



Norwegian University of
Science and Technology

Case-Based Reasoning in identifying causes of fish death in industrial fish farming

Marte Garaas
Geir Ole Hiåsen Stevning

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Agnar Aamodt, IDI
Co-supervisor: Axel Tidemann, Sintef

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

In cooperation with Sintef Fisheries and Aquaculture, within the SimFrame project under the Create program, we are studying the combined use of data-driven machine learning (ML) and case-based reasoning (CBR) for improved data analysis and decision support in fish farming (“fiskeoppdrett”). A prototype CBR system is under development, for decision support in the management of reduced fish death ratio.

In a prestudy project a set of relevant parameters were identified by studying a set of machine learning methods applied to an existing data set. This thesis work will start out from:

- an extended data set, including an additional set of relevant parameters, and possible combined indicators identified by the domain experts
- the current state of the CBR system under development at Sintef, developed in myCBR

And based on that:

- study the past machine learning methods applied to the extended data set.
- develop a CBR system framework based on the jColibri platform.
- implement a demonstrator system and test it on available data.

Assignment given: 13. January 2011
Supervisor: Agnar Aamodt, IDI

Abstract

Fish farming is a million dollar business world wide, and fish is in fact the third most important export product after oil/gas and metal in Norway¹. There are a lot of different aquaculture sites which produce fish along our long coast line and they all have some differences in the production rates and procedures. The fish farmer at these sites hold valuable information about the production, which is almost impossible to derive only from empirical data.

In this thesis we introduce Glaucus, a Case-Based Reasoning system which aims to help the fish farmers with their decision making when conduction sorting operations at their aquaculture sites. The system is built in Java and uses the jColibri development framework for Case-Based Reasoning. It retrieves cases based on similarity function from myCBR and jColibri in addition to custom made ones. The case base is generated from real world data and the case queries are populated by a combination of user input and data from a database with continuous data flow.

Our approach is just the beginning of what we hope will be a even greater journey towards a complete decision support system that will meet the expectations of the fish farmers.

Keywords: Case-Based Reasoning, Machine learning, Fish farming, jColibri, my-CBR

¹http://www.ssb.no/fiskeri_havbruk_en/

Preface

This thesis is written at the Department of Computer and Information Science at The Norwegian University of Science and Technology, in collaboration with SINTEF Fisheries and Aquaculture. The work started in autumn 2010 with a specialization project within the same domain.

We would like to thank our main supervisor Agnar Aamodt for his invaluable input throughout the project. It has been an honor to work with a man of such prestige and knowledge. We have drawn lots of inspiration from your lectures and discussions in our years at NTNU.

We would also like to thank our co-supervisor Axel Tidemann at SINTEF. You have always been there to answer our question, whether they are foolish or not. A big thanks goes to Gunnar Senneset for using his time to support us when we stumbled around with our problems with the data set just before the easter break.

Finn Olav Bjørnson at SINTEF also has our gratitude by demonstrating the prototype system with myCBR and his input throughout meetings. We would also give thanks to Arnfinn Aunsmo at SalMar, Brad Schofield, and Hans V. Bjelland, both at SINTEF, and Tore Bruland at NTNU for their valuable input during the project.

Last, but not least we would like to thank our family and friends for their support during our years of studying, it means a lot.

Contents

1	Introduction	1
1.1	Goals	2
1.1.1	Methodical goals	2
1.2	Motivation	3
1.3	Chapter Overview	3
I	Insight	5
2	Background	7
2.1	Aquaculture: Brief History and Statistics	7
2.2	Fish Farming	8
2.3	Research in the domain	9
2.3.1	Knowledge Acquisition	10
2.3.2	Image processing	11
2.3.3	Warning Systems	12
2.4	Machine Learning	12
2.5	Case-Based Reasoning	14
2.5.1	The CBR Cycle	15
2.5.2	Case representation	16
2.5.3	Similarity Mechanisms in CBR	17
2.5.4	Knowledge representation in CBR	18
2.5.5	Well-known CBR systems	19
	CYRUS	19
	PROTOS	19
	CREEK	19
	DrillEdge	19
3	Related Research	21
4	Methodological Approach	23
4.1	Equipment	24
4.1.1	Database	24
4.1.2	Development Frameworks	24
4.1.3	IDE's and Misc	25
4.2	jColibri	26
4.2.1	History	26

COLIBRI	26
jColibri	26
jColibri 2	27
4.2.2 jColibri 2 tutorial	28
Case Components and Similarity	29
4.3 Protegé + myCBR	30
4.3.1 Usage with jColibri	30
Example of use	31
4.4 Weka	33
4.5 Hibernate	34
4.5.1 Hibernate Mapping	34
4.5.2 Hibernate Configuration File	36
4.5.3 Storing data	37
4.5.4 Loading data through jColibri	37
4.6 Acquiring and analyzing the data	39
4.7 Machine Learning Methods	40
4.8 Knowledge Acquisition	42
4.9 Putting it all together	43
4.9.1 Initial Architecture for Glaucus	43
4.9.2 Case representation and solution	44
4.10 Evaluating the final system	46
II Utilization	47
5 Work process	49
6 Acquiring and analyzing the data	53
6.1 Table analysis	53
6.1.1 Empty tables	55
6.1.2 Tables containing empty columns and missing values	55
6.2 New tables	57
6.2.1 Operation	57
6.2.2 OperationType	59
6.2.3 Aquacultureprodunit_Has_Operation	59
6.2.4 Fishgroupreleation	59
6.2.5 Fishgroup_Has_StartFishGroup	60
6.2.6 Fishgroup_Has_Vaccine	60
6.2.7 Observationprocedure	60
6.2.8 Empty tables	61
6.3 Result	62
6.4 Database model	62
7 Attribute selection	67
7.1 Causeofdeath	67
7.2 Deadcount	67
7.3 Fishgroup	68

7.4	Fishgrouprelation	69
7.5	Company	69
7.6	Aquacultureprodunit	69
7.7	Feeding	70
7.8	Aquaculturesite	70
7.9	Aquacultureprodunit_has_operation	70
7.10	Operation	71
7.11	Operationtype	71
7.12	Fishgroup_has_vaccine	72
7.13	Vaccine	72
7.14	Fishgroup_has_startfishgroup	72
7.15	Startfishgroup	72
7.16	Referencepoint	73
7.17	Observation	73
7.18	Observationprocedure	74
7.19	Observationtype	74
7.20	Measurementnumerical	74
7.21	Resulttype	74
8	Machine Learning Experiments	77
8.1	Decision Tree: J48	79
8.2	Neural Network	80
8.3	Bayes Net	81
8.4	Naive Bayes	82
8.5	Comparison and Discussion	83
8.5.1	Comparing the methods	83
8.5.2	Comparing the data sets	83
9	Creating a case structure	87
9.1	Knowledge Acquisition	87
9.1.1	Inconsistencies in the data set	89
	Generating the Sorting table	92
9.1.2	Fish group relationship	92
9.2	Field Trip	94
9.3	The final case structure	95
10	Architecture and case base	99
10.1	Case representation and solution	102
10.2	CaseDescription	103
10.3	AquacultureSite	103
10.4	AquacultureProdUnit	103
10.5	HatcheryCompanyID	106
10.6	Vaccine	106
10.7	SpeciesOrigin	107
10.8	MeasurementType	108
10.9	MeasurementInstance	108

10.10	CaseSolution	110
10.11	FishDeath	112
10.12	Hibernate Mappings	113
10.13	Generating Cases	115
10.13.1	Case distribution	116
10.14	Query case generation	116
11	Similarity Assessment	119
11.1	Built-in	120
11.1.1	Equal	120
11.1.2	Interval	120
	Example	120
11.2	MyCBR	121
11.2.1	DateSimilarity	121
11.2.2	TrendSimilarity	122
11.3	Customized	123
11.3.1	ListInterval	123
11.3.2	ProdUnitAttributeSimilarity	123
11.3.3	ListSimilarity	123
12	The implemented system	125
12.1	Libraries	125
12.2	CBR Application	126
12.2.1	Configure	126
12.2.2	Precycle	126
12.2.3	Cycle	126
	Normal mode	126
	Evaluation mode	127
	Visualization mode	127
12.2.4	Postcycle	127
12.3	Deploying with pre-configured database	127
12.4	Application walk through	129
12.4.1	Creating the query(new case)	129
12.4.2	Retrieving similar cases	131
12.4.3	Reuse a case	131
12.4.4	Revise and Retain the case	136
III	Results	139
13	Evaluating the system	141
13.1	Leave One Out Evaluator	141
13.1.1	Output from evaluation	142
13.2	Hold Out Evaluator	142
13.2.1	Result, 5 % split and 4 repetitions	143
13.2.2	Result, 10 % split and 4 repetitions	143

14 Conclusion and Future Research	145
14.1 Evaluation of goals	146
14.1.1 Assess the use of CBR in the fish farming domain	146
14.1.2 Create a decision support system based on CBR	146
Frameworks	146
Case Structure	147
Other Methodical goals	147
14.2 Future Research	148

List of Figures

1.1	Scylla and Glaucus	2
2.1	Fry transport in Nan-hai Station, Kwangtung Province	7
2.2	Tristein, one of SalMar’s fish farming sites which we visited outside the coast of Sør-Trøndelag	9
2.3	Simple Knowledge Acquisition process	10
2.4	The Star Wars protocol droid, C-3PO	13
2.5	The humanoid robot created by Honda: ASIMO	14
2.6	CBR-cycle	16
2.7	Case complexity	17
2.8	Similarity Mechanisms in CBR	17
2.9	Richter’s Knowledge Containers	19
4.1	jColibri framework structure	27
4.2	jColibri 2 framework architecture	27
4.3	Persistence architecture	28
4.4	Configure Query Dialog	29
4.5	myCBR logo	30
4.6	Attribute in Protegé	31
4.7	testAttribute similarity	31
4.8	Preprocessor	33
4.9	Fish Death Class	34
4.10	Decision Tree	40
4.11	Neural Network	41
4.12	Simple Bayes Net	41
4.13	Naive Bayes	42
4.14	Component Diagram	43
4.15	Class Diagram	45
6.1	Output from the table operation	58
6.2	Operation Attributes conversion numeric to nominal	58
6.3	Output from the table OperationType	59
6.4	Fishgrouprelation table output	60
6.5	Database model part 1	64
6.6	Database model part 2	65
8.1	Threshold Curve J48 for TAPERSYNDROM of the new data	85

8.2	Threshold Curve J48 for TAPERSYNDROM of the specialization project data	85
9.1	operation table	89
9.2	The table aquaproductionunit_has_operation	90
9.3	fishgroup table	90
9.4	fishgrouprelation table	91
9.5	Fishgroup movement	91
9.6	Sorting and fishgrouprelation	92
9.7	The relationship between the different fishgroups	93
9.8	Field trip to Tristein	94
9.9	Fish farmer at work at Tristein	95
10.1	Java classes for generated similarity	100
10.2	Core Architecture	101
10.3	AquacultureSite Class	103
10.4	Data Model	104
10.5	AquacultureProdUnit Class	105
10.6	HatcheryCompanyID Class	106
10.7	Vaccine Class	107
10.8	SpeciesOrigin Class	108
10.9	MeasurementType	109
10.10	MeasurementInstance	109
10.11	TemperatureInstance	110
10.12	CaseSolution	111
10.13	FishDeath	112
10.14	Database Schema	114
10.15	Activity Diagram Case generation	115
10.16	Query case generation	117
11.1	Date Similarity	121
11.2	Trend Similarity	123
12.1	Startup	129
12.2	Query Dialog	129
12.3	Whole query	130
12.4	Visualization of the Trends	130
12.5	Similarity dialog	131
12.6	Retrieved case 1	132
12.7	Visualization of deaths	132
12.8	All operations	133
12.9	Retrieved case 2	134
12.10	Visualization of deaths, case 2	134
12.11	Retrieved case 5	135
12.12	Revise dialog	136
12.13	Retain dialog	137

13.1	Leave One Out Evaluation Chart	142
13.2	Hold Out Evaluation chart, 5% 4 rep	143
13.3	Hold Out Evaluation chart, 10% 4 rep	144
14.1	Temperature prediction	151

List of Tables

2.1	Aquaculture production(tons) by top 10 countries: 2006 - 2008. The character “F”(FAO estimate) indicates that there was no actual registered production number, but that the value is calculated given different variables and assumptions.	8
2.2	Feed Ratio comparison(pounds)	9
5.1	WorkProcess	49
5.2	WorkProcess2	50
5.3	WorkProcess3	51
6.1	Old Tables	54
6.2	All Tables	63
7.1	causeofdeath	68
7.2	deadcount	68
7.3	fishgroup	68
7.4	fishgrouprelation	69
7.5	company	69
7.6	aquacultureprodunit	69
7.7	feeding	70
7.8	aquaculturesite	70
7.9	aquacultureprodunit_has_operation	71
7.10	operation	71
7.11	operationtype	71
7.12	fishgroup_has_vaccine	72
7.13	vaccine	72
7.14	fishgroup_has_startfishgroup	72
7.15	startfishgroup	73
7.16	referencepoint	73
7.17	observation	73
7.18	observationprocedure	74
7.19	observationtype	74
7.20	measurementnumerical	74
7.21	resulttype	75
8.1	Attribute Comparison	77
8.2	Data Set Information	78
8.3	New Data	83

8.4	specialization project Data	83
8.5	Data Set Comparison	84
10.1	Case Solution Distribution	116
13.1	Leave One Out Evaluator Result	142
13.2	Output HoldOut 5% split first repetition	143
13.3	Hold Out Result 5%	143
13.4	Output HoldOut 10% split first repetition	144
13.5	Hold Out Result 10%	144

Listings

4.1	XML file for testAttribute	32
4.2	testAttribute similarity Java code	32
4.3	Hibernate mapping file for FishDeath	35
4.4	Hibernate mapping file for CaseSolution	35
4.5	Hibernate configuration file	36
4.6	Java store CaseSolution and FishDeath	37
4.7	DataBaseConnector configuration file	37
4.8	Load case from database	38
8.1	SQL query for data selection	78
8.2	Decision Tree: J48 Run configuration	79
8.3	Decision Tree: J48 Summary	79
8.4	Neural Network Run configuration	80
8.5	Neural Network Summary	80
8.6	Bayes Net Run configuration	81
8.7	Bayes Net Summary	81
8.8	Naive Bayes Run configuration	82
8.9	Naive Bayes Summary	82
9.1	Inserting Sorting data	92
10.1	Hibernate mapping file for MeasurementInstance	113
11.1	XML file for Date similarity	121
11.2	Date similarity Java conversion	121
11.3	Date similarity Java test	122
11.4	List Similarity calculation	124
12.1	Start MySQL server Java code	128
13.1	LeaveOneOutEvaluator code	141
13.2	LeaveOneOutEvaluator code inside cycle	141
13.3	HoldOutEvaluator code	142
14.1	jColibri issue	147

Chapter 1

Introduction

In this thesis we assess the use of Case-Based Reasoning within the domain of fish farming. Fish farming is a form of aquaculture where the purpose is to breed/produce fish which are to be utilized by consumers. Fish farming is current, and is covered by news on a daily basis, both negative and positive[Sved, 2011, Ringseth, 2011, Holstad, 2011a,b]. Our focus has been on specific human interactions with the fish, where we have looked at the sorting of the fish. Sorting is executed when the variation in size of the fish within one aquaculture production unit is too big. Exactly what “too big” is in this situation is mainly up to the fish farmers. When a sorting operation is done, we want to capture the information at hand within a case structure, and also encapsulate the knowledge and intuition of the fish farmer with it. The goal is to combine sensor data with the knowledge of the fish farmer. By having all the sites within a company collaborating and sharing experience with each other you can, in theory, have equally good production rate and avoid mistakes.

Case-Based Reasoning uses past experiences to solve new problems. Situations have a way of reoccurring and when they do, why not take advantage of the experience from last time. This is the foundation of Case-Based Reasoning and also what we want to utilize in this thesis. The decisions of a fish farmer are based on both knowledge and intuition. The knowledge is a combination of sensor data and the understanding of the historical data leading up to the relevant day. The intuition is mainly experience with similar situations and some good old gut feeling. It is this last part which is difficult to capture in a regular computer program.

As mentioned earlier we introduce the system Glaucus, a Case-Based Reasoning system which aims to help the fish farmers with their decision making when conducting sorting operations at their aquaculture sites. Glaucus, shown in Figure 1.1, is a Greek god who began his life as a mortal. He discovered, by accident, a magical herb which could bring the fish he had caught back to life. One day he decided to eat the magical herb, which then turned him into a fish-like figure. He started to grow a tail, instead of legs, and fins instead of arms. He also became immortal and was forced to dwell at sea forever. It is said that he frequently rescues sailors and fishermen in storms, being that he was once one of them. He is therefore known as the fisherman’s sea god.

The project is done in collaboration with an internal group at SINTEF Fishery and Aquaculture. We have our own system, and made our own choices, but we have gotten the opportunity to sit-in on several meetings, both internal and also a few where they



Figure 1.1: Scylla and Glaucus

met their clients. They are in the start phase of creating a similar system as our Glaucus, and our assignment was given, among other factors, to test out the technology they are planning to use, e.g. the Java framework jColibri. It has been great to get first-hand experience with how a scientific project is done in the real world.

In the following Sections we introduce the goals for the project and we also look at some of the motivations for making a decision support system in the fish farming domain.

1.1 Goals

Our goal in this master thesis is to study the potential for, and develop a Case-Based Reasoning demonstrator system for decision support in the fish farming domain. The test application is to help the fish farmers to know under which conditions to carry out a sorting operations in order to lower the amount of dead fish.

- Assess the use of Case-Based Reasoning in the fish farming domain
- Build a decision support system based on Case-Based Reasoning, with emphasis on fish deaths caused by human interactions that may involve massive losses
 - The operation considered is sorting

1.1.1 Methodical goals

The methodical goals are supplements to the main goal and describe how we are going to achieve our goal.

- Investigate the use of the jColibri development framework to create a Case-Based Reasoning application
- The cases should consist of both single parameters and trends over time
- Use myCBR to create similarity functions
- Case matching will be done based on a combination of single parameters and trends
- The output of the system will be a risk assessment of the situation/case
- Take into account that further data material can be added over time
- Create a graphical user interface

1.2 Motivation

We have completed a specialization project[Garaas and Hiåsen Stevning, 2010] in the previous term where we worked within the same domain, fish farming. Our goal there was to use different machine learning methods(among them Neural Network and Decision Tree) to classify causes of deaths in fish farming, given some chosen attributes. This was a global approach, where we found a generalization for the data set as a whole, on the basis of the provided data from sensors and observations logged by the fish farmer, and later mapped into a database. No considerations were taken to the opinion of the fish farmers/experts and specific happenings/cases. This is what we want to include in this project. Case-Based Reasoning captures specific events like the human mind does, and stores it for later use. But unlike the mind, the case base can store a large amount of cases and it should be pretty easy to discover it again at any time given strong indexing. This can be seen as a local approach if compared to the previous study, as we now are going to work with specific cases/happenings, and not a global generalization.

The main motivation, from the application side, is to help the fish farmers make good choices when it comes to when and how they interact with the fish, and in this way reduce the loss of fish to a minimum. There is also a motivation in the technological aspect of the project, by getting to develop a Case-Based Reasoning system in a real-world setting. The health and well being of the fish is also of course in mind.

As mentioned earlier our thesis is written in collaboration with SINTEF Fishery and Aquaculture, more specific with the SimFrame project in the CREATE program[SINTEF, 2009]. The SimFrame project's main objective is to development a framework for simulation, optimization and monitoring of all aspects of modern fish farming¹.

1.3 Chapter Overview

In the next chapter we look at some background information, both related to fish farming and to the different approaches we aim to use during the project. In Chapter 3 we

¹<http://www.sintef.no/Projectweb/CREATE/About-CREATE/Objective/>

assess some of the related research within fish farming combined with Case-Based Reasoning. Chapter 4 describes the approaches and methods used during the project, and also proposes the initial architecture for Glaucus.

In Chapter 5 we list the different milestones in the project and also in which chapters each milestone is documented in detail. Chapter 6 describes the analysis part of this project, where we look at the supplied data set and look for inconsistencies. Next is Chapter 7, where we document the work we did before the discussion over the case structure started. Chapter 8 is related to the work we did in our specialization project in autumn 2010, where we used machine learning algorithms to find patterns in a data set. Chapter 9 describes the process of knowledge acquisition and case structure generation in our project.

In Chapter 10 we describe the final architecture of the system through a couple of diagrams and describe where each of the individual parts of the architecture are documented. In addition we look at the case base with regards to the case representation and how cases and queries are generated. Next up is Chapter 11, where we introduce the similarity functions we have used in Glaucus.

Chapter 12 describes Glaucus, including some test runs and basic descriptions of the different Case-Based Reasoning components used from the jColibri framework. In Chapter 13 we use some of jColibri's built-in methods to evaluate the system, before we in Chapter 14 discuss the results and propose some future work.

Part I
Insight

Chapter 2

Background

This chapter is divided into four main parts; Aquaculture: Brief History and Statistics, Fish Farming, Machine Learning and Case-Based Reasoning. The two latter parts are related to the the Methodologies we use in our solution for the problem description. The former are related to the domain itself, in addition to some research in the domain. Chapter 3 will cover the research most related to our field of study.

Aquaculture or aqua farming, is the broad term for farming organisms in freshwater and seawater. It involves controlling the environment with regard to aspects such as temperature, pollution, feeding and medication in order to farm a specific aqua-based organism. The output from the farming process is often food to be consumed by humans or by other farmed animals. A big advantage with the farming approach is that we can cultivate much more of a specific organism than what is available in the wild.

2.1 Aquaculture: Brief History and Statistics

Aquaculture is practiced world-wide and has its roots back in China as long ago as 2000 BC[Rabanal, 1988]. The practice may be even older, but due to the lack of written texts there are no factual evidence of this, just a hypothesis that aquaculture was handed down from one generation to another, usually from those who were found in seat of power. China is to date the largest producer of aqua-based organisms, while other Asian countries, South America and Europe follow. Figure 2.1 shows the transportation of fry (small fish) on the southern coast of China¹.



Figure 2.1: Fry transport in Nan-hai Station, Kwangtung Province

¹<http://www.fao.org/docrep/005/AC862E/AC862E07.htm>

China had in 2008 a production of approximately 42.7 million ton, mainly carps and oysters. Table 2.1 illustrates the aquaculture production in the years 2006 to 2008 for the ten countries in the world with the highest production. The numbers are taken from the FAO(Food and Agriculture Organization of the United Nations) data set[FAO, 2008a], with consultation with the SOFIA reports produced by FAO[FAO, 2008b, 2010].

Country	2006	2007	2008
China	39 359 174 F	41 172 951	42 669 744
Indonesia	2 479 247 F	3 121 379	3 854 844
India	3 180 865	3 112 242	3 478 692
Vietnam	1 693 727 F	2 123 400 F	2 497 400 F
Philippines	2 092 274	2 214 785	2 407 698
Thailand	1 407 001	1 351 075	1 374 024
Japan	1 223 953	1 286 027	1 187 774 F
Bangladesh	892 049	945 812	1 005 542
Chile	832 329 F	806 166 F	870 845 F
Norway	712 373	841 560	843 730

Table 2.1: Aquaculture production(tons) by top 10 countries: 2006 - 2008. The character “F”(FAO estimate) indicates that there was no actual registered production number, but that the value is calculated given different variables and assumptions.

Note that there is a clear dominance of Asian countries on the list and it is estimated that Asia accounts for about 98% of carp, 95% oyster and 88% shrimps and prawns world-wide[FAO, 2008b]. Aquatic plant production in aquaculture accounts for about 93% of the worlds total supply, and of this approximately 72% is from China. By looking at salmon in the table we see that there are two main actors, Norway and Chile, with 33% and 31% of production respectably. Export of production overall is dominated by China, Norway and Thailand.

2.2 Fish Farming

Fish farming is a specific type of aquaculture in the same way as shrimp farming, oyster farming and seaweed farming. Fish farming is the process of breeding, feeding and taking care of fish, and later selling them to consumers. A popular type of farmed fish is the salmon. Wild salmon is born in freshwater, moves to seawater and goes back to freshwater to reproduce. The fish is considered to be very healthy given the large amount of vitamin D, omega-3 fatty acid and proteins. Studies have, however, shown that the omega-3 fatty acid level is a bit lower in farmed fish than in wild fish[Frøysa, 2011], and that farmed fish contain high levels of dioxins[Hites et al., 2004]. These are among the issues and questions raised in the aquaculture domain including others like water quality management, early warning systems and fish disease diagnosis. A typical Norwegian fish farming site is shown in Figure 2.2.

On an aquaculture site there are several production units. In each production unit there are fish in various sizes depending on the time they have been in “production”. On each site, or on the production units, there are many different types of operations which



Figure 2.2: Tristein, one of SalMar’s fish farming sites which we visited outside the coast of Sør-Trøndelag

are conducted in a production cycle such as De-lousing, Slaughtering, Sorting and Deployment. In our thesis we focus on cases where the Sorting operation is used. A Sorting operation is, as stated previously, a way of managing the size difference on the site. When the difference in fish size in each production unit reaches a certain threshold(estimated by the fish farmers), they sort the fish from the affected production units into groups of similar fish sizes in separate units. Related to this we see an increase of fish death before and after sorting, and our aim is to create a system which recognizes when there is a risk for increased fish death.

A big advantage with salmon farming is the low feed ratio. The feed ratio tells us how much feed we need in order to get one amount of fish. According to FAO the amount of feed needed to produce one pound of farmed salmon is one and a half pound. If we compare the ratio to other related industries, we get the figures shown in Table 2.2, where we range from least effective to most effective.

Wild Salmon	10:1 to 15:1
Beef	10:1
Pork	5:1
Chicken	2:1
Farmed Salmon	1.5:1

Table 2.2: Feed Ratio comparison(pounds)

2.3 Research in the domain

Fish disease diagnosis and treatment is a field which has received a lot of attention over the past decade[Zeldis and Prescott, 2000]. Fish disease diagnosis is a quite complicated process which require a lot of expertise, time and resources. There are several studies which describe expert systems and decision support systems for fish disease diagnosis, and a lot of work the last years has and is being done by the one of the biggest producers in aquaculture, China.

The methods used to acquire knowledge, learn, query and present the results vary from mathematical theorems and Rule-Based Reasoning to Case-Based Reasoning, and presentation of the results with technologies such as web applications[Zhang et al., 2004, Li et al., 2006b] and Short Message Service(SMS)[Guirong, 2009].

The different studies use methods of determining diseases from data collected in the environment such as water temperature and oxygen level, and data collected through examining the fish with regards to symptoms and other features. The output of each expert system is also varying in that some choose to give a diagnosis with related treatment plans, while others adopt a more simple solution by presenting the k-nearest cases and let the user decide which is the most likely diagnosis or treatment. This can be related to the field of Case-Based Reasoning where we have knowledge-light and knowledge-intensive applications, as described in Section 2.5.

2.3.1 Knowledge Acquisition

A common issue in creating an expert system, such as in the combined field of aquaculture and computer science, is the bottleneck that is *knowledge acquisition*[Forsythe and Buchanan, 1989]. Knowledge acquisition is the process of gathering information from the field and using it in some way. Typically it involves retrieving information from one to several experts and/or documentation, representing it in some way and translating it into a way that machines can understand.

We have personal experience with this while working on our specialization project in the autumn of 2010[Garaas and Hiåsen Stevning, 2010]. In the study we used Fayyad et. al's KDD(Knowledge Discovery in Databases) process[Fayyad et al., 1996] to extract relevant data from a given database and apply common machine learning methods with the help of the data mining software, Weka 3.6[Hall et al., 2009]. The process was in-depth and time-consuming which is what one might expect from learning something from a new field. Figure 2.3 illustrates a simple knowledge acquisition process² where a knowledge engineer queries the domain expert and formalizes the information into structured knowledge.

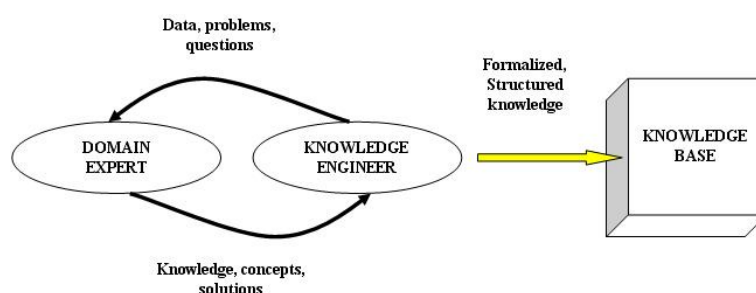


Figure 2.3: Simple Knowledge Acquisition process

However, the bottleneck of knowledge acquisition is not only related to databases, but also how to use the experts in the field to your advantage. Experts, such as veterinarians, are not people you can find plenty of. In a specific region, ranging from several hundred kilometers to country-lengths, there may be only a few people who have the qualified

²<http://www.generation5.org/content/2005/PDAMum.asp>

expertise in a certain area. The demand for a specific expert may then be quite high based on how many people or organizations require his/her expertise. To acquire the knowledge that these individuals possess is a high priority and an essential part in order to “spread” the information in a region, although it is quite time-consuming[Zhu and Li, 2008, Fu et al., 2009, Zhang et al., 2004, Li et al., 2009, 2006a].

A common way to retrieve knowledge is to hold interviews with the experts, discussing the different aspects of the field or presenting cases to see how and what the expert emphasizes in the problem solving. In relation to presenting cases, it is also possible to observe the experts working in their natural habitat, to see if there are things they do that may be important, but they are not aware of them self.

2.3.2 Image processing

Image processing has become a quite popular technique to solve many of the problem we have today. There are a range of different uses for image processing, including Surface Grading Systems(grade the quality of wood panels for the building industry), passport control and general recognition. Fish diagnosis and disease detection is one of the latest additions to a long list applicable uses for this technology.

The common way to create an application with this technology is to first preprocess the images captured from the various aquaculture environments. The images may be noisy, which will reduce the chance of getting a positive classification of the disease a fish may be carrying. There are many ways to optimize the quality and usage of the image, including noise removal, edge sharpening, leveling, edge detection and binarization[Jeong-Seon et al., 2007].

When the preprocessing is done it is common to extract some features which can be used to classify the image. The different ways to classify the problem ranges from the use of techniques like principal component analysis[Jeong-Seon et al., 2007] and Case-Based Reasoning[Lou et al., 2007].

A well-known system in the fish farming domain is Fish-Expert. Fish-Expert[Li et al., 2002] is a web-based system which diagnoses fish by using rules and images. The images are gathered from the environment and depicts different types of diseases and symptoms. The system can diagnose 126 types of diseases among primarily freshwater fish, in total nine. The reasoning technique used is rule-based, with 300 rules, and it has approximately 400 images in its database at its disposal. The system has obtained very good results and the farmers are overall very happy with the system. When a farmer experiences issues with his fish, he takes a picture of the fish and supplies the system with pond(site) information in the form of observed data. The reasoning system uses the provided image and information to find the the disease through forward chaining inference. If the disease is not found additional data must be provided, such as a microscopic examination and water quality information. A combination of Fish-Expert and Glaucus would be an interesting concept, where we could use Fish-Expert as an autonomous agent who finds diseases in the fish after a Sorting operation. This could be regarded as future work, and a more in-depth study would be necessary to assess if this is feasible.

2.3.3 Warning Systems

Warning systems are popular in China as shown by the use of finding diseases in the most up and coming fish type, flounder. A warning system has been implemented with a architecture divided into three different parts; warning module, knowledge module and maintenance module. The system monitors trends, water quality and flounder symptoms and warns the responsible people when something is wrong[Xing et al., 2009].

There has also been conducted research in monitoring the environment in different studies, such as with CORMS AI[Vafaie and Cecere, 2005]. CORMS AI is a decision support system for monitoring the US maritime environment with the combination of rule-based and case-based reasoning. The system has been in operation since 2003 and identifies suspect data and network disruption accurately and reliably. The system assist the NOAA's Center for Operational Oceanographic Products and Services personnel in monitoring maritime environment.

Related to the environment is the concern for the water quality in China. A particular work has been done in developing a knowledge-based early warning system for fish disease/health via water quality management, EWS-FDWQ[Li et al., 2006b, 2009]. Investigations in northern China show that most outbreaks of diseases are caused by water pollution. The loss of revenue due to diseases and water quality was at USD 1.4 and 1.9 billion in 2003 and 2004. EWS-FDWQ aims to monitor the water quality and provide a tool to warn of potential fish diseases and enhance the management of pond ecosystem.

2.4 Machine Learning

Machine Learning is a scientific discipline within the domain of Artificial Intelligence[Russell and Norvig, 2002]. The discipline is based on using empirical data, such as from sensors, hand written data and databases, to let computers learn and evolve their behavior. A learner(machine learning method) first tries to learn a concept or capture some patterns from a given data set, and then make an intelligent decisions based on this[Mitchell, 1997b] .When a concept has been learned the method is fed some data and then is asked to make a decision on what to do or classify the data instance. This type of learning is called eager, which means that the learner tries to find a global solution for the whole data set or to generalize before more problems are added. This is in contrast to Case-Based Reasoning which is a lazy learner. In this Section we will look at the so called eager learners, while Section 2.5 will cover a lazy learner, Case-Based Reasoning.

How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?

Tom M. Mitchell

The above quote is taken from an article called The Discipline of Machine Learning[Mitchell, 2006], and it illustrates the question in what Machine Learning tries to answer. Machine Learning has been influences by a broad range of field, including Computer Science, Statistics and Psychology. Computer Science and Statistics have had the most influence on Machine Learning to date with how we build machines that solve

problems and what can be inferred from the data. Psychology and other related fields however, have had some degree of influence, but the human knowledge of the brain in both animals and humans is too weak to make a great impact. There is however high hopes that this will change in the near future. Case-Based Reasoning has taken another turn, where psychology has had a great impact. More about this can be found in the Section 2.5

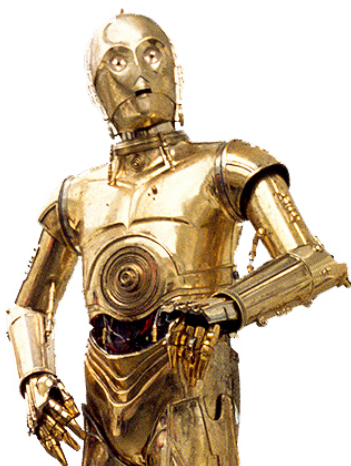


Figure 2.4: The Star Wars protocol droid, C-3PO

Over the years we have seen lots of real-world applications in different fields, such as *Computer Vision*, *Surveillance- and Warning systems*, *Speech Recognition*, *Robots*, *Automotive Vehicles* and *AI in Games*. Machine Learning has evolved from being small laboratory demonstrations to full scale commercial systems, and it in fact works[Mitchell, 1997a].

In Speech Recognition we have systems which will make it easier for people who want to perhaps limit their use of the keyboard and mouse, or because they have a disability which makes it hard to use applications. Microsoft, with arguably the most wide known solution, usual ships their operating systems with Windows Speech Recognition³ which makes it possible for a user to dictate emails, manage applications or browse the Internet.

Surveillance- and Warning systems are systems that in some way monitor their environment and does some action based on its observations. In Chapter 3, we will look at systems using Machine Learning and/or Case-Based Reasoning to monitor or warn of danger in the fish farming domain.

Robots and Automotive Vehicles are tightly coupled fields, as seen in popular science fiction movies, such as Star Wars(see C-3PO in Figure 2.4) and Blade Runner. There is much promise and potential in this field, and we have various systems ranging from the polite robot ASIMO(Figure 2.5)[Sakagami et al.], cars running autonomous in high speed desert-driving[Thrun et al., 2007], to intelligent vacuum cleaners.

In order to create systems and application which have or simulate intelligent behavior, we need to use some practical approaches. There are a range of different approaches such as:

- Decision Tree Learning

³<http://windows.microsoft.com/en-US/Windows7/What-can-I-do-with-Speech-Recognition>



Figure 2.5: The humanoid robot created by Honda: ASIMO

- Genetic Programming
- Artificial Neural Networks
- Clustering
- Bayesian Networks
- Support Vector Machines

We will not describe these approaches in detail here, but in Section 4.7 and Chapter 8 we look at four different approaches to learn from a data set in the fish farming domain. This is the same four approaches we worked with in our specialization project.

2.5 Case-Based Reasoning

Case-Based Reasoning, or CBR, is a branch on the big machine learning tree that has been around for approximately thirty years. It originated from separate work done at the University of Texas and Yale University, among others, in the U.S, with Bruce Porter and Roger Schank as two important initiative-takers and researchers. Schank is maybe most known for his dynamic memory model [Schank, 1983] and Porter for his Protos [Bareiss, 1989] system. Both have served as great inspiration to researchers across the globe.

CBR uses past experiences to solve new cases, much like our own brain. Learning in Artificial Intelligence has usually referred to the learning of generalizations. CBR is a lazy learning algorithm, as opposed to many other machine learning algorithms which are eager (e.g. Decision Trees, Bayes Net). A lazy learner awaits the generalization until after it receives a query.

David B. Leak points out five areas that CBR can contribute to the rest of the AI-community, e.g. areas that have proven themselves to be challenging for other AI-methods [Leak, 1996]. These are listed below:

1. Knowledge Acquisition
 - In theory, it is not necessary to do extensive knowledge acquisition while creating a CBR system, as we rely on specific cases instead of a all-including domain model.
2. Knowledge Maintenance

- The CBR system is always updated, and learns from experience.
 - It starts out with a collection of “start-cases” which will be maintained as the system encounters new cases.
3. Increasing problem-solving efficiency
 - Reusing of old solutions increases the problem-solving efficiency and there is no need to create solutions to similar problems from scratch each time when you can use an old, maybe adjusted, solution.
 4. Increasing quality of solutions
 - With bad or non-existing domain models
 5. User acceptance
 - A user will more likely approve of a solution that once worked on a similar problem given in a case format, than derived rule-chains. Cunningham et al. show that Case-Based Explanation is considered more convincing than the rule-based alternative through a series of tests on faculty staff and students at Trinity College Dublin[Cunningham et al., 2003].

The basic idea behind CBR is that problems have a way of reoccurring in a similar way. The solution we applied last time, may therefore also apply for this new but quite similar problem. Take for instance a doctor examining a patient with a set of given symptoms. While listing the symptoms the doctor is reminded of a previous case, a patient with the same symptoms who came in a couple of weeks ago. The patient was treated with antibiotics and told to hold the bed, and he recovered in no time. Might the same treatment work in this instance?

