Project Report
TDT4290 - Customer Driven Project
"Privacy Advisor"

GROUP 4
Ulf Nore, Nicholas Gerstle, Henrik Knutsen, Dimitry Kongevold,
Einar Afiouni, Neshahvan Karunakaran, Amanpreet Kaur

NTNU, Fall 2011

Abstract

This is report for the "Privacy Advisor" project in the TDT4290 - Customer Driven Project course. The "Privacy Advisor" software was developed as part of a larger project for SINTEF ICT, in order to investigate the applicability of machine learning to aid users in making internet privacy decisions. Previous research at SINTEF has pointed out several advantages to using a Case Based Reasoning (CBR) agent as a core algorithm in this type of application, which this project implements. SINTEF ICT has indicated the use of the project software as part of future research and therefore the emphasis was placed on implementing a broad framework, rather than actual algorithms.

The resulting product is a Java based framework with a central CBR engine that is extendable with respect to key algorithms, databases and user interfaces. It also allows for network communication with a community server that could implement a collaborative filtering approach similar to the CBR.

List of Tables

List of Figures

Contents

Introduction

This report describes the development of a machine learning systems for aiding users in Internet privacy decisions. The software is titled "Privacy Advisor" and uses a case based reasoning (CBR) approach, which is a learning method that seeks to predict preferences based on previous user choices. The project is a part of the course TDT4290 - Customer Driven Project and is a part of a SINTEF ICT research project in privacy agents. The report is written in a chronological order where each chapter represents a distinct "phase" in the development process. In reality, of course, there are no crisp boundaries, but the structure here provides a useful structure for reasoning about the process.

The structure of the report is as follows. Part I describes the initial phase of the project; the projects directive (Chapter I), the planning phase (Chapter 2.4.1) and the preliminary study phase (Chapter 1.7). The project directive gives a high level overview over the project, its objectives, how they are reached, project scope, resources available etc. The preliminary study comprise the first weeks of the project, and is a consistent effort to better grasp the problem at hand, identify how it can be broken into subproblems and what tools are required to solve these problems. It also seeks to identify to some extent what the priorities in the project are; that is, given scarce resources, which objectives are prioritized. The project planning phase seeks to make a more fine grained decision on how resources are best allocated over time and to the various tasks that comprise the project.

Part II describes the requirements specification (Chapter II), design phase, implementation and documentation. The requirements specification is a contract between the customer and the project team where the the requirements to be satisfied by the software are stated explicitly. Based on the requirements, a design (Chapter 4.6.3) is made, which is to serve as a guideline for implementing (Chapter 5.4.3) the software system.

Finally, Part III describes the testing and evaluation phase, and summarizes the project.

Part I

Initial Phase - Planning and Research

Project Directive

Contents

1.1 Purpose

This document describes the mandate, background, resources available and organizational structure of the project Privacy Advisor, henceforth referred to as the project.

1.2 Mandate

The purpose of this project is to implement the key functionality of a privacy agent as described in Nyre and Tndel (2010), that provides users with advice in making Internet privacy decision.

1.2.1 Background

This project is a part of a larger research project at SINTEF ICT that studies approaches to handling Internet privacy related issues. The underlying idea is that while users are

often concerned about the way various websites and services handle private information about them, obtaining information about this is very costly as privacy policies tend to be very long documents formulated in an inaccessible language. This has led to the idea that Internet privacy can be handled by machine learning techniques, where a particular decision is based on the users past behavior and the behavior of similar users.

Nyre and Tndel has then proposed a Privacy Agent structure that uses the case based reasoning (CBR) method for giving privacy advice. CBR is in many ways similar to the way human experts reason about problems; that is, by looking at what has been done in similar cases previously. Nyre and Tndel also describes this CBR approach to be complemented by a community database where the same information is stored, allowing for a second lookup that uses a collaborative filtering, that is, making a decision based on the behavior of similar users.

1.3 Objectives

This project identifies three key objective, arranged by order of importance:

1. Implementing a testing framework of CBR based privacy agent that is able to make privacy decisions based on previous user behavior.

2. Implement the community system/collaborative filtering part of the agent.

3. Extend the system to other standards for machine readable privacy policies.

4. Implement the system as a browser plugin.

Implementing a browser plugin considered least important, as it very much contingent on the success of early testing. It is also given a low priority given the relative small portion of major websites that implement P3P.

1.4 Resources and Duration

The system in its complete form is to be demonstrated on November 24 2011. For the project period, a total of 25 hours per week per project member is planned. With seven group members and a project spanning 13 weeks, this adds up to approximately 2300 hours.

