element or pattern counter). It should support the insertion of new patterns in the candidate patterns catalog.

Although this last suggestion would require the study of a substantial number of models that should automatically check the occurrences of candidate patterns to either validate it or suggest discarding it.

In any case the definition of the metrics and a method for validation should be further studied and supported by current literature.

5.6.3 Maintenance and expansion suggestions

We suggest that a maintenance protocol should be defined for the patterns catalog.

We propose that the maintenance protocol contain a periodical validation of the state of the catalog as well as a plan to expand the domains covered.

Moreover, those domains could include domain anti-patterns.

6 Summary

The project was conceived with the motivation of getting a product that aids designers to get models of high quality by means of reuse of abstracted knowledge from a domain.

The main objective of the project was to develop a methodology for building a domain patterns catalog. We studied other approaches in this topic and then adapted a methodology for our purposes. For this end we analyzed the method step by step and studied the alternatives to finally discover criteria to perform the steps systematically.

In order to validate the method we analyzed the healthcare management domain, where we discovered and gathered recurring patterns from domain models collected from sources of research and industry.

6.1 Conclusions

Domain analysis requires the participation of domain experts as well as domain engineers. The knowledge produced in designs of the same domain can be abstracted and prepared for reuse.

In the context of this work we discuss domain analysis and model driven development as current approaches that we use and that can take advantage of a method for collection of domain knowledge. Also, we discuss patterns as outstanding means of knowledge packaging and communicating for reuse.

With that in mind, we studied some authors that developed patterns in many areas of engineering in order to understand their methods and create one appropriate for collecting domain patterns from the sources available for us.

As a result a five-step methodology for building domain patterns has been created. Each step has clear boundaries defined, that is to say the sub-steps, the artifacts produced and how the results of the step should look like in preparation for the next step.

We illustrate a walk through the methodology with the purpose of validating it by developing a Domain Patterns Catalog for the healthcare management domain.

For supporting the methodology a metamodel for storing domain patterns and the elements of a catalog was developed, tested and implemented using the EMF framework of Eclipse.

Also an analysis of classification schemas has been performed and a general classification schema was developed and attached to enrich the metamodel for domain pattern catalogs.

The notation of the metamodel's design makes it easy to understand as well as to manipulate if needed. Its flexibility makes it ideal for developing pattern catalogs of other domains of expertise.

The use of Eclipse EMF in the implementation gives the catalog the portability necessary for amplifying the accessibility to the information contained in the catalog.

By following the method proposed, we discover that the process of identification and construction of patterns from models of the domain can be made in an efficient and systematic way; therefore its automation could be a future research step.

The resulting candidate patterns will be available at the repository of works of the DIMA department of the Technical University of Berlin and also at a CD attached to this thesis.

We are confident that the patterns can be useful for researchers interested in further developing and maintaining the catalog as well as for designers, people coming from different backgrounds of expertise, interested in modeling the domains in a uniform way.

Finally, we have made a pre validation of the accuracy of the patterns by comparing them to pattern catalogs, coming to the conclusion that other authors have identified some of our patterns as patterns themselves.

6.2 Issues

Originally the desired sources were XML based files, such as XSD, XMI, OWL among others; but the search of such model files presented several problems, i.e. irrelevant results, heterogeneity, etc. Hence, the start of the project was tremendously affected by this lack of sources available on the internet and the search of sources had to be expanded to model files such as UML diagrams, ER diagrams and SQL Schemas.

We faced a difficult decision point when we had to decide about the pattern representation, whether to use a simpler and broadly used schema or a complex semantic net. Therefore an analysis had to be made and even though knowledge representation models that offer more explanatory power, they demand higher cognitive effort. That is why we explore the option that offers simplicity, a well-known representation that does not need the effort of re-learning and that will finally allow for better systematic reuse of domain knowledge.

As part of the issues concerning the Eclipse EMF framework we identify that after making instances of the metamodel it is dangerous to go back and update the model because of loss of instances. However, some techniques and tools are available to deal with this matter in the area of model consistency.

There is a need to know if the patterns provided actually contribute to a better designing performance, for that reason the implementation of a validation method is imperative. The main issue of this matter is defining and testing clear acceptance levels and metrics.

6.3 Recommendations and Future Work

After the analysis of current literature and after the development of this work we identify opportunities in many areas to enrich and further develop in the topic of this thesis.

First of all we discover the need of a tool for source models acquisition, a first approach to it was the work of Thonggoom [60], although the theory is provided, no tool for models search is available. Therefore a research area could merge the findings of this work and the theory of the author mentioned in order to provide a tool that would retrieve model sources in all sorts of domains. Some other ideas include the extraction of models using Web SCRAPE and Yahoo BOSS API among others.

We recommend as future work to perform of experiments and case studies in order to perform validation of the current state of the catalog. From this analysis the definition of the acceptance levels and metrics should be well documented and tested for other developers to make use of it in conjunction to our method. As some general ideas we cite: Development of a survey with expert designers to evaluate the accuracy of the patterns and development of a survey with novice designers that evaluate how helpful the patterns are in the new known domain.

Another area of future work may consist on the automation of our method. This would include a sort of parser that is able to understand all source model files in order to perform the extraction of elements.

This idea is also applicable to another area of research that would deal with the validation of the candidate patterns. Meaning that having other models and a parser that can process them, one could implement a validator able to set the coincidence rate and therefore validate automatically the patterns in the catalog or even create/suggest new ones.

The Eclipse EMF API can be used as base to read and compare a set of models and a set of patterns. This can not only be an opportunity for validation, but also could be used as base for the implementation of a pattern retrieving mechanism within a graphical tool that may be implemented to aid designers in using the patterns catalog. I.e. the tool could suggest the use of patterns according to the set of domain concepts the designer introduces in the model. The work of Agt [2], deals with this kind of suggestions, although at another level of knowledge reuse.

Finally we think that this thesis can be a further developed in direction of developing a knowledge-based service to support domain-specific modeling under the context of the BIZWARE project [1].

7 Literature

1. Agt, H., Kutsche, R.-D., Natho, N., Li, Y.: The BIZWARE Research Project. In : 15th International Conference, MODELS 2012, Innsbruck, Austria (2012)
2. Agt, H., Kutsche, R.-D.: Supporting Software Language Engineering by Automated Domain Knowledge Acquisition. In Springer, ed. : MODELS 2011, vol. 7167 (2011)
3. Boronat, A., Meseguer, J.: An Algebraic Semantics for MOF, Fundamental Approaches to Software Engineering. In : Lecture Notes in Computer Science, vol. 4961/2008, pp.377-391 (2008)
4. Ludewig, J.: Models in software engineering – an introduction. In Springer, ed. : Software and Systems Modeling Vol. 2. Springer (March 2003) 5–14
5. Stachowiak, H.: General Model Theory. Springer (1973)
6. Fakhroutdinov, K. In: UML diagrams graphical notation overview, tutorials, examples, and reference. (Accessed 10-04-2013 2010) Available at: <http://www.uml-diagrams.org/>
7. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process 1st edn. Addison Wesley (1999)
8. Stahl, T., Voelter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons (2006)
9. Object Management Group: OMG. (Accessed December 2012) Available at: <http://www.omg.org/>
10. Kutsche, R.-D.: Lecture notes Advanced Information Modeling. DIMA-TU Berlin, Berlin (Summer Term 2013)
11. Kleppe, A.: Episode 120: MDD, DSL, UML, OCL with Anneke Kleppe. (Accessed November 2008) Available at: <http://www.se-radio.net/2008/12/episode-120-ocl-with-anneke-kleppe/>
12. Stahl, T., Voelter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons (2006)
13. Prieto-Díaz, R., Arango, G.: Domain Analysis. Press, IEEE Computer Society, USA (1991)

14. Prieto-Diaz, R.: Domain Analysis for reusability. In : COMPSAC 87, pp.22-29 (1987)
15. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute (1990)
16. Fowler, M.: Domain-Specific Languages 1st edn. Addison-Wesley Professional (2010)
17. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Professional (2008)
18. Voelter, M.: DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform (2013)
19. Parr, T.: Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages. Pragmatic Programmers (2010)
20. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF - Eclipse Modeling Framework 2nd edn. Addison Wesley (2009)
21. Eclipse Foundation: Eclipse Modeling Framework Project (EMF). (Accessed March 20013) Available at: <http://www.eclipse.org/modeling/emf/>
22. Gronback, R.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional (2009)
23. Alexander, C.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press (1977)
24. Fowler, M.: Analysis patterns for reusable Object Models. Addison-Wesley (1997)
25. Gamma, E. .: Design Patterns: Elements of reusable software components 2nd edn. Addison-Wesley (1997)
26. Tesanovic, A.: What is a pattern? Dr. ing. course DT8100 (prev. 78901/45942/DIF8901) Object-oriented Systems, Linköping University (2005)
27. Alexander, C. In: Pattern Language. (Accessed April 2013) Available at: <http://www.patternlanguage.com>

28. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons (1996)
29. Coplien, J.: The Patterns Handbook: Techniques, Strategies, and Applications 1st edn. Cambridge University Press (1998)
30. Alexander, C.: The Timeless Way of Building. Oxford University Press (1979)
31. Ward, C., Kent, B.: Using Pattern Languages for Object-Oriented Programs. In : OOPSLA'87 workshop on the Specification and Design (1987)
32. Coad, P., Mayfield, M.: Object Model Patterns. Workshop Report 1-512-795-0202, Portland (1994)
33. Coplien, J., Schmidt, D.: Pattern Languages of Program Design. In Addison-Wesley, ed. : Pattern Languages of Programming - PLoP (1995)
34. Czarnecki, K., Eisenecker, U.: Generative programming: methods, tools, and applications. Addison Wesley (2000)
35. Buschmann, F., Henney, K.: Pattern-Oriented Software Architecture. (Accessed September 2012)
36. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice-Hall (1991)
37. Fraser, S., Leishman, D., McLellan, R.: Patterns, Teams and Domain Engineering. In ACM, ed. : SSR '95 - Symposium on Software reusability , New York, pp.222-224 (1995)
38. Fülleborn, A., Heisel, M.: Methods to Create and Use Cross-Domain Analysis Patterns. In : EuroPLoP' 2006, Eleventh European Conference on Pattern Languages of Programs (2006)
39. Fülleborn, A., Meffert, K., Heisel, M.: Problem-Oriented Documentation of Design Patterns. In Springer-Verlag, ed. : FASE '09 Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, pp.294 - 308 (2009)
40. Tepandi, J., Piho, G., Puusep, V.: Archetypes Based Development from the Perspective of Domain Engineering Research Topics. In : MIPRO 2012, pp.686-691 (2012)

41. Hay, D.: Data model patterns: Conventions of thought. Dorset House Pub (1996)
42. Silverston, L.: The Data Model Resource Book - A library of Universal Patterns for all Enterprises 1. John Willey & Sons (2001)
43. Silverston, L.: The Data Model Resource Book - A library of Universal Data Models by Industry Types 2. John Wiley & Sons (2001)
44. Silverston, L., Agnew, P.: The Data Model Resource Book - Universal Patterns for Data Modeling 3. Wiley Publishing (2009)
45. Vlissides, J., Coplien, J., Kerth, N.: Pattern Languages of Program Design 2 2. Addison-Wesley Professional (1996)
46. Fernandez, E., Yuan, X.: Semantic Analysis Patterns., Florida Atlantic University (2000)
47. Sorgente, T., Fernandez, E., Larrondo, M.: Analysis Patterns for Patient Treatment Records. In : Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2004), pp.8-12 (2005)
48. Sorgente, T., Fernandez, E., Larrondo, M.: The SOAP Pattern for Medical Charts. In : Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2005), pp.7-10 (2005)
49. Vaccare, R., Germano, F., Masiero, P.: A Pattern Language for Business Resource Management. In : In Proceedings of the 6th Pattern Languages of Programs Conference (PLoP'99) (1999)
50. Vaccare, R., Germano, F., Masiero, P.: A Confederation of Patterns for Resource Management. In : Proceedings of Pattern Language of Programs' 98 (PLOP'98) (1998)
51. Arsanjani, A.: Service Provider: A Domain Pattern and its Business Framework Implementation. In : Proceedings of Pattern Language of Programs' 99 (PLOP'99) (1999)
52. Oxford University Press In: The Oxford English Dictionary - Online. (Accessed April 2013) Available at: <http://dictionary.oed.com/>
53. IEEE In: Draft Standard for Learning Object Metadata. (Accessed April 2013) Available at: http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf

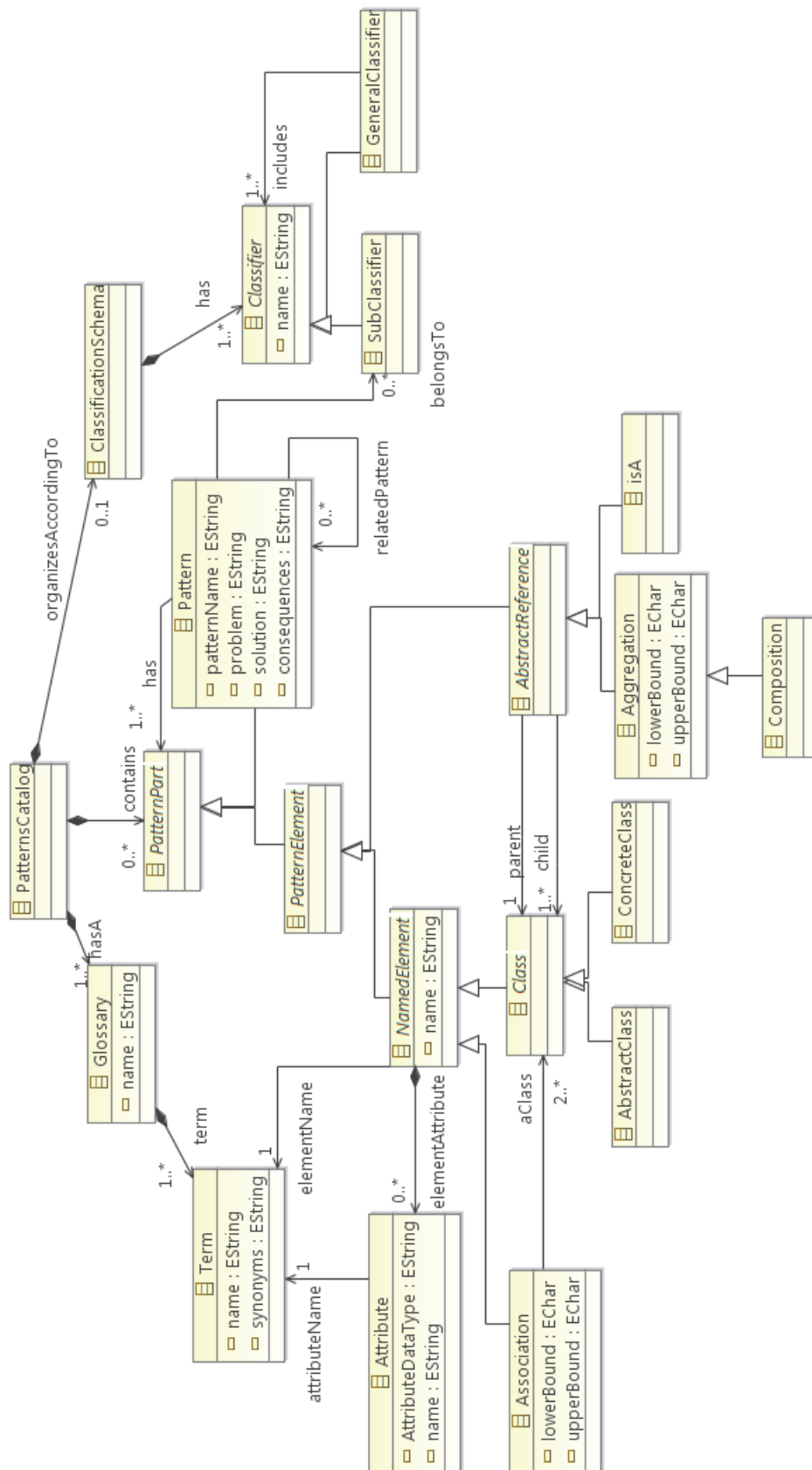
54. Löbe, M., Knuth, M., Roland, M.: TIM: A Semantic Web Application for the Specification of Metadata Items in Clinical Research. Workshop on Semantic Web Applications and Tools for Life Sciences, University of Leipzig, Germany (2009)
55. Health Level Seven International (HL7) In: Health Level Seven. (Accessed April 2013) Available at: <http://www.hl7.org>
56. Federal Geographic Data Committee: Street Address Data Standard. Standard, FGDC (2005)
57. ANZLIC Metadata Profile 1.1. Standard 978-0-646-46940-9 , ANZLIC - The Spatial Information Council, Australia-New Zeland (2007)
58. Carreon, C.: “Construcción de un catálogo de patrones de requisitos funcionales para ERP” – English: Construction of a Functional-Requirements-Patterns Catalogue. Master Thesis, Technical University of Catalunya, Barcelona, Spain (2008)
59. Botella, P., Burgués, X., Carvallo, J. P., Franch, X., Grau, G., Marco, J., Quer, C. In: GESSI: Publications. (Accessed November 2012) Available at: <http://www.essi.upc.edu/~webgessi/publicacions/SMEF'04-ISO-QualityModels.pdf>
60. Thonggoom, O., Song, I.-Y., An, Y.: Semi-automatic Conceptual Data Modeling Using Entity and Relationship Instance Repositories. In Springer, ed. : Conceptual Modeling – ER 2011. (2011) 219-232
61. Han, T.-D., Purao, S., Storey, V.: A Methodology for building a Repository of Object-Oriented Design Fragments. In Springer, ed. : ER, pp.203-217 (1999)
62. Riehle, D.: Composite Design Patterns. In Press, A., ed. : Object-Oriented Programming Systems, Languages and Applications (OOPSLA), pp.218-228 (1997)
63. Vlissides, J.: Composite Design Patterns (They aren't what you think). In Report, C., ed. : Pattern Hatching (1998)
64. Fakhroutdinov, K. In: UML Diagrams. (Accessed March 2013) Available at: www.uml-diagrams.org
65. iSchool at Drexel In: Knowledge Repository of Schemas and Semantics. (Accessed March 2013) Available at: <http://cluster.ischool.drexel.edu:8080/kross>

66. Williams, B. In: Database Answers. (Accessed February 2013) Available at:
<http://www.databaseanswers.org>
67. Bvbsoft Community In: BVBSOft. (Accessed December 2012) Available at:
www.bvbsoft.com
68. Wandelt, B. In: DIMA Repositories: project-bizware - Revision 2647. (Accessed March 2013) Available at: https://svn.dima.tu-berlin.de/svn/project-bizware/public/trunk/theses/benedict_wandelt/
69. Sandy-Martinez, S., Laime, C.: SIIH - Sistema Integrado de Informacion Hospitalaria. Bachelor's Theses, Universidad Mayor Real y Pontificia de San Francisco Xavier de Chuquisaca, Sucre, Bolivia (2009)
70. Lucredio, D.: Models Repository. (Digital Resource) (2012)
71. Repository for Model Driven Development (ReMoDD). (Accessed January 2013) Available at: <http://www.cs.colostate.edu/remodd/v1/>
72. OneTree Technologies S.A.: Metamodel Zoos. (Accessed November 2012) Available at:
<http://www.emn.fr/z-info/atlanmod/index.php/Zoos>
73. The Wikimedia Project: Wikipedia. (Accessed June 2013) Available at:
<http://www.wikipedia.org/>

APPENDIX

Appendix A. Metamodel of the Domain Patterns Catalog

I. UML metamodel



II. Ecore metamodel



Appendix B. Elements Union

I. Entities

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
CN-001	entity	concrete	Hospital name address email website phone	clinic, infirmary	String Address String String Phone	
CN-002	entity	abstract	Person name title gender birthdate address phone		Name String Gender Date Address Phone	
CN-003	entity	concrete	Department idDepartment type name description	area, branch int string string text		
CN-004	entity	abstract	Staff staffId joined education certification languages name title gender birthdate home_address emergencyPhone phone otherStaffDetails	crew, employees, personnel, workers	Date String String String Name String Gender Date Address string Phone	person person person person person person
CN-005	entity	concrete	Patient patientId accepted sickness	inpatient, outpatient, referral,sick person, deseased person patientNo, Social Security numbers, driver's license number inPatientDate Illness	String Date History	

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
			prescriptions	Drugs	String	
			allergies		String	
			special_reqs		String	
			name	Name	Name	person
			title	Occupation	String	person
			gender	Sex	Gender	person
			birthdate		Date	person
			home_address	Address	Address	person
			birthplace		String	
			phone	telephone, landline, homePhone	Phone	person
			height		double	
			weight		double	
			workPhone		String	
			cellPhone		String	
			otherPatientDetails		text	
			maritalStatus		string	
			nextOfKin		string	
CN-006	entity	abstract	OperationsEmployee			
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			Pone		Phone	person
CN-007	entity	abstract	AdministrativeEmployee			
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-008	entity	abstract	TechnicalEmployee			
			Joined		Date	employee
			education		String	employee
			certification		String	employee

E ID	Element Type	as	Value	Synonims	type - cardinalit y	Parent
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-009	entity	concrete	Doctor	doctor, physician, medic		
			doctorID		String	
			specialty		String	
			locations		String	
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-019	entity	concrete	ConsultantDoctor	attending physician, staff physician		
			specialty		String	doctor
			locations		String	doctor
			physician_id		integer	identifier
			name		Name	person
CN-010	entity	concrete	Nurse			
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-011	entity	concrete	Surgeon			
			specialty		String	doctor
			locations		String	doctor
			joined		Date	employee
			education		String	employee
			certification		String	employee

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-012	entity	concrete	FrontDeskEmployee			
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-013	entity	concrete	Receptionist			
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			Pone		Phone	person
CN-014	entity	abstract	Technician			
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-015	entity	abstract	Technologist			
			joined		Date	employee
			education		String	employee
			certification		String	employee

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-016	entity	abstract	Surgical_Technologist			
			joined		Date	employee
			education		String	employee
			certification		String	employee
			languages		String	employee
			name		Name	person
			title		String	person
			gender		Gender	person
			birthdate		Date	person
			home_address		Address	person
			phone		Phone	person
CN-017	entity	concrete	Team	firms		
			Name		String	
CN-018	entity	concrete	Room			Facility
			roomId		RoomId	
			patientsGender		Gender	
			capacity		Integer	
CN-031	entity	concrete	Facility			
			facilityId		String	
			description		Text	
			squareFootage			
CN-020	entity	concrete	Junior_Doctor			
			specialty		String	doctor
			locations		String	doctor
CN-022	entity	concrete	Examination	exam, test		
			idExamination			
			date			
CN-023	entity	concrete	Diagnosis			CIE10
			idDiagnosis			
			date			
CN-024	entity	concrete	Prescription			
			idPrescription			
			date			
			Status			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
CN-025	entity	concrete	Treatment treatmentId treatmentDate treatmentTime	patientDrugsTreatments, intervention	String Date Time	
CN-026	entity	concrete	Therapy			
CN-027	entity	concrete	Surgery	operation, surgery, surgical operation, surgical procedure		
CN-028	entity	concrete	Insurance cardholder company type copay		integer String Numeric(5,2)	
CN-029	entity	concrete	Medical_Procedure Procedure_id name risk Price			
CN-030	entity	concrete	MedicalRecord RedordId PatientId DoctorId physicalExam tests diagnosis recordComponents Date medicalCondition UserName otherRecordDetails	referredTo	String(20)	
CN-032	entity	concrete	Bed idBed status	bedNumber	IdBed BedStatus	
CN-033	entity	concrete	Bill RecordID PatientId Quantity patientBillId dateBillPaid totalAmountDue PaymentType otherBillDetails Items			record

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
CN-034	entity	concrete	Item itemSequenceNr itemName Total UnitId BuyPrice SalePrice		String[50] int int float float	identifier record
CN-035	entity	concrete	cleaningEmployee			
CN-036	entity	concrete	driver			
CN-037	entity	concrete	kitchenEmployee			
CN-038	entity	concrete	laboratoryEmployee			
CN-039	entity	concrete	childPatient legalGuardian			patient
CN-040	entity	concrete	service idService type name price valid?		string float bool	
CN-042	entity	concrete	vaccine idVaccine name description		int string text	
CN-043	entity	concrete	relative idRelative name birthdate relation maritalStatus occupation address telephone		int string date string string string string string	
CN-044	entity	concrete	epicrisis idEpicrisis testResults evolution treatment complications recommendations		int text text text text text	
CN-045	entity	concrete	habit name		string	

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
CN-046	entity	concrete	clinicalBackground idClinicalBackground infections behaviorProblems surgeries hospitalizations other updateDate			
CN-047	entity	concrete	allergy idAllergy type			
CN-048	entity	concrete	order date service patient bill personnel price			
CN-049	entity	concrete	request idRequest date doctorRequesting diagnosis patient type status		{emited, in process, done}	
CN-050	entity	concrete	emergencyConsult broughtBy conditionOfAdmission contactPerson ContactAddress contactPhone legalNotification typeOfAcccident	text		
CN-051	entity	concrete	circulatoryEmergency periferalPulse pulseComment cyanosis capillaryRefillTime cardiacSounds thoracalgia locationOfThoracalgia characteristicsThorocalgia			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
			edema locationOfEdemas	oedema		
CN-052	entity	concrete	digestiveEmergency abdomen abdominalPain vesicalHabits hydroaerialNoises symptoms observations vaginalBleeding observationsBleeding			
CN-053	entity	concrete	vitalSignsEmergency heartRate respiratoryRate bloodPressure temperature oralMucous observations skin injuries		[bpm] [mmHg] [°]	
CN-054	entity	concrete	respiratoryEmergency toraxicalExpansion respiratoryNoises bronchialSecretions auscultation observations			
CN-055	entity	concrete	movementDisorderEmergency limitation partialLimitation membersMovility characteristicsOfMobility walkingAbility deformities			
CN-056	entity	concrete	perceptionEmergency discomfort pain duration alterations characteristicsOfAlteration intensityOfCephalea observationsCephalea nausea fobias emotionalState			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
			personalConflicts alcoholism familiarViolence suicideIntent			
CN-057	entity	concrete	neurologicalEmergency glasgowResult glasgowAO glasgowRV glasgowRM consciousness pupils seizures psychomotorRestlessness motorDeficit	convulsions		
CN-058	entity	concrete	ICD10 Code descriptor version	International Classification of Diseases, International Statistical Classification of Diseases and Related Health Problems		
CN-059	entity	concrete	familiarBackground anemia autism asthma ADHD birthDefects bleedingDisorders cancer cerebralPalsy depression diabetes downSyndrome drugAbuse geneticDisease headaches hearingLoss heartDisease highCholesterol highBloodPressure kidneyDisease liverDisease seizures skinDisease stroke thyroidDisease tuberculosis			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
			ulcers other			
CN-060	entity	concrete	pediatricFamilyBackground smokingExposure dayCare numberOfSiblings pets			
CN-061	entity	concrete	pediatricBirthBackground fullTerm birthWeight delivery deliveryComplications pregnancyIssues brestfeed brestfeedingDuration			
CN-062	entity	concrete	dentalConsult dentalHealth currentDisease bleeding hospitalized headaches pregnancy metalallergies currentDiscomforts mainDiscomfort durationOfDoscomfort lastXRay treatment observations			
CN-063	entity	concrete	dentalTreatmentPlan treatmentPlan observations			
CN-064	entity	concrete	endodonticsFile organ symptoms diagnosis treatment pain discomfort conductometry observations			
CN-065	entity	concrete	dentalExamination missingTeeth caries			

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
			restaurations diastemas cervicalErosion surfaceLoss sensibility			
CN-066	entity	concrete	periodontalExam pockets gingivalRecession furcations mobility redness bleeding swelling frenulumInsertion gumLoss plaque			
CN-065	entity	concrete	pediatricConsult height weight guardian			
CN-066	entity	concrete	obstetricFamilyBackground tbc diabetes hipertension preclampsia others			
CN-067	entity	concrete	obstetricPersonalBackground tbc diabetes hipertension preclampsia previousPregnancies abortions deliveries vaginalDeliveries caesareanDeliveries bornAlive bornDead others			
CN-068	entity	concrete	obstetricConsult prenatalControls previousGenitalTractSurgery infertility hiv			