A case can have many forms, ranging from simple data, information or knowledge. In regards to this, the data may be non-readable sensor data, while information is data with a meaning, and knowledge is some learned information. It is only when a case is interpreted as knowledge that we can say that we have a real Case-Based Reasoning system. Systems that interpret cases just as data uses case-based methods to retrieve and index the cases[Aamodt, 2004].

There are two main styles of CBR, problem solving CBR and interpretive CBR[Kolodner, 1992]. Problem solving CBR is when you use old cases to create a solution to a new case. It is in other words heavily dependent of adaption mechanisms. Interpretive CBR is when you interpret a new case by using the context of old situations. They are both very much alike when it comes to the dependency of retrieval mechanisms and retaining of cases in order to learn from experience. There is also another way a Case-Based Reasoner learns, and this is that it can become more efficient as it remembers old solutions and adapts these, instead of creating/deriving them from scratch each time.

2.5.1 The CBR Cycle

Aamodt and Plaza proposed to divide the process of CBR into subprocesses, the four RE’s[Aamodt and Plaza, 1994]. The processes are:

- **Retrieve** similar case(s)
- **Reuse** the information stored in these cases to create a solution the new problem
- **Revise** this solution by trying it out in the real-world or asking a domain expert
- **Retain** the revised solution for later use

The four steps are illustrated in Figure 2.6 where we start the cycle with a new case(problem).

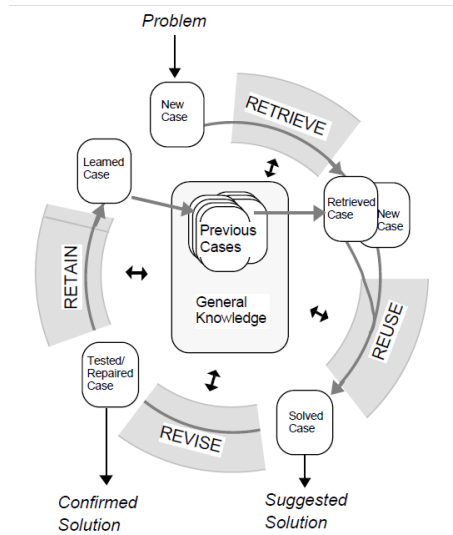


Figure 2.6: CBR-cycle

2.5.2 Case representation

A case is usually built from the most important features. The process of finding these features are a difficult process, and also very biased where subjectiveness can be an issue. One person or expert will emphasis one aspect of a situation, while another person may think something else is important. The goal is to find the golden mean. A typical case is composed of a description, also known as the problem, and a solution to the given case. In more advanced cases one might also incorporate justifications and results of applying the solution.

Case-Based Reasoning is a common denominator for a widely set of different systems that uses past experiences to cope with new problems, like described in general above. These systems range from knowledge-light approaches on one side, to more knowledge-intensive on the other. In a knowledge-light CBR approach the knowledge of the system is only based on the content of the cases. In knowledge-intensive CBR general domain knowledge is added to the system as well. This supplemented knowledge can be used among other factors to enrich the cases, describe relationship between entities or find more advanced similarities between attributes [Aamodt, 2004]. A knowledge-light approach is usually used when the system should operate in a domain which is unknown or difficult to understand. It can also be used when you want to partially bypass the knowledge

acquisition bottleneck mentioned in Section 2.3.1, where for instance the adaption logic will be found inside the already acquired cases instead of in the general domain knowledge.

Figure 2.7[Aamodt, 2004] demonstrates the two extreme points of CBR, with the first simple KNN systems at the left and the CREEK [Aamodt, 2004] system at the right, where the cases contains complex attributes and general knowledge to both adaption and to enrich the cases.



Figure 2.7: Case complexity

The case base should include both positive and negative cases. A positive case is one that successfully solved a goal or subgoal of the Case-Based Reasoner. The case base should have enough cases to cover all of its goals and subgoals. A negative case is one that failed, which can be as or even more, useful than a successful one. This means that the Case-Based Reasoner will avoid mistakes, or at least only make the mistake once.

2.5.3 Similarity Mechanisms in CBR

When we look at two different cases, such as a query case and a case from the case base, we have to formalize a way of representing how similar they are each other. The similarity between two cases is often calculated by reviewing how similar its individual features are, with weights on each. A taxonomy has been suggested to divide the similarity mechanisms into four different categories [Cunningham, 2009]. These categories and their subcategories are illustrated in Figure 2.8

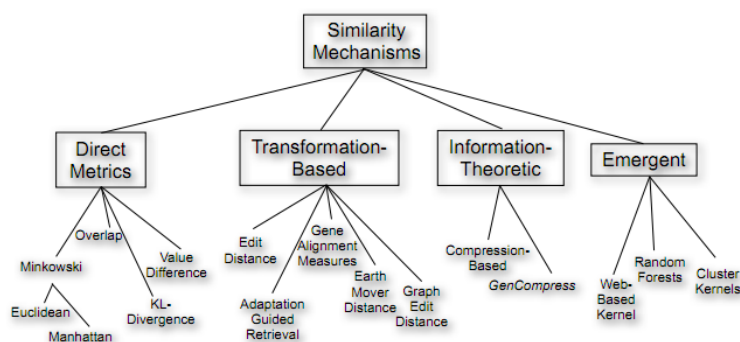


Figure 2.8: Similarity Mechanisms in CBR

The similarity mechanisms are highly dependent of the representation of the feature values. A case can range from a simple feature vector to very complex case structures. Cunningham categorizes case representations in three types; **1. feature-value representation**, **2. structural representation** and **3. sequences and strings**.

1. Cases built up from simple numeric feature values ranging normalized from 0.0 to 1.0. Can be both internal and external value representations.
2. The feature values in the case can have an internal structure more sophisticated than a feature vector, like a hierarchical structure, a semantic network or another non-atomic structure.
3. A bag-of words(free text) or a sequence of words/strings

Direct Mechanisms This is the dominant strategy according to Cunningham. In most cases the features will be represented by vectors and the similarity is computed directly based on these vectors. A k-nearest neighbor is usually used to find the k most similar cases and then the class is determined by these in some way(e.g a majority vote or a distance weighted vote). These mechanisms has a computational advantages, tied to feature-valued representation of cases.

Transformation-based Mechanisms The basic idea is to try to transform one case into another case, instead of evaluate the distance/similarity between objects. Edit Distance is the most basic, where you count number of transformations that have to be executed to transform one string into another.

Information Theoretic Mechanisms Information theory offers several ways of determine the similarity between cases. One of them is compression-based similarity works directly on the raw case data. If two documents/cases are very similar the compressed and combined documents/cases will not be much greater than the compressed size of a single document/case. Information-based similarity has some of the same characteristics as transformation-based similarity.

Emergent Measure Mechanisms Includes different ways of using machine learning to find similarity, using significant processing power to produce a characterization of the data. Random forests and Cluster Kernels are of the internal type, and they uncover new relationships from analysis within the data set. Where the first one typically creates a lot of unpruned decision trees that work together; an ensemble technique. Cluster Kernels are semi-supervised learning algorithms. Web-based Kernels are external by bringing new knowledge from outside the data set by searching the web.

2.5.4 Knowledge representation in CBR

Richter's knowledge containers model[Richter, 2005], Figure 2.9, is a known method of structuring a Case-Based Reasoning system.

There are in all four knowledge containers:

- The **Case Base** contains the cases
- The **Similarity Measure** contains the different similarity measures
- The **Adaption Knowledge** (also known as The Solution Transformation) contains the information needed to adapt an old solution case from the case base to best suit the new problem case.
- The **Vocabulary** contains all the information about the attributes and which data structures that are used to represent them

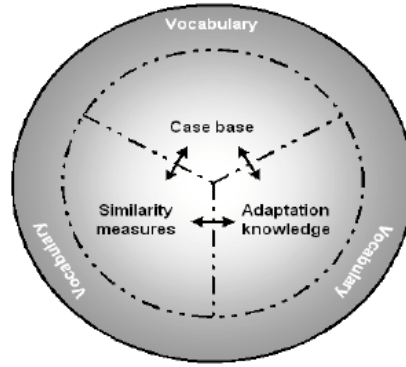


Figure 2.9: Richter's Knowledge Containers

The boundaries between the different knowledge containers are fluid, in the way that information can be moved from one container to another. Each container can also have several sub-containers if necessary.

2.5.5 Well-known CBR systems

There have been many Case-Based Reasoning system implemented over the years. Below we have listed some of the earlier and to our knowledge, most known work/study done in the real-world, and also a commercial solution in the oil drilling industry.

CYRUS

Kolodner's CYRUS[Kolodner, 1983] was the first implemented CBR system and was based on Schank's dynamic memory theory. It was a system which featured travels and meetings of an US Secretary of State.

PROTOS

PROTOS[Bareiss, 1989, Porter et al., 1990] is a Case-Based problem solving and learning system for heuristic classification tasks, made by Bruce W. Porter and Ellis R. Bareiss.

CREEK

Creek[Aamodt, 2004] is a knowledge-intensive CBR approach, with tight coupling between general domain knowledge and cases. It is targeted at addressing problems in weak or open theory domains.

DrillEdge

Verdande Technology was formed by a group of professors and students from the Norwegian University of Science and Technology(NTNU), and is founded on the CBR principle and uses it in the domain of oil drilling. Their product DrillEdge[Sørmo, 2009], is formed from the basis of CREEK, more specific TrollCreek. The product is in their own words:

designed to reduce risk, increase rate of penetration and reduce non-productive time while drilling. We help operators to re-use knowledge in order to diagnose and avoid costly drilling problems before they escalate.

Verdande Techonology

Chapter 3

Related Research

Studying related research has not been a major focus in our thesis, as we have focused more on getting a functional system up and running. We have however chosen to list some related research below.

In Chapter 2 we mentioned some of the work and studies being conducted in the field of Aquaculture/Agriculture. The most relevant work is however related to the different approaches in the domain when using Case-Based Reasoning. The early warning system EWS-FDWQ[Li et al., 2006b] mentioned in Section 2.3.3 uses a Case-Based approach. The knowledge and cases are stored in a relation database, which makes it easy to create new, update and delete cases.

As mentioned in Section 2.3.2, a web-based expert system, Fish-Expert, was created. The authors of the paper proposed in 2008 a way to deal with some of the issues related to the system[Zhu and Li, 2008], such as:

- The system had too complex rules to be understood by those without deep knowledge
- The inference engine does not work efficiently and is quite time-consuming

The proposition was to use CBR so that one would not be so heavily dependent on general domain knowledge, but instead use previously solved cases. The system developers interviewed several fish experts who had plenty of knowledge, represented the knowledge in a structured way before they created case representations and indexing.

Wang and Li presented in 2009 a fish disease diagnosis system using SMS(Short Message Service) to communicate with the users[Guirong, 2009]. The system uses CBR to solve problems and allegedly has an incredible accuracy of approximately 93.6% for fresh-water-fish.

The CBR approach was also adopted in relation with Rule-Based Reasoning for fish disease diagnosis[Fu et al., 2009]. A generic algorithm was proposed in order to solve the problem feature vector space and to integrate CBR and RBR.

Zhu and Li demonstrated a two-step case-retrieve model in fish disease diagnosis with the help of CBR[Zhu and Li, 2008]. The cases in the knowledge-base consists of an object(fish), the given symptoms and the treatment given. The similarity assessment between a query and the case-base is done by a clustering algorithm to find which part of the case-base the query belongs to. A simple nearest neighbor algorithm is then used to find the closest matching cases.

CBR in combination with image processing has also been studied[Lou et al., 2007], where the user uploads an image of a fish and the system finds the most similar image(case) in the case base. Features are extracted from the image, based on pre-processing of the image, and a case is created and used as a query for the case base.

Yuan, Mao and Zhao proposed in 2010 a novel interference method that is a combination of CBR, grey theory and Back propagation Artificial Neural Network[Hongchun et al.]. The system uses the four RE's briefly mentioned in Section 2.5, but the retrieval step is replaced by the combination of grey theory and Neural Networks to find similar cases. The system has been successfully applied to the Vannamei expert system for disease diagnosis and treatment.

Our group is also working on creating a commercial CBR system which is based on the SimFrame architecture[Tidemann et al., 2011]. A demonstrator system was created based on Protegé and myCBR and it was there discovered that there was need for a structured registration portal for fish mortality. The hand-written reports, which the cases were based on, were lacking in quality, both in regards to missing attributes and different names for the same thing.

CARMA[Branting et al., 1999] proposed the use of *model based adaption* as a technique for integrating case-based reasoning and model-based reasoning. The main motivation for this was that the domain of biological systems is limited by incomplete models and empirical data. So since the different approaches, case-based reasoning and model-based reasoning, could not understand the domain alone, why not incorporate them. The technique is then to use case-based reasoning to find an approximate solution, while model-based reasoning is used to adapt this solution to a more precise solution. When evaluating the performance they found that using this approach in the domain of range-land pest management yielded good results as opposed to the individual methods on their own.

In addition to the domain related work above, there has been work done at our department at NTNU, Department of Computer and Information Science, with regards to the framework we are basing our system on. Bacchus[Gravem, 2010], a combined Case-Based Reasoning and Bayesian reasoner for decision support in choosing wine to specified meal is introduced. The system uses the jColibri framework to achieve its Case-Based Reasoning capabilities, and we have drawn inspiration and knowledge from this thesis.

Chapter 4

Methodological Approach

In this Chapter we describe how we planned to reach the goals we specified in Section 1.1. The following issues will be addressed:

Equipment describes the different tools which are used in the project.

jColibri will give an overview of the jColibri framework with emphasis on the example Travel Recommender.

Protegé + myCBR describes a knowledge-base framework and its sub components.

Weka describes the Weka software, and how it will be used in our project.

Hibernate will give a basic description of how it can be used and why.

Acquiring and analyzing the new data from the fish farming sites.

Machine Learning Methods we will use on the new data set.

Knowledge Acquisition will describe how we aim to gather knowledge in order to create the CBR system.

Putting it all together will propose the initial implementation of Glaucus.

Evaluating the final system will cover how we aim to evaluate the final system.

4.1 Equipment

The following development frameworks, software and other equipment are used during the project:

4.1.1 Database

MySQL¹ is the worlds most popular open source database and it has a variety of different tools which are used during the majority of the project.

MySQL Community Server² is the worlds most popular open source database. The database is easy to handle and maintain, while also having a great and active community at your disposal if you should ever face an issue or problem.

MySQL Workbench (GUI Tool)³ is a visual database design application which is used maintain and manage the database schemata.

MySQL Connector/J 5.1.15⁴+ **Connector/MXJ** is the official JDBC driver for MySQL. It will be used in the development of the Java application.

4.1.2 Development Frameworks

jColibri⁵ is a Case-Based Reasoning framework written completely in Java.

Protegé⁶ is a free, open source ontology editor and knowledge-base framework. The framework is written in Java and it supports several plug ins.

MyCBR⁷ is an open source case-based reasoning tool which is built on top of Protegé as a plug in. The tool enables fast and easy prototyping of a CBR system.

Weka 3.6⁸ is an application for data mining tasks. It includes a vast collection of machine learning algorithms which can be used directly on some data set or through Java code. The tools included in are data pre-processing, classification, regression, clustering, association rules and visualization. The software is also open source.

Hibernate⁹ is an open source Java persistence framework. It provides a way to map an object-oriented domain model to a relational database.

¹<http://www.mysql.com/>

²<http://www.mysql.com/downloads/mysql/>

³<http://www.mysql.com/downloads/workbench/>

⁴<http://www.mysql.com/downloads/connector/j/>

⁵<http://gaia.fdi.ucm.es/projects/jcolibri/>

⁶<http://protege.stanford.edu/>

⁷<http://mycbr-project.net/>

⁸<http://www.cs.waikato.ac.nz/ml/weka/>

⁹<http://www.hibernate.org/>

4.1.3 IDE's and Misc

Netbeans IDE 6.7.1 and 6.9¹⁰ is an IDE(Integrated Development Environment) which is part of an open-source project called Netbeans. The IDE is written in Java and it aims to give developers a rock-solid development environment in a broad range of languages. The IDE can be run on any system where a Java Virtual Machine is installed and has support for a wide range of third-party software such as MySQL database, GlassFish and Versoning Control Tools. Netbeans 6.7.1 is used to create UML diagrams.

TeXnicCenter¹¹ is an IDE for developing latex documents under the Windows-platform. This is used to document our work.

EndNote X4¹² is a reference managment software. The program is used to keep track of your bibliography and references when writing documents such as articles and reports.

Gephi¹³ is used to illustrate the relationship between the different fish groups in the provided data set.

Gliffy¹⁴ is a online diagram software. We use this to create diagrams of various types in the project.

¹⁰<http://netbeans.org/index.html>

¹¹<http://www.texniccenter.org/>

¹²<http://www.endnote.com/>

¹³<http://www.gephi.org/>

¹⁴<http://www.gliffy.com/>

4.2 jColibri

jColibri is a Java framework for creating Case-Based Reasoning systems. The framework is a product of the Group for Artificial Intelligence Applications, short GAIA, which is a group composed of professors and student interested in the field of Artificial Intelligence at the Complutense University of Madrid. The first thing we need to know is the motivation and history behind the framework.

4.2.1 History

The group's approach to Case-Based Reasoning came from integrating applications that combined both general domain knowledge and case specific knowledge. COLIBRI(Cases and Ontology Libraries Integration for Building Reasoning Infrastructures) is the product of Belén Díaz-Agudo's Phd thesis[GAIA, 2011] with the guidance of Pedro A. González-Calero, in which they propose a domain independent architecture in order to design knowledge intensive CBR systems.

COLIBRI

COLIBRI[Díaz-Agudo et al., 2002a] is based on knowledge acquisition from a library of independent ontologies and CBROnto[Díaz-Agudo et al., 2002b]. CBROnto is a ontology with CBR terminology which guides the case representation and CBR problem solving methods to solve CBR tasks. COLIBRI and CBROnto were developed in the languages/environments LISP and LOOM, and are therefore quite unusable by users not familiar with Description Logic(DL). In order to broaden the community jColibri was created. COLIBRI is the predecessor of jColibri 1.x and jColibri 2.x.

jColibri

jColibri, illustrated in Figure 4.1, is as mentioned above a further development of the architecture COLIBRI. It was originally created by Juan José Bello while Antonio A. Sanchez-Ruiz Granados greatly contributed to the project[GAIA, 2011]. The design of the framework consist of a number of XML-files and Java classes organized around the following four elements:

Tasks and Methods which are supported by the framework. They are described by XML-files.

Case Base which are supported, along with connectors to support different persistences such as database and file system.

Cases are provided in the framework in form of abstract classes and interfaces to support an actual real-world case.

Problem solving methods are represented by actual code in the framework.

One of the main issues with a framework is how to use it. jColibri supports a semiautomatic configuration(authoring) tools which helps the user create a CBR system through a dynamic graphical interface. This feature is as of June 10, 2011, not implemented in jColibri 2.x.

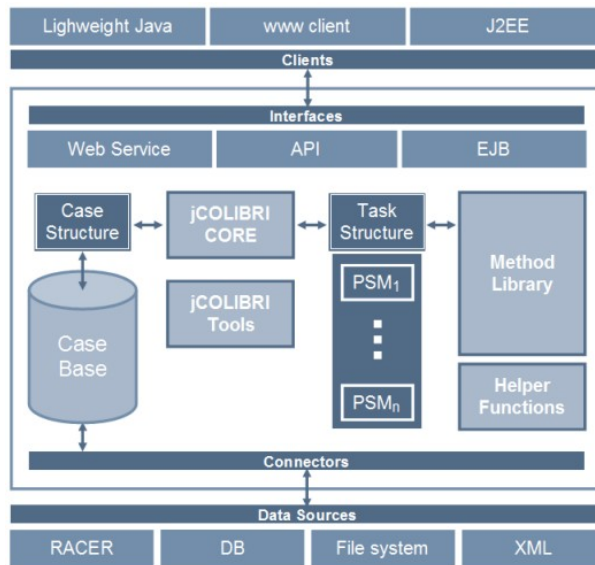


Figure 4.1: jColibri framework structure

jColibri 2

jColibri 2 is a major release where most of the features from the jColibri 1.x framework has been reimplemented. Juan A. Recio-Garcia is the main developer on the jColibri 2 project and he also completed the first version of the framework. The development of the second version of the framework was also the base of his Phd[Recio-García, 2008].

The main reason for the reimplementation of jColibri is to make the framework easier to extend with your own code and to make it more robust. The framework is also designed as regards to separating the problem solving techniques and the domain knowledge. Authoring tools as those present in jColibri 1.x are not included but they are under development.

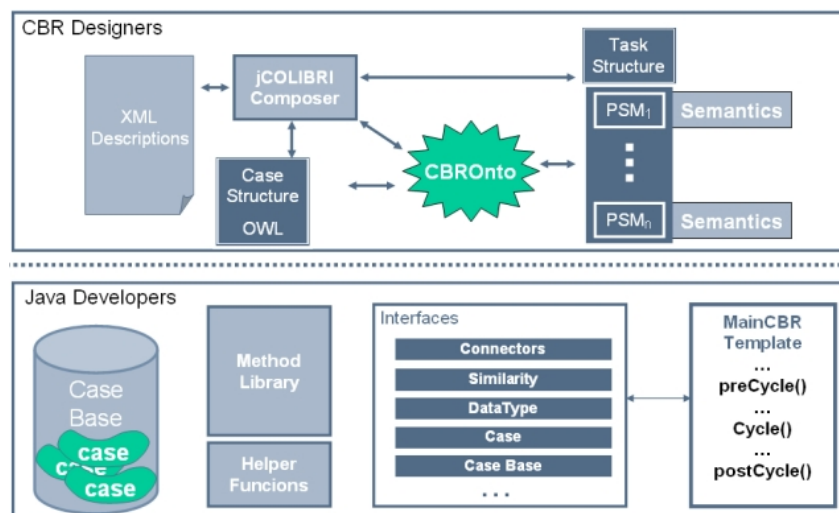


Figure 4.2: jColibri 2 framework architecture

A useful feature in jColibri which will be addressed in Section 4.5 is the support

for the use of Hibernate. Hibernate will make it easier to map the database to their corresponding Java classes. This will limit the amount of actual SQL queries we would have to generate and use in our application.

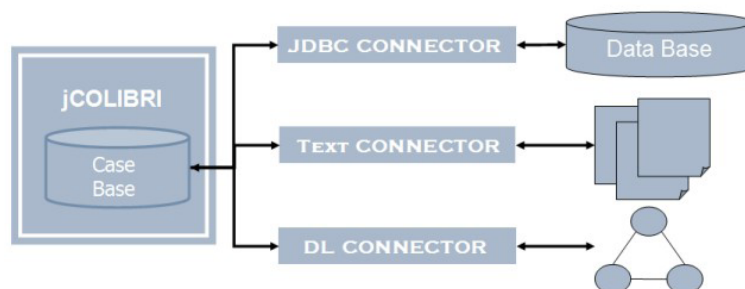


Figure 4.3: Persistence architecture

jColibri's two layered persistence architecture, illustrated in Figure 4.3, is composed of Connectors and In-memory organization of cases. The Connectors are interfaces which know how to communicate and load cases from a physical medium. There are currently three types of Connectors in jColibri; DatabaseConnector, PlainTextConnector and OntologyConnector. The most appropriate Connector for us would be the DatabaseConnector, since its easier to maintain and debug a database if problems should occur. The DatabaseConnector also supports Hibernate internally.

Each Connector is configured through XML configuration files, which state which class in the project is the Description for the Case, and which is the Solution. We also supply the Hibernate configuration file, which will be covered in more detail in Section 4.5. Basically its a file which says how class objects are mapped to tables in a given database schema.

The second layer in the architecture is related to how the cases are organized once they are loaded to memory. There are three basic in-memory organizations of cases supported in jColibri; LinealCaseBase, CachedLinealCaseBase and IDIndexedLinealCaseBase. There is also a Interface which makes it possible for developers to create their own representations if the would like to.

Mechanisms for Retrieving, Reusing, Revising and Retaining cases are included and the code is well documented. For the purpose of learning how to use the framework we use the jColibri tutorial[Recio-García et al., 2008] and implement the Travel Recommender system. The Travel Recommender system is a test example created by the authors of jColibri, and it uses many of the components required to build a working CBR system.

4.2.2 jColibri 2 tutorial

jColibri can be downloaded from the jColibri web site¹⁵. There are three versions available from the site(June 10, 2011), and we will focus on using version 2.1. The Travel Recommender system is a system which has a case base consisting of different trips, or cases. Each case is composed of several simple attributes in a flat structure. Id, holiday type, number of persons and duration are among the list of available attributes. The

¹⁵<http://gaia.fdi.ucm.es/projects/jcolibri/>

solution of the case is composed of the price and hotel. The CBR cycle starts with the user input.

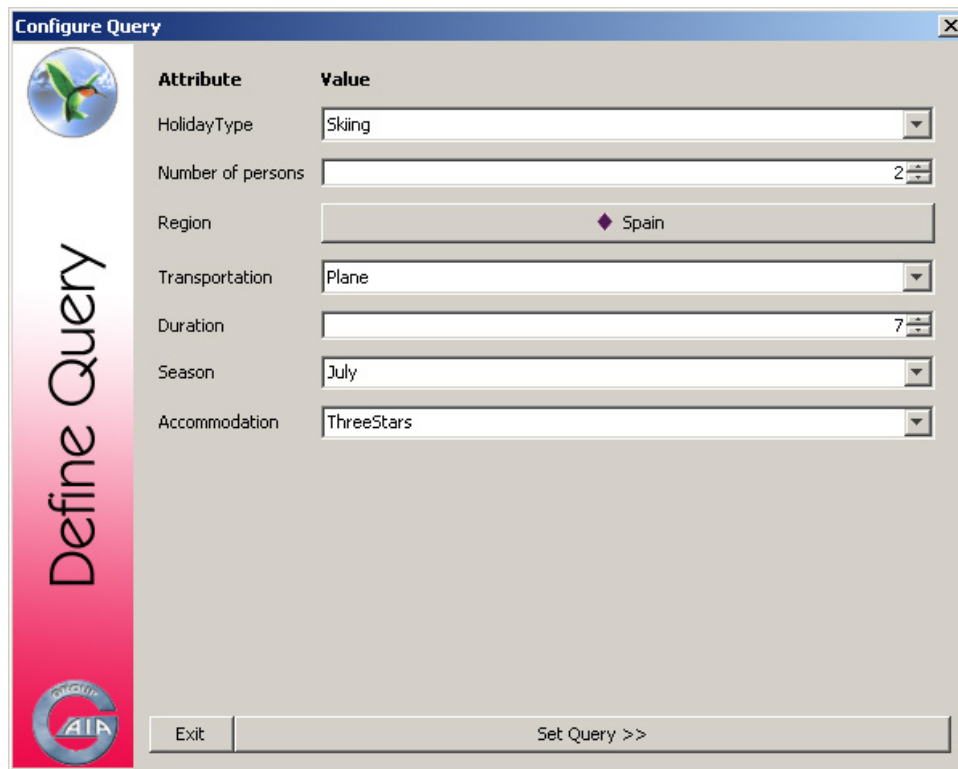


Figure 4.4: Configure Query Dialog

The user inputs some attributes for the desired trip, Figure 4.4, and the system finds the the most similar trip in the case base based on some similarity functions. The selection of the most similar cases is related to the retrieve step in the CBR cycle. The K nearest cases are then displayed to the user, and we get the option of adapting(Reuse step) the solution according to the values set in the query.

The revise step is handled by the domain expert, which is in this case the user. The solution which Travel Recommender recommends may not be available at this time, and the user has to revise the solution.

As the last task in the cycle(Retain) we can save the new case so it can be used later. This tutorial demonstrates a very simple CBR application where the jColibri framework can be used. In addition to the tutorial, there are a dozen different examples in the jColibri source of how to use different features in the framework. There are methods and features for evaluating the application, visualizing the case base, using ontologies, compound attributes, textual CBR extension and so on. The different types of methods will be covered in more detail in the actual implementation chapters.

Case Components and Similarity

A case component or compound attribute is an attribute which contains numerous simple attributes. In the Travel Recommender system the TravelDescription is in itself a compound attribute, where we have simple attributes such as id, duration and holiday type.

Every compound attribute must implement the `CaseComponent` interface and redefine the `getAttribute` method.

Similarity in `jColibri` works in two ways, `global`(for compound attributes) and `local`(for simple attributes). A common `Global` similarity function is `Average`, where we average over some collection, while `Local` similarity function such as `Equal`, `Interval` and `Threshold` work on simple attributes. In the instance of `TravelDescription` the similarity between a case and query is calculated by averaging(`global`) over the similarity between each simple attribute(`local`), where each attribute is assigned their own local similarity function.

In the `jColibri` framework it is possible to create own similarity functions by either implementing the `GlobalSimilarityFunction` interface or the `LocalSimilarityFunction` interface.

4.3 Protegé + myCBR

Protegé is an open source ontology editor and knowledge-base framework. The Protegé-Frames and Protegé-OWL editors are the two main ways of modeling the ontologies. These ontologies can be exported into a variety of formats including `RDF(S)`, `OWL`, and `XML Schema`. It is, as `jColibri`, based on `Java` and also a flexible base for rapid prototyping and application development. Protegé is, according to their website, supported by a strong community of developers and academic, government and corporate users, who are using Protegé for knowledge solutions in areas as diverse as biomedicine, intelligence gathering, and corporate modeling.



Figure 4.5: myCBR logo

myCBR is another open source tool, made for Case-Based Reasoning system development. Which can be downloaded as a plug-in to Protegé from the website¹⁶. Together with Protegé, myCBR offers advanced functionality for defining and visualizing of object-oriented case representation. It is easy to use through a clear graphical user interface, which enables quick testing and prototyping. It is developed at the DFKI(Deutsches Forschungszentrum für Künstliche Intelligenz GmbH). It is also possible to use the myCBR retrieval engine as a standalone application.

4.3.1 Usage with `jColibri`

One of the main advantages of using Protegé + myCBR is to create our own custom similarity functions. The issue is, how do we use them with the `jColibri` framework? The `jColibri` developers(GAIA) in cooperation with the myCBR developers(German Research Center for Artificial Intelligence DFKI GmbH) developed a contribution to the framework which contains wrapper methods for using similarity measures/functions created with myCBR.

¹⁶<http://www.mycbr-project.net/download.html>

Example of use

The first step in order to use the wrapper is to create the similarity function. This is done by creating an attribute in Protegé, such as the one pictured in Figure 4.6. We have chosen to set this attribute as a symbol, where we have three allowed values.

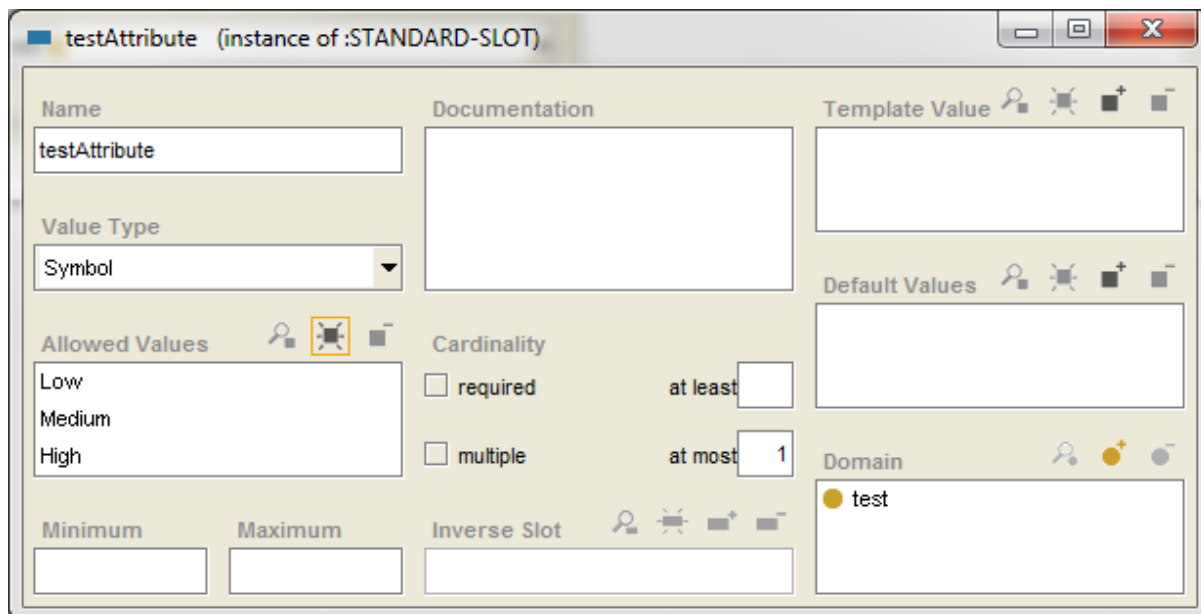


Figure 4.6: Attribute in Protegé

The second step is to create the similarity function for this attribute. This step is done in the Similarity Measure Editor Tab, usually located to the wide right in Protegé. In the editor we have several options, and we will choose Similarity mode=Table and Symmetry=Symmetric. This is a quite simple way to measure similarity between symbols where we have allowed values. Figure 4.7 illustrates the values we have given the similarity between allowed values. Given that you have a value of Low in your case base, and a query with value Medium, you would get a similarity of 0.5. This is also true vice versa.

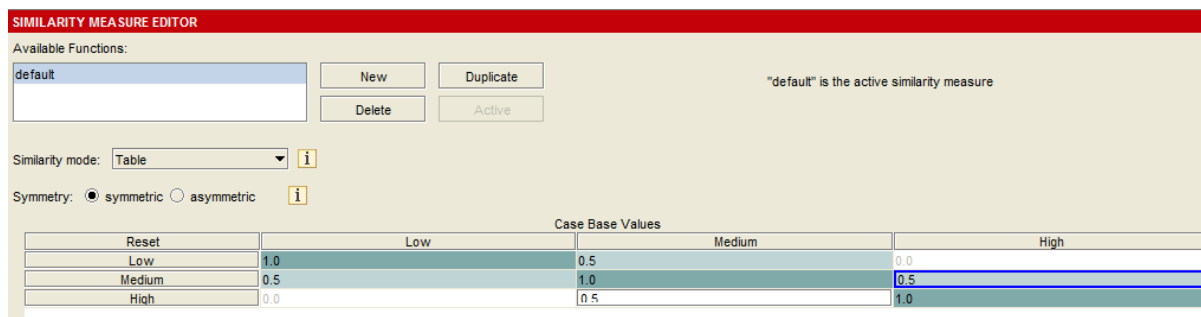


Figure 4.7: testAttribute similarity

The third step is to export the similarity measure. This is done by using the menu system located in the top of Protegé, more precisely through MyCBR->Export Similarity Measure...

Listing 4.1: XML file for testAttribute

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <SMFunction smfname="default" model_instname="testAttribute" type="Symbol">
3   <QuerySymbol symbol="Low">
4     <CBSymbol sim="1.0" symbol="Low" />
5     <CBSymbol sim="0.5" symbol="Medium" />
6   </QuerySymbol>
7   <QuerySymbol symbol="Medium">
8     <CBSymbol sim="0.5" symbol="Low" />
9     <CBSymbol sim="1.0" symbol="Medium" />
10    <CBSymbol sim="0.5" symbol="High" />
11  </QuerySymbol>
12  <QuerySymbol symbol="High">
13    <CBSymbol sim="0.5" symbol="Medium" />
14    <CBSymbol sim="1.0" symbol="High" />
15  </QuerySymbol>
16 </SMFunction>
```

The final part is to set up the similarity measure with an attribute in your system. This is done as shown in Listing 4.2.

Listing 4.2: testAttribute similarity Java code

```
1 NNConfig config = new NNConfig();
2 config.addMapping(new Attribute("testAttribute", TestClass.class), new
   MyCBRTTableSimilarity("path/testAttribute.xml");
```

4.4 Weka

WEKA is an open source data mining tool developed at the University of Waikato in New Zealand. The abbreviation stands for Waikato Environment for Knowledge Analysis.

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data preprocessing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

- Weka web¹⁷

We use the Weka explorer and its preprocessing tool, illustrated in Figure 4.8, to examine the database, and also to run the same machine learning algorithms as we did in our specialization project[Garaas and Hiåsen Stevning, 2010]. The machine learning approaches were Decision Tree J48, Naive Bayes, Bayes Net and Neural Network. The machine learning was not the focus in this project, but mainly a way to prove that our assumptions about having a better data set, would improve the performance of the algorithms and to confirm that this is in fact a better data set.

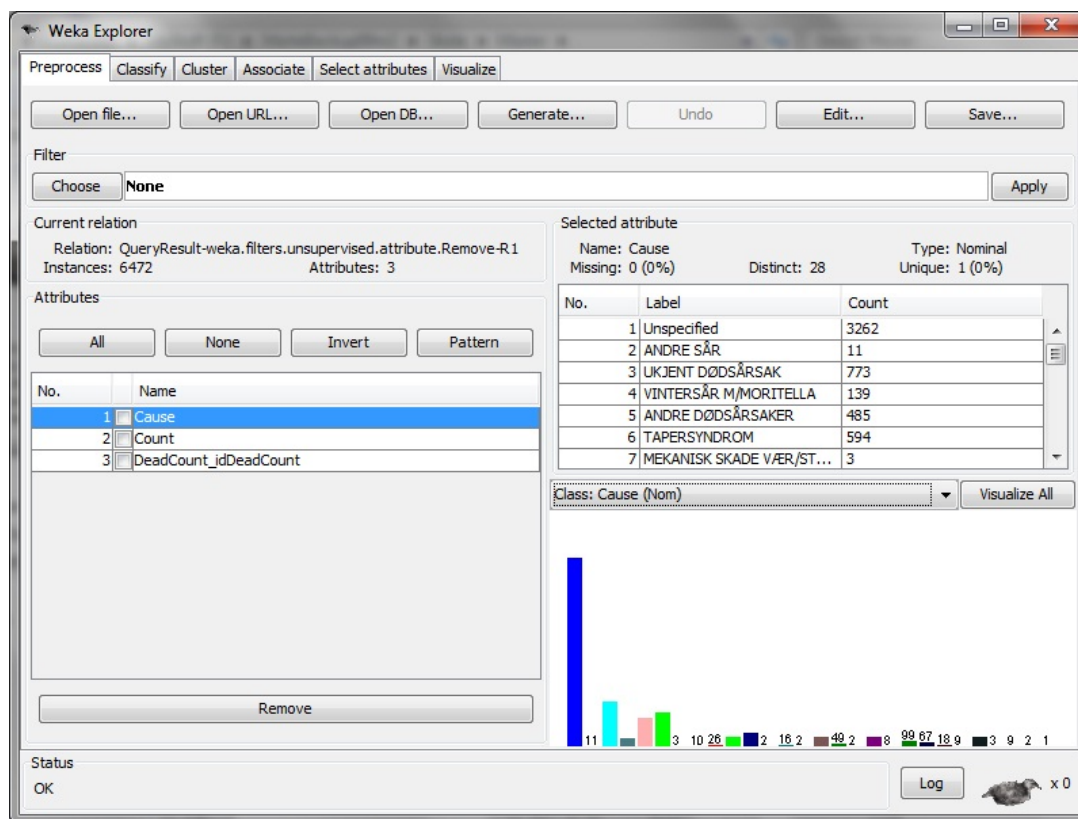


Figure 4.8: Preprocessor

¹⁷<http://www.cs.waikato.ac.nz/ml/weka/index.html>

4.5 Hibernate

Hibernate is an Object/Relational Mapping solution in Java. The way it works is that it maps all data from an object model representation to a relation model representation, and vice versa. With this in mind we can set up Hibernate between our object model, such as Java classes, and the database.

4.5.1 Hibernate Mapping

Given a Java class such as the one in Figure 4.9 we can create objects of it. In order to store instances of it in a database we would however have to create a table in the database schema, extract the values from the FishDeath object and use SQL-queries and the JDBC API directly. With more than one class, and lots of different relationship to keep track of, this would be a quite time-consuming task. This is where Hibernate has its strength for developers.

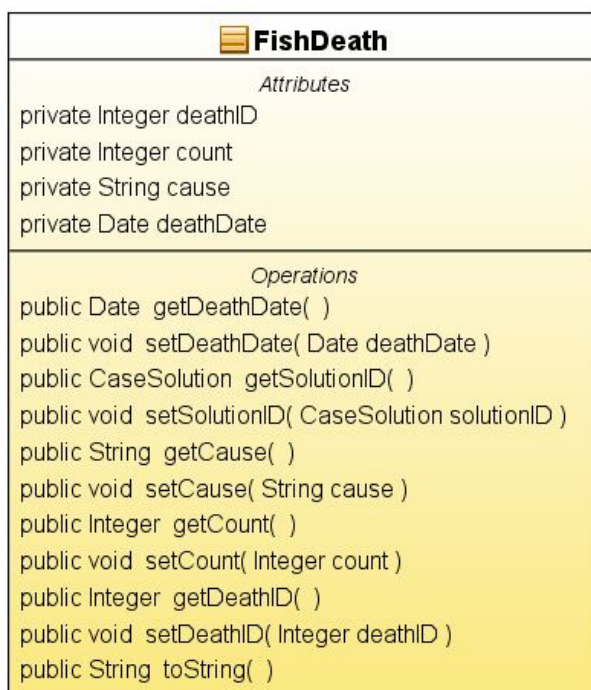


Figure 4.9: Fish Death Class

The first thing required to do is to map the Java class to a table in the database schema. Listing 4.3 illustrates how a possible mapping could look like. The two first lines denote the format of the file. Line four states that this is a class named FishDeath in the `my.cbr.database.model` package, together with some additional arguments. Once the name of the class is established, we are going to list the properties from the class which are going to be saved to the table in the database. Each property is listed in the file, with the exact same name as in the class file. If you want the property to have another name in the table, this can be achieved through the use of an additional parameter, `column="some column name"`. Each table also needs to have an identification column. Line numbers five to seven show how you set up the id to auto-increment. There are also

many other generator types which may be useful if increment does not fit the current situation.

Listing 4.3: Hibernate mapping file for FishDeath

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate_Mapping_DTD_
   3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
3 <hibernate-mapping>
4   <class dynamic-insert="false" dynamic-update="false" mutable="true" name=
     "my.cbr.database.model.FishDeath" optimistic-lock="version"
     polymorphism="implicit" select-before-update="false">
5     <id name="deathID">
6       <generator class="increment"/>
7     </id>
8     <property name="cause"/>
9     <property name="count"/>
10    <property name="deathDate"/>
11    <many-to-one class="my.cbr.casedescriptions.CaseSolution" name="
     solutionID" not-null="false"/>
12  </class>
13 </hibernate-mapping>
```

Relationships with other classes/tables are given by using one of one-to-one, many-to-one, one-to-many or many-to-many. Depending of the type, you may also be required to set up something similar in the class which this current class has a relationship to. The relationship given by this mapping is that a CaseSolution can have many FishDeath instances, but on FishDeath instance can only have one CaseSolution.