1.5 Organization

Project management is based a standard model where the customer takes on the role of *project owner* or simply "owner". The owner is the actual stakeholder and initiator of the project, and responsible for all executive decisions in the project.

The *project group* or *team* is responsible for delivering the product in accordance with the wishes of the customer as defined by the *requirements specification* document. Two project management roles are designated, one is responsible for administrative decisions, hereunder planning, reporting, calling meetings, customer contact and so forth. The second management role is that of chief system architect, who has the responsibility and final word in all technical decisions.

The project group organization is based on the modules of the system that is being implemented, this is often referred to as a *functional* structure or organization. One group member is responsible for developing one particular feature. This organization is shown in Table 1.1. In addition to this internal functional organization, two project managers are appointed, one having responsibilities for administrative decisions and reporting and one with responsible for technical decisions.

1.6 Planning

A project plan has been developed for the purpose of communicating expectations and progress within the group and to the customer and the advisor. The plan also serves as an aid in identifying problems and project management. For the software development process, a hybrid waterfall model has been chosen.

1.7 Limitations and Scope

The primary focus of this project is on developing a framework that allows for testing the CBR privacy agent framework. This entails building a module for parsing policy documents in XML format, a data structure for holding policy information in memory (henceforth "policy objects"), a set of exchangeable distance metric that compares policy objects, a generic retrieval algorithm (such as k Nearest Neighbors) that works with any distance metric and methods to store and update a knowledge base. Being a part of an ongoing research project, reusability and modularity are important success factors for in evaluating the project. This means that it should for instance be simple to swap P3P with some other privacy policy standard, that different distance metrics should be applicable, new metrics could easily be implemented and so forth.

Table 1.1: Responsibilities

Area	**Role**	**Description**	**Responsible**
Administrative	Project Manager	Customer relations Requirements specification Planning Meeting minutiae Status reporting Project report	Ulf Nore
Administrative Technical	Head Systems Architect	Overall design. Design report. User documentation. Technical Decisions.	Nicholas Gerstle
Technical	Data Storage/Databases	Flat file data storage system. Database systems .	Amanpreet Kaur
Technical	CBR - Algorithms and Data structures	Data structures for storing privacy policy information. Define and implement similarity metrics. Retrieval and learning algorithms. Parameter storage.	Dimitry Kongevold, Neshahavan Karunakaran
Technical	Testing and Evaluation	Design test cases. Criteria/methodology for model testing.	Henrik Knutsen
Technical	GUI	Implement a simple GUI for testing model framework.	Ulf Nore
Technical	Version control	Set up and maintain code repository.	Einar Afiouni
Technical	XML/P3P Parser	Implement P3P parser that produces inputs to CBR	Einar Afiouni

1.8 Quality Assurance

Preliminary Study

Contents

This section describes the preliminary study phase of the project. It focuses on gaining insight into the problem the software system is to solve. For this project, the problem is very well defined from the customer's perspective as such, little time is spent looking into similar products and various possible solutions to the overarching problem of computer aided privacy advice.

A large portion of this time was spent on investigating the proposed solution and the technologies involved, that is Case Based Reasoning (CBR) and the technologies used by it. As the customer, SINTEF ICT, is a research institution, and the project can somewhat simplified be viewed as the "implementation" part of a larger research project, the customer could from the start provide relatively clear specification of the system that is envisioned[1].

Another critical work laid down in this phase were choices with respect to project scope and development model. As will become apparent, these choices are strongly related to the nature of the problem statement.

[1]The system is outlined in broad terms in the article Inger Anne Tndel, smund Ahlmann Nyre and Karin Bernsmed: *"Learning Privacy Preferences"*, SINTEF ICT 2011.

2.1 Internet Privacy Technology

This section briefly describes the situation today with respect to Internet privacy tools and which needs the project seeks to address.

As discussed in Tndel et al. (2011), there is a disproportion between the average Internet user's concern for privacy and the actual control he has over private information. While most websites provide a privacy policy, these tend to very long documents written in a obscure "legalesque" language, intended for protecting the websites' interests than those of users. To aid this problem, machine readable standards such as P3P have been introduced. P3P seeks to compress the information contained in the privacy policy in an XML document that can be parsed and summarized by computer programs.

2.1.1 Current Privacy Technology

The AT&T *Privacy Bird* is mentioned in Tndel et al. (2011) as an example of current Internet privacy software. Privacy Bird can parse P3P documents and match a site's privacy policy with the user's preferences. The problem with this however, is that the user has to explicitly state his preferences, which, first of all, is a rather time-consuming endeavor. Secondly, while a user may have clear notions about which information he would share in a *particular* situation, such preferences may be very hard to generalize. It may be hard to give a general statement on privacy preferences in a way that is both simple enough for the user to understand and rich enough to actually describe the problem. Furthermore, preferences may be inconsistent, for instance, while in the general case, the user would not accept privacy terms similar to those of for instance Facebook; however in Facebook's particular case he will accept them nevertheless.