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
			cardiacConditions otherAcuteMedicalCondition			
CN-069	entity	concrete	obstetricCurrentPregnancy weightPreviousPregnancy abdomenSize lastPeriodDate predictedDueDate fetalMovement cigarretesPerDay alcohol drugs antitetanicImmunization antimeaslesImmunization normalOdontologicalExam normalMamalExamination normalCervixExamination bodyMassIndex bloodType rhFactor papanicolaou proteinInUrin syphilis hemoglobin anemic folates streptococcus pelvicExamination		date date	
CN-070	entity	concrete	obstetricCheckup idCheckup dateCurrentCheckup gestationalAge weight bloodPressure fundalHeight presentation fetalHeartbeat AmountFerrousSulphateSupplement obstetricExams dateNextCheckup			
CN-071	entity	concrete	obstetricPartogram time company position dilatation			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
			fcfDips observations			
CN-072	entity	concrete	obstetricNewborn gender weight height headCircumference gestationalAge physicalExamination apgarFirstMinute apgarFifthtMinute resuscitation diesInDeliveryRoom congenitalDefects congenitalIllnesses vdrlScreening tshScreening chagasScreening bilirubinScreening meconiumFirstDayScreening Observations			
CN-073	entity	concrete	obstetricDischarge dateOfNewbornDischarge statusOfNewbornDischarge ageOfNewborn Lactation weightOfNewbornDischarge dateOfMotherDischarge statusOfMotherDischarge ageOfNewborn Lactation weightOfNewbornDischarge antiMeaslesPostPartumImmunization contraception		date string	,
CN-078	entity	concrete	obstetricPuerperium Time Temperatura Pulse bloodPressure Lochia			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
CN-079	entity	concrete	obstetricBirthOrMiscarriage admissionDate antenatalCorticoidsAdministration dateStartOfDiscomfort ruptureOfAmnioticMembrane weekOfGestationalAge Presentation Delivery dateTimeOfDelivery Multiple Termination indicationsForDelivery codeInducingMiscarriage codeOperation birthPosition Episiotomy tearDegree oxytocinDelivery retainedPlacenta cordLigature			
CN-080	entity	concrete	obstetricBirthOrMiscarriageIllnesses previousHighbloodPressure inductedHighBloodPressure preEclampsia eclampsia cardioVascular diabetes chorioamnionitis urinaryInfection threatenPretermLabour fetalGrowthRestriction prelaborRuptureOfMembranes anemia hemorrhage postPartumHemorrhage postPartumInfections otherSevereCondition			
CN-081	entity	concrete	obstetricBirthOrMiscarriageAdministeredMedicine magnesiumSulfate oxytocin antibiotics analgesics anesthesia transfusions			

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
CN-082	entity	concrete	examLaboratory date		date	
CN-083	entity	concrete	examRadiology name resultsDate description		String date text	
CN-084	entity	concrete	examUltrasonography date requestingDoctor conclusion		Date Doctor text	
CN-085	entity	concrete	uroAnalysis			
CN-086	entity	concrete	chemicalAnalysis			
CN-087	entity	concrete	immunology			
CN-088	entity	concrete	stoolTest			
CN-089	entity	concrete	bacilloscopy			
CN-090	entity	concrete	hematology			
CN-091	entity	concrete	pregnancyTest			
CN-092	entity	concrete	gynecologicalUltrasound			
CN-093	entity	concrete	pregnancyObstetricUltrasound			
CN-094	entity	concrete	obstetricUltrasound			
CN-095	entity	concrete	earlyPregnancyUltrasound			
CN-096	entity	concrete	urologicalUltrasound			
CN-097	entity	concrete	abdominalUltrasound			
CN-098	entity	concrete	consult id date		visit string date	
CN-099	entity	concrete	medicalOffice			
CN-100	entity	concrete	medicalBuilding			
CN-101	entity	concrete	Clinic			
CN-102	entity	concrete	Floor			
CN-103	entity	concrete	operationRoom			
CN-104	entity	concrete	emergencyRoom			
CN-105	entity	concrete	laboratory			
CN-106	entity	abstract	party			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
CN-107	entity	concrete	organization			
CN-108	entity	concrete	guardian			
CN-109	entity	concrete	medicalEmployee			
CN-110	entity	concrete	intern			
CN-111	entity	concrete	imagineryTechnologist			
CN-112	entity	concrete	surgicalTechnologist			
CN-113	entity	concrete	employer			
CN-114	entity	concrete	insuranceProvider			
CN-115	entity	concrete	healthcareProvider			
CN-116	entity	concrete	privatePractice			
CN-117	entity	concrete	role description			
CN-118	entity	concrete	personRole			
CN-119	entity	concrete	insuredPartyRole			
CN-120	entity	concrete	OrganizationRole			
CN-121	entity	concrete	individualPractitioner specialty			
CN-122	entity	concrete	insuredIndividual			
CN-123	entity	concrete	insuredOrganization			
CN-124	entity	concrete	insuredContractHolder			
CN-125	entity	concrete	insuredDependent			
CN-126	entity	concrete	teamOfPractitioners			
CN-127	entity	concrete	institution			
CN-128	entity	concrete	thirdPartyAdministrator			
CN-129	entity	concrete	infections			
CN-130	entity	concrete	surgeries			
CN-131	entity	concrete	hospitalizations			
CN-132	entity	concrete	behaviorProblems			
CN-133	entity	concrete	personalAntecedents			
CN-134	entity	concrete	foodAllergy			
CN-135	entity	concrete	petAllergy			
CN-136	entity	concrete	seasonalAllergy			
CN-137	entity	concrete	chemicalAllergy			

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
CN-138	entity	concrete	metalAllergy			
CN-139	entity	concrete	otherAllergy			
CN-140	entity	concrete	sample amount natureOftheSample dateTimeWhenProduced			
CN-141	entity	concrete	sputum			
CN-142	entity	concrete	swabs			
CN-143	entity	concrete	cerebroSpinalFluid			
CN-144	entity	concrete	feces			
CN-145	entity	concrete	synovialFluid			
CN-146	entity	concrete	urine			
CN-147	entity	concrete	infectedTissue			
CN-148	entity	concrete	blood			
CN-149	entity	concrete	cultures			
CN-150	entity	concrete	fullBloodCounts			
CN-151	entity	concrete	biopsy			
CN-152	entity	concrete	bloodFilms			
CN-153	entity	concrete	dna			
CN-154	entity	concrete	sensibilityTesting medicine			
CN-155	entity	concrete	specializedTest			
CN-156	entity	concrete	observation date time description value			

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
CN-157	entity	concrete	variable			
CN-158	entity	concrete	unitOfMeasure			
CN-159	entity	concrete	testObservation			
CN-160	entity	concrete	physicalObservation			
CN-161	entity	concrete	pathologist			
CN-162	entity	concrete	medicalLaboratoryAssistant (MLA)			
CN-163	entity	concrete	biomedicalScientist (BMS)			
CN-164	entity	concrete	clinicalBioChemist			
CN-165	entity	concrete	pathologistsAssistant (PA)			
CN-166	entity	concrete	medicalLaboratoryTechnician (MLT)			
CN-167	entity	concrete	specimenProcessor			
CN-168	entity	concrete	plebotomist (PBT)			
CN-169	entity	concrete	transcriptionist			
CN-170	entity	concrete	leadTechnicalPersonnel			
CN-171	entity	concrete	laboratoryMedicalDirector			
CN-172	entity	concrete	secretary			
CN-173	entity	concrete	hypothesis			
CN-174	entity	concrete	projection			
CN-175	entity	concrete	activeObservation			
CN-176	entity	concrete	rejectedObservation			
CN-177	entity	concrete	deceasedRelative			
CN-178	entity	concrete	causeOfDeath			

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
CN-179	entity	concrete	tobaccoUse			
CN-180	entity	concrete	alcoholIntake			
CN-181	entity	concrete	exercise			
CN-182	entity	concrete	diet			
CN-183	entity	abstract	physicalExamination			
CN-184	entity	concrete	symptom			
CN-185	entity	concrete	movementDisorder			
CN-186	entity	concrete	neural			
CN-187	entity	concrete	circularory			
CN-188	entity	concrete	digestive			
CN-189	entity	abstract	organSystem			
CN-190	entity	concrete	perceptionDisorder			
CN-191	entity	concrete	vitalSigns			
CN-192	entity	concrete	respiratory			
CN-193	entity	concrete	inPatient			
CN-194	entity	concrete	outPatient			
CN-195	entity	concrete	note dateEntered			
CN-196	entity	concrete	medicalHystoryNote			
CN-197	entity	abstract	antecedent			
CN-198	entity	concrete	familiarHistory			
CN-199	entity	concrete	immunizationHistory			
CN-200	entity	concrete	immunizationTest			
CN-201	entity	concrete	medicalEncounterNote			

E ID	Element Type	as	Value	Synonyms	type - cardinality	Parent
CN-202	entity	concrete	inPatientNote			
CN-203	entity	concrete	complaint nature duration			
CN-204	entity	concrete	historyOfPresentIllness			
CN-205	entity	concrete	assessment			
CN-206	entity	concrete	plan			
CN-207	entity	concrete	drugAdministration			
CN-208	entity	concrete	testResult			
CN-209	entity	concrete	admissionNote			
CN-210	entity	concrete	SOAPNote			
CN-211	entity	concrete	onServiceNote			
CN-212	entity	concrete	preOperativeNote			
CN-213	entity	concrete	operativeNote			
CN-214	entity	concrete	postOperativeNote			
CN-215	entity	concrete	procedureNotes			
CN-216	entity	concrete	dischargeNote			
CN-217	entity	abstract	obstetricInPatientNote			
CN-218	entity	concrete	deliveryNote			
CN-219	entity	concrete	postPartumNote			
CN-220	entity	concrete	address			
CN-221	entity	abstract	contactInformation type			
CN-222	entity	concrete	phoneNumber			
CN-223	entity	concrete	electronicAddress			

E ID	Element Type	as	Value	Synonims	type - cardinality	Parent
CN-224	entity	concrete	postalAddress			
CN-225	entity	concrete	employment type			
CN-226	entity	concrete	position name description			
CN-227	entity	concrete	filledPosition startDate endDate			
CN-228	entity	concrete	quantity amount unit			
CN-229	entity	concrete	phenomenonType			
CN-230	entity	concrete	measurement category			
CN-231	entity	abstract	Object			

II. Relationships

ID	elementType	as	Value	cardinality	Name
RN-001	relations hip	binary-association	person-hospital	n-n	
RN-002	relations hip	aggregation	hospital-department	1-n	
RN-003	relations hip	aggregation	department-staff	1-n	
RN-004	relations hip	isA	staff-person		
RN-005	relations hip	isA	patient-person		
RN-006	relations hip	isA	operationsEmployee-employee		
RN-007	relations hip	isA	administrativeEmployee-employee		
RN-008	relations hip	isA	technicalEmployee-employee		
RN-009	relations hip	binary-association	patient-operationEmployee	n-n	
RN-010	relations hip	isA	doctor-operationsEmployee		
RN-011	relations hip	isA	nurse-operationsEmployee		
RN-012	relations hip	isA	surgeon-doctor		
RN-013	relations hip	isA	frontdesk_staff-administrativeEmployee		
RN-014	relations hip	isA	receptionist-administrativeEmployee		
RN-015	relations hip	isA	technician-technicalEmployee		
RN-016	relations hip	isA	technologist-technicalEmployee		
RN-017	relations hip	isA	surgical_technologist-technicalEmployee		
RN-018	relations hip	aggregation	hospital-team	1-1..n	
RN-019	relations hip	aggregation	hospital-room	1-n	
RN-020	relations hip	isA	consultant_doctor-doctor		
RN-021	relations hip	isA	junior_doctor-doctor		
RN-022	relations hip	composition	team-doctor	0..1-0..n	
RN-023	relations hip	binary-association	team-consultant_doctor	0..1-1	leader

ID	elementType	as	Value	cardinality	Name
RN-024	relations hip	binary-association	team-patient	1-n	
RN-025	relations hip	binary-association	doctor-patient	n-n	treats
RN-026	relations hip	binary-association	consultant_doctor-patient	1-n	
RN-027	relations hip	binary-association	room-patient admissionDate dischargeDate	1-n	
RN-028	relations hip	binary-association	patient-examination	1-n	
RN-029	relations hip	binary-association	patient-diagnosis	1-n	
RN-030	relations hip	binary-association	patient-prescription	1-n	
RN-031	relations hip	binary-association	patient-treatment	1-n	
RN-032	relations hip	binary-association	examination-diagnosis	n-n	
RN-033	relations hip	binary-association	diagnosis-prescription	n-n	resultsIn
RN-034	relations hip	binary-association	prescription-therapy	1-n	
RN-035	relations hip	binary-association	diagnosis-treatment		results_in
RN-036	relations hip	isA	therapy-treatment		
RN-037	relations hip	isA	surgery-treatment		
RN-038	relations hip	binary-association	doctor-examination	1-n	
RN-039	relations hip	binary-association	doctor-diagnosis	n-n	make
RN-040	relations hip	binary-association	doctor-prescription	n-n	write
RN-041	relations hip	binary-association	doctor-treatment	n-n	
RN-042	relations hip	binary-association	patient-insurance insuranceNumber	1-1	
RN-043	relations hip	binary-association	physician-patient visit_date	n-n Date	visits
RN-044	relations hip	terciary-association	patient-physician-medical_procedure visit_date		conducts

ID	elementType	as	Value	cardinality	Name
RN-045	relations hip	binary-association	Medical_Procedure-Medical_Procedure	1-n	follows_up
RN-046	relations hip	binary-association	record-patient	1-1	
RN-047	relations hip	binary-association	record-staff	n-n	
RN-048	relations hip	binary-association	patient-bill	1-n	
RN-049	relations hip	binary-association	room-bed	1-n	has
RN-050	relations hip	binary-association	patient-bed	1-n	
RN-051	relations hip	binary-association	department-staff	1-n	boss Of
RN-052	relations hip	isA	cleaningStaff-staff		
RN-053	relations hip	isA	driver-staff		
RN-054	relations hip	isA	kitchenStaff-staff		
RN-055	relations hip	isA	laboratoryStaff-staff		
RN-056	relations hip	isA	childPatient-patient		
RN-057	relations hip	binary-association	surgeon-surgery		
RN-058	relations hip	binary-association	diagnosis-medicalRecord		
RN-059	relations hip	binary-association	treatment-medicalRecord		
RN-060	relations hip	binary-association	patient-vaccine		
RN-061	relations hip	binary-association	patient-relative		
RN-062	relations hip	binary-association	patient-epicrisis	1-n	
			admissionDate	date	
			admissionDiagnosis	text	
			treatingDoctor		
RN-063	relations hip	binary-association	doctor-epicrisis	n-n	
			dischargeDate	date	
			dischargeCondition	text	
			dischargeDiagnosis		
RN-064	relations hip	binary-association	habit-patient	n-n	