Listing 4.4: Hibernate mapping file for CaseSolution

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate_Mapping_DTD_
   3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
3 <hibernate-mapping>
4   <class dynamic-insert="false" dynamic-update="false" mutable="true" name=
     "my.cbr.casedescriptions.CaseSolution" optimistic-lock="version"
     polymorphism="implicit" select-before-update="false">
5     <!-- Use the same ID as CaseDescription -->
6     <id name="caseID">
7       <generator class="foreign">
8         <param name="property">caseDescription</param>
9       </generator>
10    </id>
11    <one-to-one class="my.cbr.casedescriptions.CaseDescription" constrained
     ="true" name="caseDescription"/>
12    <!-- Added lazy loading in order to counter a problem with
     persistent bag-->
13    <bag cascade="all" inverse="true" lazy="false" name="fishdeath" table="
     FishDeath">
14      <key column="solutionID" not-null="true"/>
15      <one-to-many class="my.cbr.database.model.FishDeath"/>
16    </bag>
17    <property name="risk"/>
18    <property name="aggregatedCount"/>
19    <property name="deathRatio"/>
```

```
20 | </class>
21 | </hibernate-mapping>
```

For the CaseSolution, which is referenced by FishDeath, to be able to know which FishDeath instances it is hooked to we have to use Collection mapping. To do this we create a property in CaseSolution which is a subtype of java.util.Collection, such as java.util.List. A List in the Hibernate mapping environment is denoted as a bag, while there are also similar types such as map and set. Lines 13-16 in Listing 4.4 states that the CaseSolution class has a bag named fishdeath, and that this bag contains instances of the class my.cbr.database.model.FishDeath. The id of CaseSolution is also put on the many side(FishDeath) to maintain the relationship(line 14 key column). The inverse and cascade parameters indicate that this class is the owner of the referenced one. What this means is that when we come to the point where we want to store the objects in a database, we only need to supply the session with the owner object. The session will automatically insert the objects which are coupled to the owner object.

4.5.2 Hibernate Configuration File

When all Hibernate mappings are configured we pass these to a Hibernate configuration file. This is the file where we input parameters such as which database to use, SQL dialect, drivers and other things relevant for the session factory. Listing 4.5 shows a typical format of the configuration file. We specify that in this instance we want to use a MySQL database, root as username, show SQL under run-time and if there are changes in the mapping files *update* the database schema. The hibernate mapping files are added with the <mapping resource=""/> blocks.

Listing 4.5: Hibernate configuration file

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate_
   Configuration_DTD_3.0//EN" "http://hibernate.sourceforge.net/hibernate-
   configuration-3.0.dtd">
3 <hibernate-configuration>
4   <session-factory>
5     <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</
      property>
6     <property name="hibernate.connection.driver_class">com.mysql.jdbc.
      Driver</property>
7     <property name="hibernate.connection.url">jdbc:mysql://localhost:1337/
      FishFarmingCaseBase?createDatabaseIfNotExist=true&server.baseDir
      =C:\dbtemp\mxj-tmp</property>
8     <property name="hibernate.connection.username">root</property>
9     <property name="hibernate.connection.pool_size">1</property>
10    <property name="hibernate.hbm2ddl.auto">update</property>
11    <property name="hibernate.show_sql">>true</property>
12    <mapping resource="my/cbr/database/model/resources/FishDeath.hbm.xml"/>
13    <mapping resource="my/cbr/casedescriptions/resources/CaseSolution.hbm.
      xml"/>
14  </session-factory>
15 </hibernate-configuration>
```

4.5.3 Storing data

The final step is to create some objects of both `CaseSolution` and `FishDeath`, couple them together and save them to session. The Java code given in Listing 4.6 shows that we create a object of `CaseSolution`, and then create multiple `FishDeath` objects which are added to `CaseSolution`'s `Collection` property, the `List fishDeath`. Each `fishDeathObject` is set to point to its owner, namely `CaseSolution`.

Listing 4.6: Java store `CaseSolution` and `FishDeath`

```
1 CaseSolution caseSolution = new CaseSolution ();
2 // Create FishDeath objects and insert some data
3 // For each FishDeath object do:
4 caseSolution.addFishDeath(fishDeathObject); // add to collection
5 fishDeathObject.setSolutionID(caseSolution) // set relationship with
   caseSolution
6
7 try{
8     // Load configuration and session
9     Configuration cfg = new Configuration ();
10    cfg.configure("my/cbr/database/resources/Hibernate_config.cfg.xml")
11    ;
12    SessionFactory sf = cfg.buildSessionFactory ();
13    Session session = sf.openSession ();
14
15    org.hibernate.Transaction transaction = session.beginTransaction ();
16    session.save(caseSolution);
17    transaction.commit ();
18
19    session.close ();
20 }catch (Exception e) {
21     org.apache.commons.logging.LogFactory.getLog(this.getClass()).error
   (e);
22 }
```

When we have created the objects needed we create a configuration with the hibernate configuration file. A `SessionFactory` can then be acquired from the configuration, which will make it possible to open a session. The `CaseSolution` object is saved to session and the transaction is committed. The data is now safely stored in the database.

4.5.4 Loading data through jColibri

In order to load the data back from the database we will use some of the features given to us by `jColibri`. This will be addressed later in the documentation of the implementation.

To be able to load data in `jColibri` we are going to use what is called a `Connector` in `jColibri`. A `Connector`, as mentioned in Section 4.2, is a `Interface` between the application and the physical medium where the cases are stored. This `Connector` knows how to get the data from the database, and store them to the in-memory organization indicated by the developer. Listing 4.7 is a typical configuration file for the `DataBaseConnector`.

Listing 4.7: `DataBaseConnector` configuration file

```
1 <?xml version="1.0" encoding="ISO-8859-9"?>
2 <DataBaseConfiguration>
```

```

3 <HibernateConfigFile>my/cbr/database/resources/Hibernate_config.cfg.xml
  </HibernateConfigFile>
4 <DescriptionMappingFile>my/cbr/casedescriptions/resources/
  CaseDescription.hbm.xml</DescriptionMappingFile>
5 <DescriptionClassName>my.cbr.casedescriptions.CaseDescription</
  DescriptionClassName>
6 <SolutionMappingFile>my/cbr/casedescriptions/resources/CaseSolution.hbm
  .xml</SolutionMappingFile>
7 <SolutionClassName>my.cbr.casedescriptions.CaseSolution</
  SolutionClassName>
8 </DataBaseConfiguration>

```

The first thing which is added is the `HibernateConfigFile`, which is the one created in Listing 4.5 under Section 4.5.2. The next thing is to specify the mapping and class files for Case Description and Solution. Note that when you add these you have to remove the mapping from the Hibernate Config File, or you will receive an exception when running the application. Listing 4.8 shows how you would retrieve the cases from the database in Java through jColibri.

Listing 4.8: Load case from database

```

1 // Custom embedded(mxj) MySQLDBServer
2 MySQLDBServer.init();
3
4 // create a new LinealCaseBase
5 this.caseBase = new LinealCaseBase();
6
7 // Set the Connector
8 this.connector = new DataBaseConnector();
9
10 // initialize the connector by using the database config file
11 this.connector.initFromXMLfile(jcolibri.util.FileIO.findFile(CBRApplication
  .DATABASE_CONFIG_FILE));
12
13 // store all data in Lineal Case Base
14 this.caseBase.init(this.connector);
15
16 java.util.Collection<CBRCase> cases = this.caseBase.getCases();

```

4.6 Acquiring and analyzing the data

In this project we are provided with a data set similar to that in our specialization project [Garaas and Hiåsen Stevning, 2010] autumn 2010. An analysis was then completed with this data set, and we have used some of the outcome in this project. In addition, an analysis of the new data is important in order to get an overview of how the data is distributed in the database. It is important not to take any assumptions when it comes to a new data set as there can be several changes that affect the system architecture. The MySQL Workbench is used to create a database diagram, in order to visualize the complete relationships between the entities. The plan from there was to use this model to pick out tables from this model, one by one, and examine them by using the preprocessor tool in Weka. The sequence of the analysis was as follows:

1. Find a table in the model
2. Look up its attributes by using the preprocessor in Weka
3. Run through the values of each attribute, look for missing values and corrupted data
4. Also find out what the data in the table represents
5. Use its foreign keys to find its connecting tables and do the same procedure for them

This analyzing process is crucial in our journey towards a final attribute set, which can be used as a case in our system. A more detailed explanation and the result of this analysis can be found in the Chapter 6.

4.7 Machine Learning Methods

Machine Learning is as stated in Section 2.4 a scientific discipline within Artificial Intelligence. Its main focus is to learn a concept from a given set of empirical data, such as from sensors. In our specialization project we focused on using four different machine learning methods/approaches to learn which cause of death was most likely given a set of attributes. In this thesis we aim to follow up on what we did then, and see if the changes made to the data set have any influence on the results.

The Machine Learning methods used in the specialization project were:

Decision Tree: J48 is a algorithm which is based on the the popular C4.5, developed by J. Ross Quinlan[Quinlan, 1993]. In the tree structure leaves represent classification, while branches represent conjunctions of the attributes that lead to the classification. A typical tree is illustrated in Figure 4.10[Mitchell, 1997b]. The question here is whether one should go out on some kind of activity, such as tennis or surfing, given some information from relevant attributes. This tree says that if the Outlook is sunny and the Humidity is high, then you should not go out. However, if the Outlook is overcast, you should go. The method is quite popular and also easy to extract rules from by traversing the tree, such as in the form of if-then.

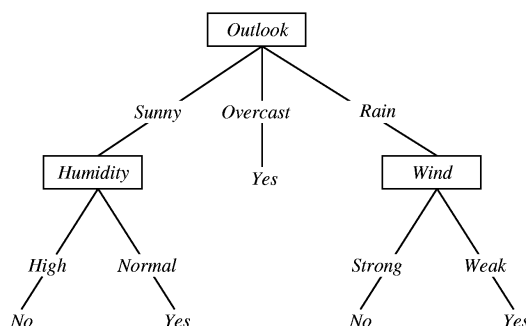


Figure 4.10: Decision Tree

Neural Network is a term that is often referred to as Artificial Neural Network(ANN)[Kröse and Smagt, 1993] in Computer Science. The model is composed of different nodes which mimic the properties of an actual biological neuron. ANNs are often used to model relationships between input and output in a data set, and from there find patterns. There are three main types of nodes in the network, including; input, hidden and output nodes. A node is given one or many inputs, where each input is assigned a weight, including a bias input which is used together with the neurons function to compute the output. The way we train or network is that we feed it with training examples and let it create new nodes and adjust the weights of the inputs. In Figure 4.11¹⁸ you can see a Neural Network with multiple input nodes, some hidden nodes and a output node. All node relationships are weighted.

Bayesian Net is a probabilistic graphical model which represents a set of variables/attributes and their conditional dependencies in a directed acyclic graph. A typical

¹⁸TDT4171 - Lecture 8(2010) Norwegian University of Science and Technology

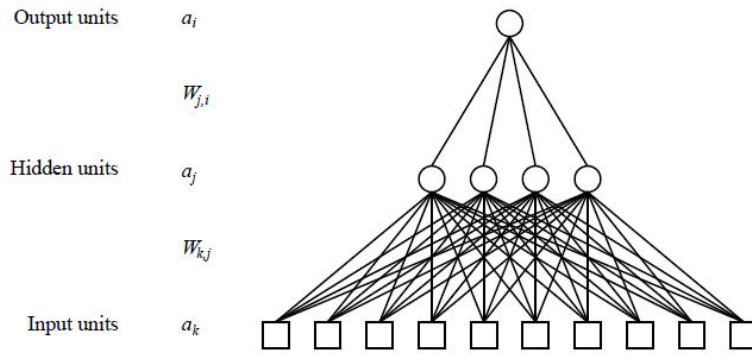


Figure 4.11: Neural Network

example used in theory books and lectures at universities is to model the relationship between diseases and symptoms. Given a list of symptoms, the Bayesian Net should be able to calculate the presence of available diseases. The network in Figure 4.12¹⁹ shows that there are two separate events which can cause the grass to be wet, a sprinkler or rain. By taking advantage of the conditional probability formula, we can calculate the probability of it raining given that the grass is wet. For us this means we can train the net to understand the domain, and then input some test cases and see if it calculates the correct solution.

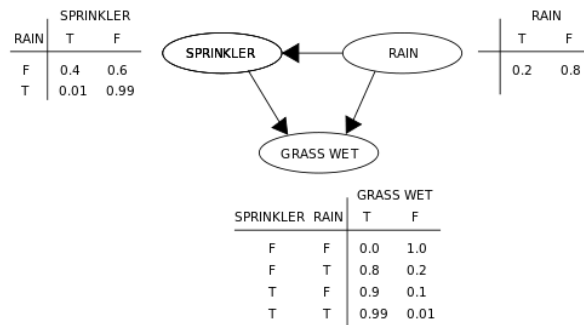


Figure 4.12: Simple Bayes Net

Naive Bayes is strongly related to Bayesian Net. It is based on applying the Bayes' theorem with a naive independence assumption. Basically, the Naive Bayes assumes that the presence or absence of an attribute is unrelated to the presence or absence of another. For example, an animal is considered a bird if it has a beak, feathers and wings. The Naive Bayes classifier assumes that all these features/attributes independently contribute to its classification(a bird), even though this might not always be the case because features may depend on one another. Figure 4.13 illustrates the example given above. Naive Bayes calculate the probability that this is a bird(Yes or No) given the presence of the independent features.

The attributes collected in the specialization project were based on a attribute selection algorithm found in Weka 3.6. The attribute selection was given a set of attributes

¹⁹<http://upload.wikimedia.org/wikipedia/en/0/0e/SimpleBayesNet.svg>

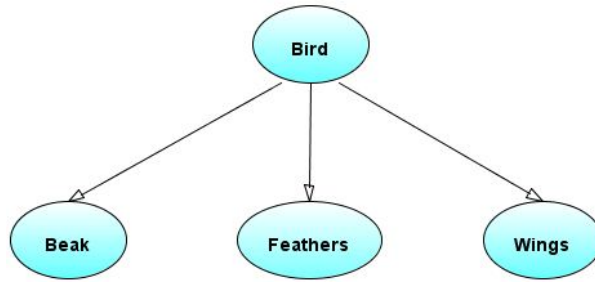


Figure 4.13: Naive Bayes

along with some data, and it outputs the most usable attributes from top to bottom. These attributes were the ones with the strongest influence of the classification of a specific cause. In the end we had a list of the following 6 attributes:

- Cause
- Count
- VaccineType
- DateTime
- SerialNumber
- StartTime

The attributes were based on only a subset of the tables and data in the available data set, but a more detailed selection and analysis would not have been feasible due to the time-constraint. The aim for this part is to see the changes in the data set through the methods of machine learning, and to confirm the assumption we had about improving the performance of the algorithm with a better, more adequate data set. To do this we run each of the four mentioned methods/approaches, note the results and discuss the difference between both the methods in the data set and across data sets.

4.8 Knowledge Acquisition

Knowledge acquisition is as stated in Section 2.3.1 a common issue when creating an expert system. Our approach to knowledge acquisition was:

- a in-depth analysis of the data set at hand
- continuous meetings with the internal group to get the latest update and information that they had acquired
- meetings with experts within the field

The knowledge acquisition is described in detail in Chapter 9.

4.9 Putting it all together

In this section we introduce Glaucus, a Case-Based Reasoning system which retrieves and reuses the solution from cases which are similar to a query case in the fish farming domain. We use the tools and technologies presented in the previous sections and describe the architecture of the system and how we intended to build the cases in our system. This architecture is what we aimed to create from a early stage, and there are some difference between this and the actual architecture. The reason we decided to include the outdated architecture in this thesis is for the reader to see the planning phase of our project, and see what changes it was exposed to.

4.9.1 Initial Architecture for Glaucus

As mentioned above we decided to propose the system architecture illustrated in the component diagram in Figure 4.14 based on the different requirements and technologies presented in the previous sections. The system consists of the core case based reasoning functionality, similarity functions, user interface and database connections.

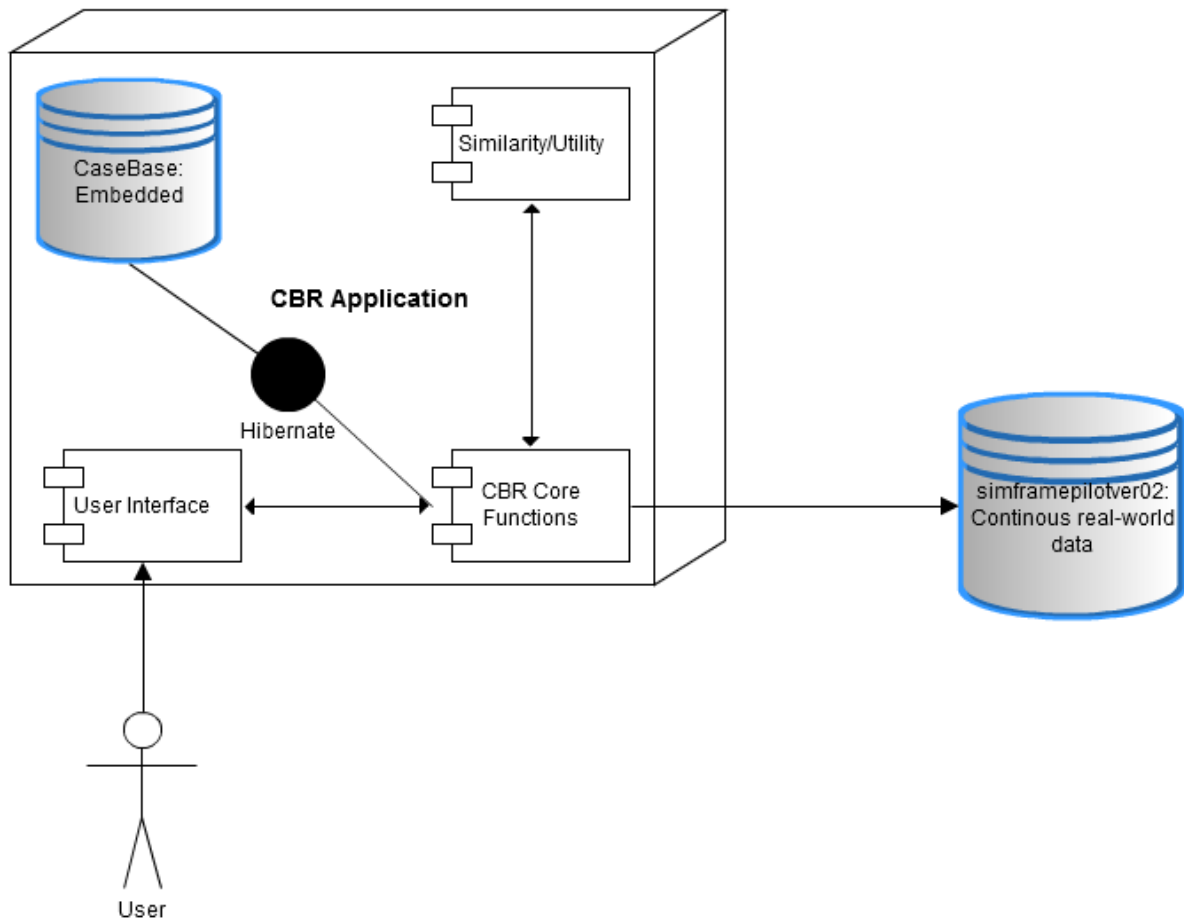


Figure 4.14: Component Diagram

The core case based reasoning is provided by the jColibri framework, and it includes different methods for each of the four phases, Retrieve, Reuse, Revise and Retain. The

similarity functions are both provided by the framework and support for the functions created in myCBR. In order to create a case base we use Hibernate to take our object-oriented data model to a relation model and vice versa. The database server chosen in this application is MySQL, and we use a special Java utility package called MySQL Connector/MXJ to deploy our databases with the application. The user interface is based on that of the TravelRecommender.

A more detailed view is shown in Figure 4.15, which is a basic class diagram. The class diagram shows which components may be needed in the system and the relationship between them. The final system deviates from the initial architecture, but the structure is basically the same.

4.9.2 Case representation and solution

In order to represent the information gathered from the knowledge acquisition, we will use some of the techniques provided by the jColibri framework. A case in our system is represented by a description and a solution. The case description will be as explained in Section 4.2 a Case Component or Compound Attribute, in addition to including several other Compound Attributes. A case in the jColibri framework will have the structure as for instance the following(**bold** are compound attributes):

- **CaseDescription**
 - id
 - date
 - **AquacultureSite**
 - * id
 - * capacity
- **CaseSolution**
 - id
 - risk

The actual case in the system will include many more attributes, but this just illustrates the basic structure of the components which are used in a case. The similarity in the system is composed by several different functions, both represented in the framework and external. In relation to what was said about the four different categories for similarity mechanism, in Section 2.5.3, we will concentrate on using direct mechanisms. For the example case structure listed above we can for instance use a similarity measure like this(functions inside parenthesis):

- **CaseDescription** (Global: Average)
 - id (Local: Equal)
 - date (Local: Date interval)
 - **AquacultureSite** (Global: Average)

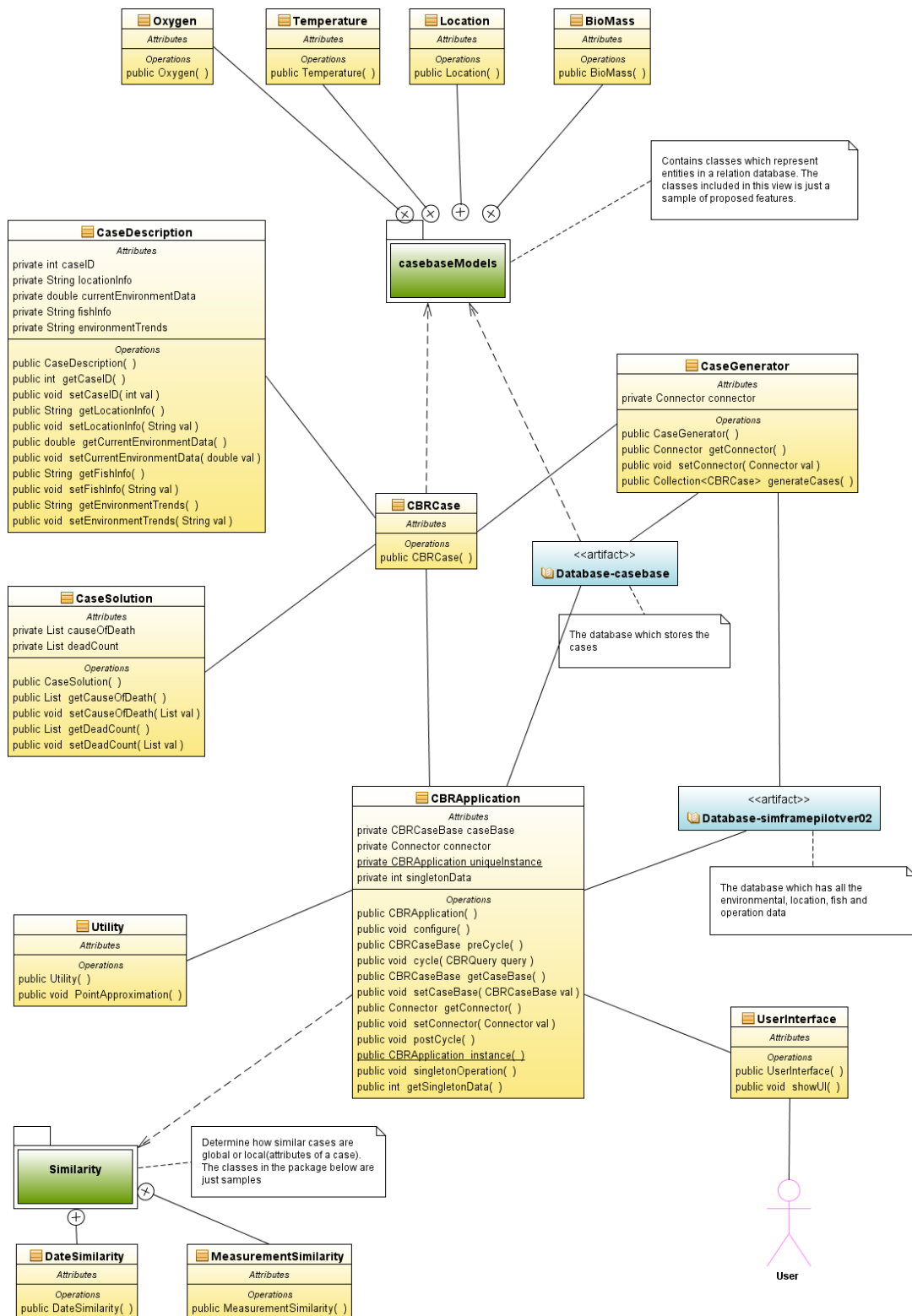


Figure 4.15: Class Diagram

- * id (Local: Equal)
- * capacity (Local: Threshold)

Note that each compound attribute needs a global similarity function.

4.10 Evaluating the final system

When creating a Case-Based Reasoning system, or any other system for that matter, it is important to evaluate and test it. In our project we aimed to use functions and methods provided by the jColibri framework in order to see the strength of the case base and the reasoning of the system.

A more thorough test phase was not feasible due to the time constraint of the project. It is however recommended that a test run with a domain expert is committed in later versions of the system. This will evaluate the utility of the system.

Part II

Utilization

Chapter 5

Work process

The work process throughout a thesis is not always a straight line, and we have therefore chosen to include this chapter for covering the basic milestones and possible bumps along the way. The Tables 5.1, 5.2 and 5.3 include what we consider as the milestones throughout the project, and each one of them is described with references to where in the document they can be found.

Milestone	Description
Literature studies	The first part of the project involved basically around research. The result from these studies can be found in the Chapters 2 and 3
Knowledge acquisition	In addition to studying articles and other written materials and the data set, we gathered information through meetings with the internal group at SINTEF, and experts within the fish farming domain. This is covered in Chapter 9
Prototyping	While doing research, some time was also spent on prototyping to test out some of the technologies like different parts of the jColibri framework and so on
Graphical User Interface	During our prototyping, we implemented the TravelRecomender system provided by jColibri. This application became a template for us when we were going to make our own GUI. A walk through of the system with figures is shown in Chapter 12

Table 5.1: WorkProcess

Milestone	Description
Implementing basic methods	Before we received our new data set, we started to implement some skeletal code, with the basic method that we were going to use from the jColibri framework. This is documented in Section 4.9.
Analyzing the new data	After receiving the new data set an analysis of this was necessary. The result from the analysis can be found in Chapter 6
Evaluating attributes	Through the analysis we got valuable knowledge about the data set, and we used this to evaluate and find attributes that could be included in a case structure. The attribute evaluation can be found in Chapter 7
Machine learning on the new data set	We executed four machine learning algorithms on the data set received in this semester. The work done can be found in Chapter 8
Establishing a case structure	We used the attributes that came as a result from our evaluation to create a case structure. We consulted the attributes with the internal group at SINTEF before settling with them. The case structure and how we created it can be found in Chapter 9 and Section 10.1
Creating the case base	We created a case generator described in Section 10.13 to create all the cases that constitutes the case base
Implementing the similarity functions	With all the attributes in place, we could create the similarity function for each of them. We used both myCBR and built-in functions in the jColibri framework, in addition to customizing our own, Chapter 11

Table 5.2: WorkProcess2

Milestone	Description
Generating the query case	To start the CBR-cycle we need a query case. We have chosen to generate this by using some user input, and some data from the database. How we did this can be found in Section 10.14.
Implementing the case retrieval	With a query case in place, the k most similar cases are retrieved from the case base. How this is done can be found in Chapter 12 Section 12.2.3
Implementing reuse, revise and retain	Even though our focus was on the retrieval part of the system, we have included some minor functionality for the last three RE's as well. The work done can be found in Chapter 12
Evaluating the application	In the end, an evaluation of the system was done. The evaluation can be found in Chapter 13

Table 5.3: WorkProcess3

Chapter 6

Acquiring and analyzing the data

This chapter contains information about what we did to gather information about the domain, and how we carried out the analysis of the data set at hand. We have worked with two different data sets during our project. At the beginning we worked with the data set from our specialization project from last semester, which will be referred to as the old data set from here on. Both data sets contains information on fish farming. The data ranges from daily sensor information like sea temperature and oxygen levels, to logged information like fish groups and their feedings. The data set we received in the middle of this project was quite similar to the old one, but contained more entries and attributes. It was among others, an addition from two to four aquaculture sites.

The first step in our specialization project was to analyze the data and clean it up, so it would be suited for machine learning methods in Weka. This analyzing process was also important to get to know the data and the structure of it. We went through each table in the database to discover missing values and how the data material was distributed throughout the database. This process was repeated also now when we received the new data set. The outcome of this data analysis was a foundation for the attribute selection process, and in the end a case structure. The data set mainly consists of the same tables as the previous data set, but there were also some completely new table additions to the data set. The tables of the old and new database are explained in the following sections. We have chosen to use Weka and its preprocessing tool to help with the analyzing process, like we did in the specialization project.

6.1 Table analysis

Our old date set is found in a database called *simframepilotver01*, and it is the first version of the data foundation we base our system on. A thorough examination of *simframepilotver01* was done in our specialization project autumn 2010. Many of the same tables are still present in the new data set, *simframepilotver02*, but some attributes and entries have also been removed and some have been added. Some of the tables had gotten new attributes, and some also remained the same only with more data in them. So it was also necessary with a new review of the old tables, especially to look for missing values so they will not be taken into account when finding case attributes later on. The tables that we used to find our attributes in our specialization project are shown in Table 6.1, together with a short description.

Table	Description
Startfishgroup	Contains the information about the origin of the fish groups
Causeofdeath	States the deaths occurred with cause and count
Deadcount	Connected to the causeofdeath table and links fish group and date to the deaths
Fishgroup	Connects the startfishgroup information to the existing groups and where they can be found
Aquacultureprodunit	Identifies all the production units with fish and which location it belongs to
Measurementnumerical	Contains sensor data measured at each site
Resulttype	States the different types of measurement types
Feeding	Records of the feeding done in the production units
Observation	When and where an observation is done
Observationtype	Types of observation done
Observationprocedure	How the observation was obtained
Sensor	Contains all the operative sensors
Sensorplatform	Where the sensor is installed with a install date
Referencepoint	Connects observations and measurements to sites and production units
Aquaculturesite	The different sites that produce fish with capacity numbers and owner
Company	The companies that own the sites
Coordinatesystem	One type in the system

Table 6.1: Old Tables

Most of these tables are still present in the new data set, but some of the tables have been completely removed, or empties for data entries, these are listed in the next Section.

6.1.1 Empty tables

The data set presented to us in the database is not complete. This has its natural explanation, as the project of creating a Case-Based Reasoning system is just in its initial phase. The data in place in the database is information derived from handwritten forms and logged sensor data, modeled into the database *past tense of the episodes*. The internal group at SINTEF has developed a web-based registration form, which is going to populate the database automatically *in the future*, to secure the data quality, but this is not available for us yet. The database at hand is developed to suit this registration form to some extent, so for now, it contains empty tables. All the empty tables have been removed to ease the process of getting an overview of the data set and selecting the case attributes.

- sensor
- feedstore
- feedstore_has_feeddelivery
- aquacultureprodunit_has_feedstore
- feedbarge
- feeddelivery
- sensor_has_observation

6.1.2 Tables containing empty columns and missing values

Even though a table is present with data entries inside, it is not always complete. Therefore it is important to identify empty columns and fields inside the tables as well. This has to be taken into consideration while choosing the case attributes, since there is no use for attributes with no data in them.

- deadcount
 - AggregatedCount
- fishgroup
 - StartCount
 - EndCount
 - StartMeanWeight
 - EndMeanWeight
 - UnitOfWeight

- EndTime has some missing values
- aquaculturesite
 - SystemSiteID
 - SiteName
 - UnitOfCapacity
 - LocationLatitude
 - LocationLongitude
- company
 - only one entry is complete
- startfishgroup
 - HatcheryID
 - CompanyGenerationID
- operation
 - NoOfPeople
- aquacultureprodunit
 - CompanyUnitId
 - Circumference
 - Depth
 - Area
 - Volume
 - UnitOfDepth
 - UnitOfArea
 - UnitOfVolume
- feeding
 - FeedStore_idFeedStore
 - UnitOfAmount
- resulttype
 - Definition
- observation
 - CoordinateX
 - CoordinateY

- CoordinateZ
- referencepoint
 - CoordinateX
 - CoordinateY
 - CoordinateZ
 - AquacultureProdUnit_idAquaCultureProdUnit has some missing values
- measurementnumerical
 - Sensor_idSensor
 - CoordinateX
 - CoordinateY
 - CoordinateZ

6.2 New tables

Now we were left with the new tables in our database, and they are described in this section. Some of them have a more detailed explanation than the old tables, since their containing information was new to us. Some of the new tables are only “connection tables”, which means they connects two tables together that have a connection.

6.2.1 Operation

Figure 6.1 illustrates the Operation table. The entries are read as rows, where one row represents an operation executed, with what kind of operation it was together with start and end time of it. idOperation contains the key values which is unique and identifies the specific operation. OperationType_IdOperationType contains key values that identifies which kind of operation type that has been used in the specific operation; a pointer to the operationtype table. It is necessary to apply a filter to get usable information from the Weka preprocessor since the key values are of a numeric type. Weka has several built-in filter features, also one to fix this, called NumericToNominal. This filtering is shown in the Figures 6.2a and 6.2b.

As seen now in the nominal representation of the OperationType_IdOperationType, it is obvious that it is only four of the operation types that are in use, with the majority of the type '2' which corresponds to the operationtype 'Sorting using well-boat' if we look up the key value '2' in the operationtype table. The columns containing the attributes for Start- and EndTime need no further explanation, as they basically contain the start and end time of the operation. The last attribute is NoOfPeople, but this is completely empty.