2.1.2 How Machine Learning Can Improve Privacy Advice

The novel feature of this project is to introduce an intelligent system that *learns* the user's preferences, seeking to limit the amount of user interference. This is to be achieved by the use of CBR agent, that can look at previous examples of user choices in similar situations. A particular advantage of the CBR approach is the feedback loop where the system can actually *explain* its choice in terms of similar cases which sets CBR apart from alternative reasoning models such as artificial neural networks. It also allows for better tuning to user response in those cases that the user disagrees with the recommendation.

2.2 Case Based Reasoning

Being, the core part of the system to be developed, some time needed to be spent on looking into CBR. In vague terms, CBR is problem solving based on past solutions to previous problems. This approach is similar in many ways to the way humans solve problems, both as domain experts, and in their daily life. For instance, a software engineer, faced with a particular problem, may identify similarities to a problem he has previously solved, using for instance a factory design pattern, so he adopts this solution to the new problem with some modifications. Similarly, a NTNU student, hungry for a late night snack, recalling past experience with favorable opening hours and culinary excellency, heads off to Sesam.

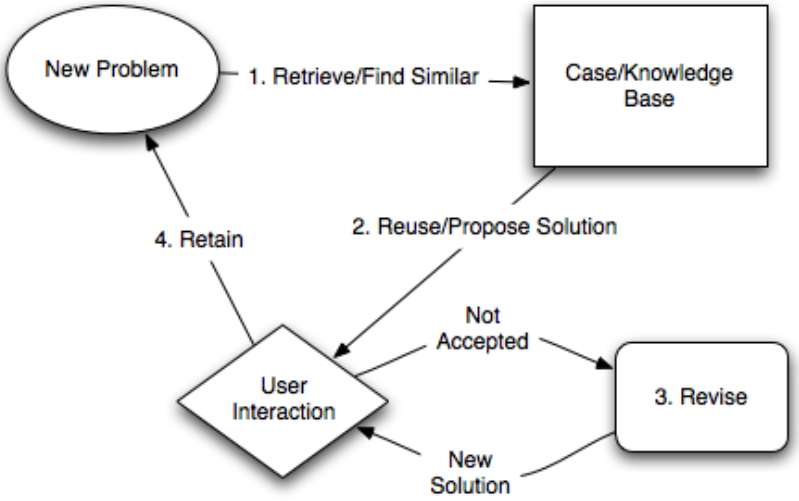


Figure 2.1: A Simplified CBR Cycle.

For computer implementation, CBR is usually formalized in terms of a four step process:

- Retrieve: Given a new site, the agent will retrieve from its knowledge base, the set of cases deemed the most similar to the one at hand. This means, that if presented with the site Facebook, for instance, the agent finds Twitter, Google and LinkedIn to be the sites that have the most similar policy to Facebook.
- Reuse: Look at the decisions made about the cases that were found and adapt this decision to the problem at hand. In this case, the agent needs to see if there are strong enough indications toward a particular behavior with respect to the type of site that is at hand. If for instance, the user has accepted the policies of all the similar sites found, he is also likely to accept that of Facebook.

- Revise: Once a conclusion is reached, it is presented to the user (along with the background for why it is reached). The user may then choose to accept the conclusion, or to overrule it, providing the system with directions as to why it was wrong. This may in turn cause the agent to update its parameters accordingly.
- Retain: Finally, the new case is stored to the database along with the users decision as a reference for the next time the same site is opened, and as a case to employ when evaluating new sites.

As can be seen from this specification, CBR is a very generic approach, and several notions need to be defined before it can be implemented. For instance, we need to settle on a knowledge representation. For most CBR purposes, a frame based approach is taken. To store this knowledge representation, an appropriate database structure must be selected, and a routine for keeping this up to date must be established. Given a representation, one needs to define a notion of *similarity* between cases, and an algorithm to do the actual retrieval. Usually, one would also like this algorithm to provide some measure of the certainty of the results as well.

While SINTEF has proposed some initial ideas for how these details can be implemented, they are very much open problems, and subject to empirical study to find an 'optimal' approach for the actual problem at hand.

2.3 Project Scope

The research nature of this project makes it a very open-ended one. The clearly most important part of the system envisioned by the customer is the CBR engine, that classifies websites based on the user's previous decisions. Implementing this is clearly necessary, regardless of which direction is followed and what limitations are made.