ID	elementType	as	Value	cardinality	Name
RN-065	relationship	binary-association	clinicalBackground-patient	n-n	
RN-066	relationship	binary-association	patient-allergy description	n-n	
RN-067	relationship	binary-association	patient-order	n-n	
RN-068	relationship	binary-association	patient-request	n-n	
RN-069	relationship	binary-association	doctor-request	n-n	
RN-070	relationship	binary-association	service-request	n-n	
RN-071	relationship	binary-association	bill-request	1-n	
RN-072	relationship	isA	emergency-consult		
RN-073	relationship	isA	circulatoryEmergency-emergency		
RN-074	relationship	isA	digestiveEmergency-emergency		
RN-075	relationship	binary-association	vitalSignsEmergency-emergency		
RN-076	relationship	isA	respiratoryEmergency-emergency		
RN-077	relationship	isA	movementDisorderEmergency-emergency		
RN-078	relationship	isA	perceptionEmergency-emergency		
RN-079	relationship	isA	neurologicalEmergency-emergency		
RN-080	relationship	binary-association	diagnosis-ICD10		
RN-081	relationship	binary-association	familiarBackground-patient		
RN-082	relationship	isA	dentalConsult-consult		
RN-083	relationship	aggregation	dentalTreatmentPlan-dentalConsult		
RN-084	relationship	aggregation	endodonticsFile-dentalConsult		
RN-085	relationship	aggregation	dentalExamination-dentalConsult		
RN-086	relationship	aggregation	periodontalExam-dentalConsult		
RN-087	relationship	isA	pediatricConsult-consult		

ID	elementType	as	Value	cardinality	Name
RN-088	relationship	binary-association	obstetricFamilyBackground-patient		
RN-089	relationship	binary-association	obstetricPersonalBackground-patient		
RN-090	relationship	isA	obstetricConsult-consult		
RN-091	relationship	aggregation	obstetricCurrentPregnancy-obstetricConsult		
RN-092	relationship	aggregation	obstetricCheckup-obstetricConsult		
RN-093	relationship	aggregation	obstetricCheckup-doctor		
RN-094	relationship	aggregation	obstetricPartogram-obstetricConsult		
RN-095	relationship	aggregation	obstetricNewborn-obstetricConsult		
RN-096	relationship	binary-association	obstetricNewborn-doctor		
RN-097	relationship	binary-association	obstetricPartogram-doctor		
RN-098	relationship	aggregation	obstetricDischarge-obstetricConsult		
RN-099	relationship	aggregation	obstetricPuerperium-obstetricConsult		
RN-100	relationship	binary-association	obstetricPuerperium-obstetricNewborn		
RN-101	relationship	aggregation	obstetricBirthOrMiscarriage-obstetricConsult timeToDueDate		
RN-102	relationship	aggregation	obstetricBirthOrMiscarriageIllnesses- obstetricBirthOrMiscarriage		
RN-103	relationship	aggregation	obstetricBirthOrMiscarriageAdministeredMedicine- obstetricBirthOrMiscarriage		
RN-104	relationship	isA	pediatricBirthBackground-familiarBackground		
RN-105	relationship	isA	pediatricFamiliarBackground-familiarBackground		
RN-106	relationship	isA	examLaboratory-examination		
RN-107	relationship	isA	examRadiology-examination		
RN-108	relationship	isA	examUltrasonography-examination		
RN-109	relationship	isA	uroAnalysis-examLaboratory		
RN-110	relationship	isA	chemicalAnalysis-examLaboratory		

ID	elementType	as	Value	cardinality	Name
RN-111	relationship	isA	immunology-examLaboratory		
RN-112	relationship	isA	stoolTest-examLaboratory		
RN-113	relationship	isA	bacilloscopy-examLaboratory		
RN-114	relationship	isA	hematology-examLaboratory		
RN-115	relationship	isA	pregnancyTest-examLaboratory		
RN-116	relationship	isA	gynecologicalUltrasound-examUltrasonography		
RN-117	relationship	isA	pregnancyObstetricUltrasound-examUltrasonography		
RN-118	relationship	isA	obstetricUltrasound-examUltrasonography		
RN-119	relationship	isA	earlyPregnancyUltrasound-examUltrasonography		
RN-120	relationship	isA	urologicalUltrasound-examUltrasonography		
RN-121	relationship	isA	abdominalUltrasound-examUltrasonography		
RN-122	relationship	IsA	Ward-facility		
RN-123	relationship	IsA	Laboratory-facility		
RN-123	relationship	IsA	Laboratory-facility		
RN-124	relationship	IsA	hospital-medicalFacility		
RN-125	relationship	IsA	medicalOffice-medicalFacility		
RN-126	relationship	IsA	medicalBuilding-medicalFacility		
RN-127	relationship	IsA	clinic-medicalFacility		
RN-128	relationship	IsA	floor-medicalFacility		
RN-129	relationship	IsA	room-medicalFacility		
RN-130	relationship	isA	operationRoom-medicalFacility		
RN-131	relationship	isA	emergencyRoom-medicalFacility		
RN-132	relationship	isA	laboratory-medicalFacility		
RN-133	relationship	isA	person-party		
RN-134	relationship	isA	organization-party		

ID	elementType	as	Value	cardinality	Name
RN-135	relationship	isA	guardian-person		
RN-136	relationship	isA	employee-person		
RN-137	relationship	isA	nurse-medicalEmployee		
RN-138	relationship	isA	doctor-medicalEmployee		
RN-139	relationship	isA	surgeon-doctor		
RN-140	relationship	isA	consultantDoctor-doctor		
RN-141	relationship	isA	intern-doctor		
RN-142	relationship	isA	laboratoryPersonnel-medicalEmployee		
RN-143	relationship	isA	imagineryTechnologist-technologist		
RN-144	relationship	isA	employer-organization		
RN-145	relationship	isA	insuranceProvider-organization		
RN-146	relationship	isA	healthcareProvider-organization		
RN-147	relationship	isA	hospital-healthcareProvider		
RN-148	relationship	isA	privatePractice-healthcareProvider		
RN-149	relationship	binary-association	party-role	actingAs	
RN-150	relationship	isA	personRole-role		
RN-151	relationship	isA	insuredPartyRole-role		
RN-152	relationship	isA	organizationRole-role		
RN-153	relationship	isA	individualPractitioner-personRole		
RN-154	relationship	isA	guardian-personRole		
RN-155	relationship	isA	patient-personRole		
RN-156	relationship	isA	employee-personRole		
RN-157	relationship	isA	insuredIndividual-insuredPartyRole		
RN-158	relationship	isA	insuredOrganization-insuredPartyRole		
RN-159	relationship	isA	insuranceContractHolder-insuredIndividual		

ID	elementType	as	Value	cardinality	Name
	hip				
RN-160	relations	isA	insuranceDependant-insuredIndividual		
RN-161	relations	isA	healthcareProvider-organizationRole		
RN-162	relations	isA	thirdPartyAdministrator-organizationRole		
RN-163	relations	isA	insuranceProvider-organizationRole		
RN-164	relations	isA	employer-organizationRole		
RN-165	relations	isA	healthcarePractice-healthcareProvider		
RN-166	relations	isA	teamOfPractitioners-healthcareProvider		
RN-167	relations	isA	institution-healthcareProvider		
RN-168	relations	isA	other-healthcareProvider		
RN-169	relations	isA	familiarAntecedent-Antecedent		
RN-170	relations	isA	obstetricFamiliarAntecedent-Antecedent		
RN-171	relations	isA	obstetricPersonalAntecedent-Antecedent		
RN-172	relations	isA	pediatricFamiliarAntecedent-Antecedent		
RN-173	relations	isA	personalAntecedent-Antecedent		
RN-174	relations	isA	clinicalAntecedent-Antecedent		
RN-175	relations	isA	pediatricBirthAntecedent-Antecedent		
RN-176	relations	isA	infection-clinicalAntecedent		
RN-177	relations	isA	surgery-clinicalAntecedent		
RN-178	relations	isA	hospitalization-clinicalAntecedent		
RN-179	relations	isA	behaviourProblem-clinicalAntecedent		
RN-180	relations	isA	other-clinicalAntecedent		
RN-181	relations	isA	foodAllergy-Allergy		
RN-182	relations	isA	petAllergy-Allergy		
RN-183	relations	isA	seasonalAllergy-Allergy		

ID	elementType	as	Value	cardinality	Name
RN-184	relationship	isA	chemicalAllergy-Allergy		
RN-185	relationship	isA	otherAllergy-Allergy		
RN-186	relationship	binary-association	sample-medicalEmployee		
RN-187	relationship	binary-association	sample-patient		
RN-188	relationship	binary-association	sample-test		
RN-189	relationship	isA	sputum-sample		
RN-190	relationship	isA	swabs-sample		
RN-191	relationship	isA	cerebrospinalFluid-sample		
RN-192	relationship	isA	feces-sample		
RN-193	relationship	isA	synovialFluid-sample		
RN-194	relationship	isA	urine-sample		
RN-195	relationship	isA	infectedTissue-sample		
RN-196	relationship	isA	blood-sample		
RN-197	relationship	isA	cultures-test		
RN-198	relationship	isA	fullBloodCount-test		
RN-199	relationship	isA	biopsy-test		
RN-200	relationship	isA	bloodFilms-test		
RN-201	relationship	isA	dna-test		
RN-202	relationship	isA	sensibilityTest-test		
RN-203	relationship	isA	specializedTest-test		
RN-204	relationship	binary-association	test-doctor	requestedBy	
RN-205	relationship	isA	testObservation-Observation		
RN-206	relationship	isA	physicalObservation-Observation		
RN-207	relationship	binary-association	observation-medicalEmployee		

ID	elementType	as	Value	cardinality	Name
RN-208	relationship	binary-association	observation-variable		
RN-209	relationship	binary-association	observation-unitOfMeasure		
RN-210	relationship	binary-association	variable-unitOfMeasure		
RN-211	relationship	binary-association	medicalEmployee-test		
RN-212	relationship	binary-association	physicalObservation-Sample		
RN-213	relationship	binary-association	physicalObservation-Patient		
RN-214	relationship	binary-association	test-testObservation		
RN-215	relationship	isA	pathologist-laboratoryPersonnel		
RN-216	relationship	isA	medicalLaboratoryAssistant (MLA)-laboratoryPersonnel		
RN-217	relationship	isA	biomedicalScientist (BMS)-laboratoryPersonnel		
RN-218	relationship	isA	clinicalBioChemist-laboratoryPersonnel		
RN-219	relationship	isA	pathologistsAssistant (PA)-laboratoryPersonnel		
RN-220	relationship	isA	medicalLaboratoryTechnician (MLT)-technicalEmployee		
RN-221	relationship	isA	specimenProcessor-technicalEmployee		
RN-222	relationship	isA	plebotomist (PBT)-technicalEmployee		
RN-223	relationship	isA	transcriptionist-administrativeEmployee		
RN-224	relationship	isA	leadTechnicalPersonnel-administrativeEmployee		
RN-225	relationship	isA	laboratoryMedicalDirector-administrativeEmployee		
RN-226	relationship	isA	secretary-administrativeEmployee		
RN-227	relationship	isA	hypothesis-observation		
RN-228	relationship	isA	projection-observation		
RN-229	relationship	isA	activeObservation-observation		
RN-230	relationship	isA	rejectedObservation-observation		
RN-231	relationship	binary-association	observation-observation		

ID	elementType	as	Value	cardinality	Name
RN-232	relations hip	binary- association	relative-familiarAntecedents		
RN-233	relations hip	isA	deceasedRelative-relative		
RN-234	relations hip	binary- association	deceasedRelative-causeOfDeath		
RN-235	relations hip	isA	familiarAntecedent-diagnosis		
RN-236	relations hip	isA	causeOfDeath-diagnosis		
RN-237	relations hip	isA	tobaccoUse-habit		
RN-238	relations hip	isA	alcoholIntake-habit		
RN-239	relations hip	isA	exercise-habit		
RN-240	relations hip	isA	diet-habit		
RN-241	relations hip	aggregation	vitalSigns-PhysicalExamination		
RN-242	relations hip	aggregation	organSystem-PhysicalExamination		
RN-243	relations hip	aggregation	movementDisorder-PhysicalExamination		
RN-244	relations hip	aggregation	perceptionDisorder-PhysicalExamination		
RN-245	relations hip	isA	Neural-organSystem		
RN-246	relations hip	isA	circulatory-organSystem		
RN-247	relations hip	isA	digestive-organSystem		
RN-248	relations hip	isA	respiratory-organSystem		
RN-249	relations hip	aggregation	Note-medicalRecord		
RN-250	relations hip	isA	inPatient-patient		
RN-251	relations hip	isA	outPatient-patient		

ID	elementType	as	Value	cardinality	Name
RN-252	relations hip	binary- association	medicalRecord-patient		belongingsT
RN-253	relations hip	binary- association	healthcareProvider-medicalRecord		jurisdictionOnO
RN-254	relations hip	binary- association	note-medicalEmployee		writtenBy
RN-255	relations hip	isA	medicalHistoryNote-note		
RN-256	relations hip	isA	medicalEncounterNote-note		
RN-257	relations hip	isA	testResult-note		
RN-258	relations hip	isA	order-note		
RN-259	relations hip	isA	allergy- medicalHistoryNote		
RN-260	relations hip	isA	antecedent- medicalHistoryNote		
RN-261	relations hip	isA	habit- medicalHistoryNote		
RN-262	relations hip	isA	familiarHistory- medicalHistoryNote		
RN-263	relations hip	isA	immunizationHistory- medicalHistoryNote		
RN-264	relations hip	isA	immunizationTest-immunizationHistory		
RN-265	relations hip	isA	vaccine-immunizationHistory		
RN-266	relations hip	isA	inPatientNote-medicalEncounterNote		
RN-267	relations hip	isA	complaint-medicalEncounterNote		
RN-268	relations hip	isA	historyOfPresentIllness- medicalEncounterNote		
RN-269	relations hip	isA	physicalExamination-medicalEncounterNote		
RN-270	relations hip	isA	admissionNote-inPatientNote		