Result

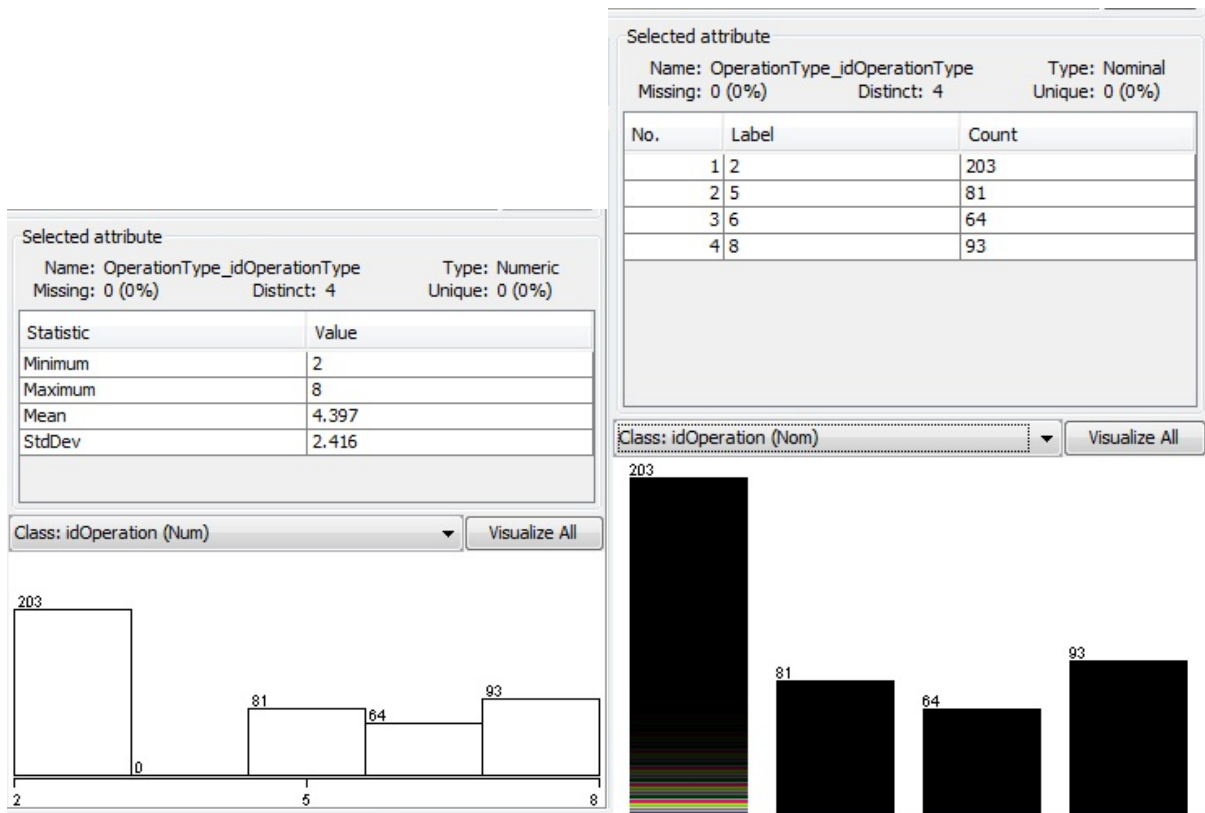
Row	idOperation	OperationType_idOperationType	StartTime	EndTime	NoOfPeople
1	1588	2	2010-07-...	2010-07-...	
2	1589	2	2010-07-...	2010-07-...	
3	1590	2	2010-07-...	2010-07-...	
4	1591	2	2010-07-...	2010-07-...	
5	1592	2	2010-07-...	2010-07-...	
6	1593	2	2010-07-...	2010-07-...	
7	1594	2	2010-07-...	2010-07-...	
8	1595	2	2010-07-...	2010-07-...	
9	1596	2	2010-07-...	2010-07-...	
10	1597	2	2010-07-...	2010-07-...	
11	1598	2	2010-07-...	2010-07-...	
12	1599	2	2010-07-...	2010-07-...	
13	1600	2	2002-09-...	2002-09-...	
14	1601	2	2002-10-...	2002-10-...	
15	1602	2	2002-09-...	2002-09-...	
16	1603	2	2002-09-...	2002-09-...	
17	1604	2	2002-09-...	2002-09-...	
18	1605	2	2003-09-...	2003-09-...	

Query1 Query2 Query3 Query4 Query5

Info

Query: select * from operation
 411 rows selected.

Figure 6.1: Output from the table operation



(a) OperationType_IdOperationType before converting (b) OperationType_IdOperationType after converting

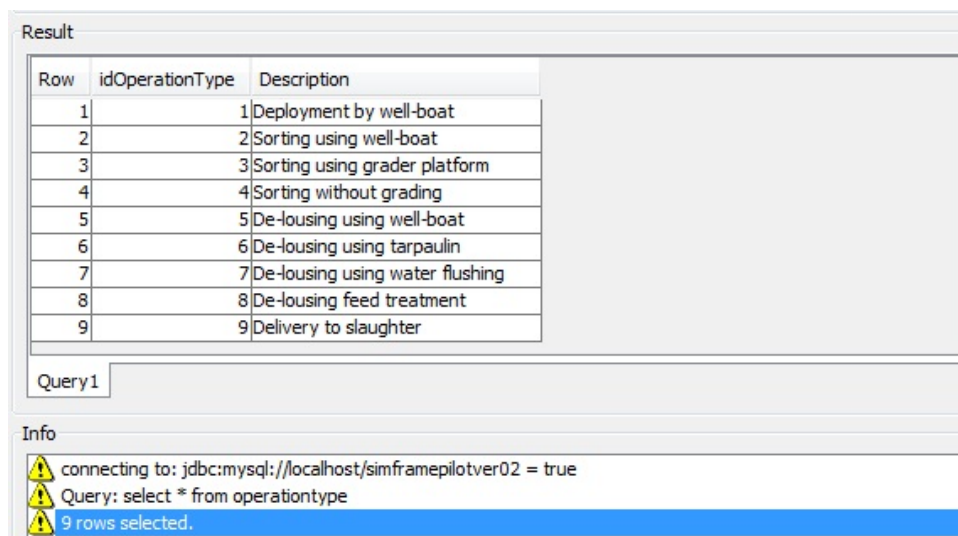
Figure 6.2: Operation Attributes conversion numeric to nominal

6.2.2 OperationType

The table contains the names of the different types of operation that are being used or is going to be used, that are stated in the operation table. The four operation types we found was in use in the operation table can now be named in this table, and they are:

1. Sorting using well-boat(id=2)
2. De-lousing using well-boat(id=5)
3. De-lousing using tarpaulin (id=6)
4. De-lousing feed treatment (id=8)

The rest of the entries in this table can be found in Figure 6.3.



Row	idOperationType	Description
1	1	Deployment by well-boat
2	2	Sorting using well-boat
3	3	Sorting using grader platform
4	4	Sorting without grading
5	5	De-lousing using well-boat
6	6	De-lousing using tarpaulin
7	7	De-lousing using water flushing
8	8	De-lousing feed treatment
9	9	Delivery to slaughter

Query 1

Info

connecting to: jdbc:mysql://localhost/simframepilotver02 = true
Query: select * from operationtype
9 rows selected.

Figure 6.3: Output from the table OperationType

6.2.3 Aquacultureprodunit _Has_ Operation

A connection table that states which aquaculture production unit the operations have been executed on. *Edit: During our case generation we found that this table contained faulty data, and we had to create our own tables for dealing with the sorting operations. More on that in Chapter 9, Section 9.1.1.*

6.2.4 Fishgroupreleation

The table fishgroupreleation contains information on the movement of fish groups when e.g. a sorting has been completed. The columns/attributes in the table can be seen in Figure 6.4. The idSendingFishGroup, and idReceivingFishGroup states the id of the fish groups the fish has moved between. The SendReceiveQuantity is the amount of fish sent, StartSend and EndSend are the time of the movement. The start- and end-time are the same. *Edit: A last attribute Sorting_idSorting has been added by our side later on to*

connect it to the sorting table we have created. This was due to the problems we met using the `Aquacultureprodunit_Has_Operation` to create cases. More about this can be found in Chapter 9, Section 9.1.1.

Query

```
select * from fishgrouprelation
```

Result

Row	idSendingFishGroup	idReceivingFishGroup	SendReceiveQuantity	StartSend	EndReceive
1	2247	2253	30930	2010-07-...	2010-07-05...
2	2247	2255	25902	2010-07-...	2010-07-05...
3	2247	2259	18134	2010-07-...	2010-07-05...
4	2248	2254	75300	2010-07-...	2010-07-05...
5	2248	2256	50856	2010-07-...	2010-07-05...
6	2249	2257	108991	2010-07-...	2010-07-05...
7	2250	2258	91602	2010-07-...	2010-07-05...
8	2251	2252	42578	2010-07-...	2010-07-05...
9	2251	2260	38849	2010-07-...	2010-07-05...
10	2252	2254	30030	2010-07-...	2010-07-05...

Query1

Info

- ⚠ connecting to: jdbc:mysql://localhost/simframepilotver02 = true
- ⚠ Query: select * from fishgrouprelation
- ⚠ 203 rows selected.

Figure 6.4: Fishgrouprelation table output

6.2.5 Fishgroup_Has_StartFishGroup

A connection table that connects fishgroup to startfishgroup

6.2.6 Fishgroup_Has_Vaccine

Yet another connection table which states which vaccines the different fishgroups have received.

6.2.7 Observationprocedure

This table contains different types of observation procedures that are possible to execute. It has two entries:

1. Manual readout
2. Manual Count

This table is connected to the Observationtype table by the Observation_idObservation attribute.

6.2.8 Empty tables

In addition to the tables mentioned there were also some of the newly added tables that were empty. These are listed below.

- Equipment
- EquipmentType
- Equipment_Has_Equipment
- Task
- TaskType
- Task_Has_Equipment
- Task_Has_Observation
- StartFishgroup_Has_Vaccine

6.3 Result

After completing the steps above we were now left with only the tables which contained data. A recap of all the tables present in the database with descriptions can be found in Table 6.2.

6.4 Database model

A database model of these tables can be found in Figures 6.5 and 6.6. Then we examined all these tables to find attribute suggestions to possibly include in the cases. A detailed study of the attributes in these tables can be found in Chapter 7.

Table	Description
Vaccine	The vaccines fish can get
Fishgroup_has_vaccine	A connection table which links which vaccine each fish group has received
Fishgrouprelation	Contains the tracking of the movement of the fish after e.g. a sorting
Operation	The operations executed on a site is kept here
Operationtype	States the names of the operations
Aquacultureprodunit_has_operation	A
Fishgroup_has_startfishgroup	Links the start fish group to the existing fish groups
Startfishgroup	Contains the information about the origin of the fish groups
Causeofdeath	States the deaths occurred with cause and count
Deadcount	Connected to the causeofdeath table and links fish group and date to the deaths
Fishgroup	States the different fish groups with their corresponding companies and in which production unit they can be found
Aquacultureprodunit	Identifies all the production units with fish and which location it belongs to
Measurementnumerical	Contains sensor data measured at each site
Resulttype	States the different types of measurement types
Feeding	Records of the feeding done in the production units
Observation	When and where an observation is done
Observationtype	Types of observation done
Observationprocedure	How the observation was obtained
Sensor	Contains all the operative sensors
Sensorplatform	Where the sensor is installed with a install date
Referencepoint	Connects observations and measurements to sites and production units
Aquaculturesite	The different sites that produce fish with capacity numbers and owner
Company	The companies that own the sites
Coordinatesystem	One type in the system

Table 6.2: All Tables

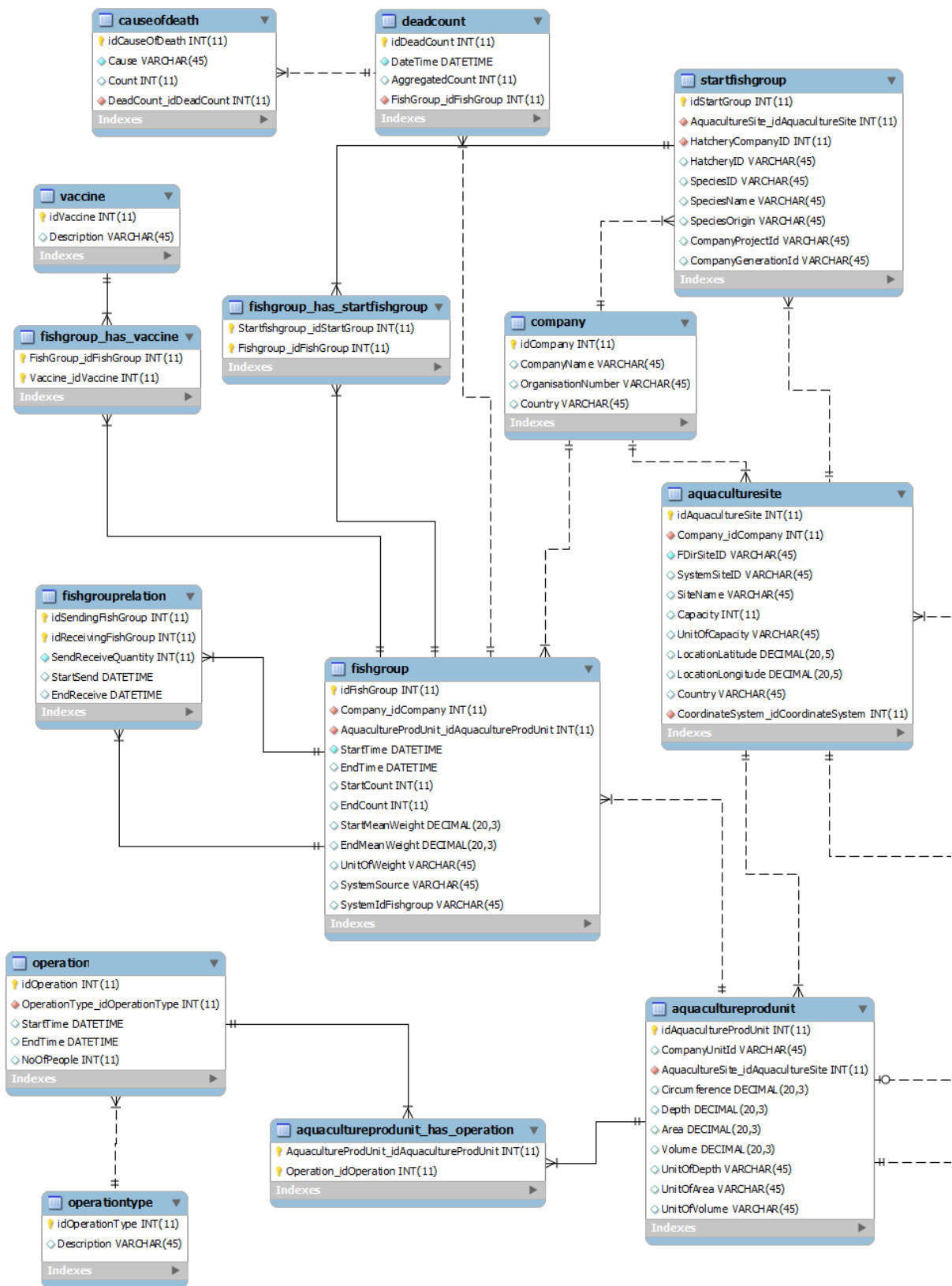


Figure 6.5: Database model part 1

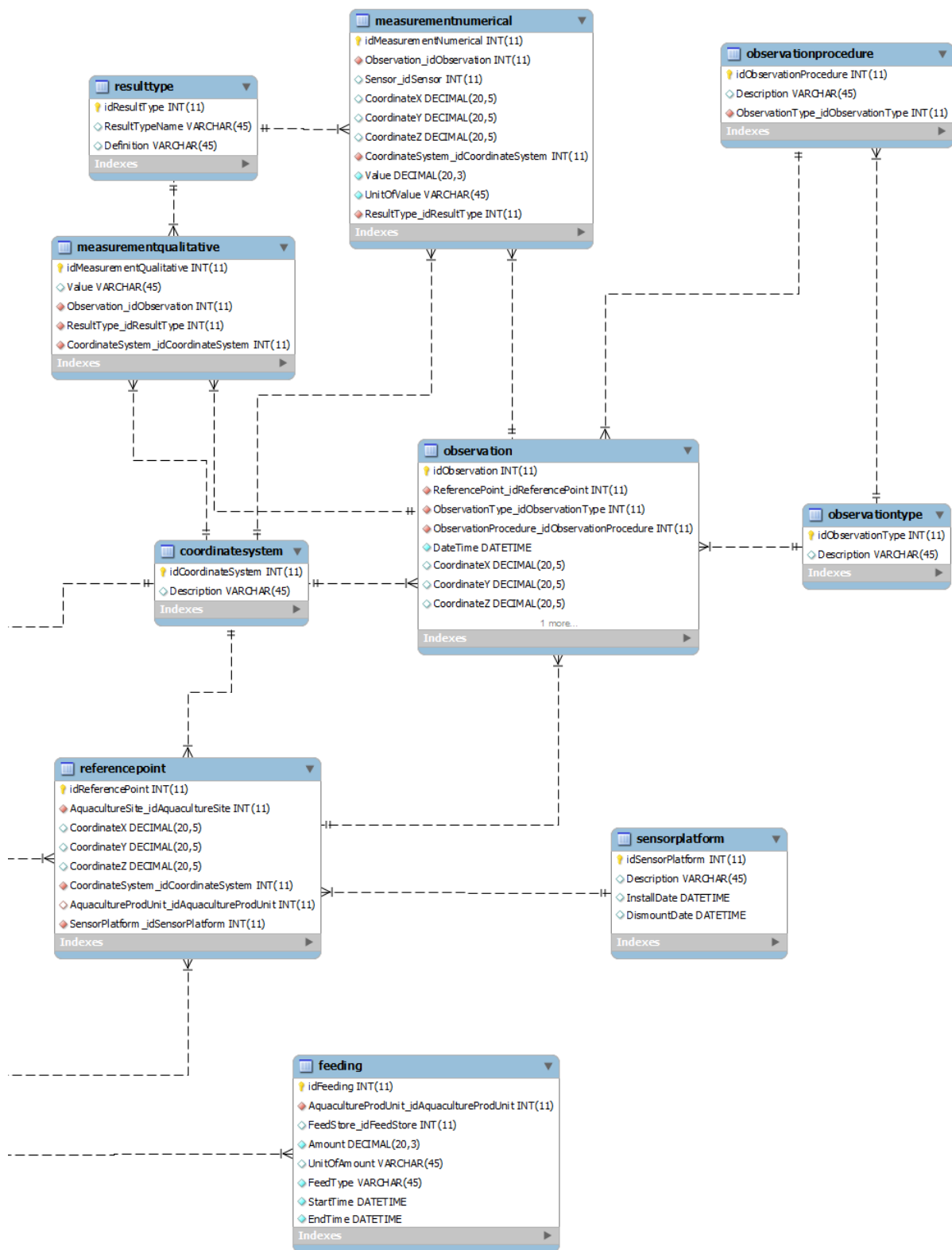


Figure 6.6: Database model part 2

Chapter 7

Attribute selection

After getting an overview of the database through the analysis presented in the previous chapter, we chose to list up all the attributes we regarded as most important. This assessment was done prior to our meetings, concerning the case structure, with the internal group, and it set the foundation for most of the discussion during the meetings. The unique key values that link tables together are also listed in this attribute selection. This was the first step towards setting a final case structure which is described in Chapter 9 and 10.1. Each Section below represents a table from the database, where a table may consist of several attributes and unique keys.

The naming formats for attributes and keys depend on their type, where they vary between normal, identification and key attributes. The normal attribute starts with an uppercase character for each word in the name, such that for instance “start time” will be **StartTime**. The identification attribute identifies each entry in the table and is started by two lowercase characters “id” preceding the table name with the naming convention as for normal attributes. An identification attribute for the table “causeofdeath” will for instance be given by **idCauseOfDeath**. The last type, key attribute, is related to referencing other tables through their identification attributes. The attribute is started with the referenced table name preceding the underscore symbol() and the referenced identification attribute. To reference the table “deadcount” the following notation is used; **DeadCount_idDeadCount**

7.1 Causeofdeath

The table causeofdeath has to do with the different instances of death. Table 7.1 shows the whole table, we see every attribute important for the system. The reference to the table deadcount is necessary in order to do the matching in our queries.

7.2 Deadcount

Table 7.2 shows the attributes which are important in the deadcount table. It links a death occurrence entry in the causeofdeath table to when it happened and in which fish group.

Name	Description
idCauseOfDeath	The id of a specific death occurrence
Cause	The cause of death of the fish
Count	The number of fish that died of a specific cause
DeadCount_idDeadCount	A key value to the table deadcount which says in which group the fish that died came from and the date it happened

Table 7.1: causeofdeath

Name	Description
idDeadCount	The id that is used in causeofdeath
DateTime	The date the fish died
FishGroup_idFishGroup	A key value to the table fishgroup which says in which fish group the death has occurred in

Table 7.2: deadcount

7.3 Fishgroup

The StartTime and EndTime of a fish group are recorded and these can be used further in an application. Which Company each fishgroup belongs to may also prove useful. In addition, Table 7.3 shows a reference to the aquacultureprodunit table which should be used to do query matching. The StartTime attribute states the time a new fish group has been established. This happens after an operation like sorting or that a new fish group has been put in the sea, after being hatched in a hatchery. The EndTime can be both when the fish has been sorted into new production units, or sent of to slaughtering.

Name	Description
idFishgroup	The id of a specific fish group
StartTime	The start time of the fish
EndTime	When the fish group ends
Company_idCompany	A key value to the table company, it says which company the fish belongs to
AquacultureProdUnit_idAquacultureProdUnit	A key value to the aquacultureprodunit table which states the corresponding production unit the fish is/was in

Table 7.3: fishgroup

7.4 Fishgrouprelation

Table 7.4 shows the columns/attributes present in the fishgrouprelation table. This table contains information on the movement of the fish, as new fish groups are created and ended when a fish group is moved from one production unit to another in a sorting.

Name	Description
idSendingFishgroup	The id of the fish group that is being sent
idReceivingFishGroup	The id of the receiving fish group
SendReceiveQuantity	Number of fish being sent
StartSend	When the sending started
End Receive	When the receiving ended

Table 7.4: fishgrouprelation

Edit: The column `Sorting_idSorting` is added to connect this table to the sorting table that we created due to the data set problem explained in Subsection 9.1.1.

7.5 Company

Table 7.5 shows the company table which is referenced in Table 7.3. Only one company is linked to the production sites.

Name	Description
idCompany	The id to the company
CompanyName	The name of the company

Table 7.5: company

7.6 Aquacultureprodunit

Aquacultureprodunit is a table which has a lot of fields, but lacks content. The only useful feature for us will be the reference to aquaculturesite. The description is given in Table 7.6.

Name	Description
idAquacultureProdUnit	The id of the specific production unit
AquacultureSite_idAquacultureSite	A key value to the table aquaculturesite which states the corresponding fish-farming site to the production unit

Table 7.6: aquacultureprodunit

7.7 Feeding

In the feeding table there are several important attributes which we want to use in an application. Feeding is, as indicated by studies[Craig and Helfrich, 2009], quite important and represent approximately 40-50% of the production costs. Table 7.7 shows the attributes we think can be used. There is also a reference to the table aquaculturesite which is used with our queries.

Name	Description
idFeeding	The id of the feeding
AquacultureSite_idAquacultureSite	A key value to the aquaculturesite table which states the corresponding production unit the feeding was done in
Amount	The amount of feed given
FeedType	The type of feed used
StartTime	The start time of the feed
EndTime	The time the feed ended

Table 7.7: feeding

7.8 Aquaculturesite

Name	Description
idAquacultureSite	The id of the site
Capacity	The capacity of the fishfarming site
Company_idCompany	A key value to the table company

Table 7.8: aquaculturesite

The table aquaculturesite contain many attributes with missing values, so not many of these are useful a present time. We have however the Capacity of each site and also a coupling to different companies. This is shown in Table 7.8.

7.9 Aquacultureprodunit_has_operation

aquacultureprodunit_has_operation is just a table which deals with many-to-many cardinality between the tables aquacultureprodunit and operation. There are two attributes which should be used to couple these two tables together. Those are shown in Table 7.9.

Edit: As mentioned earlier, this table contained faulty data, and are not used in the development of the end system. See Section 9.1.1 for details.

Name	Description
AquacultureProdUnit_idAquacultureProdUnit	A key value to the table aquaculture-produnit which states the corresponding unit
Operation_idOperation	A key value to the operation executed

Table 7.9: aquacultureprodunit_has_operation

7.10 Operation

Table 7.10 shows the table we mentioned above, operation. It has three attributes which might be of interest, namely StartTime, EndTime and a reference to which operationType this is.

Name	Description
idOperation	The id of the specific operation
OperationType_idOperationType	A key value to the table operationtype which states which type of operation
StartTime	The time the operation started
EndTime	The time the operation ended

Table 7.10: operation

7.11 Operationtype

OperationType, Table 7.11, is a table for describing the different operations present in the domain. The interesting attribute(s) are shown in the table. The operations conducted in the domain may include operations such as: Deployment by well-boat, Sorting using well-boat and De-lousing using water flushing.

Note that it was at a later stage we narrowed our scope to the Sorting operations, more precisely the Sorting using well-boat.

Name	Description
idOperationType	The id of the type of operation
Description	The name of the operation type

Table 7.11: operationtype

7.12 Fishgroup_has_vaccine

Just as the table `aquacultureprodunit_has_operation` 7.9, `fishgroup_has_vaccine` is a table which connects the two tables `fishgroup` and `vaccine`. The relevant attributes can be found in Table 7.12

Name	Description
<code>FishGroup_idFishGroup</code>	A key value to the table <code>fishgroup</code> which states the corresponding group of fish
<code>Vaccine_idVaccine</code>	A key value to the table <code>vaccine</code>

Table 7.12: `fishgroup_has_vaccine`

7.13 Vaccine

Table 7.13 shows the attributes we think are the most important from `vaccine`, which is the id and name of it.

Name	Description
<code>idVaccine</code>	The id of the vaccine
<code>Description</code>	The name of the vaccine

Table 7.13: `vaccine`

7.14 Fishgroup_has_startfishgroup

The third connecting link between different tables, `fishgroup_has_startfishgroup`, shown in Table 7.14 has just a couple of identification attributes which will be used in queries to map where the fish originated from.

Name	Description
<code>FishGroup_idFishGroup</code>	A key value to the table <code>fishgroup</code> which states the corresponding group of fish
<code>StartFishGroup_idStartGroup</code>	A key value to the table <code>startfishgroup</code>

Table 7.14: `fishgroup_has_startfishgroup`

7.15 Startfishgroup

`Startfishgroup`, Table 7.15, has four attributes which are of interest at current time due to some missing values in other attributes. `SpeciesID` and `SpeciesOrigin` are information about the fish, while the two other are used to couple tables together.

Name	Description
idStartGroup	The id of the startfishh group
AquacultureProdUnit_idAquacultureProdUnit	A key value to the table aquaculture-produnit
HatcheryCompanyID	The same as the Company_idCompany
SpiecesID	The id of the fish spieces
SpiecesOrigin	The origin of the spieces

Table 7.15: startfishgroup

7.16 Referencepoint

Referencepoint has two attributes which seem important in addition to its id. They are described in Table 7.16. The referencepoint connects observations and measurements to production sites and units.

Name	Description
idReferencePoint	The id of the referencepoint
AquacultureSite_idAquacultureSite	A key value to the table aquaculturesite
AquacultureProdUnit_idAquacultureProdUnit	A key value to the table aquaculture-produnit

Table 7.16: referencepoint

7.17 Observation

Table 7.17 displays the attributes which say something about the environment. It states how the observation was done and where, which can be found by following the key values to its table.

Name	Description
idObservation	The id of the observation
ReferencePoint_idReferencePoint	A key value to the table referencepoint
ObservationType_idObservationType	A key value to the table observation-type
ObservationProcedure_idObservationProcedure	A key value to the table observation-procedure
DateTime	The time of when the observation has been done

Table 7.17: observation

7.18 Observationprocedure

Table 7.18 has to do with how an operation is conducted. It has a description and also a reference to the observationtype table.

Name	Description
idObservationProcedure	The id of the observation procedure
Description	The name of the procedure
ObservationType_idObservationType	A key value to the table observation-type

Table 7.18: observationprocedure

7.19 Observationtype

Table 7.19 shows the description of each observation.

Name	Description
idObervationType	The id of the type of observation
Description	The name of the observation

Table 7.19: observationtype

7.20 Measurementnumerical

Measurementnumerical, shown in Table 7.20, has a observation coupled to it along with a value for the given observation. Useful in monitoring the environment, like wind, oxygen levels and temperature.

Name	Description
idMeasurementNumerical	The id of the measurement
Observation_idObservation	A key value to the table observation which states which observation that has been registered
Value	The value of the observation
ResultType_idResultType	A key value to the table that states the type of measurement done

Table 7.20: measurementnumerical

7.21 Resulttype

The Table 7.21 shows the type of result the measurement done is. This is useful when we want to e.g. only find the sea temperature registered in the Table 7.20.

Name	Description
idResultType	The id of the result type
ResultTypeName	States what kind of result type that is used

Table 7.21: resulttype

Chapter 8

Machine Learning Experiments

The Machine Learning part of this project is as stated in Chapter 4 Section 4.7, a way for us to pick up some loose ends with regards to our specialization project autumn 2010 [Garaas and Hiåsen Stevning, 2010]. In this chapter we use our new data set with the same four machine learning approaches, divided into the following sections:

1. Decision Tree: J48(Section 8.1)
2. Neural Network(Section 8.2)
3. Bayes Net(Section 8.3)
4. Naive Bayes(Section 8.4)

More about each algorithm can be found in the Section 4.7.

The different approaches are fed with data from various tables in a database. The data is composed of different attributes such as cause of death, count and vaccine type. The attributes are located in a range of tables, and each attribute's inclusion is based on the the work we did in our specialization project autumn of 2010. Some of the attributes are however changed due to refactoring of the database. VaccineType which was part of the table startfishgroup is now a own table called Vaccine which has attributes idVaccine and Description. Description is the same as startfishgroup's VaccineType. SerialNumber has also been removed from AquacultureProdUnit. In order to fix this issue we used idAquacultureProdUnit which, after consulting the old data from the specialization project, yields the same results. These changes are illustrated in the Table 8.1 below. The changes are *emphasized* and **bold**.

specialization project Attributes	New Attributes
causeofdeath.Cause	causeofdeath.Cause
causeofdeath.Count	causeofdeath.Count
startfishgroup.VaccineType	<i>vaccine.Description</i>
deadcount.DateTime	deadcount.DateTime
aquacultureprodunit.SerialNumber	<i>aquacultureprodunit.idAquacultureProdUnit</i>
fishgroup.StartTime	fishgroup.StartTime

Table 8.1: Attribute Comparison

In order to retrieve the data from the the database the SQL query shown in Listing 8.1 is used.

Listing 8.1: SQL query for data selection

```

1 select  aquacultureprodunit.idAquacultureProdUnit , causeofdeath.Cause ,
        causeofdeath.Count , deadcount.DateTime , fishgroup.StartTime , vaccine .
        Description
2 from    causeofdeath , deadcount , fishgroup , startfishgroup ,
        fishgroup_has_startfishgroup , fishgroup_has_vaccine , vaccine ,
        aquacultureprodunit
3 where
4 causeofdeath.DeadCount_idDeadCount=deadcount.idDeadCount and
5 deadcount.FishGroup_idFishGroup=fishgroup.idFishGroup and
6 fishgroup_has_startfishgroup.FishGroup_idFishGroup=fishgroup.idFishGroup
   and
7 fishgroup_has_startfishgroup.StartFishGroup_idStartGroup=startfishgroup .
   idStartGroup and
8 fishgroup_has_vaccine.FishGroup_idFishGroup=fishgroup.idFishGroup and
9 fishgroup_has_vaccine.Vaccine_idVaccine=vaccine.idVaccine and
10 fishgroup.AquacultureProdUnit_idAquacultureProdUnit=aquacultureprodunit .
    idAquacultureProdUnit ;

```

By using Weka 3.6 to retrieve the data some keywords are worth mentioning, illustrated in Table 8.2.

Keyword	Value
Number of Instances	42905
Number of Attributes	6
Causes of Death	36

Table 8.2: Data Set Information

The following sections show some output from the different machine learning methods along with a comparison between the different methods and the results gained in specialization project in Section 8.5. Each machine learning method is run with regards to classifying the *cause of death*, attribute **Cause**, given the data.

The configurations of each machine learning method are based on earlier work in our specialization project. Each machine learning method has default values for different options and attributes, which means that it is not certain that an optimal solution can be found. It is common to *tweak* the options so that the results improve, and this is unique for each data/problem set. However, since optimizing is not our main objective, we have run the machine learning methods with both the default values and the values found from improving our solution in the specialization project. The results point to that using the adjusted option values will give a better solution, and the results given in the following sections are therefore retained with regards to these.

8.1 Decision Tree: J48

As described earlier in Chapter 4 Section 4.7, Decision Tree learning is one of the simplest machine learning methods, and can be easily ported to if-then statements. A special type of decision tree is the J48 which is based on the very popular C4.5. J48 is just one of many methods included in Weka. Using this machine learning method with the configurations given in Listing 8.2 gives us the results in Listing 8.3. Line 3 in the run configuration, Listing 8.2, names which machine learning method should be run together with some attributes such as C = confidenceFactor(how much pruning should be done) and M = minNumObj(minimum number of instances per leaf node). We use a 10-fold cross-validation to test how good the decision tree is.

Listing 8.2: Decision Tree: J48 Run configuration

```
1  == Run information ==
2
3  Scheme:          weka.classifiers.trees.J48 -C 0.15 -M 4
4  Relation:        QueryResult
5  Instances:       42905
6  Attributes:      6
7                   idAquacultureProdUnit
8                   Cause
9                   Count
10                  DateTime
11                  StartTime
12                  Description
13 Test mode:       10-fold cross-validation
```

The results are quite good, as shown in Listing 8.3. The correctly classified instances is equal to 94.6% which is much more than we expected. The time to build is also quite low which makes this method very good.

Listing 8.3: Decision Tree: J48 Summary

```
1  Number of Leaves :      652
2  Size of the tree :      1293
3  Time taken to build model: 3.75 seconds
4
5  == Stratified cross-validation ==
6  == Summary ==
7
8  Correctly Classified Instances  40581  94.5834 %
9  Incorrectly Classified Instances  2324   5.4166 %
10 Kappa statistic                0.9167
11 Mean absolute error              0.0043
12 Root mean squared error          0.0496
13 Relative absolute error          11.7821 %
14 Root relative squared error      36.8651 %
15 Total Number of Instances       42905
```

8.2 Neural Network

Neural Networks are based on the concept of the human brain. And As described earlier in Chapter 4 Section 4.7, the network is composed of multiple input, hidden and output nodes in a directed graph. Each layer of nodes are connected to the next all the way through the output nodes. Every node in the network, except input nodes, are processing elements which takes some input from the previous layer coupled with adjusted weights, and forwards a value based on its function to the next layer. Such nodes are called neurons. Listing 8.4 shows the configuration for the Neural Network in Weka. The default option values usually seen on line 3 are adjusted such that L = learningRate(the amount the weights are updated), M = momentum(momentum applied to the weights during updating) and H = hiddenLayers(defines the number of hidden layers in the network) are working better with the provided data. A percentage split of 66% is used as testing.

Listing 8.4: Neural Network Run configuration

```
1  == Run information ==
2
3  Scheme:          weka.classifiers.functions.MultilayerPerceptron -L 0.05 -M
   0.05 -N 500 -V 0 -S 0 -E 20 -H 4
4  Relation:       QueryResult
5  Instances:      42905
6  Attributes:     6
7                  idAquacultureProdUnit
8                  Cause
9                  Count
10                 DateTime
11                 StartTime
12                 Description
13 Test mode:      split 66.0% train , remainder test
```

The time it took to build the model was roughly 600 seconds / 10 minutes. That is a very long time for a method which Correctly Classifies 77.5% of the instances, even for a complex algorithm like this. The output is shown in Listing 8.5 below.

Listing 8.5: Neural Network Summary

```
1  Time taken to build model: 599.2 seconds
2
3  == Evaluation on test split ==
4  == Summary ==
5
6  Correctly Classified Instances   11315   77.5638 %
7  Incorrectly Classified Instances  3273   22.4362 %
8  Kappa statistic                  0.6229
9  Mean absolute error              0.0194
10 Root mean squared error          0.1009
11 Relative absolute error          53.3403 %
12 Root relative squared error      74.5114 %
13 Total Number of Instances        14588
```

8.3 Bayes Net

A Bayes Net is a probabilistic graphical model which represents a set of variables and their conditional dependencies via a directed acyclic graph. The method is described in more detail in Chapter 4 Section 4.7. In order to get the Bayes Net to work with Weka we have to run the data through a filter to make all attributes nominal. This is illustrated in Line 4 in Listing 8.6 where an unsupervised filter, Discretize, is used. This filter takes all numeric attributes in the data set and turns it into nominal attributes. The run configuration is shown in Listing 8.6 and on line 3 we find that we are using a SimpleEstimator to calculate the conditional probability table once the net has been created. In addition we find that the search algorithm used is SimulatedAnnealing which finds a well scoring network structure. The method of testing is based on 10-fold cross-validation.

Listing 8.6: Bayes Net Run configuration

```
1  == Run information ==
2
3  Scheme:          weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.
   net.search.local.SimulatedAnnealing -- -A 10.0 -U 10000 -D 0.999 -R 1 -S
   BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator -- -A 0.5
4  Relation:       QueryResult-weka.filters.unsupervised.attribute.Discretize-
   B10-M-1.0-Rfirst-last
5  Instances:      42905
6  Attributes:     6
7                  idAquacultureProdUnit
8                  Cause
9                  Count
10                 DateTime
11                 StartTime
12                 Description
13 Test mode:      10-fold cross-validation
```

The results are shown in Listing 8.7 where the correctly classified instances are roughly 77.7%. The time it took to build the model was 145 seconds which is acceptable.

Listing 8.7: Bayes Net Summary

```
1  Time taken to build model: 145.05 seconds
2
3  == Stratified cross-validation ==
4  == Summary ==
5
6  Correctly Classified Instances   33354   77.7392 %
7  Incorrectly Classified Instances  9551   22.2608 %
8  Kappa statistic                  0.6311
9  Mean absolute error              0.0166
10 Root mean squared error          0.0925
11 Relative absolute error          45.8193 %
12 Root relative squared error      68.7019 %
13 Total Number of Instances        42905
```

8.4 Naive Bayes

A Naive Bayes classifier is a simple probabilistic classifier based on using Bayes' theorem with strong independence assumptions. Naive Bayes assumes that the presence/absence of a feature is unrelated to any other feature in the same scope. The Naive Bayes is run with the default values and the filtered data mentioned in Section 8.3, the Discretize filter. Listing 8.8 shows the configuration where we also use a 10-fold cross-validation to test the method.