However, once the local CBR engine and auxiliary modules such as parsing P3P documents and so forth are in place, there are several directions that the project can take, and pursuing all of them is an unlikely scenario given the resource limitations. To some extent, the direction of the project also depends in a large extent on how well preliminary testing goes; that is, how well does the CBR system actually predict user preferences. Depending on the results of these tests, several possible directions were deemed possible:

1. Given test failure, making improvements to the algorithm, hereunder, the retrieval methods, the amount of data stored per case, the distance metric used to compare cases, and the weighting of the different features of a case/policy.
2. Extending the system by implementing the community portion/collaborative filtering part of the proposed system.

3. Given a test success, implementing a working browser plugin.
4. Closely related to the previous point, extending the system to work with other privacy policy standards.

Direction 1 above takes the project from a system engineering direction more over to a scientific and statistical analysis type project, and while placing some emphasis on tuning the algorithms, this is not our primary focus[2]. Item 3 relies heavily on the effectiveness of the algorithm and may require exactly the type of statistical analysis discussed. In agreement with the customer, it was therefore concluded that the collaborative filtering part was to be the second focus after the CBR part.

2.4 Development Model and Workflow

Having briefly identified and outlined the project "flow", a development method or model must be chosen. A two-stage implementation process is envisioned; the first in which the core CBR is implemented. In the second part, the focus is on the community/CF portion. By thus splitting the work in two stages, time is made available for more thorough testing and reworking of the CBR portion, while work on the CF system, which is given a lower priority is pushed back. Given this project flow, we deemed that the work fits well within waterfall model framework. This is described in more detail in the next section.

2.4.1 The Waterfall Model

The waterfall model describes the development process as a linear process that starts with planning and requirements specification, and flows sequentially through design, implementation and finally (in this case) testing as is illustrated in Figure **??**. Arguments *for* the waterfall approach is that it encourages rigorous planning and design, which means that inconsistencies and problems can be discovered earlier in the process, which is generally less costly than if they are discovered late, since this often means re-writing a large portion of code.

Another advantage of the waterfall model, which relates directly to the nature of this particular project, is the focus on design. Since the software product to be delivered is a very early prototype that is to be used in further research, and likely to be further modified in the future, providing solid modularity and interfaces so as to allow code reuse is critical.

[2]This would require gathering a dataset of some size as well as setting up testing scenarios, which not only requires a sophisticated statistics background, but also likely group of test users. It was decided that this should not be prioritized in a software engineering project of limited scope such as this.

Properly documenting the program structure, as encouraged by the waterfall model, will also be highly beneficial to anyone who is later to modify the program.

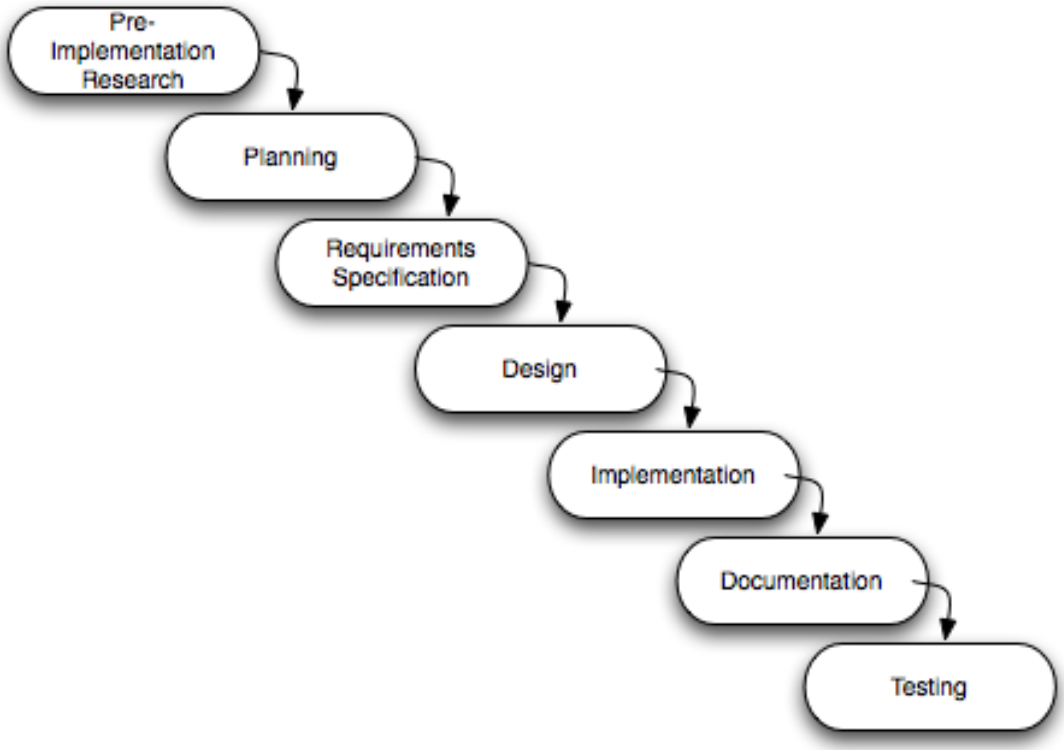


Figure 2.2: The Waterfall Model.