ID	elementType	as	Value	cardinality	Name
RN-271	relations hip	isA	SOAPNote-inPatientNote		
RN-272	relations hip	isA	onServiceNote-inPatientNote		
RN-273	relations hip	isA	preOperativeNote-inPatientNote		
RN-274	relations hip	isA	operativeNote-inPatientNote		
RN-275	relations hip	isA	postOperativeNote-inPatientNote		
RN-276	relations hip	isA	procedureNote-inPatientNote		
RN-277	relations hip	isA	dischargeNote-inPatientNote		
RN-278	relations hip	isA	obstetricInPatientNote-inPatientNote		
RN-279	relations hip	isA	deliveryNote-obstetricInPatientNote		
RN-280	relations hip	isA	postpartumNote-obstetricInPatientNote		
RN-281	relations hip	aggregation	symptom- historyOfPresentIllness		
RN-282	relations hip	aggregation	observation-assessment		
RN-283	relations hip	binary- association	medicalEncounterNote-assessment		leadsTo
RN-284	relations hip	binary- association	assessment -plan		leadsTo
RN-285	relations hip	binary- association	assessment-diagnosis		leadsTo
RN-286	relations hip	aggregation	order-plan		
RN-287	relations hip	binary- association	order-medicalEmployee		givenTo
RN-288	relations hip	binary- association	order-therapy		of
RN-289	relations hip	binary- association	order-drugAdministration		of

ID	elementType	as	Value	cardinality	Name
RN-290	relationship	binary-association	order-test		of
RN-291	relationship	binary-association	test-testResults		of
RN-292	relationship	binary-association	address-party		locationOf
RN-293	relationship	isA	person-party		
RN-294	relationship	isA	organization-party		
RN-295	relationship	isA	phoneNumber-contactInformation		
RN-296	relationship	isA	postalAddress-contactInformation		
RN-297	relationship	isA	electronicAddress-contactInformation		
RN-298	relationship	binary-association	organization-position		offers
RN-299	relationship	isA	filledPosition-position		
RN-300	relationship	binary-association	filledPosition-employment		filledBy
RN-301	relationship	binary-association	employment-organization		employer
RN-302	relationship	binary-association	employment-person		employee
RN-303	relationship	binary-association	physicalObservation-phenomenonType		measure
RN-304	relationship	binary-association	physicalObservation-measurement		in
RN-305	relationship	binary-association	measurement-quantity		
RN-306	relationship	binary-association	physicalObservation-object		over
RN-307	relationship	isA	party-object		

Appendix C. Domain Candidate Patterns Catalog

Domain Patterns Catalog

The Candidate Patterns

I. Catalog Organization

In this section we explain how to understand the catalog and how to understand the patterns by describing the parts they consist of.

A. Catalog's Classification Schema

The classifiers Level of Design and Domain are described next in the section.

Level of Design			
Early Design		Intermediate Design	Advanced Design
Domain	Patient's Allergy (p.154) Antecedent Types (p.155) Clinical Antecedent (p.155) Patient's Habit (p.156) Pediatric Environment Antecedent (p.156) Familiar Antecedent (p.157) Obstetric Antecedent (p.158) Pediatric Birth Antecedent (p.158) Hospital Types (p.159) Vital Signs (p.159) Laboratory Employee (p.160) Medical Facility (p.160) Ultrasound Types (p.161) Movement Disorder Physical Examination (p.161) Perception Disorder Physical Examination (p.162) Organ System Physical Examination (p.162)	Department Types (p.163) Familiar History (p.163) Healthcare Party (p.164) Healthcare Role (p.165) Observation States (p.165) Observation (p.166) Supporting Unit Types (p.167) Sample (p.168) Test (p.169)	Healthcare Physical Examination (p.170) Hospital Organization (p.171) Medical Record (p.172)
	Healthcare Management		
	Domain-Specific		
Cross-Domain		Contact Information (p.174) Party (p.174) Employment (p.175) Physical Observation (p.175)	

Domain

Domain is the area of application of the pattern, where it can be found and applied.

- **Domain-Specific**
Domain specific patterns are patterns that cannot be applied in more domains than the one stated.
- **Cross-Domain**
Cross-domain patterns are the ones that are universally applicable; they can be found and applied in multiple domains.

Level of Design

The level of design is the kind of models the patterns could be part of.

- **Early Design**
Gives a general overview of the problem domain that is simple. Contains patterns that illustrate and communicate information requirements, show specific entities and attributes within entities.
- **Intermediate Design**
Gives an overview of the problem domain that is more advanced than the previous, the application of abstraction of some concepts can be seen. It is a hybrid approach between early and advanced design.
- **Advanced Design**
Gives a solution that can be found when the design is ready for implementation.
Containing patterns that are foundation for database design, they incorporate flexibility and the application of design patterns, for example.

B. Pattern template description

Here you find a description of how the pattern should be understood

<i>Pattern Name</i>	The name of a pattern describes a design problem, its solution, and consequences in a few words. Naming a pattern makes it easier to think about design and improve communication with the designers who will make use of it.	Level of design The classification the pattern belongs to according to the Level of design classifier
<i>Problem</i>	The problem describes when to apply the pattern by explaining the problem and its context. It addresses the question: What particular design issue or problem does this pattern address?	
<i>Solution</i>	The solution describes the elements that make up the design, their relationships, and attributes. It includes the pattern in its graphical form.	
<i>Consequences</i>	The consequences are the results of applying the pattern. The section aims to aid the designer evaluate design alternatives by understanding the benefits and trade-offs of using the pattern.	
<i>Related patterns</i>	It is the relationship between patterns within the catalog. The section aids the designer know which patterns are closely related or may be also used in combination to the one being observed.	

II. The Candidate Patterns

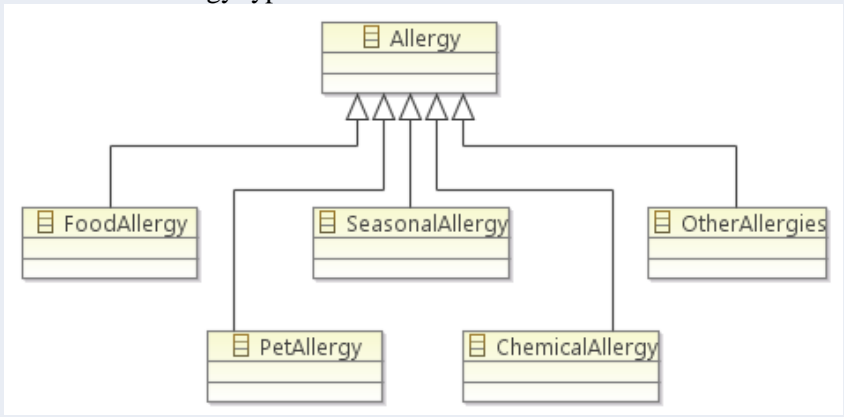
In this section you find the candidate patterns identified.

A. Domain-Specific Patterns

1. Healthcare Management Domain

The patterns in this category are patterns for enterprises that provide treatment of illness and or injuries. Such enterprises include hospitals, emergency rooms, private practices, etc.

a) Patterns for Early Design

<i>Pattern Name</i>	Patient's Allergy	Level of design Early
<i>Problem</i>	This pattern addresses the problem of identifying the allergies of a patient.	
<i>Solution</i>	Abstract the allergy types.  <pre> classDiagram class Allergy { } class FoodAllergy { } class SeasonalAllergy { } class OtherAllergies { } class PetAllergy { } class ChemicalAllergy { } Allergy < -- FoodAllergy Allergy < -- SeasonalAllergy Allergy < -- OtherAllergies Allergy < -- PetAllergy Allergy < -- ChemicalAllergy </pre>	
<i>Consequences</i>	The model using Patient's Allergy has a clear representation of the possible allergy types within the domain.	
<i>Related patterns</i>		

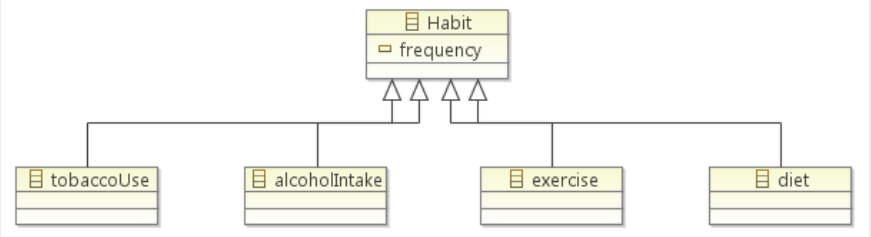
Candidate Pattern 1: Patient's Allergy

<i>Pattern Name</i>	Antecedent Types	Level of design Early
Problem	This pattern addresses the problem of identifying the antecedent of a patient.	
Solution	Abstract the important information necessary to identify patient antecedents.	
	<pre> classDiagram class Antecedent class FamiliarAntecedents class ObstetricFamilyAntecedents class PediatricEnvironmentAntecedents class PersonalAntecedents class ObstetricPersonalAntecedents class ClinicalAntecedents class PediatricBirthAntecedents Antecedent < -- FamiliarAntecedents Antecedent < -- ObstetricFamilyAntecedents Antecedent < -- PediatricEnvironmentAntecedents Antecedent < -- PersonalAntecedents ObstetricFamilyAntecedents < -- ObstetricPersonalAntecedents PersonalAntecedents < -- ClinicalAntecedents PersonalAntecedents < -- PediatricBirthAntecedents ClinicalAntecedents --> date </pre>	
Consequences	The model using Antecedent Types has a clear representation of the antecedents of the patient.	
Related patterns	<u>Medical Record</u>	


Candidate Pattern 2: Antecedent Types

<i>Pattern Name</i>	Clinical Antecedent	Level of design Early
Problem	This pattern addresses the problem of identifying the clinical antecedent of a patient.	
Solution	Identify the important information necessary to identify patient antecedents.	
	<pre> classDiagram class ClinicalAntecedents class Infections class Surgeries class Hospitalizations class behaviorProblems class Other ClinicalAntecedents < -- Infections ClinicalAntecedents < -- Surgeries ClinicalAntecedents < -- Hospitalizations ClinicalAntecedents < -- behaviorProblems ClinicalAntecedents < -- Other ClinicalAntecedents --> date </pre>	
Consequences	The model using Clinical Antecedents has a clear representation of the possible clinical antecedents of a patient.	
Related patterns	<u>Antecedent Types</u>	

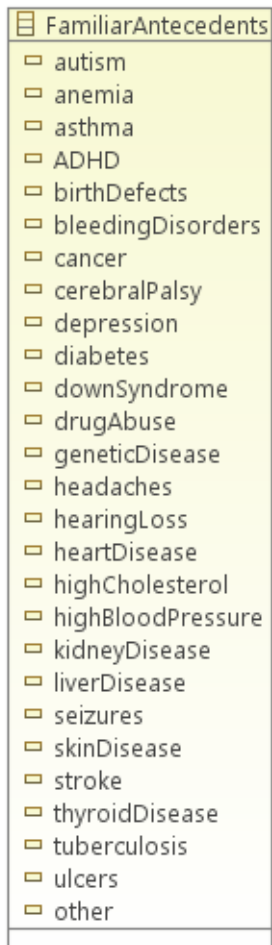
Candidate Pattern 3: Clinical Antecedent

<i>Pattern Name</i>	Patient's Habit	Level of design Early
Problem	This pattern addresses the problem of identifying the habits of a patient.	
Solution	Abstract the habit types.  <pre> classDiagram class Habit { frequency } class tobaccoUse class alcoholIntake class exercise class diet Habit < -- tobaccoUse Habit < -- alcoholIntake Habit < -- exercise Habit < -- diet </pre>	
Consequences	The model using Patient's Habit has a clear representation of the possible allergy types within the domain.	
Related patterns		

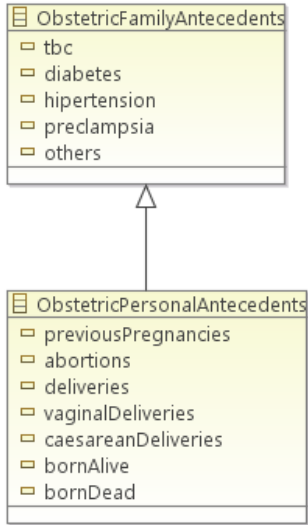
Candidate Pattern 4: Patient's Habit

<i>Pattern Name</i>	Pediatric Environment Antecedent	Level of design Early
Problem	This pattern addresses the problem of identifying the relevant environmental antecedents of a pediatric patient.	
Solution	Abstract the important information necessary to identify patient antecedents.  <pre> classDiagram class PediatricEnvironmentAntecedents { smokingExposure attendsDaycare numberOfSiblings pets } </pre>	
Consequences	The model using Pediatric Environment Antecedent takes into account the relevant information of antecedents for a pediatrics patient.	
Related patterns	<u>Antecedent Types</u>	

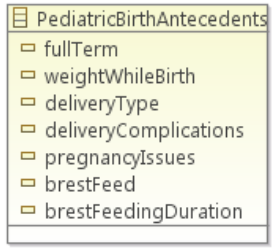
Candidate Pattern 5: Pediatric Environment Antecedent

<i>Pattern Name</i>	Familiar Antecedent	Level of design Early
<i>Problem</i>	This pattern addresses the problem of identifying the familiar antecedent of a patient.	
<i>Solution</i>	<p>Abstract the important information necessary to identify patient antecedents.</p> 	
<i>Consequences</i>	The model using Familiar Antecedents has a clear representation of the possible familiar antecedents of a patient.	
<i>Related patterns</i>	<u>Antecedent Types</u>	