Listing 8.8: Naive Bayes Run configuration

```
1 == Run information ==
2
3 Scheme:          weka.classifiers.bayes.NaiveBayes
4 Relation:        QueryResult-weka.filters.unsupervised.attribute.Discretize-
                    B10-M-1.0-Rfirst-last
5 Instances:       42905
6 Attributes:      6
7                  idAquacultureProdUnit
8                  Cause
9                  Count
10                 DateTime
11                 StartTime
12                 Description
13 Test mode:      10-fold cross-validation
```

The results show the lowest Correctly Classified Instances by the selected machine learning methods, which is quite surprising and disappointing. The method correctly classifies approximately 70% of the instances, while the time to model is the fastest of the methods, at 0.03 seconds. This is most likely because the assumption about the attributes being totally independent did not hold. The results are depicted in Listing 8.9.

Listing 8.9: Naive Bayes Summary

```
1 Time taken to build model: 0.03 seconds
2
3 == Stratified cross-validation ==
4 == Summary ==
5
6 Correctly Classified Instances  30067  70.0781 %
7 Incorrectly Classified Instances 12838  29.9219 %
8 Kappa statistic                 0.5734
9 Mean absolute error              0.0212
10 Root mean squared error         0.1104
11 Relative absolute error         58.412 %
12 Root relative squared error     81.9867 %
13 Total Number of Instances      42905
```


8.5 Comparison and Discussion

In this section we look at how the performance of the machine learning methods compare to each other and also across the different data sets used in this project and the specialization project.

8.5.1 Comparing the methods

The most interesting values from the results are shown in Table 8.3. There is one method which really sticks out in regards to correctly classified instances. The Decision Tree: J48 correctly classifies nearly 94.6% of the given instances. The method which is nearest is the Bayes Net, with approximately 77.74%. This is a difference of 17%, which is huge. We should also note that the execution time is quite varying in the different methods. The Neural Network and Bayes Net have nearly the same amount of correctly classified instances, but the time it took to create the model is in favour of the Bayes Net with roughly seven and a half minutes. The Decision Tree and Naive Bayes are the methods which use the shortest time.

Type	Correctly Classified	Time to Build Model
Decision Tree: J48	94.5834%	3.75 seconds
Neural Network	77.5638%	599.2 seconds
Bayes Net	77.7392%	145.05 seconds
Naive Bayes	70.0781%	0.02 seconds

Table 8.3: New Data

8.5.2 Comparing the data sets

In our specialization project we got the machine learning results as shown in Table 8.4. We concluded with that the Decision Tree: J48 was the overall best method due to its classification and time to build, even though the Bayes Net was slightly better with regard to classification.

Type	Correctly Classified	Time to Build Model
Decision Tree: J48	70.1545%	0.28 seconds
Neural Network	65.4093%	1331.06 seconds
Bayes Net	70.853%	766.92 seconds
Naive Bayes	60.5946%	0.03 seconds

Table 8.4: specialization project Data

By combining the two tables, Table 8.3 and Table 8.4, we can easily see the difference between the machine learning methods on the different data sets in Table 8.5. We see an increase of about 24, 12, 7 and 10 percent to Decision Tree, Neural Network, Bayes Net and Naive Bayes respectively. The time to build the model also went down on all of the methods except for the Decision Tree which increased with roughly three and a half

seconds. Our conclusion is that the data is more complete in this batch, something that also is shown in the amount of instances increasing from 6862 to 42905.

	specialization project Data		New Data	
	% correct	timeToModel	% correct	timeToModel
Decision Tree: J48	70.1545%	0.28 seconds	94.5834%	3.75 seconds
Neural Network	65.4093%	1331.06 seconds	77.5638%	599.2 seconds
Bayes Net	70.853%	766.92 seconds	77.7392%	145.05 seconds
Naive Bayes	60.5946%	0.03 seconds	70.0781%	0.02 seconds

Table 8.5: Data Set Comparison

Weka gives us the ability to study the result even more thoroughly by using various graphical representations of data. We have chosen to illustrate the threshold curve, like we did in our specialization project as well. The minimum probability required to classify an instance is called the threshold value. For each threshold value Weka calculates the several performance values, e.g. True Positives, False Negatives, False Positive Rate and True Positive Rate. Each of the performance variables can be used for the x- or y-axis. We have chosen to use the False Positive Rate as x and true Positive Rate as y. The color of the graph is the threshold value. This is a so called ROC curve[Fawcett, 2004] which is a good technique for visualizing and selecting classifiers based on performance.

To compare classifiers we may want to reduce ROC performance to a single scalar value representing expected performance. A common method is to calculate the area under the ROC curve, abbreviated AUC [Hanley and McNeil, 1982].

-[Fawcett, 2004]

The AUC of a classifier corresponds to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. The AUC is shown at the top of the graph; a higher number here indicates better performance as the classifier is able to get high true positive rates with low false positive rates quicker. We have illustrated the ROC curves for cause “Tapersyndrom” and the J48 algorithm now and in the specialization project in Figure 8.1 and 8.2. The “Tapersyndrom” cause was chosen because it was the cause with the most instances disregarding the “unknown” cause such as Unspecified, Unknown, Andre Dødsårsaker. And as for the classification correctness values earlier, the improved performance of the J48 algorithm on the new data set is obvious.

This machine learning experiment has shown that the data set provided in this project is stronger than the one in our specialization project. By conducting this experiments we gained insight into how the data set is composed and gave us a good foundation for further work with case structuring.

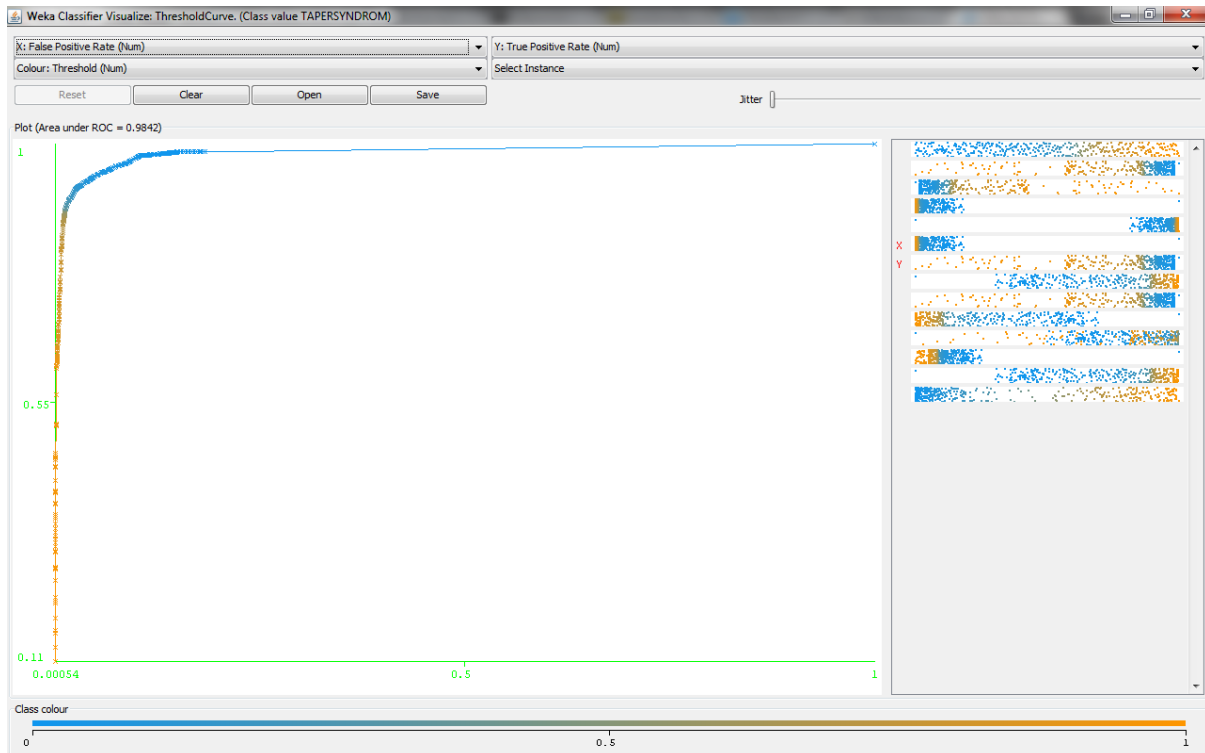


Figure 8.1: Threshold Curve J48 for TAPERSYNDROM of the new data

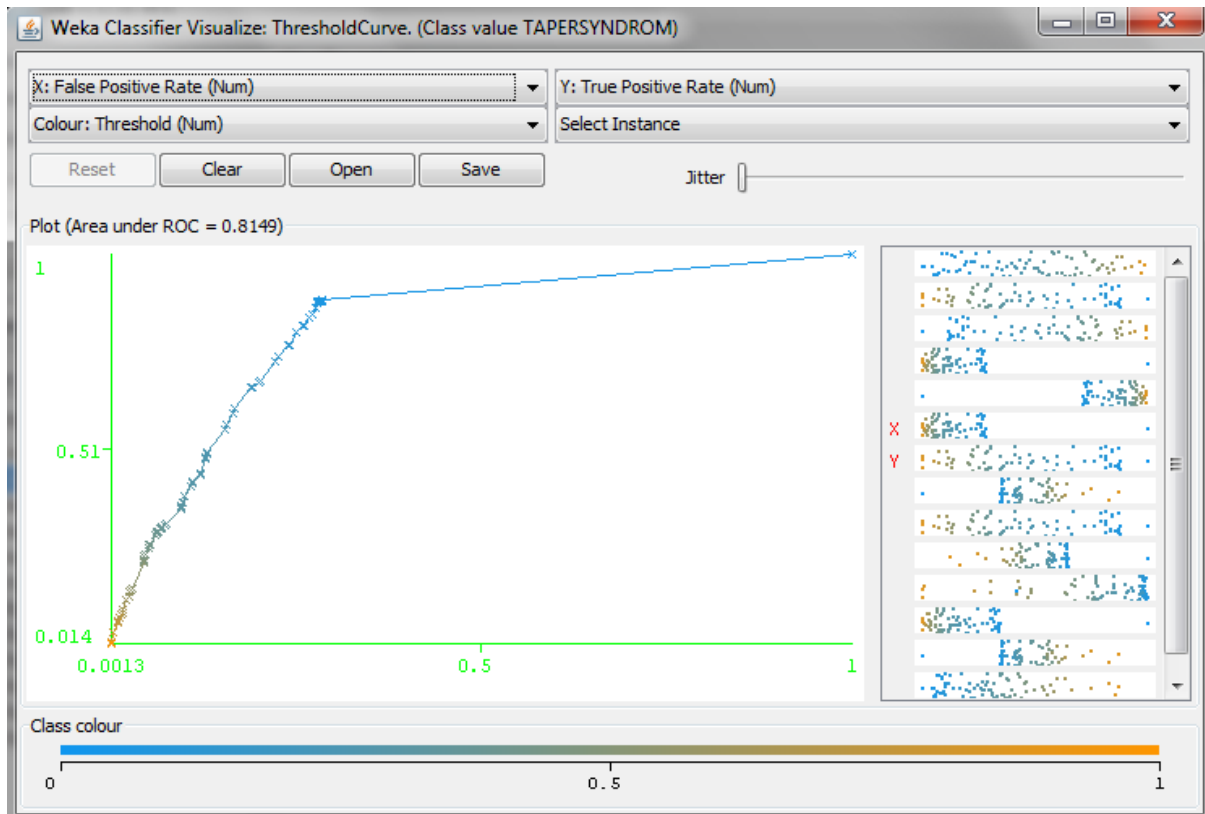


Figure 8.2: Threshold Curve J48 for TAPERSYNDROM of the specialization project data

Chapter 9

Creating a case structure

In this chapter we explain the steps taken to create a case structure based on the work done in the previous chapters and other factors like information gained through different meetings which are described in this chapter. One of our main focuses has been the establishment of a rich but yet simple case structure. It is a fine line between too many and too few attributes present in a case, and also how the attributes present are represented. In Section 2.5.2 we went through the structure of a case, and the numerous forms it can have. Our goal was to create a case structure which would cover the different features that a fish farmer would take into consideration when deciding to do an operation. On the quest to find the final structure we went through several meetings with different groups, that had different viewpoints.

9.1 Knowledge Acquisition

Knowledge acquisition is important while designing a system, as pointed out earlier in Section 2.3.1, especially when it comes to understanding why and how the system is needed. The first meeting concerning the case structure was with the internal group at SINTEF Fisheries and Aquaculture. At this meeting it was decided that our focus should be at the sorting operations, as there was some talk in the beginning of including more operations. The internal group would also concentrate their work around the same operation type. The new restriction of the domain further specialized our assignment, and it helped us get a better grasp on how the development was going proceed.

The following meeting was with Arnfinn Aunsmo, Manager Biology and Nutrition at SalMar. As preparation to the meeting we had gone through our new data material and drawn up a list of possible attributes which would affect a operation. These attributes can be found in Chapter 7. The meeting was hands-on experience with knowledge acquisition, when people from different backgrounds meet to discuss a common subject. One of the mistakes we made was to assume that everyone has insight into the data set we were given, but this was not the case. We also learned that as a group, the fish farmers would find it difficult to believe a decision support system which only gave previous data as output. The predictions about the future was also of concern for the same reasons and we were advised to output a risk associated with the sorting operation from the system. A combination of human assessment with the data would also be appropriate. With these two meetings in mind we worked out a case structure with the following main attributes:

- Operation Time
- Oxygen level
- Temperature
- Capacity
- Fish group
- Vaccine
- Species Origin
- Feeding
- Biomass
- Cause
- Dead Count
- Risk

The middle of April 2011 marked the last touches on the case structure and some last minute input. The case structure was made final on a workshop with the internal group at SINTEF, where some additional attributes were added. They pointed out that the start-fish group information was important to maintain, and that it would be interesting to involve starving time before an sorting. The starving is being done so that the equipment like the fishing nets and so on will be littered with excrements from the fish as little as possible. We also had a discussion regarding the biomass numbers registered in the data set, which were not measurements, but really calculated using the amount of feed given the fish and temperature in the water. The biomass was an attribute which we wanted to include if there was sufficient development time and knowledge. Since this was not the case we opted to not include it in this project. We also decided to track deaths up to two weeks after a sorting has been carried out. The same time line was used when monitoring environmental changes before the operation, to get temperature and oxygen trends.

Some other aspects of the process of retrieving knowledge was that it worked like a type of evaluation of the data set. If we could find any issues with the data set then that would contribute much to our project, and also for the internal group which were in the startup phase of their project. What we found when we started to work on the implementation is that there were some issues with the tables concerning the operations in the system. Section 9.1.1 describes the issue, and also how we “solved” it in our system.

In our system we also assumed that an aquaculture production unit could receive fish from multiple production units, but not the other way round. The reason for this was that in working with a many-to-many relation between receiving and sending production units, there would be too much work given our development time in the project. More about this is covered in Section 9.1.2.

9.1.1 Inconsistencies in the data set

When we were working with writing SQL-statements to create cases from our data, we discovered some inconsistencies within the database. This led us to change the structure of how the cases were going to be built. An explanation will now be given of how we discovered the problem. The problem was due to operations not being correctly connected to the production unit it had been executed on. First we chose to look at operations for a given date. We chose the date 2010-07-05, and all the operations done on this date are marked with green in Figure 9.1

idOperation	OperationType_idOperationType	StartTime	EndTime	NoOfPeople
1588	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1589	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1590	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1591	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1592	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1593	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1594	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1595	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1596	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1597	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1598	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1599	2	2010-07-05 00:00:00	2010-07-05 00:00:00	
1600	2	2002-09-26 00:00:00	2002-09-26 00:00:00	
1601	2	2002-10-12 00:00:00	2002-10-12 00:00:00	
1602	2	2002-09-30 00:00:00	2002-09-30 00:00:00	
1603	2	2002-09-29 00:00:00	2002-09-29 00:00:00	
1604	2	2002-09-30 00:00:00	2002-09-30 00:00:00	
1605	2	2003-09-11 00:00:00	2003-09-11 00:00:00	
1606	2	2003-09-11 00:00:00	2003-09-11 00:00:00	
1607	2	2003-09-11 00:00:00	2003-09-11 00:00:00	

Figure 9.1: operation table

Second we used the identification of the operations to find which production unit it was registered to, this is shown in Figure 9.2, also marked with green. It was here that we discovered that something may be wrong. We thought it was strange that only one production unit had played a part in all these sorting operations.

Our third step was to investigate the fishgroup table, a table which states where the different fish group are located(production unit) and their start- and end time. We looked for the production units marked as green in Figure 9.2, unit 1414. There were two instances of these in the table. We also registered that a lot of other fish groups also had start- or end-times on the same date as we were looking at, marked in red. This is illustrated in Figure 9.3.

To investigate this further, we looked at the table fishgrouprelation seen in Figure 9.4, to see if there had been more fish group movements this day. The two fish groups that are registered at the production unit 1414 are marked with green. But as you can see there were also here signs of more sorting operation being done the same day, and to get an overview we created a kind of flow diagram, seen in Figure 9.5, to see were the fish moved to and from. The ovals represents the different fish groups, the small rectangles

AquacultureProdUnit_idAquacultureProdUnit	Operation_idOperation
1414	1588
1414	1589
1414	1590
1414	1591
1414	1592
1414	1593
1414	1594
1414	1595
1414	1596
1414	1597
1414	1598
1414	1599
1432	1600
1432	1601
1432	1602
1432	1603
1432	1604
1432	1605
1432	1606

Figure 9.2: The table aquaproductionunit_has_operation

idFishGroup	Company_idCompany	AquacultureProdUnit_idAquacultureProdUnit	StartTime	EndTime
2247	1	1404	2009-10-27 00:00:00	2010-07-05 00:00:00
2248	1	1405	2009-10-21 00:00:00	2010-07-05 00:00:00
2249	1	1406	2009-10-24 00:00:00	2010-07-05 00:00:00
2250	1	1407	2009-10-24 00:00:00	2010-07-05 00:00:00
2251	1	1408	2009-11-06 00:00:00	2010-07-05 00:00:00
2252	1	1409	2010-07-05 00:00:00	
2253	1	1410	2010-07-05 00:00:00	2010-07-05 00:00:00
2254	1	1410	2010-07-05 00:00:00	
2255	1	1411	2010-07-05 00:00:00	2010-07-05 00:00:00
2256	1	1411	2010-07-05 00:00:00	
2257	1	1412	2010-07-05 00:00:00	
2258	1	1413	2010-07-05 00:00:00	
2259	1	1414	2010-07-05 00:00:00	2010-07-05 00:00:00
2260	1	1414	2010-07-05 00:00:00	
2261	1	1415	2002-09-22 00:00:00	2003-09-11 00:00:00
2262	1	1416	2002-09-22 00:00:00	2003-09-02 00:00:00
2263	1	1417	2002-09-26 00:00:00	2002-09-26 00:00:00
2264	1	1417	2002-09-26 00:00:00	2003-09-01 00:00:00
2265	1	1418	2002-10-11 00:00:00	
2266	1	1419	2002-10-12 00:00:00	2002-10-12 00:00:00

Figure 9.3: fishgroup table

are their production unit and the arrows point in the directions of where the fish groups moved. The green ovals are the fish groups that resulted in the sort, e.g. where the fish groups ended up after the sorting of the day were completed.

idSendingFishGroup	idReceivingFishGroup	SendReceiveQuantity	StartSend	EndReceive
2247	2253	30930	2010-07-05 00:00:00	2010-07-05 00:00:00
2247	2255	25902	2010-07-05 00:00:00	2010-07-05 00:00:00
2247	2259	18134	2010-07-05 00:00:00	2010-07-05 00:00:00
2248	2254	75300	2010-07-05 00:00:00	2010-07-05 00:00:00
2248	2256	50856	2010-07-05 00:00:00	2010-07-05 00:00:00
2249	2257	108991	2010-07-05 00:00:00	2010-07-05 00:00:00
2250	2258	91602	2010-07-05 00:00:00	2010-07-05 00:00:00
2251	2252	42578	2010-07-05 00:00:00	2010-07-05 00:00:00
2251	2260	38849	2010-07-05 00:00:00	2010-07-05 00:00:00
2253	2254	30930	2010-07-05 00:00:00	2010-07-05 00:00:00
2255	2256	25902	2010-07-05 00:00:00	2010-07-05 00:00:00
2259	2260	18134	2010-07-05 00:00:00	2010-07-05 00:00:00
2261	2273	54691	2003-09-11 00:00:00	2003-09-11 00:00:00
2261	2275	64148	2003-09-11 00:00:00	2003-09-11 00:00:00
2261	2277	47387	2003-09-11 00:00:00	2003-09-11 00:00:00
2262	2279	63693	2003-09-02 00:00:00	2003-09-02 00:00:00

Figure 9.4: fishgrouprelation table

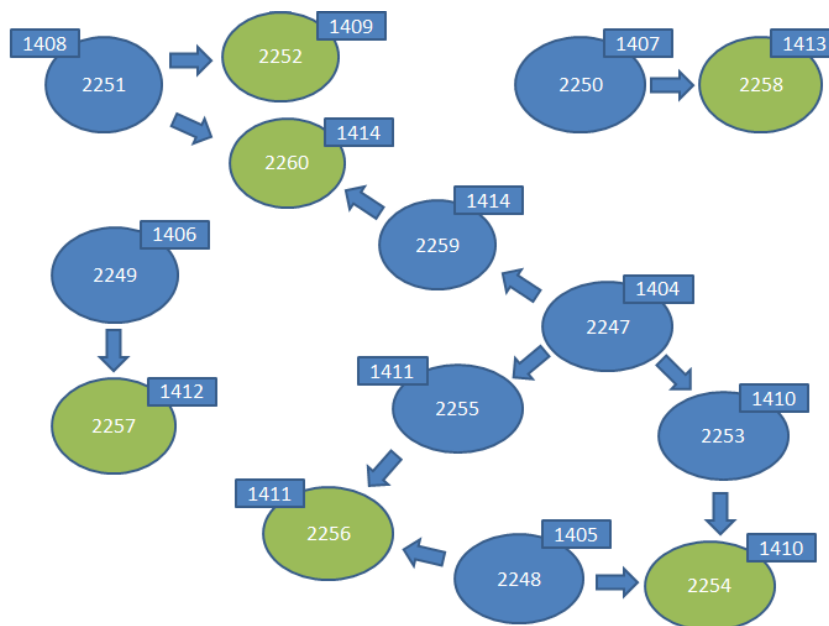


Figure 9.5: Fishgroup movement

It was now obvious that something did not add up. There were both more fish groups and production units included in the sorting than was stated in the production_unit_has_operation table shown in Figure 9.2. The additional fish groups and their production units are marked of in red in the Figure 9.3 and the relationship between the fish groups is also marked with red in the Figure 9.4.

The findings were discussed in detail at an emergency meeting close to Easter 2011. It confirmed our beliefs about something being wrong in the database, more accurately some issues regarding the parsing of the raw data from XML to the database. Fortunately for us our assignment focused on operations connected to sorting, and this data can be generated from the fishgrouprelation table, which is shown in Figure 9.4.

Generating the Sorting table

The quick and dirty solution for us was to create a table called Sorting. The Sorting table was generated with the following three fields:

- idSorting
- DateTime
- AquacultureSite_idAquacultureSite (foreign key to AquacultureSite)

In order to populate the table we used the SQL-query in Listing 9.1. What this query does is that it finds each distinct Sorting operation and insert some of the attributes into the Sorting table.

Listing 9.1: Inserting Sorting data

```

1 insert into simframepilotver02.sorting (DateTime,
   AquacultureSite_idAquacultureSite)
2 select distinct StartSend, AquacultureSite_idAquacultureSite
3 from simframepilotver02.fishgrouprelation, simframepilotver02.fishgroup,
   simframepilotver02.aquacultureprodunit
4 where idSendingFishGroup=idFishGroup and
5 AquacultureProdUnit_idAquacultureProdUnit=idAquacultureProdUnit

```

An extra field(column) was also inserted into fishgrouprelation in order to link each instance or sub sort to the main sorting operation. The population of this field was actually done manually due to some trouble with the creation of the SQL-query. The time it took to do this was about fifteen minutes, so not much loss of time.

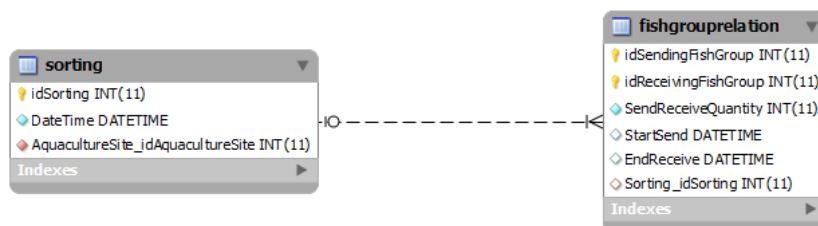


Figure 9.6: Sorting and fishgrouprelation

9.1.2 Fish group relationship

A sorting operation is an operation which is done on multiple production units, many receiving and many sending. When we started the work on this project our aim was to include both many receiving and many sending production units in the final system.

After much evaluation we came to the conclusion that we should concentrate on a one-to-many relationship in our work. This was due to the share amount of data which is distributed among the different production units and the complexity of their relationship with respect to the fish groups. Figure 9.7 gives an abstract view of the relationship between the fish groups in the system.

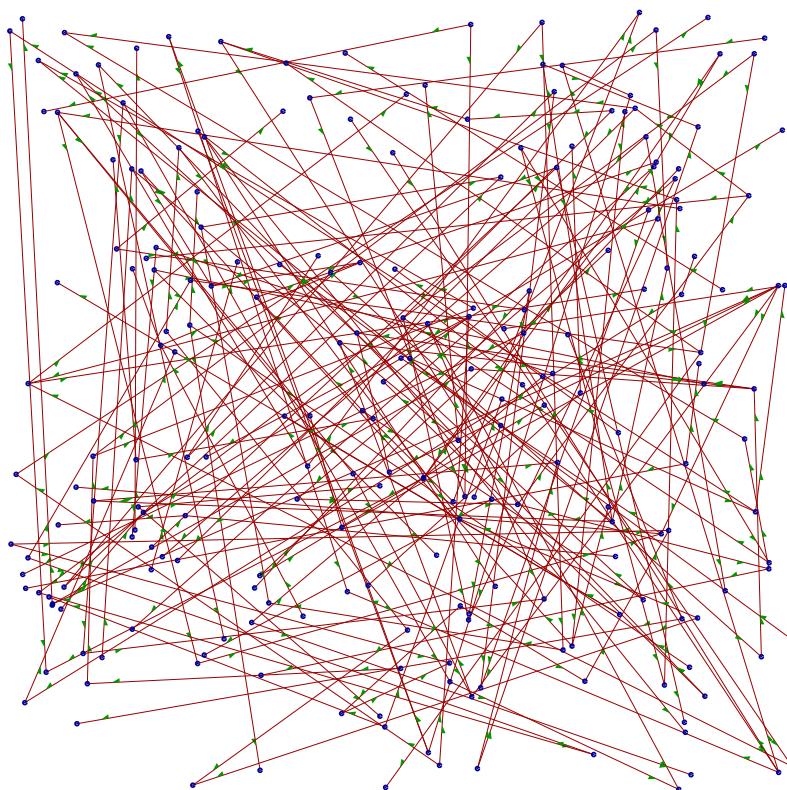


Figure 9.7: The relationship between the different fishgroups

The small blue circles represent a fish group while the red arcs represent a relationship to another fish group. The green arrow indicates which of the fish groups are sending and which are receiving.

The second reason for the proposal of using the one-to-many relationship is tightly coupled with the first point. When using the one-to-many relationship we are able to concentrate on handling similarity easier and make the data model less complex. This would also mean that we would get the time to actually implement a system given our

time schedule.

9.2 Field Trip

To get a better idea of what all of our data and attributes really meant, we got the opportunity to visit one of SalMar’s fish farming sites, at Tristein outside Vallersund, in early spring. Figure 9.8 is from when we were shown around the facility.



Figure 9.8: Field trip to Tristein

We got a nice tour where the everyday routines were explained to us. One of the most interesting things we got to see was the feeding mechanism. The feed was kept in containers below deck, and it was distributed from here into long pipes and out to the different production units in the sea. The site had at the moment 10 production units, distributed as a matrix of 2x5, and each had its own feeding routine. A fun observation we made was that there seemed to be a couple of cut off toilet brushes near the feeding tubes. The reason for this was that when feeding the fish through the feeding pipes it would sometimes clog due to the amount of feed and dust that over time stuck to the sides of the pipe. If this happened the usually used a specialized form made from expanded polyester and plastic to scrape the inside of the cable by applying pressure in one end and push the form through. What they actually experienced was that using the toilet brushes provided much better results! This was one of many fun facts we were treated with from the personnel at Tristein.

They also showed us how they monitor the daily life of the fish by sensor data and real-time video surveillance on computer screens indoor, see Figure 9.9.

The technical aspect of the work the fish farmers do was quite surprising to us, and we did not expect to find one of them sitting in front of multiple computer screens. The trip was very informative and helped us understand the system we were going to build



Figure 9.9: Fish farmer at work at Tristein

much better. It was especially helpful when it came to the choosing of the final attributes for the case structure.

9.3 The final case structure

After all these different inputs and inspiration we felt suited to define our case structure. Our description case, or query case, became as shown below:

1. case id
2. operation date
3. site
4. sending production unit
5. receiving production unit
6. current temperature
7. temperature trend
8. current oxygen level
9. oxygen trend
10. hatchery
11. species origin

12. vaccine
13. site capacity
14. fish count in unit
15. starving days

The solution case, or the output of the system became

1. risk
2. cause
3. count

Now we had our attributes and we needed to create a structure we could use with jColibri. Compound attributes are as indicated in Section 4.2 attributes which contain numerous simple attributes. In our attribute list we saw that there were instances where we could utilize this technique. The list below is a rough sketch of what we intended the case structure to look like before implementing it. The compound attribute are marked with **bold** while those marked with *italic* are lists.

- **Case Description**

- caseID
- dateTime
- **AquacultureSite**
 - * siteId
 - * capacity
- **receivingProdUnit**
 - * id
 - * *speciesOrigin*
 - * *hatcheries*
 - * *Vaccines*
 - * receiveAmount
- **sendingProdUnits**
 - * id
 - * individCount
 - * startvationDays
- **Temperature**
 - * **currentInstance**
 - value
 - date
 - * trend

- * *pastInstances*
- **Oxygen**
 - * **currentInstance**
 - value
 - date
 - * trend
 - * *pastInstances*
- **Case Solution**
 - caseID
 - Risk
 - *fishDeath*
 - percentage dead
 - aggregated count

With the attributes set we were now ready to implement them in a case structure by using the jColibri framework and start to generate our case base, and to design our systems architecture.

Chapter 10

Architecture and case base

In this chapter we look at the final architecture of Glaucus and the case base. The architecture is essentially the same as the one we proposed at the start of the analysis of the technologies and approaches, Chapter 4. One of the changes made to the architecture is the use of multiple databases on different locations. Our initial plan was to have a embedded database in Glaucus which would serve as the case base, while another database outside the application would serve as the database for continuous data flow to our application. This is illustrated in Figure 4.14 in Chapter 4, Section 4.9. What we ended up doing was to create two databases, one for the case base and one for the data, in a embedded database server. The reason we did this was to be able to deploy the application without having the user use their time with setting up their own database server outside the application. In a commercial system we would however divide the two as in the proposed architecture. The implementation for the embedded database server is described in Chapter 12.

In our proposed architecture we did not know which data models or similarity functions we needed in the system. The Case Description, Solution and data models we put aside to its own chapter in Chapter 10.1, and in Figure 10.2 they are marked as green packages named models and casesdescriptions. The similarity functions in Glaucus are described in detail in Chapter 11, where we give a detailed explanation of the each of the functions used ranging from jColibri's built-in functions, myCBR similarity functions, in addition to our own custom made ones. The ones that we made our selfs are illustrated in Figure 10.1 and shown as a green package in Figure 10.2.

The rest of the classes/components in Figure 10.2 such as dialogs and CBRApplication are discussed in more detail in Chapter 12.

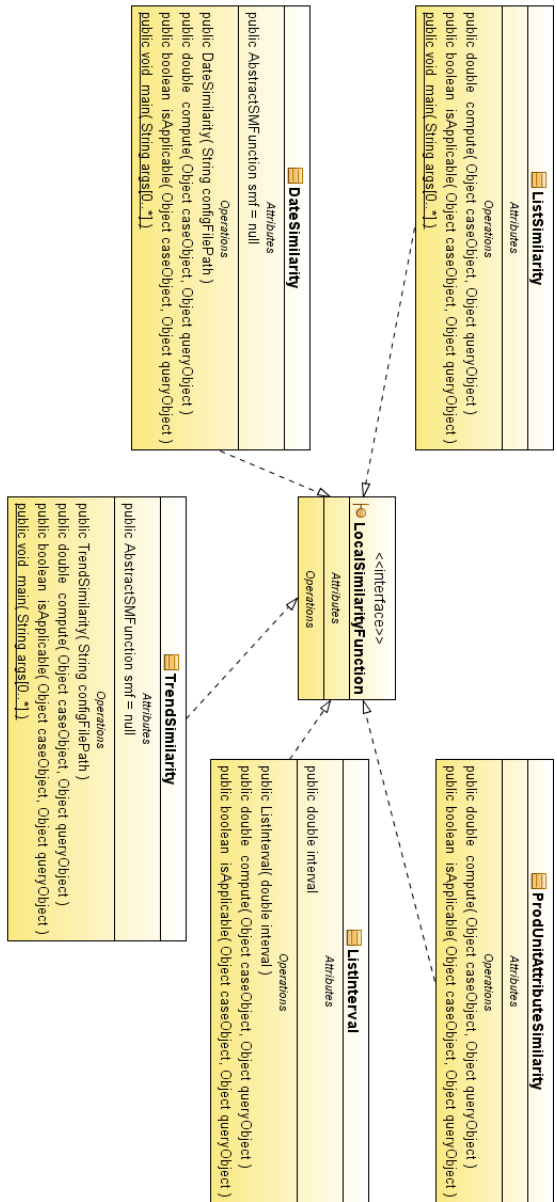


Figure 10.1: Java classes for generated similarity

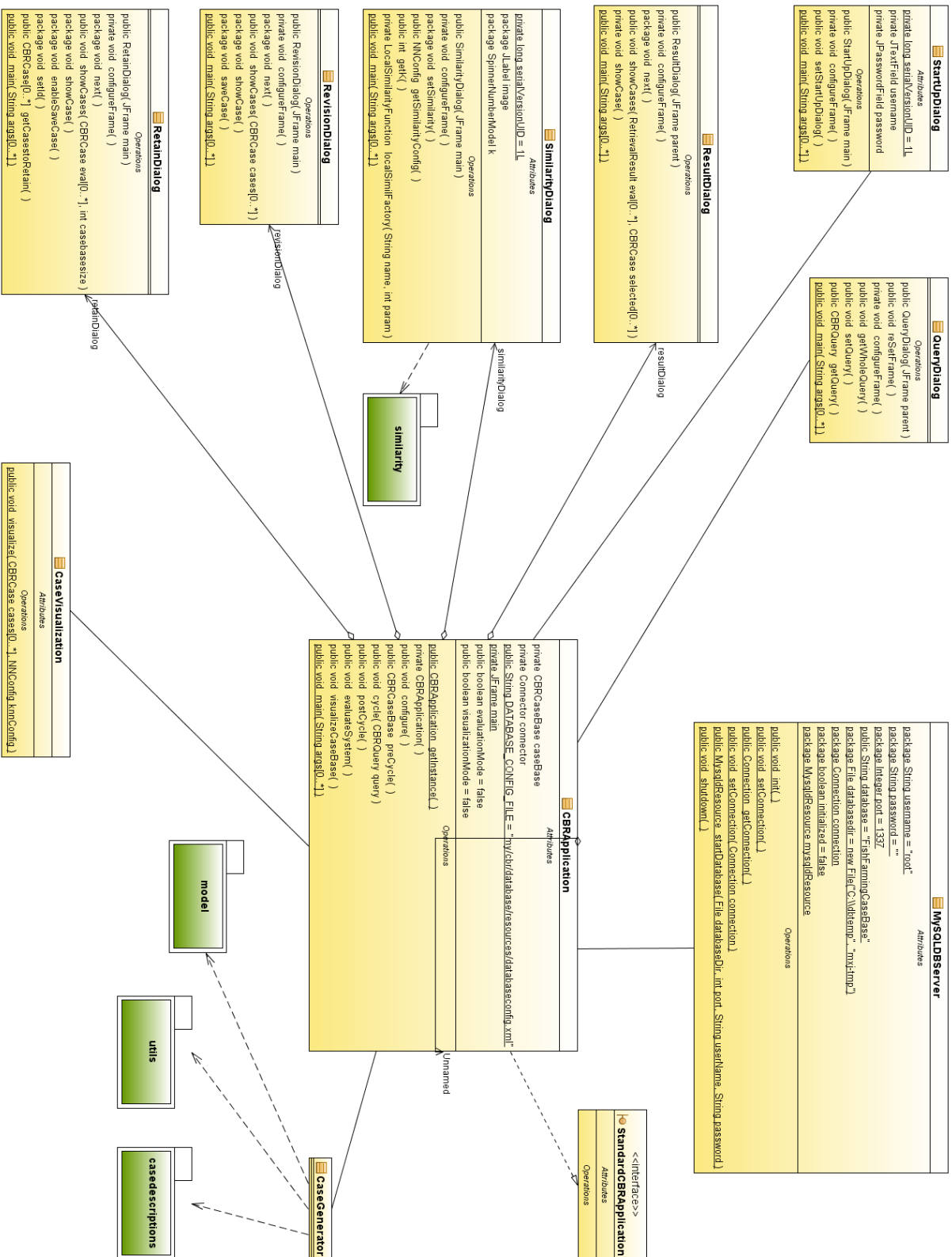


Figure 10.2: Core Architecture

10.1 Case representation and solution

Based on the findings in Chapter 9 we had to formalize our case structure. There was to our knowledge two obvious ways to do this in the combination jColibri, Hibernate and MySQL, although there may be others as well. The first alternative is to create the data model with classes in Java. Once the classes are ready, use hibernate to map the classes into tables in the MySQL database. This is done by creating the hibernate mappings described in Section 4.5 for each class and then opening a session with the hibernate.auto field set to either create or update.

The second way is actually the other way around, create the relation model first, add Hibernate mappings and reverse engineer the classes using a IDE such as Netbeans or Eclipse. Since we had most experience and knowledge with working from a object-oriented view, we opted for the first alternative. The classes representing the data model are as follows:

- CaseDescription
- AquacultureSite
- AquacultureProdUnit
- HatcheryCompanyID
- Vaccine
- SpeciesOrigin
- MeasurementType
- MeasurementInstance
 1. TemperatureInstance
 2. OxygenInstance
- CaseSolution
- FishDeath

In addition to these classes, some of the hibernate mappings will be discussed, but not in as great detail as in Section 4.5. All diagrams are created using the UML plugin in Netbeans 6.7.1. Note that this feature was removed from later versions due to funding issues¹.