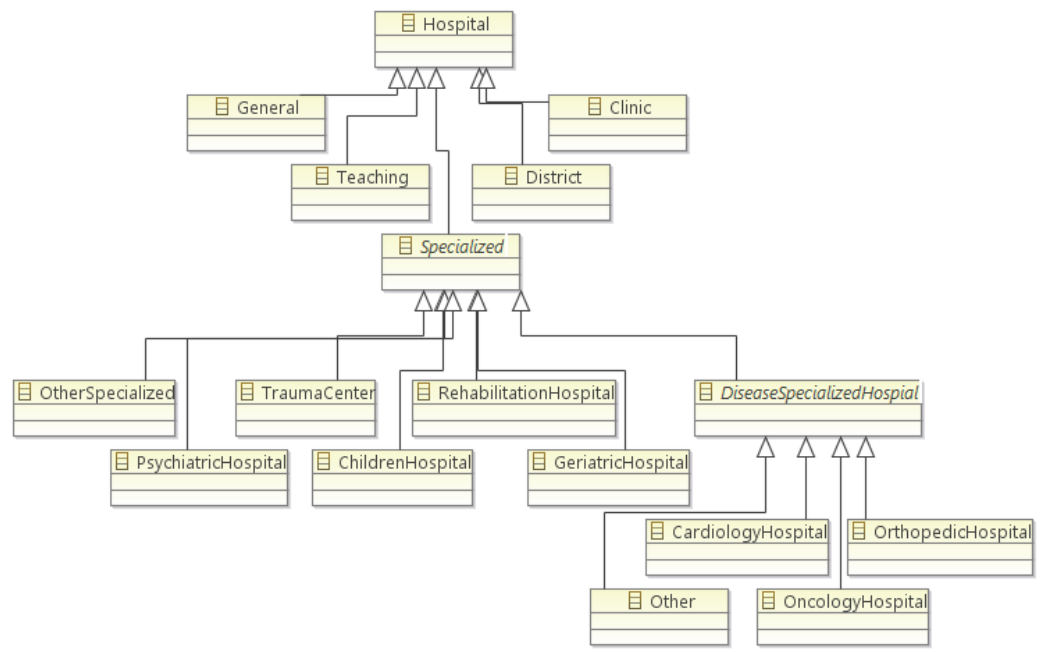
Candidate Pattern 6: Familiar Antecedent

<i>Pattern Name</i>	Obstetric Antecedent	Level of design Early
Problem	This pattern addresses the problem of identifying the relevant antecedent of an obstetrics patient.	
Solution	<p>Abstract the important information necessary to identify patient antecedents.</p>  <pre> classDiagram class ObstetricFamilyAntecedents { tbc diabetes hipertension preclampsia others } class ObstetricPersonalAntecedents { previousPregnancies abortions deliveries vaginalDeliveries caesareanDeliveries bornAlive bornDead } ObstetricPersonalAntecedents -- > ObstetricFamilyAntecedents </pre>	
Consequences	The model using Obstetric Antecedents has a clear representation of the possible obstetric antecedents of a patient.	
Related patterns	<u>Antecedent Types</u>	

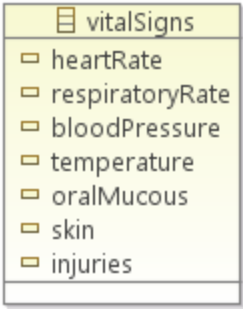
Candidate Pattern 7: Obstetric Antecedent

<i>Pattern Name</i>	Pediatric Birth Antecedent	Level of design Early
Problem	This pattern addresses the problem of identifying the relevant information of birth antecedents of a pediatric patient.	
Solution	<p>Abstract the important information necessary to identify patient antecedents.</p>  <pre> classDiagram class PediatricBirthAntecedents { fullTerm weightWhileBirth deliveryType deliveryComplications pregnancyIssues brestFeed brestFeedingDuration } </pre>	
Consequences	The model using Pediatric Birth Antecedent takes into account the relevant information of antecedents for a pediatrics patient.	
Related patterns	<u>Antecedent Types</u>	

Candidate Pattern 8: Pediatric Birth Antecedent

Pattern Name	Hospital Types	Level of design Early
Problem	This pattern addresses the problem of identifying the hospital types within the healthcare domain.	
Solution	Abstract in Hospital the possible types and subtypes of hospitals.  <pre> classDiagram class Hospital class General class Teaching class District class Clinic class Specialized class OtherSpecialized class TraumaCenter class RehabilitationHospital class DiseaseSpecializedHospital class PsychiatricHospital class ChildrenHospital class GeriatricHospital class CardiologyHospital class OrthopedicHospital class Other class OncologyHospital Hospital < -- General Hospital < -- Teaching Hospital < -- District Hospital < -- Clinic Hospital < -- Specialized Specialized < -- OtherSpecialized Specialized < -- TraumaCenter Specialized < -- RehabilitationHospital Specialized < -- DiseaseSpecializedHospital DiseaseSpecializedHospital < -- PsychiatricHospital DiseaseSpecializedHospital < -- ChildrenHospital DiseaseSpecializedHospital < -- GeriatricHospital DiseaseSpecializedHospital < -- CardiologyHospital DiseaseSpecializedHospital < -- OrthopedicHospital DiseaseSpecializedHospital < -- Other DiseaseSpecializedHospital < -- OncologyHospital </pre>	
Consequences	The model using Hospital Types has a clear representation of the taxonomical relationship of Hospital.	
Related patterns		

Candidate Pattern 9: Hospital Types

Pattern Name	Vital Signs	Level of design Early
Problem	This pattern addresses the problem of identifying the relevant information of vital signs in a patient.	
Solution	Abstract the important information necessary to identify patient's vital signs.  <pre> classDiagram class vitalSigns { heartRate respiratoryRate bloodPressure temperature oralMucous skin injuries } </pre>	
Consequences	The model using Vital Signs takes into account the relevant information of vital signs for a patient.	
Related patterns	<u>Physical Examination</u>	

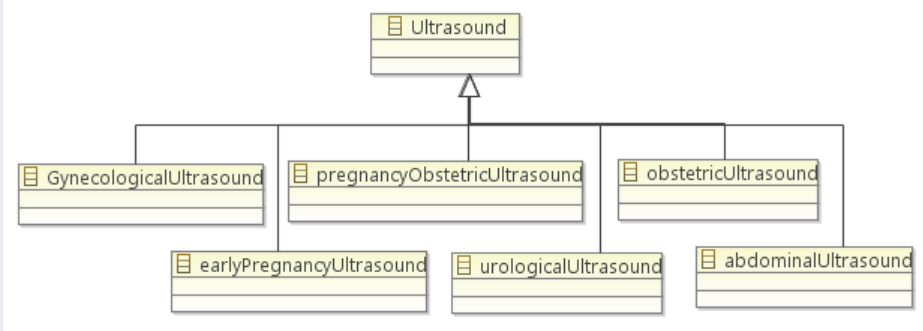
Candidate Pattern 10: Vital Signs

Pattern Name	Laboratory Employee	Level of design Early
Problem	This pattern addresses the problem of designing all employee types hierarchy within the healthcare domain.	
Solution	Abstract in employee the commonalities to laboratory employees and their organization.	
	<pre> classDiagram class Employee class MedicalEmployee class TechnicalEmployee class AdministrativeEmployee class LaboratoryPersonnel class Pathologist class ClinicalBiochemist class MedicalLaboratoryAssistant [MLA] class BiomedicalScientist [BMS] class MedicalLaboratoryTechnician [MLT] class PathologistsAssistant [PA] class SpecimenProcessor class Phlebotomist [PBT] class Transcriptionist class LeadTechnicalPersonnel class Secretary class LaboratoryMedicalDirector Employee < -- MedicalEmployee Employee < -- TechnicalEmployee Employee < -- AdministrativeEmployee MedicalEmployee < -- LaboratoryPersonnel LaboratoryPersonnel < -- Pathologist LaboratoryPersonnel < -- ClinicalBiochemist LaboratoryPersonnel < -- MedicalLaboratoryAssistant LaboratoryPersonnel < -- BiomedicalScientist TechnicalEmployee < -- MedicalLaboratoryTechnician TechnicalEmployee < -- PathologistsAssistant TechnicalEmployee < -- SpecimenProcessor TechnicalEmployee < -- Phlebotomist TechnicalEmployee < -- Transcriptionist TechnicalEmployee < -- LeadTechnicalPersonnel AdministrativeEmployee < -- Secretary AdministrativeEmployee < -- LaboratoryMedicalDirector </pre>	
Consequences	The model using Laboratory Employee identifies the location of laboratory employees within the organizational structure of employees of a healthcare providing institution.	
Related patterns	<u>Healthcare Party</u>	

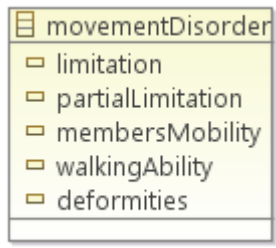
Candidate Pattern 11: Laboratory Employee

Pattern Name	Medical Facility	Level of design Early
Problem	The pattern addresses the problem of designing a medical facility used by the healthcare service provider.	
Solution	Medical facility helps abstract the healthcare providing institution and its facilities.	
	<pre> classDiagram class MedicalFacility class Hospital class MedicalOffice class MedicalBuilding class Floor class Clinic class Room class OperationRoom class Laboratory class EmergencyRoom MedicalFacility < -- Hospital MedicalFacility < -- MedicalOffice MedicalFacility < -- MedicalBuilding MedicalFacility < -- Floor MedicalFacility < -- Clinic MedicalFacility < -- Room MedicalFacility < -- OperationRoom MedicalFacility < -- Laboratory MedicalFacility < -- EmergencyRoom </pre>	
Consequences	The model using Medical Facility eases the design for covering requirements such as scheduling and accounting of the facilities.	
Related patterns	<u>Hospital Organization</u>	

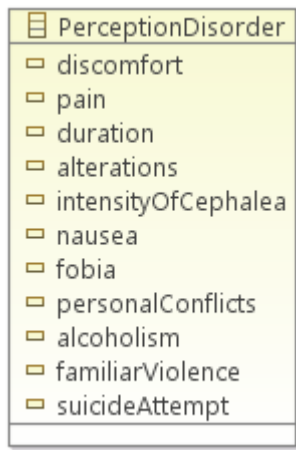
Candidate Pattern 12: Medical Facility

<i>Pattern Name</i>	Ultrasound Types	Level of design Early
Problem	This pattern addresses the problem of identifying the types of ultrasound exams.	
Solution	Abstract the important information necessary to identify ultrasound exams.  <pre> classDiagram class Ultrasound class GynecologicalUltrasound class pregnancyObstetricUltrasound class obstetricUltrasound class earlyPregnancyUltrasound class urologicalUltrasound class abdominalUltrasound Ultrasound < -- GynecologicalUltrasound Ultrasound < -- pregnancyObstetricUltrasound Ultrasound < -- obstetricUltrasound Ultrasound < -- earlyPregnancyUltrasound Ultrasound < -- urologicalUltrasound Ultrasound < -- abdominalUltrasound </pre>	
Consequences	The model using Ultrasound Types takes into account the types of ultrasonography exams.	
Related patterns		

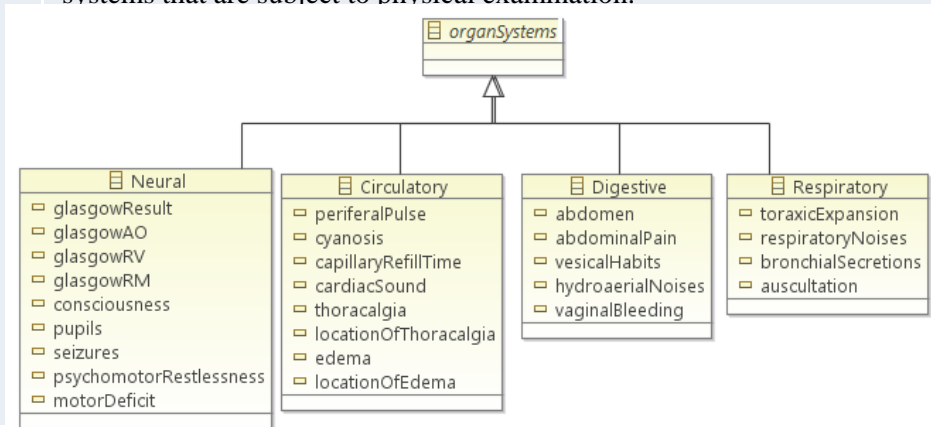
Candidate Pattern 13: Ultrasound Types

<i>Pattern Name</i>	Movement Disorder Physical Examination	Level of design Early
Problem	This pattern addresses the problem of identifying the relevant information of a movement disorder physical examination.	
Solution	Abstract the important information necessary to identify movement disorder.  <pre> classDiagram class movementDisorder { limitation partialLimitation membersMobility walkingAbility deformities } </pre>	
Consequences	The model using Movement Disorder Physical Examination takes into account the relevant information of movement disorders of a patient.	
Related patterns	<u>Physical Examination</u>	

Candidate Pattern 14: Movement Disorder Physical Examination

<i>Pattern Name</i>	Perception Disorder Physical Examination	Level of design Early
Problem	This pattern addresses the problem of identifying the relevant information of perception disorders in a patient.	
Solution	Abstract the important information necessary to identify patient's perception disorders.  <pre> classDiagram class PerceptionDisorder { discomfort pain duration alterations intensityOfCephalea nausea fobia personalConflicts alcoholism familiarViolence suicideAttempt } </pre>	
Consequences	The model using Perception Disorder Physical Examination takes into account the relevant information of perception disorders for a patient.	
Related patterns	<u>Physical Examination</u>	

Candidate Pattern 15: Perception Disorder Physical Examination

<i>Pattern Name</i>	Organ System Physical Examination	Level of design Early
Problem	This pattern addresses the problem of identifying the relevant information of organ system physical examination to a patient.	
Solution	Abstract the important information necessary to identify patient's organ systems that are subject to physical examination.  <pre> classDiagram class organSystems class Neural { glasgowResult glasgowAO glasgowRV glasgowRM consciousness pupils seizures psychomotorRestlessness motorDeficit } class Circulatory { periferalPulse cyanosis capillaryRefillTime cardiacSound thoracalgia locationOfThoracalgia edema locationOfEdema } class Digestive { abdomen abdominalPain vesicalHabits hydroaerialNoises vaginalBleeding } class Respiratory { toraxicalExpansion respiratoryNoises bronchialSecretions auscultation } organSystems < -- Neural organSystems < -- Circulatory organSystems < -- Digestive organSystems < -- Respiratory </pre>	
Consequences	The model using Organ System Physical Examination takes into account the types of organ systems and the relevant information of each during a physical examination.	
Related patterns	<u>Physical Examination</u>	