¹Netbeans IDE 7.0 has external support: <http://netbeans.org/features/uml/>

10.2 CaseDescription

Figure 10.4 illustrates the data model used in our application. The CaseDescription class is composed of several attributes, ranging from simple datatypes to more complex objects. The most simple attributes are caseID and dateTime, which are of type java.util.Integer and java.util.Date respectively. The caseID represent the identification of this case, while the dateTime is the date of the operation. The other attributes in the class are made up of Compound Attributes(see Section 4.9.2) and java.util.List<Object>.

10.3 AquacultureSite

Each case is connected to a specific aquaculture site, and this is given by the class AquacultureSite shown in Figure 10.3. The class has two attributes which are deemed important in our view, siteID and capacity. The siteID is represented by an Integer, and in our data set we have sites one to six, although only 1, 2, 4 and 6 are used. The capacity indicates the maximum amount of fish in ton that the site supports.

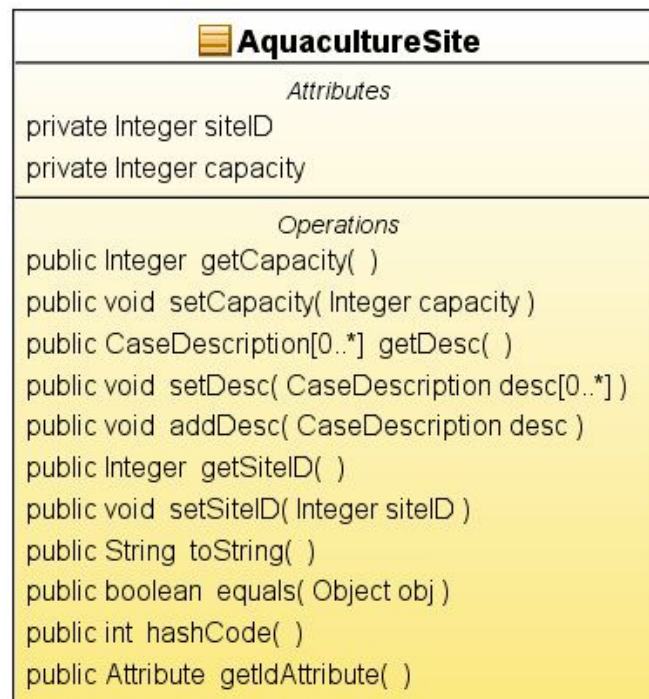


Figure 10.3: AquacultureSite Class

10.4 AquacultureProdUnit

The CaseDescription also has two other attributes named receivingProdUnit and sendingProdUnits. The receivingProdUnit is of type AquacultureProdUnit, while sendingProdUnits is a java.util.List of AquacultureProdUnit. An aquaculture production unit is

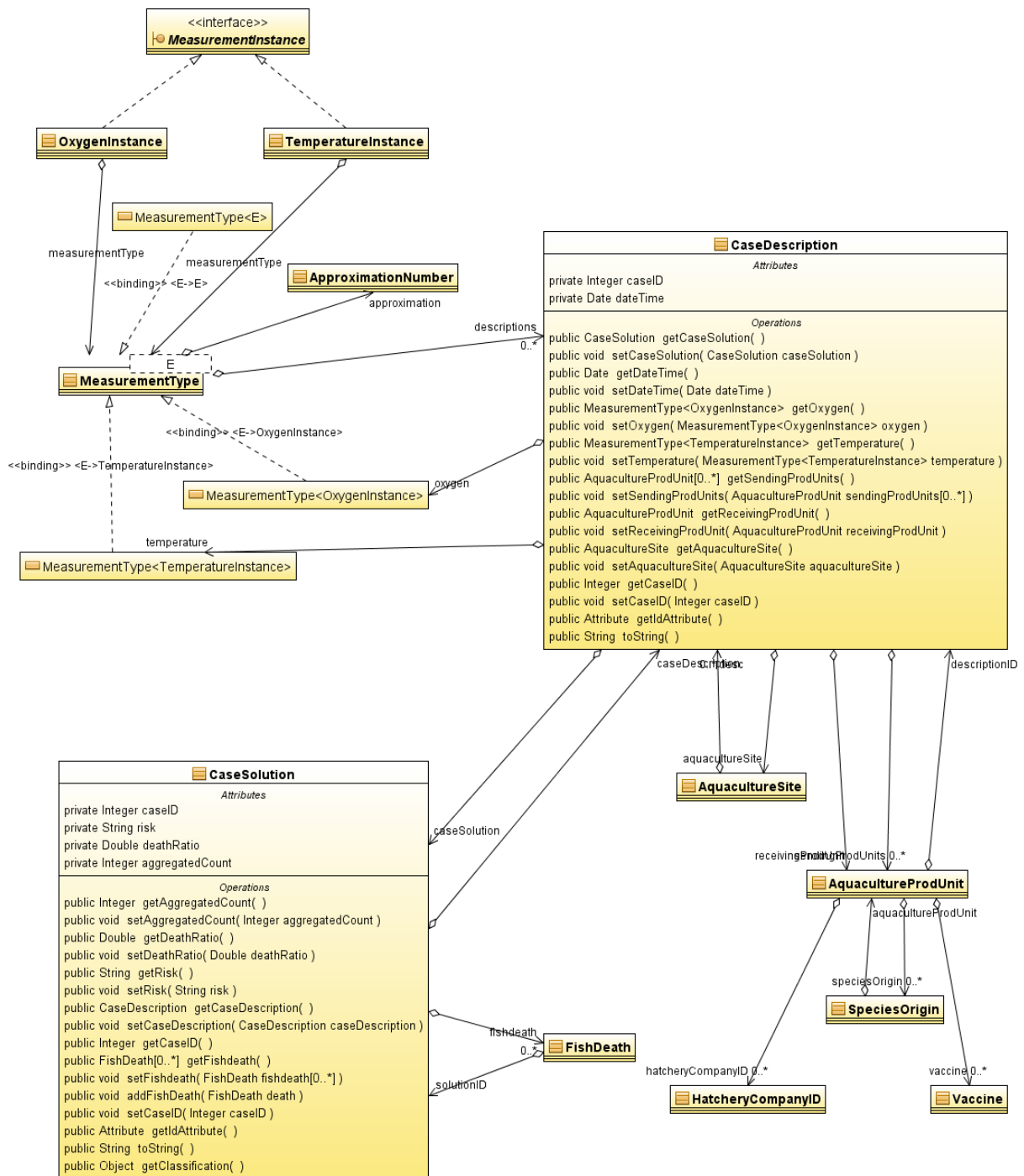


Figure 10.4: Data Model

where the fish live and feed in most of its time during production. The class diagram for AquacultureProdUnit is shown in Figure 10.5.

The class diagram may lead to some confusion when you look at two of the attributes, ID and aquacultureProdUnitID, both of type Integer. In the development of the system we initially said that a CaseDescription could have many sending production units, but that a production unit only had one case, or what is called a many-to-one relation. The attribute aquacultureProdUnitID was used for this purpose. It was later discovered that this would not work since we have several cases with the same production units. The quick and dirty solution was just to add an attribute called ID which is incremented for each production unit regardless of it already being in the table. This is of course not the optimal solution since we would get several rows displaying the same thing.



Figure 10.5: AquacultureProdUnit Class

The other attributes in the class are individCount, receivingCount and starvation-Days. The individCount is of type Double and it indicates the amount of fish in the production unit before the actual sorting operation. In a receiving production unit, this attribute would normally be null. The same would apply to starvationDays, which indi-

cates how long the fish in the production unit has been starved prior the sorting. The last attribute is related to the receiving production unit, more precisely the amount of fish received from the sending production units.

The remaining attributes are not actually given in the attribute list in Figure 10.5. They are however displayed on the full class diagram in Figure 4.15 with relationship arrows and labels. From this figure we can see that we have three different attributes, each an object of their own class. HatcheryCompanyID is specified in more detail in Section 10.5, while Section 10.6 and 10.7 explain Vaccine and SpeciesOrigin respectably.

10.5 HatcheryCompanyID

Figure 10.6 is the HatcheryCompanyID, which indicates where the fish in the production unit where hatched. It only has one attribute, hatcheryID. HatcheryID is an Integer which tells you which hatchery this is.

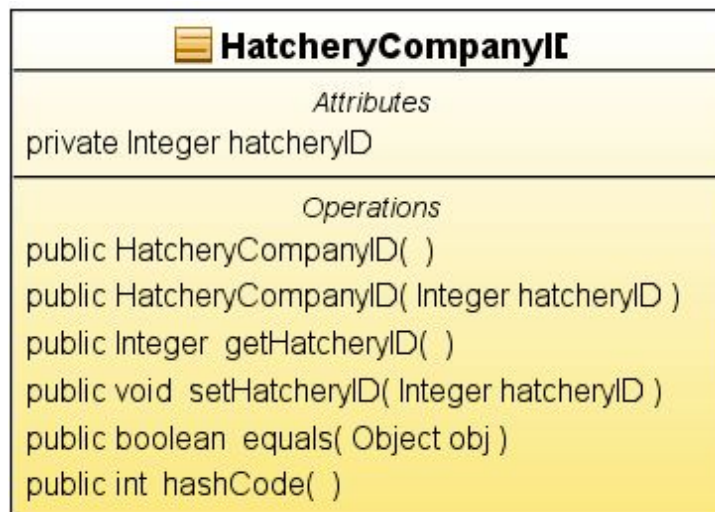


Figure 10.6: HatcheryCompanyID Class

10.6 Vaccine

Figure 10.7 shows the class diagram for Vaccine which tells us what kind of vaccine the fish group in the production unit was treated with. Each vaccine treats different diseases, often multiple, and its quite unusual to vaccinate against all diseases. The fish farmers are said to “choose” the diseases they want the fish to get in order to get the diseases where they have their main expertise.

Each vaccine in the system has an attribute named vaccineID, which of type Integer, and a name of the vaccine, which is of type String. In this application we operate with the following four vaccines:

1. Pentium Forte
2. Alphaject 6-2

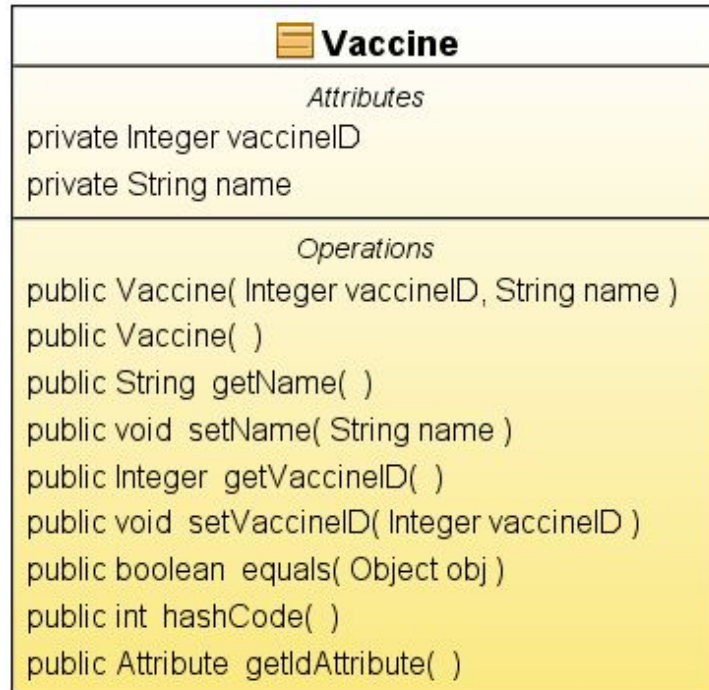


Figure 10.7: Vaccine Class

3. PD
4. Alphaject Micro 6

10.7 SpeciesOrigin

SpeciesOrigin indicates what kind of origin this fish group has. In our system all the fish are of type Atlantic salmon, also known as *Salmo salar*. There are however different origins for each fish group, and in our system we operate with the following SpeciesOrigins:

1. AquaGen
2. MOWI
3. SALMOBREED
4. terningen
5. Slørdal

Figure 10.8 displays the class for SpeciesOrigin. The class has two attributes, an Integer named speciesName and a String named speciesName, populated by the values given in the enumarted list.

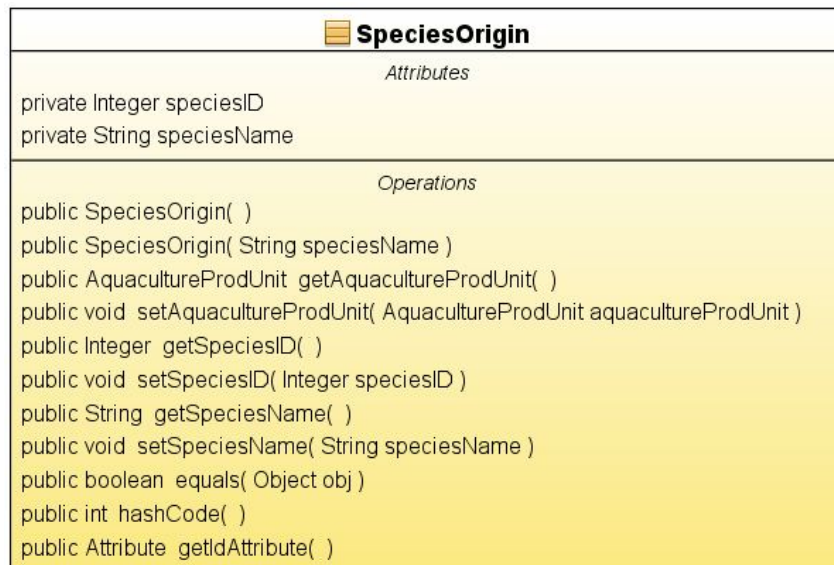


Figure 10.8: SpeciesOrigin Class

10.8 MeasurementType

MeasurementType, shown in Figure 10.9, is a generic class, where E extends MeasurementInstance. The different MeasurementInstances are described in Section 10.9. Each CaseDescription is connected to one instance of MeasurementType<E> where E is TemperatureInstance, named temperature, and an attribute called oxygen which is also of type MeasurementType<E>. In the oxygen attribute E is of type OxygenInstance.

Each MeasurementInstance has an Integer named ID and a String named Trend. As mentioned above, the class is generic, which means that the attribute currentInstance is of type E, as is the instances stored in the List pastInstances. The Trend attribute is based on an approximation done to the pastInstances. In our application we have used a basic Linear Approximation to the data. The last attribute in the class is strongly related to the Trend, as it is the numbers generated from the Linear Approximation. Based on these numbers we set the Trend to either of six different values including *Increasing*, *Weakly Increasing*, *Stable*, *Weakly Decreasing*, *Decreasing* and *Unknown*.

10.9 MeasurementInstance

MeasurementInstance is an Interface which is used to record a measurement instance in the system. There are several types of measurements in the system, although we only use two, temperature and oxygen. Figure 10.10 shows the interface and its methods. The classes that implement the MeasurementInstance interface must implement and create their own versions of the methods.

In each sub class we also create the relevant attributes for the methods. Each sub class therefore has an Integer named measurementID, a Date called date and a value of type Double. Each measurementinstance is also connected to the MeasurementType through an attribute called measurementType.

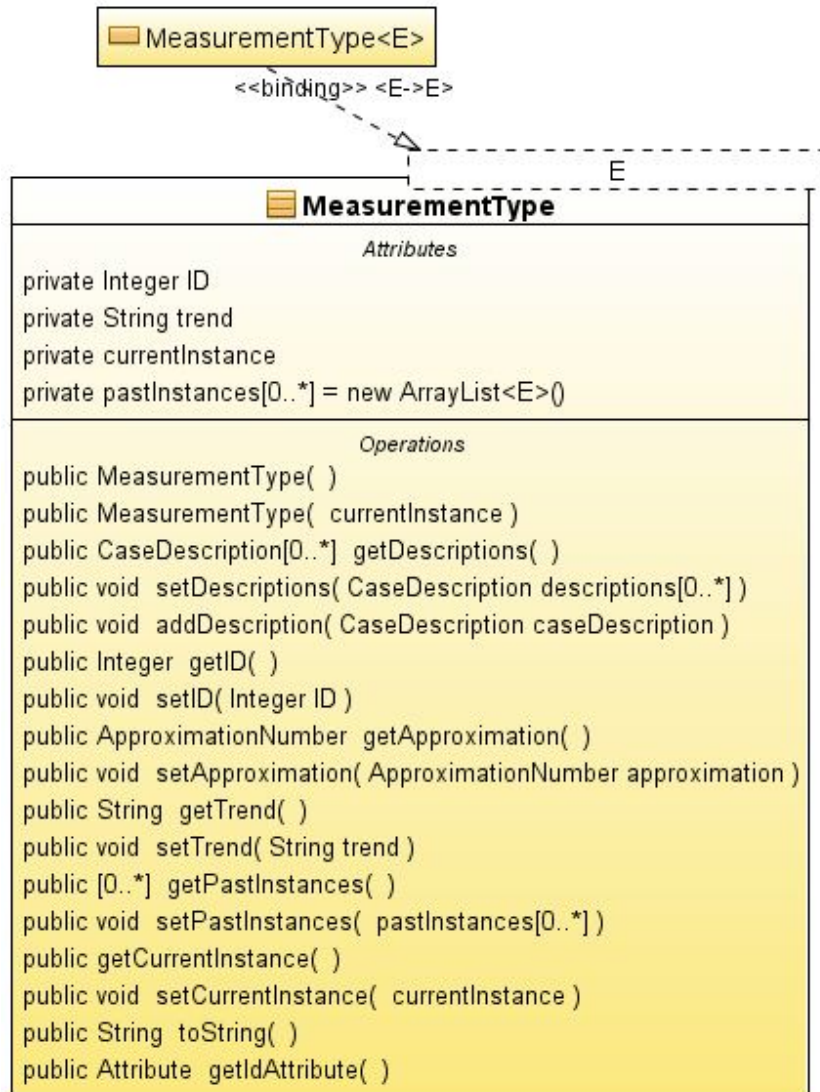


Figure 10.9: MeasurementType

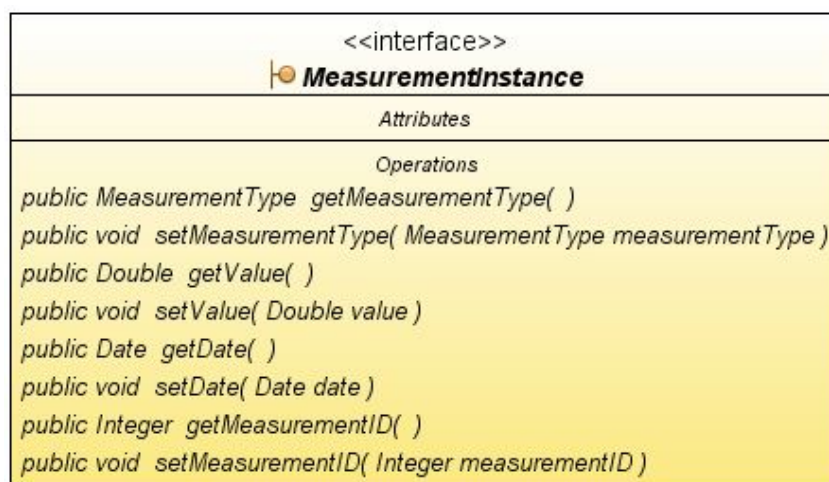


Figure 10.10: MeasurementInstance

Figure 10.11 illustrates the implementation of a MeasurementInstance. TemperatureInstance implements the interface MeasurementInstance and redefines the methods, in addition to creating the appropriate attributes.

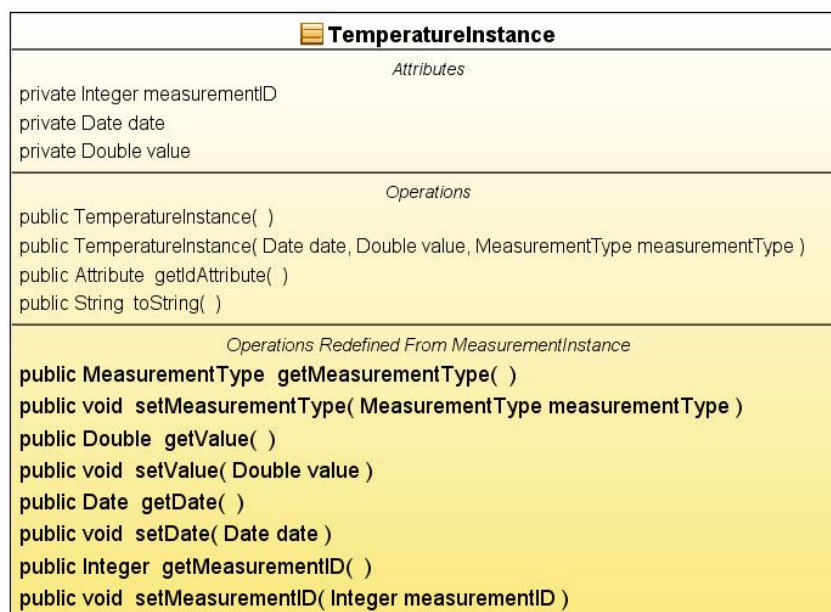


Figure 10.11: TemperatureInstance

10.10 CaseSolution

The CaseSolution, pictured in Figure 10.12, is the solution for this case. The class includes a caseID which is the same as for CaseDescription, a risk of type String, a death ratio of type Double and an aggregated death count of the type Integer. The deathRatio attribute is calculated by dividing the aggregated death count by the amount of fish received by the receiving production unit. The risk is then based on using this ratio to set the one of the following risks:

1. No death
2. Low
3. Low/Medium
4. Medium
5. Medium/High
6. High

The aggregated death count is calculated by using another attribute present in CaseSolution, a List of FishDeath. The attribute is called fishDeath and includes instances of FishDeath, which are described in more detail in Section 10.11. What is basically done is that we iterate through the list of FishDeath and add the death count of each to the aggregated count.

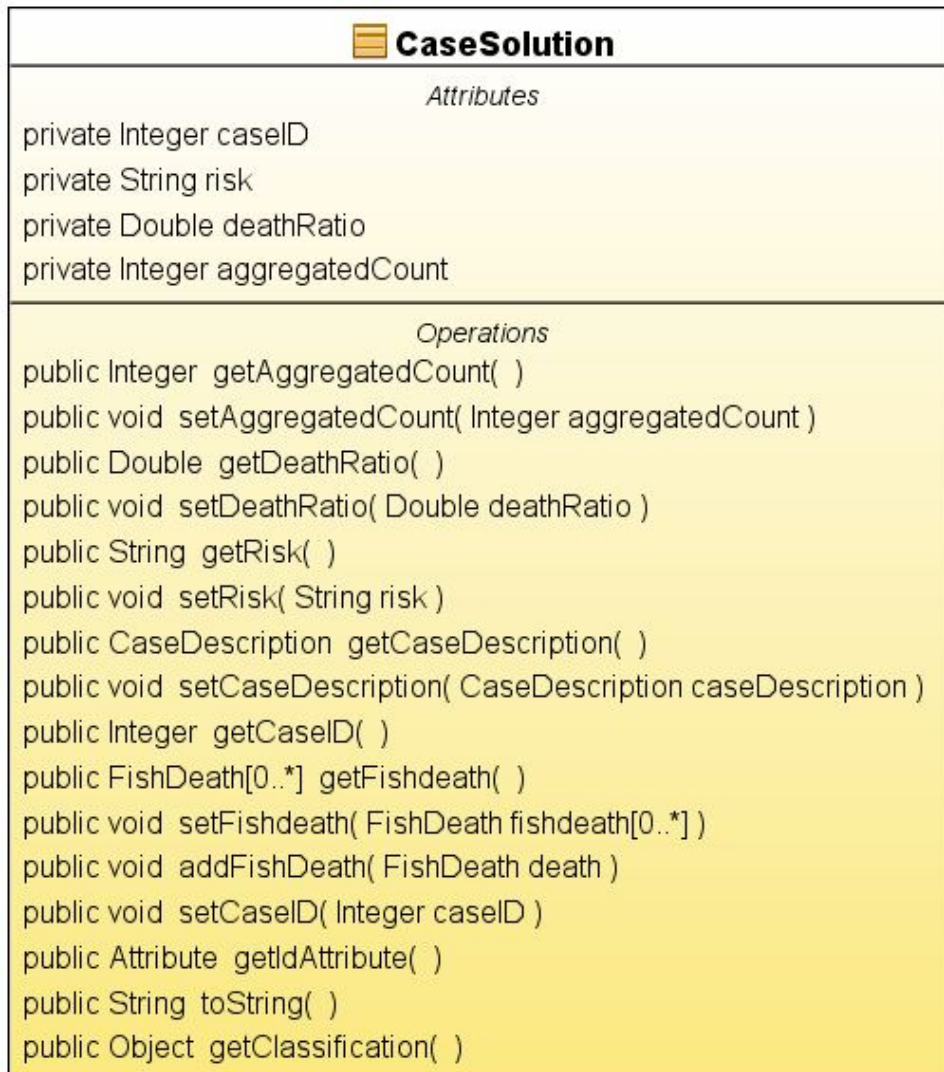


Figure 10.12: CaseSolution

10.11 FishDeath

FishDeath is a class for handling the death which occurs. When fish die in a production unit, an entry is created in a table called causeofdeath. An entry has the date it was registered, the count and the cause of death. The same thing is present in the FishDeath class, as seen in Figure 10.13.

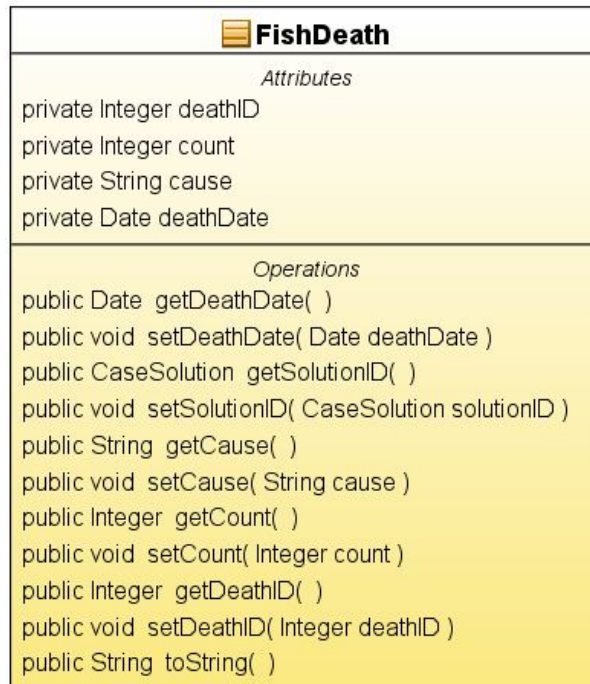


Figure 10.13: FishDeath

The class has four attributes; deathID, count, cause and deathDate. The causes of death which occur in the cases we have are highly dominated by causes related to mechanical injury/damage, as one would expect when working only with sorting operations. There is also a high occurs of causes such as Unspecified and Unknown.

10.12 Hibernate Mappings

In order to map the data model in a object-oriented view to the relation view we use Hibernate. In Section 4.5 we took a look at how we map the CaseSolution and FishDeath to the tables in the database, in addition to the relationship between them. The mappings used for the above Java classes are pretty similar, but one of the mappings we used in the implementation is an interesting one.

MeasurementInstance is an Interface which in itself should not be created instances of, but the classes who implement it however should be able. What we did at first was to create a mapping file for each of the classes which implemented the interface. The mapping files where approximately equal if you disregard the class and table name set at one of the first lines.

It seemed as there should be a more elegant solution to this, and the answer was using something called union-subclass. What is done is that the mapping for the Interface is set up in the same way you would any other class, with an abstract=true in the class tag as shown in Line 4 in Listing 10.1. Then for each subclass of the Interface create a tag called union-subclass. What this tag says is that the Interface has a subclass which has the same properties as the parent class, plus the properties added inside the tag, if there are any. In this case we do not have any additional properties to add, we just specify the name of the subclass.

Listing 10.1: Hibernate mapping file for MeasurementInstance

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate_Mapping_DTD_
   3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
3 <hibernate-mapping>
4   <class abstract="true" dynamic-insert="false" dynamic-update="false"
      mutable="true" name="my.cbr.database.model.measurement.
      MeasurementInstance" optimistic-lock="version" polymorphism="implicit"
      select-before-update="false">
5     <id name="measurementID">
6       <generator class="increment"/>
7     </id>
8     <property name="date"/>
9     <property name="value"/>
10    <many-to-one class="my.cbr.database.model.measurement.MeasurementType"
      name="measurementType" not-null="false"/>
11    <union-subclass name="my.cbr.database.model.measurement.
      TemperatureInstance"/>
12    <union-subclass name="my.cbr.database.model.measurement.OxygenInstance"
      />
13    <union-subclass name="my.cbr.database.model.measurement.FeedInstance"/>
14  </class>
15 </hibernate-mapping>
```

The result of the mappings are shown by the generation of the database schema in Figure 10.14.

10.13 Generating Cases

Based on the completion of the case representation in the previous sections, we could start to create instances for our case base. We have made an activity diagram to illustrate the program flow in the case generation process. The diagram is shown in Figure 10.15. The first thing that happens when creating the cases in the case base is that a connection has to be established to the database with all the data regarding the fish. When the connection is established we query the database for the information in the sorting table, which contains information about all the sorting that has been executed.

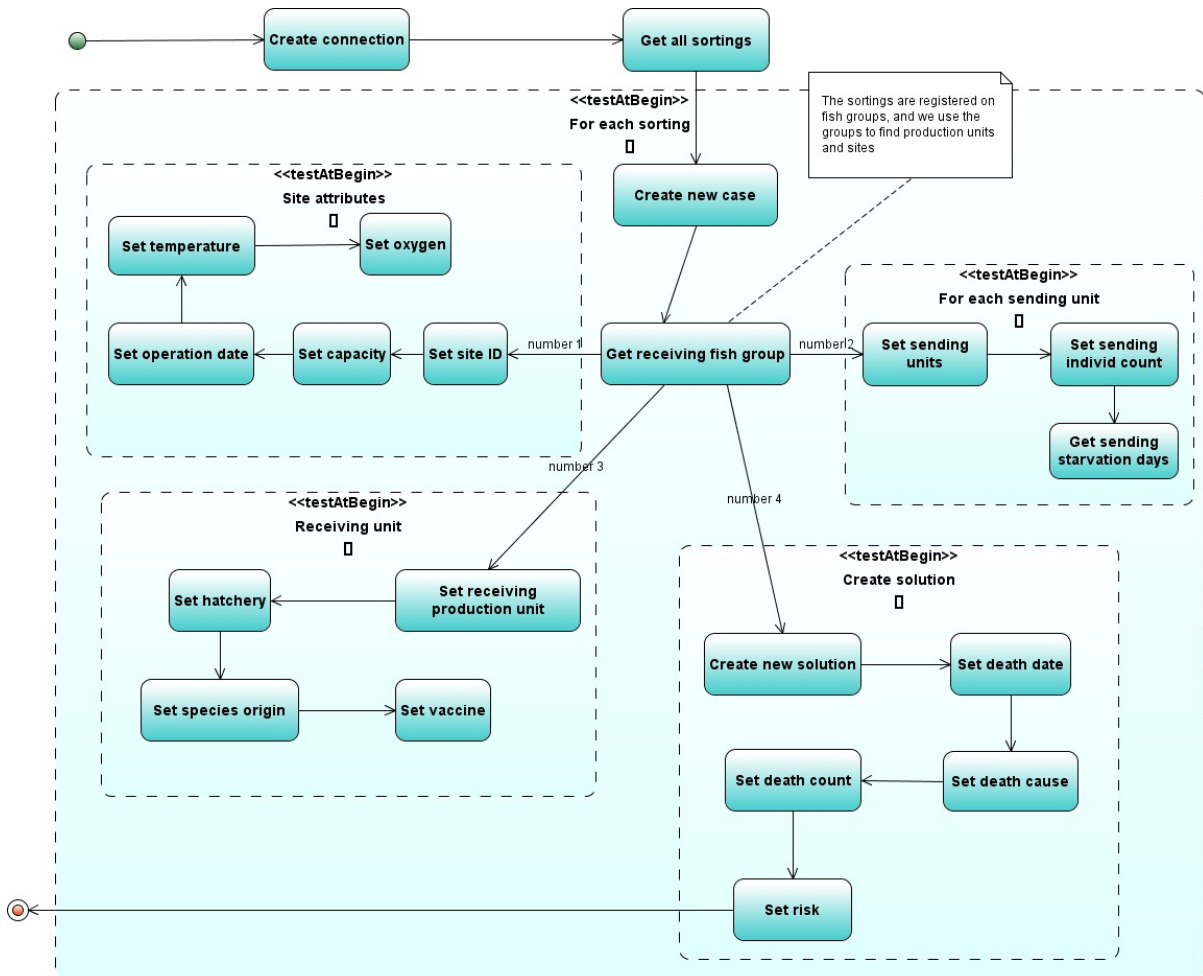


Figure 10.15: Activity Diagram Case generation

For each distinct sub-operation, where we regard one receiving unit and many sending units as a sub-operation, we create a new case. The information gathered from each sub-operation is registered to a sending and receiving fish group which are used to retrieve information related to those specific fish groups. We use the retrieved site attribute to get the related capacity, and populate a new AquacultureSite object. The operation date is also extracted and set in the case description.

The AquacultureSite and date are used further to create objects of Temperature and Oxygen in order to monitor the environment in the from date minus x days to the given

date. For each of Temperature and Oxygen, its pastInstances are passed to a linear approximation calculation in order to get the trends.

The receiving fish group is used to find the related hatcheries, vaccines and species origin that the sending units contain. We also retrieve the amount of fish the receiving fish group gets from the sending production units. Note that this is only possible during case generation, since this is data which is not in real-time. The retrieved data is set in the receiving production unit. For the sending production unit we set the individ count and starvation days.

Finally, we need to create the case solution. This is done by searching for deaths registered at the receiving production unit in the following 14 days after the operation date. A risk evaluation is done based on the aggregated count of dead fish in this period, compared to how many fish are present in the receiving unit. A death ratio is also calculated to give the user a number representation of the risk.

10.13.1 Case distribution

The generated case base has 109 distinct cases. Table 10.1 illustrates the distribution of the risk in the solution, across the cases. What we see is that 101 of the 109 cases have a low risk, and we conclude that this is due to the already good work the fish farmers do at their sites. A negative case in the system is difficult to retrieve because it has to happen first, and the fish farmers are apparently very good at judging the environmental data and observations before an operation. But there is always room for improvement, and once more data will be made available there might be some changes to the distribution. The risk assessment is also quite primitive, which may affect the distribution shown below.

Risk	Number of cases
Low	101
Low/Medium	1
Medium	0
Medium/High	1
High	6

Table 10.1: Case Solution Distribution

10.14 Query case generation

When the user is starting up the application, the first thing he must do is to create a query case. We have chosen to help the process of creating the query, so the process is half input from the user, and half look ups in the database. The process itself looks quite like the case generating described above, as it uses some of the same methods. Figure 10.16 is a sequence diagram of the generation of the query case.

All the methods referred to in the CaseGeneration class within the generateQueryCase method are all also included in the normal case generation. The user inputs the sending production units, the site id, starvation days and operation date, and the system uses

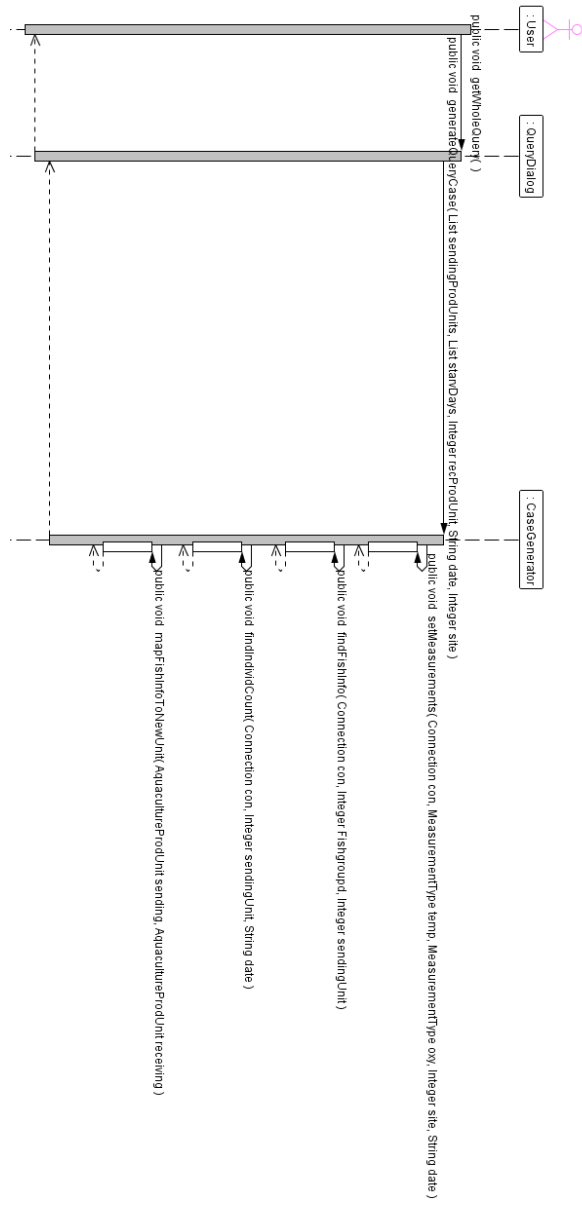


Figure 10.16: Query case generation

these inputs to find the whole query, by using the methods `setMeasurements`, `findFishInfo`, `findIndividCount` and `mapFishInfoToNewUnit`. A more illustrative walk through of system in use is shown in Section 12.4 in Chapter 12.

Chapter 11

Similarity Assessment

Now that we have established our Case Description and Case Solution, we have to represent the way we find the similarity between two cases and their attributes. We have already covered some of the basic concepts in Sections 2.5 and 4.3. The similarity functions we have used are the following:

Built in(jColibri):

- Equal
- Interval

MyCBR:

- DateSimilarity
- TrendSimilarity/MyCBRTTableSimilarity

Custom:

- ListInterval
- ListSimilarity
- ProdUnitAttributeSimilarity

11.1 Built-in

11.1.1 Equal

Equal returns either 1 or 0 depending on whether object one is identical to object 2. The class' compute method uses the equal method implemented in the superclass Object, in the form of: `object1.equals(object2)`. It is important that if you would like to use this similarity measure with compound attributes, that you redefine the class' equals method in order for this to work properly.

11.1.2 Interval

The Interval similarity returns the similarity of two numbers inside an interval. The similarity measure is applicable to `java.lang.Number`, and works therefore great with `java.lang.Double`. The class takes one parameter, the interval of the type double. The similarity is measured by taking 1 minus the absolute value of x minus y divided by the interval:

$$sim(x, y) = 1 - \frac{|x - y|}{interval} \quad (11.1)$$

Example

The Case Description has an attribute called `temperature`(compound attribute), which is of type `Temperature`. In `Temperature` we have an attribute called `currentInstance` which states the latest temperature value and date. For the temperature value we have used the Interval similarity and the interval is set to 4. Given a case in the case(x) base with a `currentInstance` of 8.5 and a query(y) with 9.4 we find a similarity of 0.775(see Equation 11.2).

$$\begin{aligned} sim(x, y) &= 1 - \frac{|x - y|}{interval} \\ sim(8.5, 9.4) &= 1 - \frac{|8.5 - 9.4|}{4} \\ &= 1 - \frac{0.9}{4} \\ &= 0,775 \end{aligned} \quad (11.2)$$

Equation 11.3 shows an example where there is not much similarity between the case and query.

$$\begin{aligned} sim(x, y) &= 1 - \frac{|x - y|}{interval} \\ sim(15, 11.2) &= 1 - \frac{|15 - 11.2|}{4} \\ &= 1 - \frac{3.8}{4} \\ &= 0,05 \end{aligned} \quad (11.3)$$

For the oxygen attribute the interval is set to 10.

11.2 MyCBR

11.2.1 DateSimilarity

The DateSimilarity measure is applicable for datatypes of `java.util.Date`. The similarity measure is taken from a prototype system created at SINTEF Fisheries and Aquaculture autumn 2010. The system was created in myCBR, which made it possible to use in our system through the wrapper function described in Section 4.3.1. Figure 11.1 shows the graphical representation of the similarity function created in myCBR, while Listing 11.1 is the representation in XML.

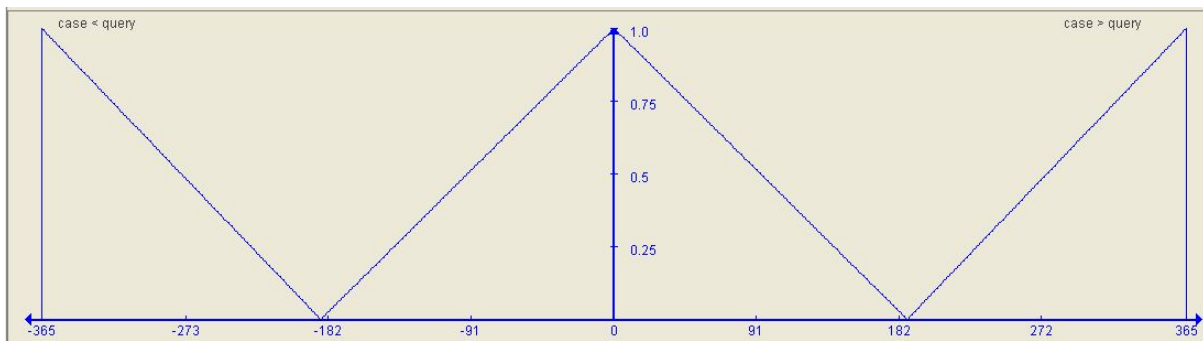


Figure 11.1: Date Similarity

Listing 11.1: XML file for Date similarity

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <SMFunction smfname="default" model_instname="date" type="Integer" maxval="
   465.0" minval="100.0" modeDiffOrQuotient="0">
3   <SamplingPoint xValue="-365.0" yValue="1.0" />
4   <SamplingPoint xValue="-187.0" yValue="0.0" />
5   <SamplingPoint xValue="0.0" yValue="1.0" />
6   <SamplingPoint xValue="187.0" yValue="0.0" />
7   <SamplingPoint xValue="365.0" yValue="1.0" />
8 </SMFunction>
```

The similarity function works with Integers, while we in our system have a `java.util.Date`. In order to get Date and the similarity function to work with each other we had to transform the Date to a number. What we did was to create our own class called DateSimilarity, which implements the LocalSimilarityFunction and all its methods, and copy the related XML parsing from MyCBRNumberAdvancedSimilarity.

Inside the compute method we used a class called Calendar to get the day of the year for a given Date. By doing this we can represent our Date without taking into consideration the year, which means that 20.04.2011 is most likely quite similar to 25.04.2005. The Java code for this is shown in Listing 11.2.

Listing 11.2: Date similarity Java conversion

```
1 Calendar calendar = Calendar.getInstance();
```

```

2 // Get the query representation
3 calendar.setTime(queryDate);
4 int queryDateOfYear = calendar.get(Calendar.DAY_OF_YEAR);
5
6 // Get the query representation
7 calendar.setTime(caseDate);
8 int caseDateOfYear = calendar.get(Calendar.DAY_OF_YEAR);
9
10 // find the similarity
11 double res = smf.getSimilarityBetween(queryDateOfYear, caseDateOfYear, null
    );

```

Listing 11.3, Line 19, illustrates the point made above with the similarity between two date which are close in days but not in years. Testing 2 in Line 20 shows that the dates, given in Lines 10-17, are not similar at all, where the query will be located near the -182 mark on the x-axis in Figure 11.1, giving it a value of approximately zero on the y-axis.

Listing 11.3: Date similarity Java test

```

1 DateSimilarity dateSimilarity = new DateSimilarity("my/cbr/similarity/
    resources/date.xml");
2 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
3 try {
4     // Testing 1
5     Date ourCase = sdf.parse("2005-04-25");
6     Date query = sdf.parse("2011-04-20");
7     double compute = dateSimilarity.compute(ourCase, query);
8     System.out.println("Testing 1: "+compute);
9
10    // Testing 2
11    ourCase = sdf.parse("2005-04-20");
12    query = sdf.parse("2011-10-25");
13    compute = dateSimilarity.compute(ourCase, query);
14    System.out.println("Testing 2: "+compute);
15    } catch (Exception ex) {
16        ex.printStackTrace();
17    }
18
19 Testing 1: 0.9732620320855615
20 Testing 2: 0.005617977528089901

```

11.2.2 TrendSimilarity

The similarity for the trends in the system is built the same way as the example in Section 4.3.1. Figure 11.2 demonstrates the allowed values we have and the relationship between each value. Given a query with Increasing trend and a case with Stable trend we find a similarity of 0.5. The numbers are not based on any research, but rather on intuition.

Case Base Values						
Reset	Increasing	IncreasingWeakly	Stable	DecreasingWeakly	Decreasing	Unknown
Increasing	1.0	0.75	0.5	0.25	0.0	0.0
IncreasingWeakly	0.75	1.0	0.75	0.5	0.25	0.0
Stable	0.5	0.75	1.0	0.75	0.5	0.0
DecreasingWeakly	0.25	0.5	0.75	1.0	0.75	0.0
Decreasing	0.0	0.25	0.5	0.75	1.0	0.0
Unknown	0.0	0.0	0.0	0.0	0.0	0.0

Figure 11.2: Trend Similarity

11.3 Customized

11.3.1 ListInterval

ListInterval is just a reimplementaion of Interval, Section 11.1.2, where it takes the input as two java.util.List. What it does is to take the size of the case list and compare it to the size of the query list by using the equation shown in Equation 11.1.

The ListInterval can be used with any kind of List, but in our application it is used with the sending production units. This is however not the default similarity function for the sending production units.

11.3.2 ProdUnitAttributeSimilarity

ProdUnitAttributeSimilarity does a more thorough similarity assessment of the sending production units. It still takes two object of List, where the list now must contain objects of AquacultureProdUnit. Inside each AquacultureProdUnit we have multiple different attributes, such as starvationDays and individCount. One List contains several AquacultureProdUnit's where each contains a value for a given attribute. As Equation 11.4 indicates, we take the interval similarity of each attribute and summarize them. Each attribute may have different interval values.

$$combinedSimilarity(x, y) = \sum_{a=attribute}^n 1 - \frac{|x.a - y.a|}{a.interval} \quad (11.4)$$

The total similarity is the value from Equation 11.4 divided by the number of distinct attributes, as shown in Equation 11.5.

$$sim(x, y) = \frac{combinedSimilarity(x, y)}{distinctAttributes} \quad (11.5)$$

11.3.3 ListSimilarity

ListSimilarity is related to the similarity between the receiving production units. The input to the compute method must be two object of java.util.List or null. In our system we have a List for each vaccine, species origin and hatchery. The similarity is calculated by using Equation 11.6, where hits(x,y) is the number of equal objects in case x and query y and maxListSize is the size of the largest List.

$$sim(x, y) = \frac{hits(x, y)}{maxListSize} \quad (11.6)$$

The hits are calculated by using the code shown in Listing 11.4.

Listing 11.4: List Similarity calculation

```
1 // Set initial values
2 int denominator = caseO.size();
3 List first = caseO;
4 List second = queryO;
5
6 // Check if the query case is the largest list
7 if(queryO.size() > denominator){
8     denominator = queryO.size();
9     second = caseO;
10    first = queryO;
11 }
12
13 // compute the hits
14 int hits = 0;
15 for(Object v1 : first) {
16     if(second.contains(v1))
17         hits++;
18 }
```

Chapter 12

The implemented system

This chapter describes the system features and includes a walk through of the whole system, where the functionality described in the previous chapters is put into action.

12.1 Libraries

List of libraries which are used in the application:

- **jColibri**
 - antlr-2.7.6.jar
 - asm-attrs.jar
 - asm.jar
 - c3p0-0.9.0.jar
 - cglib-2.1.3.jar
 - Chart2D.jar
 - commons-collections-3.2.jar
 - commons-logging-1.1.jar
 - dom4j-1.6.1.jar
 - hibernate3.jar
 - InfoVisual-CB.jar
- jcolibri2.jar
- jdom-1.0.jar
- jta.jar
- junit.jar
- log4j-1.2.14.jar
- **jCalendar**
 - jcalendar-1.3.3.jar
 - looks-2.0.1.jar
- **jFreeChart**
 - jfreechart-1.0.13.jar
 - jcommon-1.0.16.jar
- **myCBR wrapper**
 - myCbr.jar
 - myCBRSimilarityFunctions.jar
- **MySQL**
 - mysql-connector-java-5.1.14-bin.jar
 - mysql-connector-mxj-gpl-5-0-11-db-files.jar
 - mysql-connector-mxj-gpl-5-0-11.jar
 - stax-api-1.0.1.jar

12.2 CBR Application

When building a Case-Based Reasoning system using the jColibri framework you need to implement the interface `jcolibri.cbrapplications.StandardCBRAApplication`. This interface contains the following four methods; `configure`, `precycle`, `cycle` and `postcycle`. The `configure` method sets up the application with regards to creating the objects of a case base, connector and so on. The `precycle` reads the cases from the case base into an in-memory case base structure. The `cycle` method executes the CBR cycle given a specific query. The `postcycle` is run to shut down the application, typically closing the connector and database server. The following subsections describe how we have used and implemented the methods in our system.

12.2.1 Configure

The `configure` method starts a database server, and initializes a database connector with the database configuration file. Then it creates a `CachedLinealCaseBase`, which only persists cases when closing the application, if the system is being run in evaluation mode. If this is not the case, it organizes the cases into a normal `LinealCaseBase`, where cases can be stored without closing the application. The method also initializes the similarity, result, revision and retain dialogs. The query dialog is initialized in the main method.

12.2.2 Precycle

The first thing that happens in the `precycle` method is that all the cases are being loaded through the connector into the case base. Each case is printed out with its solution attached to it, although this is just for debugging puposes. A check is done to see if every case has a solution, since there are some holes and missing data in our data set. The cases without solution are removed from the case base.

12.2.3 Cycle

The `cycle` method executes the CBR cycle with the query set in the query dialog. There are two different ways through the cycle method. One if you are running the application in evaluation mode, and one where the application is run in normal mode.

Normal mode

The first thing that happens is that the similarity dialog is set visible to the user. When the user has assessed the similarity configurations, the application uses the similarity configurations to retrieve the k-nearest neighbors by using the two methods `NNScoringMethod.evaluateSimilarity` and `SelectCases.selectTopK`. `NNScoringMethod.evaluateSimilarity` computes the similarity between the query and each case in the case base. The cases are stored in a list where the first instance is the most similar while the last is the least similar case. The `SelectCases.selectTopK` is used to retrieve the K first cases from this list. The K is set by the user in the similarity dialog.

The retrieved cases are printed out (debugging) and shown in the result dialog. The program flow from here is quite straight forward. First the user chooses which case he

wants to use from the retrieved ones, where the first case shown is the most similar. When the solution from this case has been applied to the current case, we are shown a revise dialog. The revise dialog is meant to be used by the user to correct the application with regards to the actual solution/outcome. If the solution was correct the user may simply press next in the dialog. If however, the solution was not correct, the user can act as the domain expert and revise the solution for this current case. Regardless of the scenarios, we are met with a new dialog called retain. In this dialog the user can either choose to store the query case with the applied solution, or simply ignore it.

The CBR cycle is now complete and the user can start over with a new query or end the application. If the user opts to end the application, postcycle is called to close the appropriate objects used.

Evaluation mode

The evaluation mode is a mode for evaluating the CBR system with regards to its correctness. The user dialogs from the normal mode are not used in this mode. The system uses the default similarity configurations to compute the similarity of the cases.

We have used two different evaluation methods provided in the jColibri framework; HoldOutEvaluator and LeaveOneOutEvaluator. The different evaluators uses the similarity configuration to evaluate the system in each way, and then output a graph visualizing the outcome of the evaluation. The evaluation of the system is described in more detail in Chapter 13.

Visualization mode

A last minute feature which was added to Glaucus was a method for visualizing the case base. The visualization methods provided by jColibri estimate the distance each case in the case base has with the other cases. The visualization will therefore show where the different cases are clustered according to the distance between them.

12.2.4 Postcycle

Shuts down the application by closing the connector and database server.

12.3 Deploying with pre-configured database

Our application is deployed with a pre-configured database, which reduces the installation barriers for the end-user. We accomplish this by using an embedded database server in the form of MySQL, with Connector/MXJ. What MySQL Connector/MXJ does is that it makes the MySQL database appear to be a Java component, not depending on the platform used.

The MySQL Connector/MXJ can be downloaded from the MySQL website¹, and it includes two libraries which are of interest to us:

- mysql-connector-mxj-gpl-5-0-11

¹<http://dev.mysql.com/downloads/connector/mxj/>

- mysql-connector-mxj-gpl-5-0-11-db-files

The mysql-connector-mxj-gpl-5-0-11-db-files is where our database will be stored, more precisely in a .jar-file called data_dir. The data_dir contains the normal database schemas such as mysql, where user profiles are stored, and test, which is just a test database. The step-by-step walkthrough to creating the custom db-files.jar is located in the mysql reference manual².

Listing 12.1 shows how the MySQL server is started through the Java code. The databasedir in Line 1 is the local folder we want the related server and database files to be stored. The start method of the MysqlResource starts an instance of the MySQL server on the port specified.

Listing 12.1: Start MySQL server Java code

```
1 MysqlResource mysqlResource = new MysqlResource(databaseDir);
2
3 // Database properties
4 Map database_options = new HashMap();
5 database_options.put(MysqlResourceI.PORT, Integer.toString(port));
6 database_options.put(MysqlResourceI.INITIALIZE_USER, "true");
7 database_options.put(MysqlResourceI.INITIALIZE_USER_NAME, userName);
8 database_options.put(MysqlResourceI.INITIALIZE_PASSWORD, password);
9
10 // Start local server
11 mysqlResource.start("test-mysqld-thread", database_options);
```

A normal java.sql.Connection can now be obtained by calling DriverManager.getConnection(jdbc-url, properties). In order to stop the server the shutdown method for MysqlResource has to be called. If this is not done the server will continue to run after the application is done executing.

²<http://dev.mysql.com/doc/refman/5.6/en/connector-mxj-usagenotes-customdb.html>

12.4 Application walk through

The first thing the user will meet while running Glaucus system is a question if you would like to create the case base. This has to be done the first time the application is being run. The dialog is shown in the Figure 12.1.

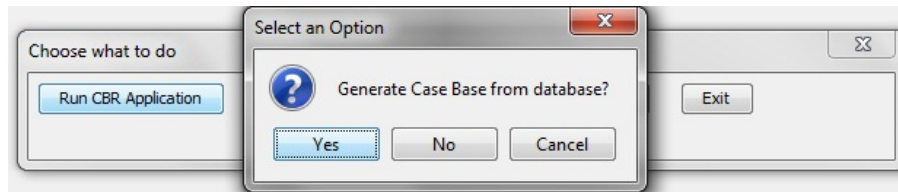


Figure 12.1: Startup

12.4.1 Creating the query(new case)

After creating the case base, the query dialog in Figure 12.2 appears. The user has to fill in the appropriate inputs in the open text fields. In this example production unit “1506” is sending fish to production unit “1509” at aquaculture site “6”. The fish have been starved for “8” days, and the date is “2010-08-05”. This information present in a case in the case base, as our application is dependent on real-time data from the site to be tested properly, something we do not have access to at this point.

Figure 12.2: Query Dialog

When all inputs are provided, the user has to click on the “Get all attributes” button, and the system will collect the rest of the attributes out from the database and fill in the

rest of the attribute slots in the query. We decided to do this both because of the ease of the user, and also to know that the input data is correct and consistent. How the query is generated can be seen more detail in the sequence diagram in Chapter 10.1, Section 10.14 Figure 10.16. The result of clicking the “Get all attributes” button with the values provided in Figure 12.2 is shown in Figure 12.3.

Sending production units	1506
Receiving production unit	1509
Aquaculture Site	6
Capacity of site	4680
Individ Count in sending unit	330422.0
Number of starvation days	8
Operation Date	2010-08-05
Vaccines	Alphaject Micro 6
Hatcheries	38
Species Origins	AquaGen
Current Temperature	13.7 C
Temperature Trend	IncreasingWeakly
Current Oxygen Level	96.3 %
Oxygen Trend	IncreasingWeakly
Note	Input OK

Figure 12.3: Whole query

With the rest of the attributes collected, the user has the opportunity to visualize the oxygen and temperature trend for the last 14 days. The trends for this test case are shown in Figure 12.4a and 12.4b. If the user is satisfied with the complete query, he can now click the “Set query” button to get to the next dialog. Should he not be, it is possible to change the input to the system and repeat the steps described above.

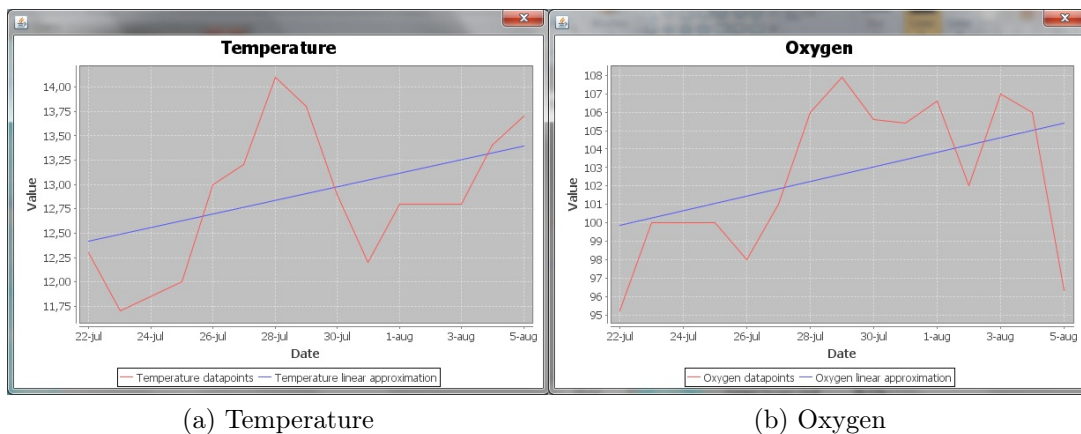


Figure 12.4: Visualization of the Trends

12.4.2 Retrieving similar cases

After the query case has been set, the user gets to the similarity dialog illustrated in Figure 12.5. Each attribute has its own similarity function attached to it, and the user can choose from a couple of functions with attached weights, or he can choose to let them all be in the default configuration. It is also possible to select how many retrieved cases to get. When finished the user has to click the “Set similarity configuration” button. The K nearest cases are calculated and sent to the next dialog for review.

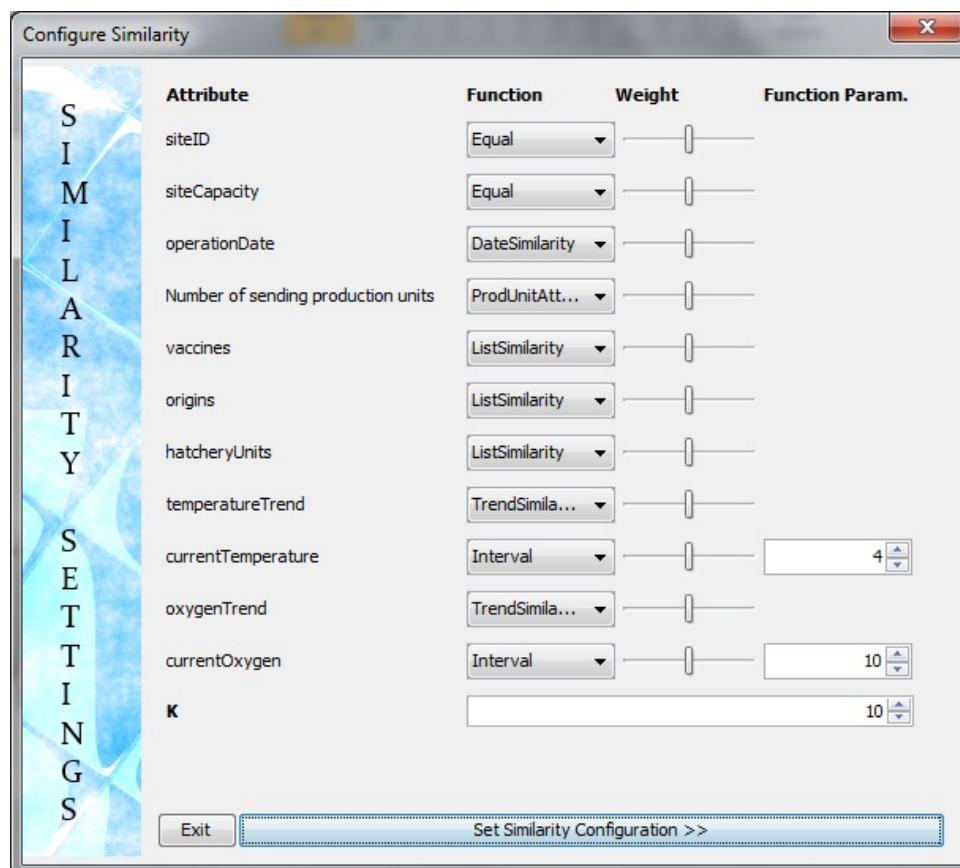


Figure 12.5: Similarity dialog

12.4.3 Reuse a case

Now the result dialog, illustrated in Figure 12.6 pops up. It is possible to alternate between the retrieved cases at the top of the dialog. The similarity number is also shown here, closer to “1” is more similar. The outcome, or result, of the retrieved cases is shown at the bottom of the dialog. The user is supplied with the aggregated number of fish death the following 14 days after the operation, together with a percentage of dead fish compared to all the fish in the receiving production unit, and a risk factor of either Low, Low/Medium, Medium, Medium/High or High. The user can also choose to visualize the deaths in a diagram, shown in Figure 12.7, where he can see the total death for each cause of death in the period.

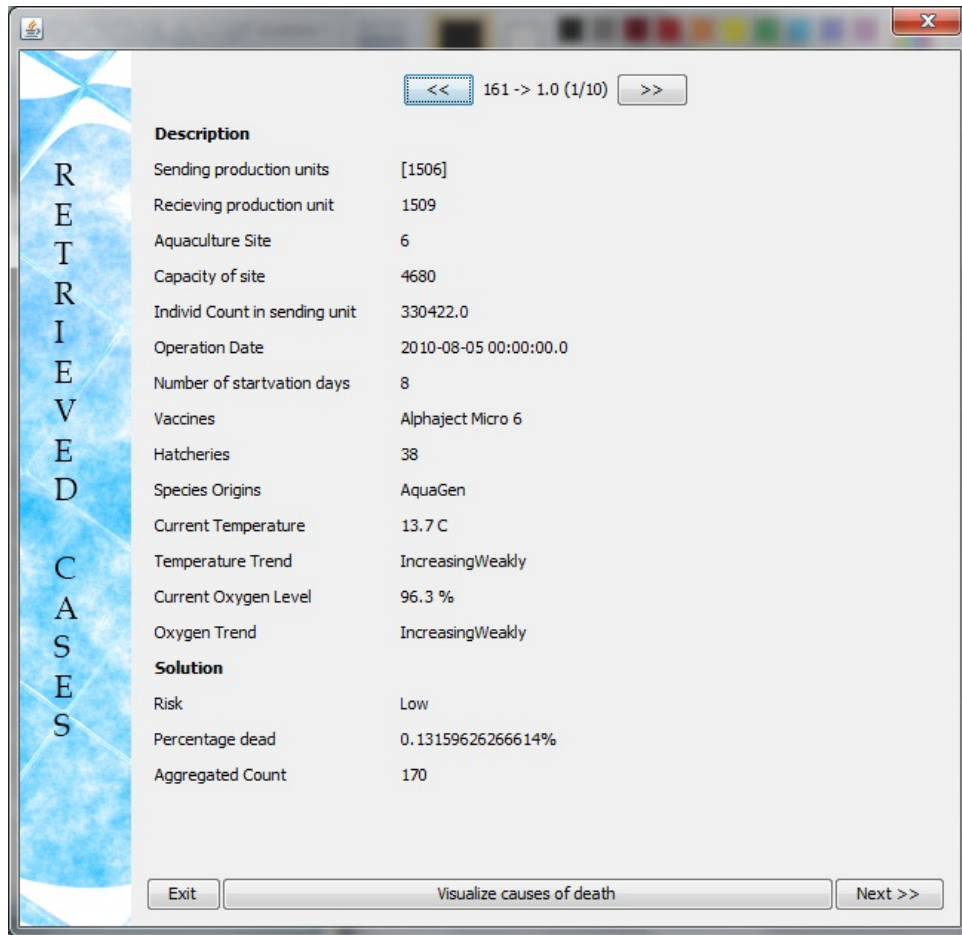


Figure 12.6: Retrieved case 1

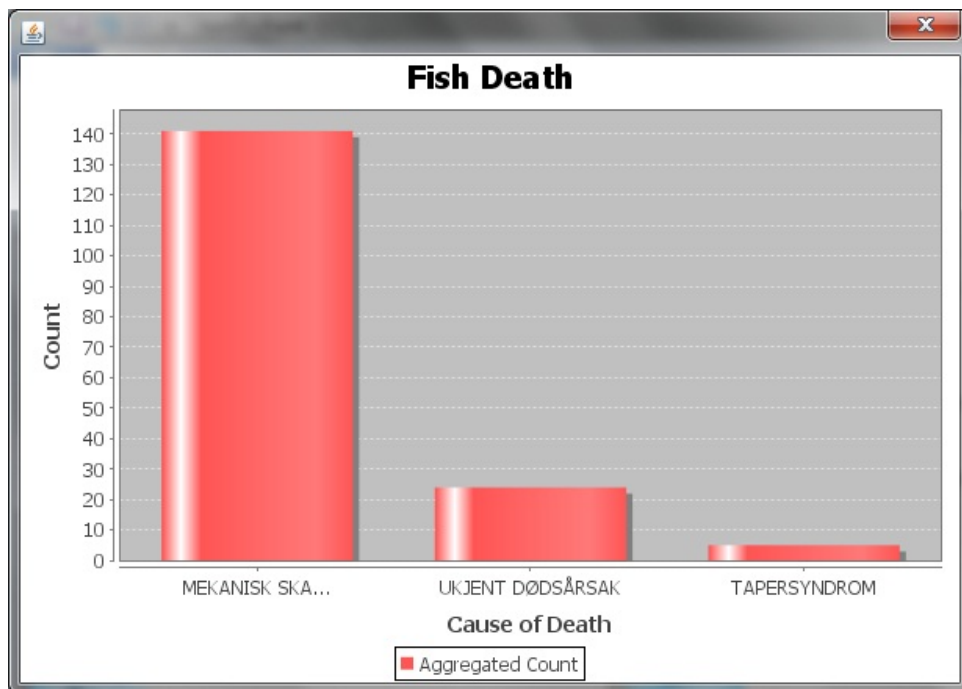


Figure 12.7: Visualization of deaths

The case returned in this instance has a similarity of “1”, not very surprising, since the query case used was an example from the case base itself. All the operations done at the same site at the given date are registered as sub sorting operations, and illustrated in Figure 12.8

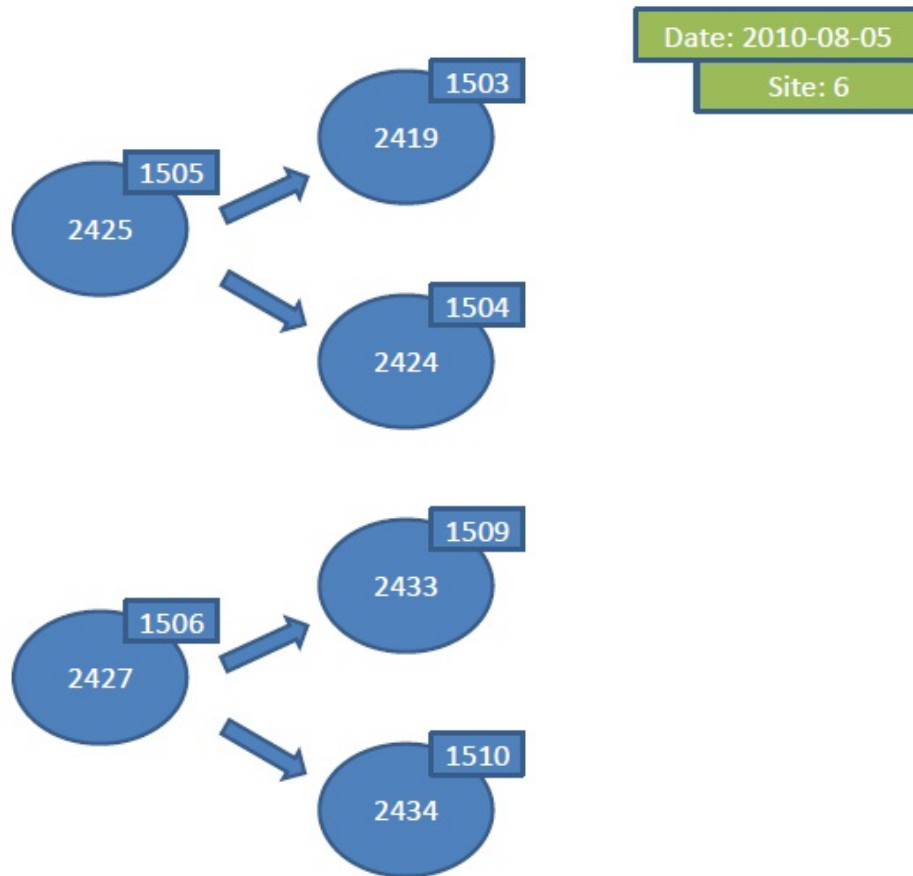


Figure 12.8: All operations

Retrieved case number 2 is shown in Figure 12.9. This case is exactly like the last one, and this has its natural explanation, when we look at the Figure 12.8. The only thing that has changed in this retrieved case is the result, since more fish died in the following 14 days in this receiving unit. The distribution of the different causes of death are shown in Figure 12.10.

It is also not surprising that the two next cases retrieved are from the same day and the same site, as shown in Figure 12.8. The cases share many features, including site and environmental measurements.

The fifth retrieved case can be seen in Figure 12.11. This is a sorting that has been done a week after the four others, as can be seen from the operation date. The environment has changed a little, but not much, and the similarity number at the top of the page reflects this.

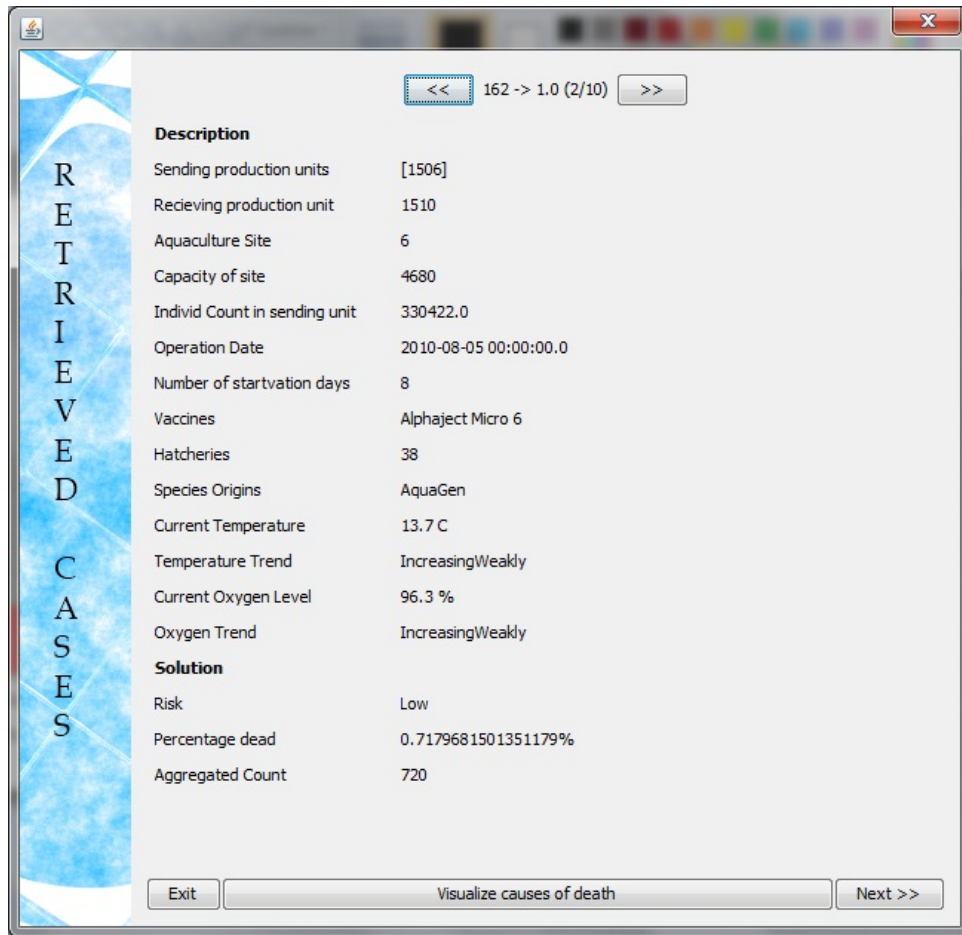


Figure 12.9: Retrieved case 2

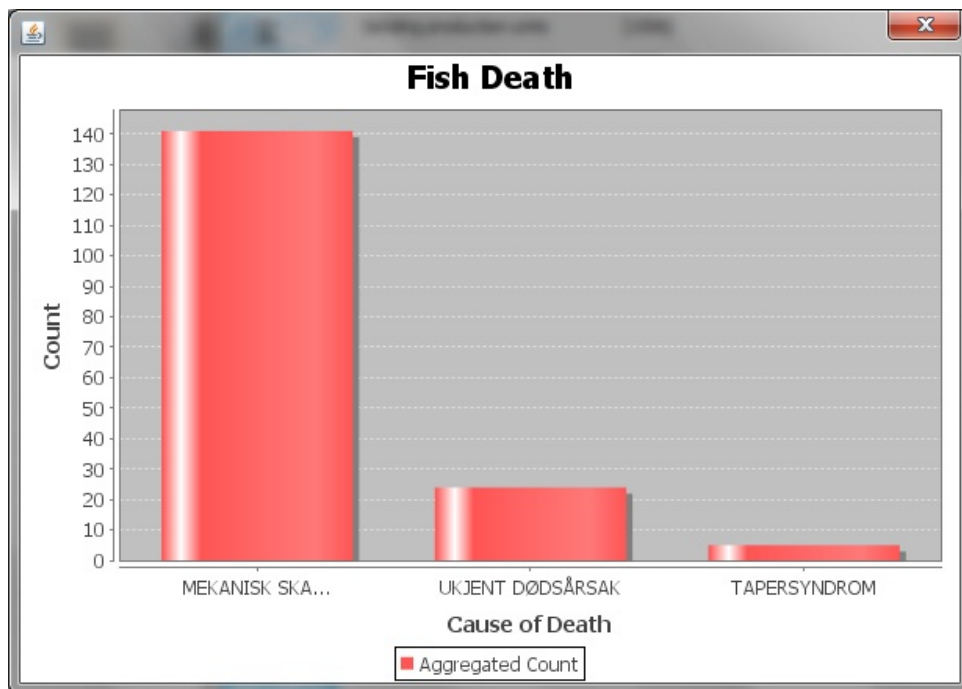


Figure 12.10: Visualization of deaths, case 2

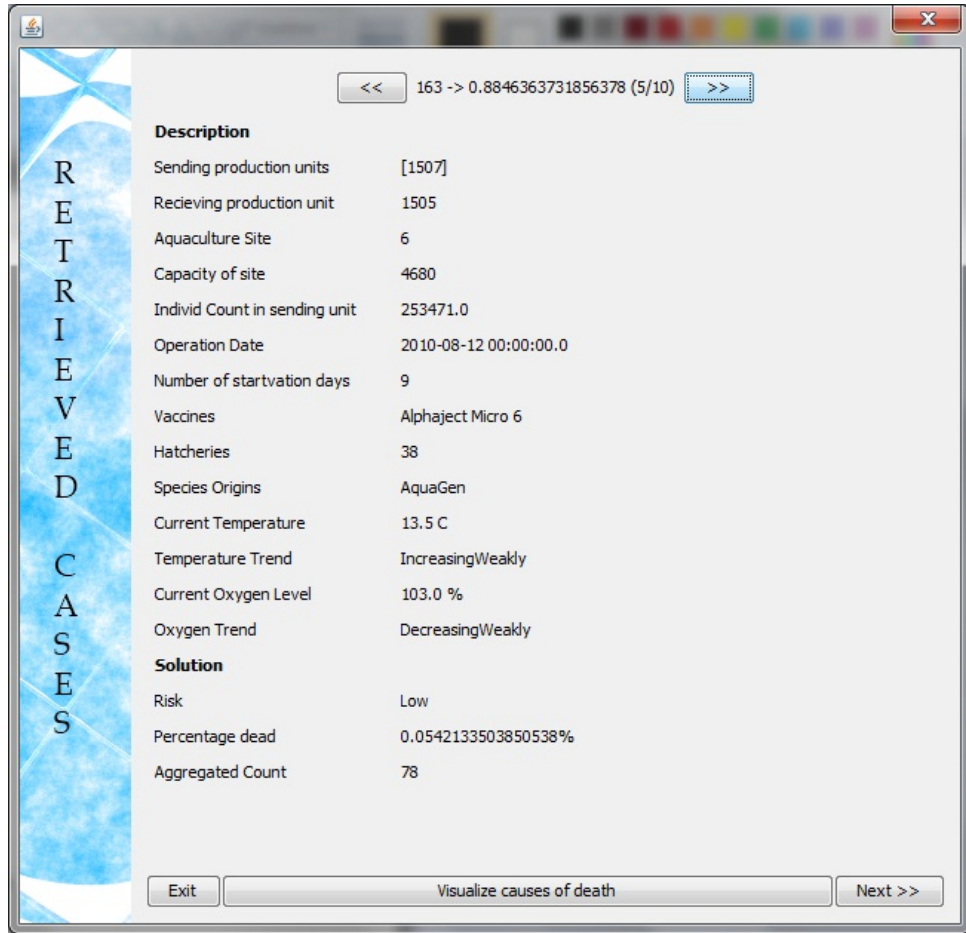


Figure 12.11: Retrieved case 5

12.4.4 Revise and Retain the case

After examining the retrieved cases, the user can choose the case he thinks has the best solution to his query and click on the “Next” button. The dialog which appears now is the revise dialog, shown in Figure 12.12. This dialog is meant to be used if there were any deviations from the proposed solution to what actually happened. The user can change the solution to what the real outcome was, or let it stay as it was in the retrieved case. When the user is satisfied, he can go on by clicking the “Set revision” button, which will take him to the retain dialog shown in Figure 12.13. The user can now decide to retain the case for later use, or to discard it. The retaining is done by checking the check box and clicking the “Apply” button.