Candidate Pattern 16: Organ System Physical Examination

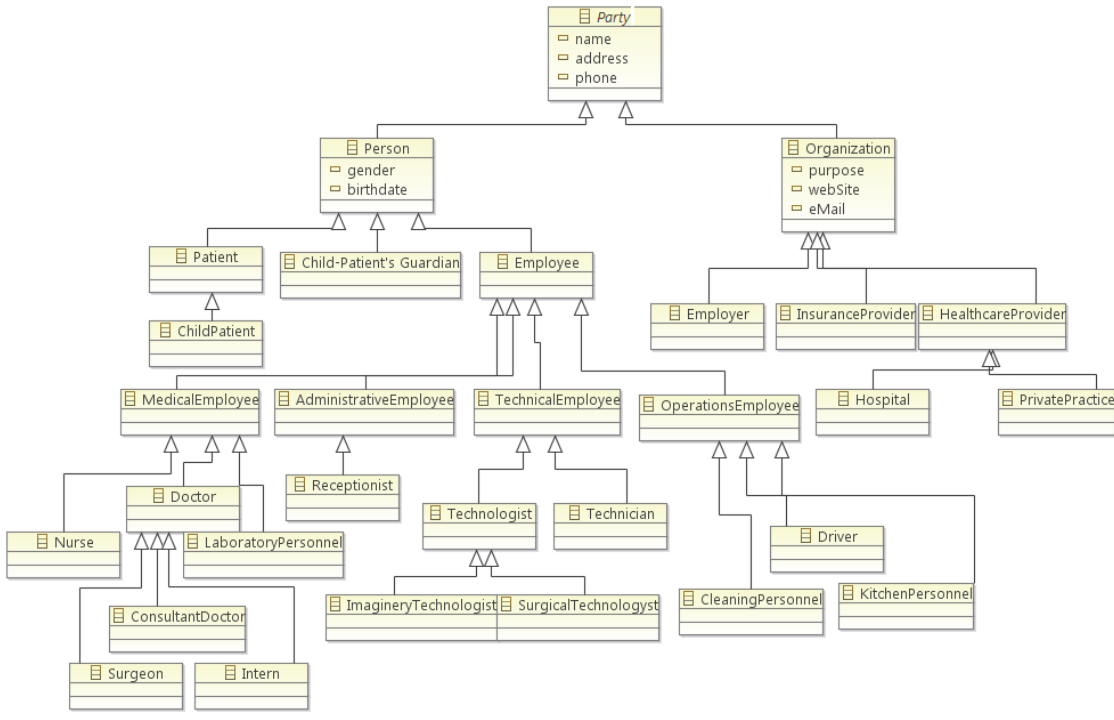
b) Patterns for Intermediate Design

Pattern Name	Department Types (H. Organization)	Level of design Intermediate
Problem	This pattern addresses the problem of identifying the department types within the healthcare domain.	
Solution	Abstract all possible type of departments and the organization of them.	
	<pre> classDiagram class Department { name description } class PhysicalTherapy class Oncology class Obstetrics class Emergency class Trauma class Burn class Nephrology class Dermatology class Gynecology class Neurology class Cardiology class Orthopedics class Gastroenterology class Surgery class Psychiatric class Dentistry class Ophthalmology class IntensiveCare class Otolaryngology class Pediatric class Neonatal class CardioVascular class Anaesthetics Department < -- PhysicalTherapy Department < -- Oncology Department < -- Obstetrics Department < -- Emergency Department < -- Trauma Department < -- Burn Department < -- Nephrology Department < -- Dermatology Department < -- Gynecology Department < -- Neurology Department < -- Cardiology Department < -- Orthopedics Department < -- Gastroenterology Department < -- Surgery Department < -- Psychiatric Department < -- Dentistry Department < -- Ophthalmology Department < -- IntensiveCare Department < -- Otolaryngology Department < -- Pediatric Department < -- Neonatal Department < -- CardioVascular Department < -- Anaesthetics IntensiveCare < -- Pediatric IntensiveCare < -- Neonatal IntensiveCare < -- CardioVascular Surgery < -- Anaesthetics </pre>	
Consequences	The model using Department Types contains an abstraction of the taxonomy of departments in the healthcare domain.	
Related patterns	<u>Hospital Organization</u>	

Candidate Pattern 17: Department Types

Pattern Name	Familiar History	Level of design Intermediate
Problem	This pattern addresses the problem of designing the relevance of the familiar history in relation to a patient within the healthcare domain.	
Solution	<p>The familiar history lists the health status of immediate family members as well as their causes of death (if known)</p> <pre> classDiagram class Patient class Relative { relationship } class DeceasedRelative class FamiliarAntecedent class CauseOfDeath class Diagnosis Patient --> Relative : isRelatedTo Relative < -- DeceasedRelative DeceasedRelative --> CauseOfDeath : dueTo CauseOfDeath --> Diagnosis Relative --> FamiliarAntecedent : sufferFrom FamiliarAntecedent --> Diagnosis </pre>	
Consequences	The model using Familiar History considers a representation of the family history of the patient.	
Related patterns	<u>Antecedent Types</u>	

Candidate Pattern 18: Familiar History

Pattern Name	Healthcare Party	Level of design Intermediate
Problem	This pattern addresses the problem of identifying the people and organizations within the domain.	
Solution	Abstract in party the parties present within the domain of healthcare.	
 <pre>classDiagram class Party { name address phone } class Person { gender birthdate } class Organization { purpose webSite eMail } class Patient class ChildPatient class ChildPatientGuardian["Child-Patient's Guardian"] class Employee class MedicalEmployee class AdministrativeEmployee class TechnicalEmployee class OperationsEmployee class Hospital class PrivatePractice class InsuranceProvider class Employer class Doctor class Nurse class Receptionist class LaboratoryPersonnel class ConsultantDoctor class Surgeon class Intern class Technologist class Technician class ImaginaryTechnologist class SurgicalTechnologist class Driver class CleaningPersonnel class KitchenPersonnel Party < -- Person Party < -- Organization Person < -- Patient Person < -- ChildPatient Person < -- ChildPatientGuardian Person < -- Employee Employee < -- MedicalEmployee Employee < -- AdministrativeEmployee Employee < -- TechnicalEmployee Employee < -- OperationsEmployee Organization < -- InsuranceProvider Organization < -- Employer Organization < -- HealthcareProvider HealthcareProvider < -- Hospital HealthcareProvider < -- PrivatePractice MedicalEmployee < -- Doctor MedicalEmployee < -- Nurse MedicalEmployee < -- Receptionist MedicalEmployee < -- LaboratoryPersonnel MedicalEmployee < -- ConsultantDoctor MedicalEmployee < -- Surgeon MedicalEmployee < -- Intern TechnicalEmployee < -- Technologist TechnicalEmployee < -- Technician TechnicalEmployee < -- ImaginaryTechnologist TechnicalEmployee < -- SurgicalTechnologist OperationsEmployee < -- Driver OperationsEmployee < -- CleaningPersonnel OperationsEmployee < -- KitchenPersonnel</pre>		
Consequences	The model using Healthcare Party has a clear representation of the taxonomical relationships within the domain.	
Related patterns	<u>Healthcare Role</u>	

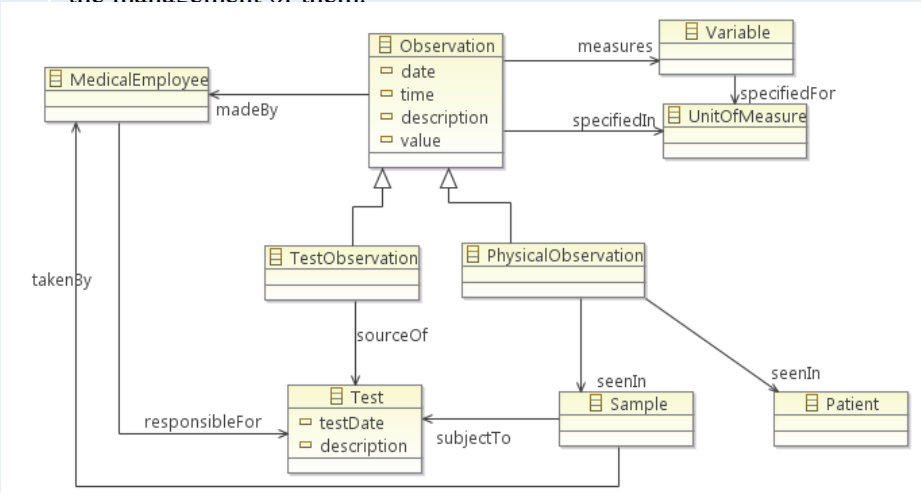
Candidate Pattern 19: Healthcare Party

Pattern Name	Healthcare Role	Level of design Intermediate
Problem	This pattern addresses the problem of identifying the roles of a party within the domain.	
Solution	Abstract in Role the interacting roles of parties within the domain of healthcare.	
Consequences	The model using Healthcare Role has a clear representation of the taxonomical relationships of roles played by parties within the domain.	
Related patterns		

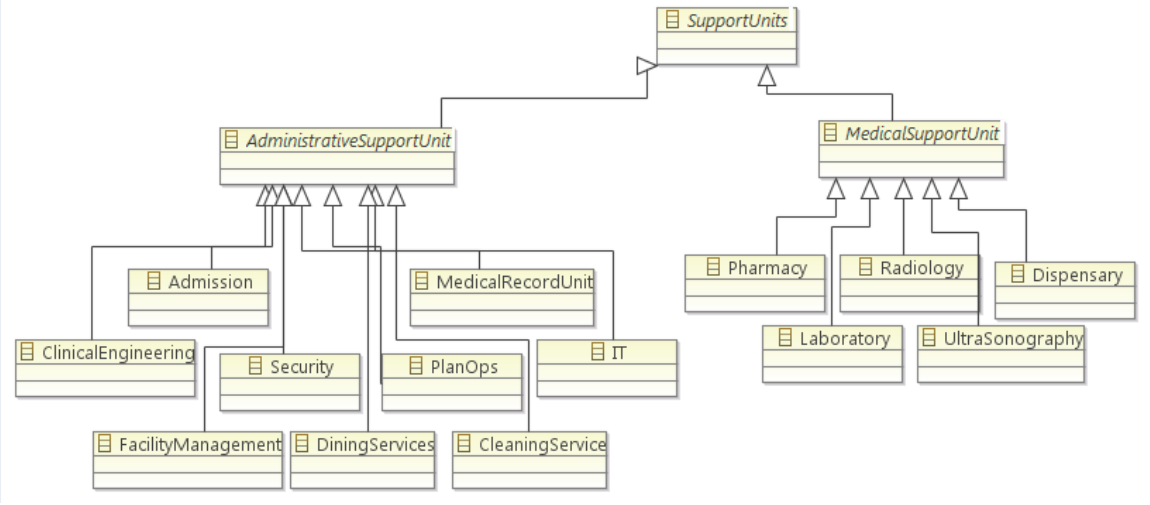
Candidate Pattern 20: Healthcare Role

Pattern Name	Observation States	Level of design Intermediate
Problem	This pattern addresses the problem of understanding the states that an observation can be in as well as the relationship between observations.	
Solution	Abstract the types of an observation and the relationship between observations.	
Consequences	The model using Observation States has a clear representation of the states observations are in.	
Related patterns	<u>Observation</u> , <u>Physical Observation</u>	

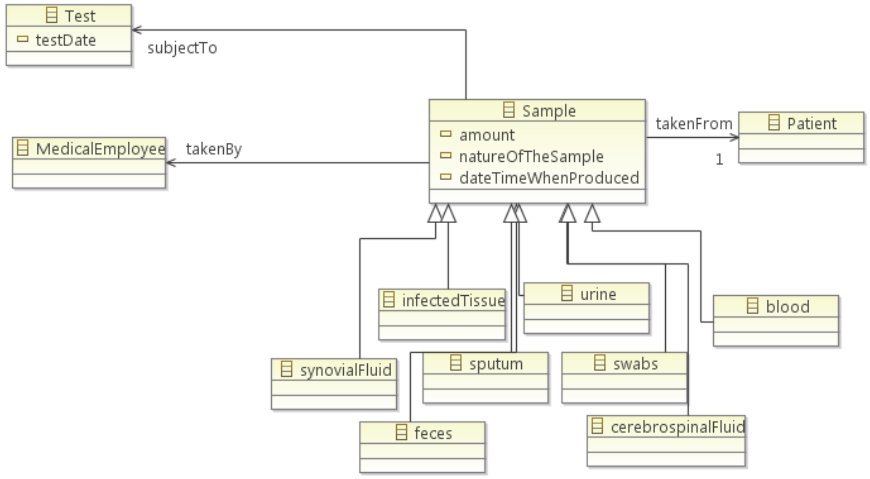
Candidate Pattern 21: Observation States

<i>Pattern Name</i>	Observation	Level of design Intermediate
Problem	This pattern addresses the problem of managing observations within the healthcare domain.	
Solution	<p>Abstract the types of observations and identify the personnel involved in the management of them.</p>  <pre> classDiagram class MedicalEmployee { } class Observation { date time description value } class Variable { } class UnitOfMeasure { } class TestObservation { } class PhysicalObservation { } class Test { testDate description } class Sample { } class Patient { } Observation < -- TestObservation Observation < -- PhysicalObservation Observation --> MedicalEmployee : madeBy Observation --> Variable : measures Observation --> UnitOfMeasure : specifiedIn TestObservation --> Test : sourceOf PhysicalObservation --> Sample : seenIn PhysicalObservation --> Patient : seenIn MedicalEmployee --> Test : responsibleFor MedicalEmployee --> Observation : takenBy Test --> Sample : subjectTo </pre> <p>The diagram illustrates the Observation pattern. It features a base class Observation with attributes <code>date</code>, <code>time</code>, <code>description</code>, and <code>value</code>. Two subclasses, TestObservation and PhysicalObservation, inherit from it. MedicalEmployee is associated with Observation via a <code>madeBy</code> relationship and with Test via a <code>responsibleFor</code> relationship. Observation also has a <code>measures</code> relationship with Variable and a <code>specifiedIn</code> relationship with UnitOfMeasure. TestObservation has a <code>sourceOf</code> relationship with Test. PhysicalObservation has <code>seenIn</code> relationships with both Sample and Patient. Test has a <code>subjectTo</code> relationship with Sample. MedicalEmployee is also associated with Observation via a <code>takenBy</code> relationship.</p>	
Consequences	The model using Observation has a clear representation of the types of observations and the interacting parties that are involved in the management of them.	
Related patterns	<u>Observation States</u>	

Candidate Pattern 22: Observation

<i>Pattern Name</i>	Supporting Unit Types (H. Organization)	Level of design Intermediate
Problem	This pattern addresses the problem of identifying the supporting unit types that are components of a healthcare institution within the healthcare domain.	
Solution	<p>Abstract all possible type of supporting units and the organization of them.</p>  <pre> classDiagram class SupportUnits class AdministrativeSupportUnit class MedicalSupportUnit class Admission class ClinicalEngineering class Security class FacilityManagement class DiningServices class CleaningService class MedicalRecordUnit class PlanOps class IT class Pharmacy class Radiology class Laboratory class UltraSonography class Dispensary SupportUnits < -- AdministrativeSupportUnit SupportUnits < -- MedicalSupportUnit AdministrativeSupportUnit < -- Admission AdministrativeSupportUnit < -- ClinicalEngineering AdministrativeSupportUnit < -- Security AdministrativeSupportUnit < -- FacilityManagement AdministrativeSupportUnit < -- DiningServices AdministrativeSupportUnit < -- CleaningService MedicalSupportUnit < -- MedicalRecordUnit MedicalSupportUnit < -- PlanOps MedicalSupportUnit < -- IT MedicalSupportUnit < -- Pharmacy MedicalSupportUnit < -- Radiology MedicalSupportUnit < -- Laboratory MedicalSupportUnit < -- UltraSonography MedicalSupportUnit < -- Dispensary </pre>	
Consequences	The model using Supporting Unit Types contains an abstraction of the taxonomy of supporting units in the healthcare domain.	
Related patterns	<u>Hospital Organization</u>	

Candidate Pattern 23: Supporting Unit Types

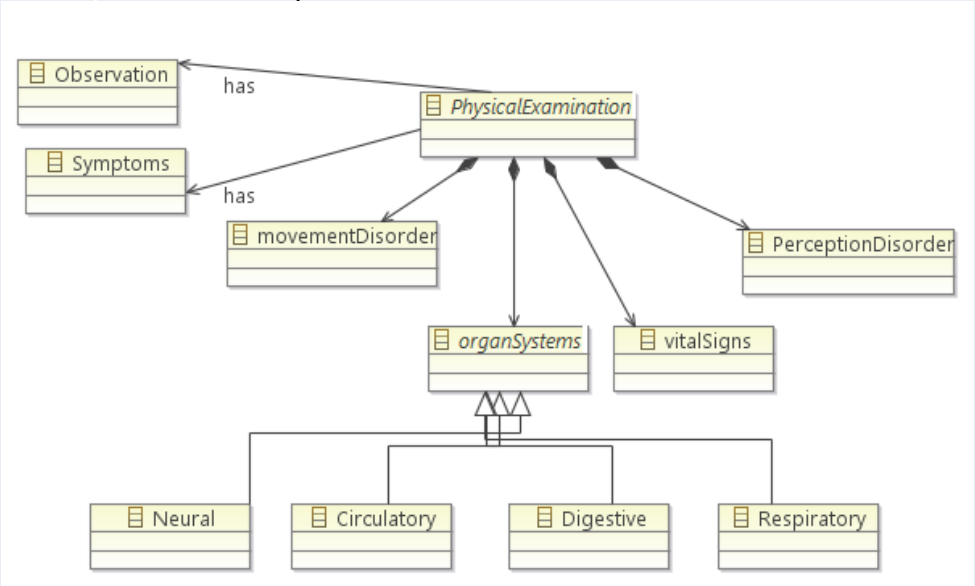
<i>Pattern Name</i>	Sample	Level of design Intermediate
Problem	This pattern addresses the problem of designing the management of samples within the healthcare domain.	
Solution	<p>Abstract the possible samples and the common attributes in a Sample class and identify the interacting elements in the domain.</p> 	
Consequences	<p>The model using Sample has a clear representation of the sample types and the elements related to it within the domain.</p> <p>A combination of Sample and Test would require the design of constraints over the “subjectTo” association in order to allow only some tests over certain sample.</p>	
Related patterns	<u>Test</u>	

Candidate Pattern 24: Sample

Pattern Name	Test	Level of design Intermediate
Problem	This pattern addresses the problem of designing the management of tests within the healthcare domain.	
Solution	Abstract the possible tests and the common attributes in a Test class and identify the interacting elements in the domain.	
<pre>classDiagram class Test { +testDate } class Sample { +amount +natureOfTheSample +dateTimeWhenProduced } class MedicalEmployee { } class Doctor class Nurse class LaboratoryPersonnel class Patient class biopsy class bloodFilms class fullBloodCounts class Cultures class dna class SensibilityTesting { +medicine } class specialisedTests Test < -- biopsy Test < -- bloodFilms Test < -- fullBloodCounts Test < -- Cultures Test < -- dna Test < -- SensibilityTesting Test < -- specialisedTests MedicalEmployee < -- Doctor MedicalEmployee < -- Nurse MedicalEmployee < -- LaboratoryPersonnel Sample --> Test : subjectTo Sample --> MedicalEmployee : takenBy Sample --> Patient : takenFrom 1 Doctor --> Test : requestedBy</pre> <p>The diagram illustrates the Test pattern in a healthcare domain. It features a central Test class with a testDate attribute. Several test types are shown as subclasses of Test: biopsy, bloodFilms, fullBloodCounts, Cultures, dna, SensibilityTesting (which includes a medicine attribute), and specialisedTests. A MedicalEmployee class is the superclass for Doctor, Nurse, and LaboratoryPersonnel. A Sample class has attributes amount, natureOfTheSample, and dateTimeWhenProduced. A Patient class is also present. The relationships are as follows: Test is the superclass for the various test types. Sample is associated with Test via a subjectTo relationship, with Test being the target. Sample is also associated with MedicalEmployee via a takenBy relationship, with MedicalEmployee being the target. Sample is associated with Patient via a takenFrom relationship, with Patient being the target and a multiplicity of 1 indicated near Patient. Finally, Doctor is associated with Test via a requestedBy relationship, with Test being the target.</p>		
Consequences	The model using test has a clear representation of the test types and the elements related to it within the domain. A combination of Sample and Test would require the design of constraints over the “subjectTo” association in order to allow only some tests over certain sample.	
Related patterns	<u>Sample</u>	

Candidate Pattern 25: Test

c) Patterns for Advanced Design

Pattern Name	Healthcare Physical Examination	Level of design Intermediate
Problem	This pattern addresses the problem of identifying the components of a physical examination.	
Solution	<p>Abstract physical examination and identify the relevant information extracted in its process.</p>  <pre> classDiagram class PhysicalExamination class Observation class Symptoms class movementDisorder class PerceptionDisorder class organSystems class vitalSigns class Neural class Circulatory class Digestive class Respiratory PhysicalExamination --> Observation : has PhysicalExamination --> Symptoms : has PhysicalExamination --> movementDisorder PhysicalExamination --> PerceptionDisorder PhysicalExamination --> organSystems PhysicalExamination --> vitalSigns organSystems < -- Neural organSystems < -- Circulatory organSystems < -- Digestive organSystems < -- Respiratory </pre> <p>The diagram illustrates the Healthcare Physical Examination pattern. It features a central PhysicalExamination class. This class has directed associations (labeled 'has') to Observation and Symptoms. It also has directed associations to movementDisorder, PerceptionDisorder, organSystems, and vitalSigns. The organSystems class is a generalization (indicated by a hollow triangle) for four specific classes: Neural, Circulatory, Digestive, and Respiratory.</p>	
Consequences	The model using Healthcare Physical Examination gives a clear of the components of a physical examination within in the healthcare domain.	
Related patterns	<u>Organ System Physical Examination</u> , <u>Movement Disorder Physical Examination</u> , <u>Perception Disorder Physical Examination</u> , <u>Vital Signs</u>	

Candidate Pattern 26: Healthcare Physical Examination

<i>Pattern Name</i>	Hospital Organization	Level of design Advanced
<i>Problem</i>	This pattern addresses the problem of identifying the organization of a hospital in departments and supporting units	
<i>Solution</i>	<p>Identify the components of the organization of a hospital and the responsible personnel.</p> <pre> classDiagram class Hospital { } class SupportUnits { } class Department { name description } class Employee { } class MedicalEmployee { } class Doctor { } Hospital "1" -- "0..*" SupportUnits Hospital "1" -- "1..*" Department SupportUnits "1" -- "1..*" Employee Department "1" -- "1..*" MedicalEmployee Doctor "1" -- "1" Department : chiefOf </pre>	
<i>Consequences</i>	The model using Hospital Organization gives a view of the components of a hospital's organization and the employees and responsables.	
<i>Related patterns</i>	<u>Healthcare Party</u> , <u>Department Type</u> , <u>Supporting Unit Type</u>	

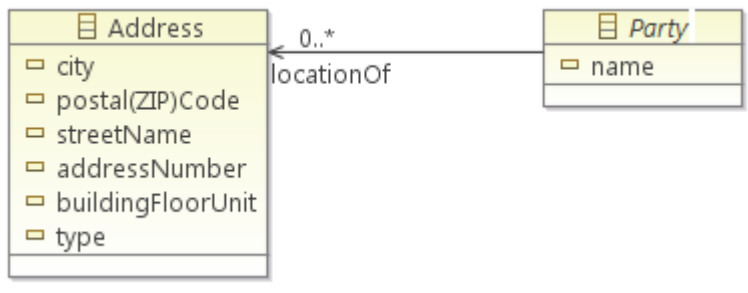
Candidate Pattern 27: Hospital Organization

Pattern Name	Medical Record	Level of design Advanced
Problem	This pattern addresses the problem of identifying the relevant information of a medical record document and the relationships between the pieces of information.	
Solution	<p>Identify the components of a medical record document.</p>	
Consequences	The model using Medical Record	
Related patterns	<u>Healthcare Party</u> , <u>Observation</u> , <u>Antecedent Types</u> , <u>Family History</u>	

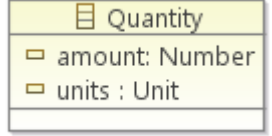
B. Cross-Domain Patterns

Cross-domain patterns are the ones that are universally applicable; they can be found and applied in multiple domains.

a) Patterns for Early Design

Pattern Name	Address	Level of design Early
Problem	This pattern addresses the problem of expressing one or multiple addresses for a party.	
Solution	Separate the attributes of address in a class related to the parties that need a location  <pre> classDiagram class Party { name } class Address { city postal(ZIP)Code streetName addressNumber buildingFloorUnit type } Party "0..*" --> "1" Address : locationOf </pre>	
Consequences	The model using Address fulfills the need of expressing that a party may not have, have one or multiple addresses.	
Related patterns	<u>Contact Information</u>	

Candidate Pattern 29: Address

Pattern Name	Quantity	Level of design Early
Problem	This pattern addresses the problem of storing measurements and their units of measure.	
Solution	Abstract the important information necessary to identify a quantity.  <pre> classDiagram class Quantity { amount: Number units : Unit } </pre>	
Consequences	The model using Quantity considers storing results not only as values because they are difficult to interpret and prone to conversion errors.	
Related patterns		

Candidate Pattern 30: Quantity

b) Patterns for Intermediate Design

Pattern Name	Contact Information	Level of design Intermediate
Problem	This pattern addresses the problem of having the need of being able to express one or multiple contact information for a party. i.e. workAddress, deliveryAddress, emergencyPhone, etc.	
Solution	<p>Identify the possible contact information types and abstract them to relate them to a party.</p> <pre> classDiagram class ContactInformation { type } class Party { name } class PhoneNumber { countryCode areaCode phoneNumber } class PostalAddress { City Postal(ZIP) Code streetName addressNumber buildingFloorUnit } class ElectronicAddress { electronicAddressString } ContactInformation < -- PhoneNumber ContactInformation < -- PostalAddress ContactInformation < -- ElectronicAddress Party "1..*" --> "1" ContactInformation : has </pre>	
Consequences	The model using Contact Information expresses in a flexible manner the means for contacting a party.	
Related patterns		

Candidate Pattern 31: Contact Information

Pattern Name	Party	Level of design Intermediate
Problem	This pattern addresses the problem of relating information (phone number, address) to many parties (people or organizations) similar to each other.	
Solution	<p>Abstract the similar information in an abstract entity that reflects the parties of the domain.</p> <pre> classDiagram class Party { name address phone } class Person { gender birthdate } class Organization { purpose webSite eMail } Party < -- Person Party < -- Organization </pre>	
Consequences	The model using Party fulfills the need of abstracting the commonalities of parties, people or organizations.	
Related patterns	<u>Employment</u>	

Candidate Pattern 32: Party

Pattern Name	Employment	Level of design Intermediate
Problem	This pattern addresses the problem of having to deal with contractual relationship of employment between organization and person.	
Solution	<p>Abstract the interacting concepts of employmen and their relationships.</p> <pre> classDiagram class Party { name } class Person { } class Organization { } class Employment { type } class Position { name description } class FilledPosition { startDate endDate } Party < -- Person Party < -- Organization Person --> Employment : employee Organization --> Employment : employer Employment --> Position : offers Position < -- FilledPosition FilledPosition --> Employment : filledBy </pre>	
Consequences	The model using Employment represents in a simple manner the contractual relationship of employment.	
Related patterns		

Candidate Pattern 33: Employment

Pattern Name	Physical Observation	Level of design Intermediate
Problem	This pattern addresses the problem of having the need to make observation over a physical object, i.e. physical exam to a patient or sample.	
Solution	<p>Abstract the interacting concepts of making observatios over a physical object.</p> <pre> classDiagram class Object { } class PhysicalObservation { details date } class PhenomenonType { } class Measurement { category } class Quantity { } Object --> PhysicalObservation : over PhysicalObservation --> PhenomenonType : measures PhysicalObservation --> Measurement : in Measurement --> Quantity </pre>	
Consequences	The model using Physical Observation represents in a simple manner the interacting concepts during an observation of a physical object.	
Related patterns		

Candidate Pattern 34: Physical Observation