Description	
Sending production units	[1506]
Receiving production unit	1509
Aquaculture Site	6
Capacity of site	4680
Individ Count in sending unit	330422.0
Operation Date	Thu Aug 05 00:00:00 CEST 2010
Number of startvation days	8
Vaccines	Alphaject Micro 6
Hatcheries	38
Species Origins	AquaGen
Current Temperature	13.7 C
Temperature Trend	IncreasingWeakly
Current Oxygen Level	96.3 %
Oxygen Trend	IncreasingWeakly

Solution	
Risk	<input type="text" value="Low"/>
Percentage dead	<input type="text" value="0.13159626266614%"/>
Aggregated Count	<input type="text" value="170"/>

Figure 12.12: Revise dialog

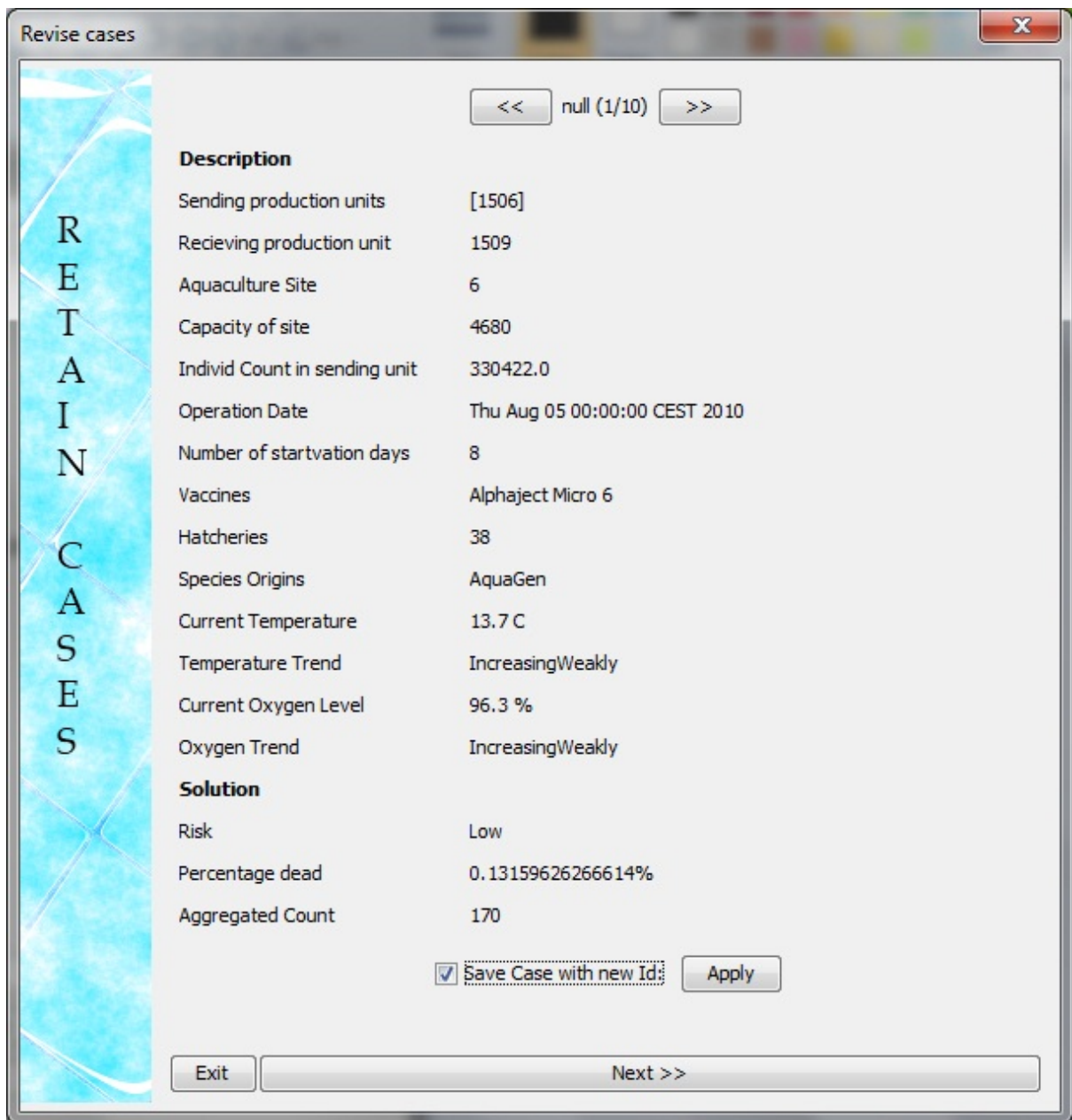


Figure 12.13: Retain dialog

The user has now completed the whole CBR cycle, and can now choose to exit or to go again.

Part III

Results

Chapter 13

Evaluating the system

This chapter contains the evaluation done on our system Glaucus. A good way of evaluating a CBR system, is to get an expert within the domain to examine the retrieved cases, and evaluate if the results are satisfying. We did not have time or the opportunity to do this, so the only “user tests” we have performed is by checking if the system returns the right cases by creating query cases which already exists in the database. The system walk through in Chapter 12 Section 12.4 proves that it does just that.

In addition we have used the built in library in jColibri to evaluate instead. We have used two different evaluators, The LeaveOneOut Evaluator and The HoldOut Evaluator. The results can be found in the following two sections.

13.1 Leave One Out Evaluator

This methods uses all the cases as queries. In each cycle one case is used as query. The code for the evaluator is shown in Listing 13.1. The “this” used in Line 2 is a object of the CBRApplication, which contains the basic methods for a CBR application in the jColibri framework.

Listing 13.1: LeaveOneOutEvaluator code

```
1 LeaveOneOutEvaluator evaluator = new LeaveOneOutEvaluator();
2 evaluator.init(this);
3 evaluator.LeaveOneOut();
4
5 System.out.println(Evaluator.getEvaluationReport());
6 jcolibri.evaluation.tools.EvaluationResultGUI.show(Evaluator.
    getEvaluationReport(), "Evaluation", true);
```

In addition to the code in Listing 13.1, two lines of code must be added inside the cycle method straight after the part where we have retrieved the most similar cases. This code is shown in Listing 13.2.

Listing 13.2: LeaveOneOutEvaluator code inside cycle

```
1 Double result = new Double(eval.iterator().next().getEval());
2 Evaluator.getEvaluationReport().addDataToSeries("Similarity", result);
```

The result variable is the similarity between the best matching case and query, 1 to 0. This number is added to a data series which is used to create a graphical representation of

the overall similarity for all cases. This code is also applicable to the HoldOutEvaluation code listed below in Section 13.2.

13.1.1 Output from evaluation

Table 13.1 illustrates some of the key numbers from the evaluation. The Number of Cycles is actually the number of cases in the case base, since each case is used as a query to the CBR cycle. Each cycle uses approximately 2.6 seconds and 286.5 seconds overall.

Number of Cycles:	109
Time per Cycle:	2629.1100917431195 ms
Total time:	286573 ms

Table 13.1: Leave One Out Evaluator Result

Figure 13.1 shows the graphical representation of the similarity of each case. Case one(1) is on the far left, while case one hundred and nine(109) is on the far right. The similarity for each case is mostly in the 1.0 - 0.8 region with some exceptions.



Figure 13.1: Leave One Out Evaluation Chart

13.2 Hold Out Evaluator

This method splits the case base in two sets: one used for testing where each case is used as query, and another that acts as normal case base. This process is performed several times and the test cases are selected randomly from the case base for each repetition.

Listing 13.3: HoldOutEvaluator code

```
1 HoldOutEvaluator evaluator = new HoldOutEvaluator();
2 evaluator.init(this);
3 evaluator.HoldOut(_testPercentage, _numOfRep);
4
5 System.out.println(Evaluator.getEvaluationReport());
6 jcolibri.evaluation.tools.EvaluationResultGUI.show(Evaluator.
    getEvaluationReport(), "Evaluation", true);
```

13.2.1 Result, 5 % split and 4 repetitions

With 109 cases in the casebase and a split in 5 % we get 5 test cases. The result with exact similarity numbers from the first repetition are shown in the Table 13.2. The test cases all score a similarity of 0.85 or above with the case base, which is quite good.

Test Case	1 repetition
1	1.0
2	0.8962910714285715
3	0.8571428571428571
4	0.9707767857142857
5	0.8858097880061114

Table 13.2: Output HoldOut 5% split first repetition

The number of cycles is equivalent to number of test cases times repetitions. In this case it is twenty(20), as in four(5 test cases) times four(4 repetitions). The reason for the high Time per Cycle is a bit of a mystery, and we are not quite sure why this has happened. The time should be less for each cycle since the case base size is reduced by 5% compared to that of Leave One Out Evaluation in Section 13.1.

Number of Cycles:	20
Time per Cycle:	5278.3 ms
Total time:	105566 ms

Table 13.3: Hold Out Result 5%

The whole result is illustrated in Figure 13.2, with all cases from all repetitions.

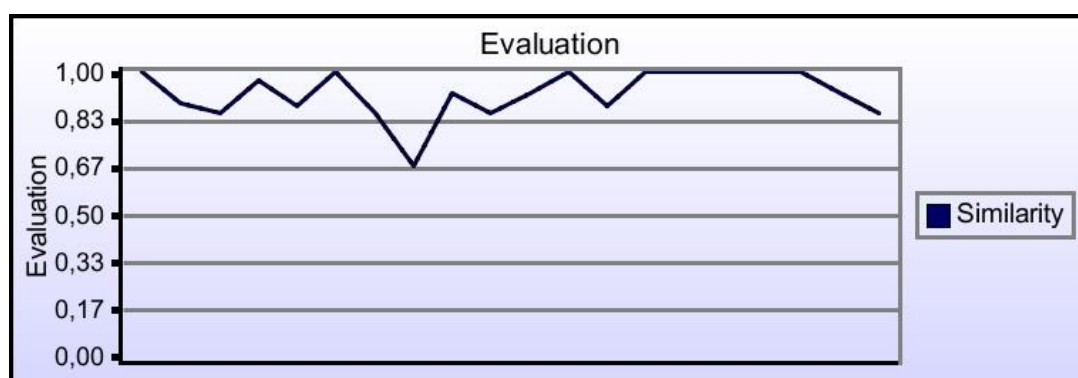


Figure 13.2: Hold Out Evaluation chart, 5% 4 rep

13.2.2 Result, 10 % split and 4 repetitions

A split on 10 % and 109 cases in the casebase, results in 10 test cases. The first repetition can be seen in Table 13.4. Also here we acquire quite good results with a minimum similarity of 0.83 for test case 3 in the first repetition.

Test Case	1 repetition
1	0.885214880952381
2	0.8571428571428571
3	0.8340272472625415
4	1.0
5	0.9704035714285715
6	1.0
7	0.9563154761904763
8	1.0
9	0.8571428571428571
10	1.0

Table 13.4: Output HoldOut 10% split first repetition

The Time per Cycle has also halved from the 5% split evaluated above. The reasons are to us unknown, and the same run on other machines yield the same results. Number of cycles are forty(40), with a total run time of 104.7 second.

Number of Cycles: 40
Time per Cycle: 2617.8 ms
Total time: 104712 ms

Table 13.5: Hold Out Result 10%

The graphical representation of the run is shown in Figure 13.3. As the figure shows, there are not many times the query case receives a similarity below 0.8 with a case from the case base.

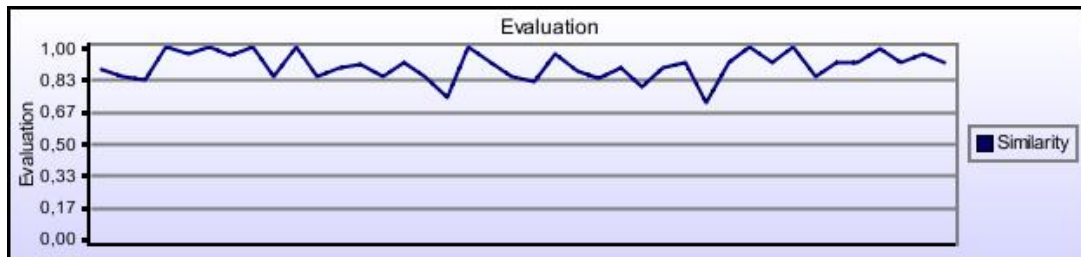


Figure 13.3: Hold Out Evaluation chart, 10% 4 rep

The results we have obtained, illustrated in Figures 13.1, 13.2 and 13.3, point to a system which is able to find similar cases in our case base. We base this on the observation of overall similarity which roughly varies between 0.8 and 1 in all of the above evaluation modes. The evaluation of the system was however, not the most important factor in this project, and we only use this part as proof of concept. A more explicit evaluation would in the future be of interest and in the next chapter we discuss the need for a thorough evaluation of the system, both with additional methods and with a domain expert.

Chapter 14

Conclusion and Future Research

In this thesis we have presented Glaucus, a Case-Based Reasoning system for decision support in the fish farming domain. The system uses past cases(situations) to solve a new case(situation), which is created on the basis of user input and continuous data flow from the aquaculture site. Glaucus is specialized at working with situations which are related to fish sorting operations between aquaculture production units on the site. We have seen that the mortality rate before and after an operation is increasing, but it varies to some degree. Glaucus recognizes the current situation from the case base and uses the outcome/result of that situation to output a risk with the current situation. The case base(memory of situations) is composed of real-world situations which we have captured from raw data in an industrial database supplied by SINTEF Fisheries and Aquaculture.

Glaucus is built in the programming language Java, with the support of frameworks and technologies such as jColibri, myCBR, Hibernate, MySQL and Weka. The work we have done has been in parallel with a group at SINTEF Fisheries and Aquaculture, and our hope is that the job we have done will serve as a foundation for future work, not just at SINTEF, but also for other interested students.

One minor weakness with the system is that it is based on continuous data flow from the fish farmers and sensor data. Therefore it is quite difficult to test the system with a hypothetical case. This is related to how we create our query case through a combination of sensor data and input from our Graphical User Interface. Extending the system to work with hypothetical cases would be a nice feature in future versions.

With regards to future work we look especially at the possibility of adding additional data to the system. The data most suited would be data which describe the different procedures in connection with the sorting operation, such as; What kind of boat is used? How many people are handling the fish? What kind of equipment is used during the operation? Are there any environmental data from outside the water? There are countless possibilities for extending the data basis. The reason we see this as important is that the fish death seem to vary on different sites, given relatively equal situations. There might be something in the procedures at the different sites which make the big difference.

As testing of the system is concerned, we have used methods integrated in the jColibri framework, which use cases from the existing case base as test cases to the system. This is not the optimal way of testing a system of this type, but due to time constraints both we nor a domain expert had time to test the system in detail. We do however, have great confidence in such type of system in the fish farming domain, and we believe that we, in

the not so distant future, will see commercial CBR systems used as decision support in the fish farming domain.

14.1 Evaluation of goals

14.1.1 Assess the use of CBR in the fish farming domain

Our system is just a foundation for what we hope will be a bigger and more comprehensive system in the future. The assignment focused on exploration of Case-Based Reasoning in a domain which is not well-known; fish farming. We have assessed the work/studies with Case-Based Reasoning in combination with aquaculture through a theoretical study. The related research we have focused on has roughly speaking been from the last couple of year up to a decade old. It is quite clear that the amount of research is increasing world-wide and we also hope this is a start of something big here in Norway. The studies done seem very positive about the usefulness of Case-Based Reasoning in the fish farming domain, and we have also shown that a CBR approach might be beneficial in the domain.

14.1.2 Create a decision support system based on CBR

Our second and largest goal was to create a decision support system based on CBR in the fish farming domain, with emphasize on fish death during certain human interactions with the fish. The operation considered was sorting operations.

Frameworks

In the study we investigated the use of the jColibri framework to create a CBR application and also tried to incorporate some of the contribution made to the framework, such as with myCBR wrapper functions and ARFFConnector. The implementation was a success as it works within its given limits. We did however have certain problems with the framework due to its complexity and some insufficient documentation. A lot of time was used on debugging the application when we had problems with the mapping structure between data models, Hibernate and jColibri.

Some of the problems/issues were strongly related to the use of Hibernate, where we had no prior experience and knowledge. The simple examples were simple enough to understand, but when we started implementing a rich case structure, we started to get lots of different issues with simple mapping files, objects which were already saved to session and lazy loading of objects. Most of the issues were solved by debugging and reading documentation and forum posts with similar issues.

A problem in particular was the use of inheritance in case components. Initially we created a abstract Java class called MeasurementInstance. The class had four fields/attributes with getters and setters. We then created two children, classes TemperatureInstance and OxygenInstance which inherited MeasurementInstance. The reason we did this was that we wanted each individual MeasurementInstance to be its own type if we ever wanted to add something new to them. The problems started to come when we tried to assign similarity functions to TemperatureInstance and OxygenInstance. For instance,

we tried to use a simple Interval similarity on each classes value-attribute, as shown in Listing 14.1.

Listing 14.1: jColibri issue

```
1 NNConfig nnConfig = new NNConfig();
2 nnConfig.addMapping(new Attribute("value", TemperatureInstance.class), new
  Interval(param));
3 nnConfig.addMapping(new Attribute("value", OxygenInstance.class), new
  Interval(param));
```

When we used this configuration we got an exception that the field “value” was not found in TemperatureInstance or in OxygenInstance. What we found out is that the jColibri code uses a call to `Class.getField(“value”)` which essentially looks for a field declared *in* the given class. Our solution was to create an Interface called MeasurementInstance and let the children create the fields and methods them selfs. This was however after we tried to just use the MeasurementInstance alone for each of Temperature and Oxygen. The problem with that is that you link the similarity to a specific class and not to the variable names and there is therefore no way to assign to separate similarity functions to objects of the same class.

Case Structure

The case structure in Glaucus was discussed in detail during several meetings between people with different views and disciplines. It was a long process which ended the last week before the easter break and was also of subject to discussion and arguments between the project members throughout the implementation.

The case structure is composed of several different case components ranging from simple numbers, to complex attributes, which again are composed by other components. Trends are a big part of the case structure where we use past instances of environmental measurements to find abstractions which are more readable by humans.

The case generation was a part we used a lot of effort. Our vision was that it should be easy to add own attribute with a couple lines of code. We had to abandon this vision due to the amount of work this would take, which essentially would make the system non functioning with regards to the Case-Based Reasoning phases. There was simply not enough time to construct a system based on this.

Other Methodical goals

The risk assessment in Glaucus is very primitive. It relies on using the aggregated death count and the amount of fish in a production unit to establish a simple High - High-/Medium - Medium - Medium/Low - Low scale. A more sophisticated risk assessment is left as future work.

The use of Protegé and myCBR was a success and it was fairly easy to implement the generated similarity functions in the jColibri framework. A further use of this would probably be with the use of ontologies, which would be very interesting.

As far as the Graphical User Interface goes, we simply based it on the GUI used in the Travel Recommender system from the jColibri 2 tutorial. The interface is neat and

very easily expendable, but a more suitable replacement in the future would perhaps be with a web application.

14.2 Future Research

There are many issues and aspects which should be taken into consideration with further work with this application. One of the main focuses should be to have a reference group or domain expert who can evaluate the application in order pinpoint obvious flaws with the system.

The Case Structure of Glaucus, described in Chapters 9 and 10.1, is also subject to future work. There are some attributes which were omitted in our final structure in order to get a working system. One important attribute related to fish farming is the feeding, as was emphasized by Arnfinn Aunsmo at one of our meetings during the project. We actually made the data models for the feeding attribute but never got around to implement the data field population due to the strict time schedule we had for the implementation part of the project. Another attribute which was taken into consideration was tracking of “normal” or daily fish death prior to an operation. If we track the death we could establish a normal death trend and maybe use this to predict and evaluate the solution of a case based on this.

The monitoring systems at the fish farming sites we have data from are being upgraded, which means that in the future there will be data available for each individual aquaculture production unit and not just global data for the specific site. Additional information about each specific operation is also due to be available, with variables such as:

- Which type of boat has been used during the operation
- How many people worked during the operation
- Which procedures were followed

The air temperature is also an attribute which could be of interest due to the nature of how the fish are sorted during a sorting operation. The fish are actually handled outside the water, which may have a direct correlation with fish death.

In our system we concentrated on creating cases based on many sending production units to one receiving production unit. The actual relationship in a case in the domain when we look at Sorting operations are many to many. This means that what we have made cases of are actually sub operations. What we envision is a case structure based on something similar to this:

- **Case**
 - OperationDate
 - OperationType
 - Site
 - List of sub operations(such as our cases)

An example of a case would then possible be the following:

- **Case 5**

- OperationDate: 2011-05-06
- OperationType: Sorting using Well-boat
- Site: 2
- Sub operations:
 - * Sub operation: 1
 - Sending prod units: 1412, 1414, 1415
 - Receiving prod unit: 1420
 - additional attributes++
 - * Sub operation: 2
 - Sending prod units: 1412, 1413
 - Receiving prod unit: 1421
 - additional attributes++
 - * additional sub operations++

The significance of the time between a fish group has been deployed and it is sorted could also be important. In a later version this should be taken into consideration. The possibility for analyzing which type of fish dies could also be interesting to look at. Is there a normal distribution of death among the fish if we look at the size and weight of the fish. Are smaller fish more vulnerable to sorting operations or is it the other way around?

In this project we also took a look at generating case descriptions and solution dynamically through the use of a contribution to jColibri, ARFFConnector. ARFFConnector is a connector for (Weka) ARFF-files, and can be used to generate the case components without having to write the code for it. It would be interesting to see if it is possible to use this to create complex case components with several compound attribute in a rapid way. This would make it very easy to extend the system with additional attributes if they should be made available on a later stage.

The similarity functions presented in Glaucus, Chapter 11, are based on the built-in function in jColibri, myCBR similarity functions and some custom made. A further assessment on which similarity functions are most applicable for each attribute in the data set should be conducted. The choices made in this regard for our system is lousily based on assumptions about the domain and some discussion with co-workers. In the similarity configuration dialog for Glaucus there is a possibility for weighting each feature/attribute through some graphical sliders. These are set to a default value for the slider, which is one(1). One possibility is for the system to learn these weights through training when we input cases into the system. Wettschereck and Aha proposed a framework of automated weight-setting methods which set weights to feature according to their relevance with little or no domain knowledge[Wettschereck and Aha, 1995]. It would be interesting to try this or something similar.

Another aspect should be to extend the system with functionality for use of different operations, such as De-lousing and Deployment. The operation supported by our system

is the Sorting operation, more precisely Sorting using well-boat. The De-lousing operation is an operation which has a very high cost, often in millions, and it is therefore very important that the fish death connected to the operation is as low as possible. The industry has however started to assess the use of wrasse(fish) to eat the lice as opposed to chemical treatment. An example where things did not work out in favor of some fish farmers was in December 2009, in Flekkefjord[Fish.no, 2009]. Fish worth millions died after a chemical De-lousing operation on two aquaculture sites. The operation was done to be sure that they did not have any lice even though they also used wrasse to hold the amount of lice down.

There is no direct general domain knowledge in our system. The system does not know whether the water is hot or cold, it only has a numerical value without knowledge of the significance of it. There are some basic abstraction in the systems such as the temperature and oxygen trends which are calculated from the Linear Approximation of environmental data. A typical scenario where the general domain knowledge would be used is if for instance the water temperature was above a certain threshold, hot, and the oxygen level was below another, low. Low oxygen level combined with a high temperature should immediately be regarded as a scenario where we should not do the operation. The general domain knowledge should be acquired through working with or interviewing the fish farmers, in order to see what they emphasize in their decision making process.

There should also be some kind of adaption in the system. The Reuse phase of the CBR cycle is in our system limited to only reuse the exact solution from the retrieved case. There are methods for adaption provided by the jColibri framework, and this would be a very interesting point to look at in the future. Our plan was originally to look at adaption, although it was out of the scope of the assignment, if time was on our side.

Another interesting point is to look at the possibility of using some kind of prediction in the system. A possible scenario is that the fish farmers are determined to do a sorting operation on a site a week into the future. What our system essentially does is that it create a query case based on the environmental data for the x last measurement instances from the operation date. If for instance todays date is 2011-06-01 and we propose an operation on 2011-06-08, we would in a best case scenario have environmental data in the past and too 2011-06-06(today). When this case is used a query for the system, the retrieved cases' solution might not be applicable for this query case due to changes in the environment the last seven days. By predicting the future temperatures, as shown in Figure 14.1, we can retrieve the cases which are more similar based on the approximations of the past instances.

This also opens up a whole new chapter where we might also look at predicting the environment after the operation has been committed. Lets say that a sorting operation has been acted out, given relatively good environmental data and positive solutions. What if the environment changes drastically over the next fourteen days? The fish might be very vulnerable and weak after being exposed to some tough processing in the sorting operation. If this is the case there might be lots of unnecessarily death in the production unit.

In our example in Figure 14.1 we use a linear approximation to find a function for the data points in the form of $y = a + bx$. There might be other approximations which are better than this one, and this should also be examined in the future.

The risk assessment in Glaucus is pretty primitive. The risk is calculated by dividing

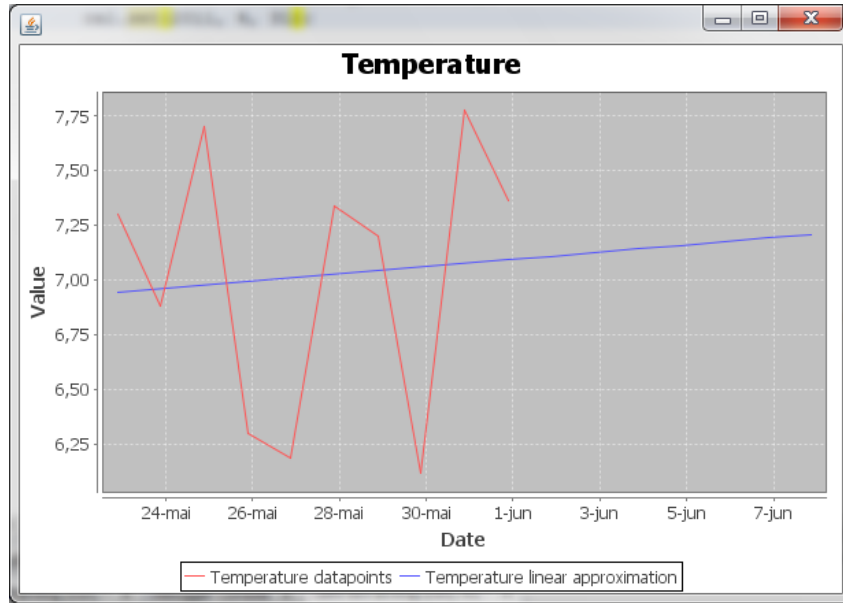


Figure 14.1: Temperature prediction

the total death count after an operation by the amount of fish present in the production unit. What we get from this simple arithmetic operation is a decimal number between 0 and 1. The risk assessment is then based on if-then statement where for instance a high risk could be given by a number above 0.2 or 20% death. A more complex risk assesment would be quite useful, and again it comes down to interviewing the people with the right knowledge to find out what they regard as high and low risks. There might be additional attributes which are considered besides the death count, such as the cause of death. Two thousand(2000) fish which died from mechanical injuries might not be as bad as three hundred(300) fish which died due to sores.

Use of additional technologies such as ontology, which is supported by jColibri[Recio-garcía et al., 2006], machine learning or perhaps model base adaption[Branting et al., 1999] are all things that might be worth taking a look at in the future.

Bibliography

- A Aamodt. Knowledge-intensive case-based reasoning in creek. *Advances in Case-Based Reasoning, Proceedings*, 3155:1–15, 2004. Times Cited: 13 Funk, P Calero, PAG 7th European Conference on Case-Based Reasoning Aug 30-sep 02, 2004 Madrid, SPAIN.
- A. Aamodt and E. Plaza. Case-based reasoning - foundational issues, methodological variations, and system approaches. *Ai Communications*, 7(1):39–59, 1994. Times Cited: 1114.
- Ray Bareiss. *Exemplar-based knowledge acquisition : a unified approach to concept representation, classification, and learning / Ray Bareiss*. Perspectives in artificial intelligence ; v. 2. Academic Press, Boston :, 1989. Includes index. Bibliography: p. 159-166.
- Karl L Branting, John D Hastings, and Jeffrey A Lockwood. Integrating cases and models for prediction in biological systems. *AI Applications*, 11:29–48, 1999.
- Steven Craig and Louis A. Helfrich. Understanding fish nutrition, feeds, and feeding. pages 1–4, 2009.
- P. Cunningham. A taxonomy of similarity mechanisms for case-based reasoning. *Knowledge and Data Engineering, IEEE Transactions on*, 21(11):1532–1543, 2009.
- Pádraig Cunningham, Dónal Doyle, and John Loughrey. An evaluation of the usefulness of case-based explanation. In Kevin Ashley and Derek Bridge, editors, *Case-Based Reasoning Research and Development*, volume 2689 of *Lecture Notes in Computer Science*, pages 1065–1065. Springer Berlin / Heidelberg, 2003.
- Belén Díaz-Agudo, Pablo Gervás, and Pedro González-Calero. *Poetry Generation in COLIBRI*, volume 2416 of *Lecture Notes in Computer Science*, pages 157–159. Springer Berlin / Heidelberg, 2002a.
- Belén Díaz-Agudo, Pablo Gervás, and Pedro González-Calero. Cbronto: A task/method ontology for cbr, 2002b.
- FAO. Global aquaculture production 1950-2008, 2008a.
- FAO. The state of world fisheries and aquaculture - 2008 (sofia). Technical report, FAO Fisheries and Aquaculture Department, 2008b.
- FAO. The state of world fisheries and aquaculture - 2010 (sofia). Technical report, FAO Fisheries and Aquaculture Department, 2010.

- T Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine Learning*, 31(HPL-2003-4):1–38, 2004.
- Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. *From data mining to knowledge discovery: an overview*, pages 1–34. American Association for Artificial Intelligence, 1996.
- Fish.no. Massiv fiskedød etter avlusing, 2009.
- D. E. Forsythe and B. G. Buchanan. Knowledge acquisition for expert systems: some pitfalls and suggestions. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(3): 435–442, 1989.
- Kaja Frøysa. Lite omega 3 i oppdrettslaks, February 2011. URL <http://www.nrk.no/helse-forbruk-og-livsstil/1.7487978>.
- Zetian Fu, Guoyong Hong, and Jian Sun. *CASE-BASED REASONING MODEL OF THE FISH DISEASE DIAGNOSIS*, pages 1433–1442. IFIP International Federation for Information Processing. Springer Boston, 2009.
- GAIA. Gaia people description, 2011.
- Marte Garaas and Geir Ole Hiåsen Stevning. Intelligent fish farming. 2010.
- Anne-Marit Gravem. Integrating case-based and bayesian reasoning for decision support. 2010.
- Wang Guirong. A fish disease diagnosis expert system using short message service. volume 3, pages 299–303, 2009.
- Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.
- J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.
- Ronald A. Hites, Jeffery A. Foran, David O. Carpenter, M. Coreen Hamilton, Barbara A. Knuth, and Steven J. Schwager. Global assessment of organic contaminants in farmed salmon. *Science*, 303(5655):226–229, 2004.
- Grete Holstad. Trøndelags største lakserømming noensinne, February 2011a. URL <http://www.adressa.no/nyheter/sortrondelag/article1590136.ece>.
- Grete Holstad. Må slakte 700 000 laks pga sykdom, April 2011b. URL <http://www.adressa.no/nyheter/nordtrondelag/article1620963.ece>.
- Yuan Hongchun, Mao Zhuo, and Zhao Bo. Expert system based on cbr and grey bp for vannamei disease diagnosis. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 3, pages 1015–1019.

- Park Jeong-Seon, Oh Myung-Joo, and Han Soonhee. Fish disease diagnosis system based on image processing of pathogens' microscopic images. In *Frontiers in the Convergence of Bioscience and Information Technologies, 2007. FBIT 2007*, pages 878–883, 2007.
- Janet L. Kolodner. Reconstructive memory: A computer model. *Cognitive Science*, 7(4): 281–328, 1983.
- Janet L. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34, 1992.
- Ben Kröse and Patrick van der Smagt. *An introduction to Neural Networks*. Amsterdam, 8 edition, 1993.
- David B. Leak. *Case-Based Reasoning: Experiences, Lessons and Future Directions*. MIT Press, 1996.
- Daoliang Li, Zetian Fu, and Yanqing Duan. Fish-expert: a web-based expert system for fish disease diagnosis. *Expert Systems with Applications*, 23(3):311–320, 2002.
- Daoliang Li, Wei Zhu, Yanqing Duan, and Zetian Fu. *Toward developing a tele-diagnosis system on fish disease*, volume 217 of *IFIP International Federation for Information Processing*, pages 445–454. Springer Boston, 2006a.
- Nan Li, Zetian Fu, Ruimei Wang, and Xiaoshuan Zhang. Developing a web-based early warning system for fish disease based on water quality management. In *Industrial Electronics and Applications, 2006 1ST IEEE Conference on*, pages 1–6, 2006b.
- Nan Li, Ruimei Wang, Jian Zhang, Zetian Fu, and XiaoShuan Zhang. Developing a knowledge-based early warning system for fish disease/health via water quality management. *Expert Systems with Applications*, 36(3, Part 2):6500–6511, 2009.
- Dongmei Lou, Ming Chen, and Jiechao Ye. Study on a fish disease case reasoning system based on image retrieval. *New Zealand Journal of Agricultural Research*, 50(5):887 – 893, 2007.
- Thomas M. Mitchell. Does machine learning really work? *AI Magazine*, 18(3):11–20, 1997a.
- Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., 1997b.
- Thomas M. Mitchell. The discipline of machine learning. *Machine Learning*, July:1–9, 2006. Carnegie Mellon University, School of Computer Science, Machine Learning Dept.
- Bruce W. Porter, Ray Bareiss, and Robert C. Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45(1-2):229–263, 1990.
- Ross Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1993.

- Herminio R. Rabanal. History of aquaculture. Technical report, ASEAN/UNDP/FAO Regional Small-Scale Coastal Fisheries Development Project, 1988.
- Juan Recio-García. *jCOLIBRI: A multi-level platform for building and generating CBR systems*. PhD thesis, 2008.
- Juan Recio-García, Belén Díaz-Agudo, and Pedro A. González Calero. jcolibri 2 tutorial. Technical report, University Complutense of Madrid, September 16, 2008 2008. <http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri2/docs.html> ISBN 978-84-691-6204-0.
- Juan A. Recio-garcía, Belén Díaz-agudo, Pedro González-calero, Antonio Sánchez ruiz granados, and Dep Sistemas Informáticos. Ontology based cbr with jcolibri. In *Applications and Innovations in Intelligent Systems XIV*, pages 149–162. Springer-Verlag London, 2006.
- Michael M. Richter. Knowledge containers. page 7, 2005.
- Birger Ringseth. 12 800 laks rømte på hitra, April 2011. URL <http://www.adressa.no/nyheter/okonomi/article1618653.ece>.
- Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.
- Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2478–2483 vol.3.
- Roger C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, NY, USA, 1983.
- SINTEF. Create: Annual report 2009, 2009.
- Frode Sørmo. Real-time drilling performance improvement. *Scandinavian Oil & Gas*, 2009.
- Børge Sved. Salmar leverte kjemperesultat, May 2011. URL <http://www.adressa.no/nyheter/okonomi/article1637750.ece>.
- Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. *Stanley: The Robot That Won the DARPA Grand Challenge*, volume 36 of *Springer Tracts in Advanced Robotics*, pages 1–43. Springer Berlin / Heidelberg, 2007.
- Axel. Tidemann, Finn Olav. Bjørnson, and Agnar Aamodt. Case-based reasoning in a system architecture for intelligent fish farming, 2011.

- Haleh Vafaie and Carl Cecere. *Corms ai: decision support system for monitoring us maritime environment*, 2005.
- Dietrich Wettschereck and David Aha. *Weighting features*, volume 1010 of *Lecture Notes in Computer Science*, pages 347–358. Springer Berlin / Heidelberg, 1995.
- Bin Xing, Daoliang Li, Jianqin Wang, Qingling Duan, and Jiwen Wen. *AN EARLY WARNING SYSTEM FOR FLOUNDER DISEASE*, pages 1011–1018. IFIP International Federation for Information Processing. Springer Boston, 2009.
- Daniel Zeldis and Shawn Prescott. Fish disease diagnosis program – problems and some solutions. *Aquacultural Engineering*, 23(1-3):3–11, 2000.
- Xiaoshuan Zhang, Zetian Fu, and Ruimei Wang. Development of the es-fdd: an expert system for fish disease diagnosis. In *OCEANS '04. MTTs/IEEE TECHNO-OCEAN '04*, volume 1, pages 482–487, 2004.
- Wei Zhu and Daoliang Li. *A CBR System for Fish Disease Diagnosis*, pages 1453–1457. IFIP International Federation for Information Processing. Springer Boston, 2008.