A common criticism raised against the waterflow model is that development rarely occurs in completely distinct stages; it is often hard to completely finish one phase of the project before moving on to the next. For instance, in many projects, requirements may be subject to uncertainty or change, so that design in turn must be altered. While recognizing this and other inherit weaknesses in the waterflow model, we think that for this project, requirements are quite clear, and since its scope is limited, a slightly modified waterfall model will still serve the project well.

Among the modifications that we have made is, as indicated in Figure 2.2, a slight overlap between the different phases. This is justified by the fact that some stages are not dependent entirely on the previous stage. For instance, given a detailed design, a test plan and testing procedures can be worked out independently, and testing of one module does not require the entire system to be integrated.

Planning Phase

Contents

3.1 Purpose

This document details the planning process that seeks to identify the different phases in the development process and allocate resources over time to the various activities that comprise each phase. Planning also needs to account for a number of risk factors that may impact the process, either by allowing for enough preventive measures or by allocating extra time to activities that may be affected. Thus a risk report enumerating several potential risks has been worked out.

3.2 Project Phases

The choice of development model is detailed in Chapter1.7. Here the various phases of the project are described.

3.2.1 Preliminary Study and Research

In this phase the aim is for each project member to acquire a certain level domain knowledge in the field of Internet privacy and to learn the necessary technology and tools required to implement the model as proposed by the customer. This entails having a working knowledge of the Java programming language, version control using Git and the CBR framework.

3.2.2 Planning

Planning seeks to identify the activities needed to reach the project objective. This entails breaking down the objectives into sub-problems, identifying the relationship between these, and allocating time for each of them.

3.2.3 Requirement Specification

The requirements specification is a document listing the functional and non-functional requirements of the software to be developed, which is a standard that the results is to be measured against, thus serving as not only a contract between the customer and the project team, but as a basis for developing testing methods.

3.2.4 Design/Architecture

This phase consists of a broad structuring and specification of the overall system. It defines the program structure in terms of program flow, modules, classes and interfaces as well as coding standards and other conventions that will serve as guidelines for the implementation phase.

3.2.5 Implementation

In this phase the design is realized as a working Java program according to the models developed in the Design phase.

3.2.6 Evaluation and Documentation

This phase consists of testing the system and documenting the structure of the system and how it is operated. From a software engineering perspective, the primary testing grounds are against the standards prescribed by the requirements specification rather than applicability of the underlying models performance. As mentioned, among the primary objectives of the project is to provide a testing framework to verify the applicability of the given system in making privacy decisions.

3.2.7 Ongoing Activities

Reporting and Administrative Tasks

Under this heading are more project management related activities, such as routine organizational work (ie. arranging meetings and writing status updates), more refined distribution of tasks as the project is underway, and preparation of the project report (this document).

Study and Lectures

To solve several of the problems posed by this project, most group members have had to learn new tools and technologies. This includes, but is not limited to Case Based Reasoning, version control (Git), certain features of Java and so on. Lectures on project management and software development are also subsumed under this heading.

3.3 Risk Report

The term 'risk' is usually defined defined as the possibility of an undesirable outcome (loss) as a consequence of a choice or an action made.

In this section we have identified some risk factors that can impact the project. Every project does risk management at some level, wether explicit stated or not. By identifying and quantifying the *likelihood* and *consequence* of undesirable events, the project plan can be adapted so as to allow for certain contingencies. Risks are quantified in two dimensions on a scale from 1-5 in severity, both in terms of the probability of occurrence and in terms of the consequence for the project.

In table 3.2 below is a description of how the risks would look like and what they mean. Beneath that the actual risk follows.

Table 3.2: Risks explained

Risk item	An arbitrary number identifying the risk factor.
Activity	The activity affected by this risk.
Risk Factor	A short textual description of the risk factor.
Probability	The probability of the event occurring. Measured on a scale from 1(unlikely) to 5(almost certain).
Consequence	What the consequences of the event occurring. Measured on a scale from 1(not critical) to 5(disastrous).
Action taken	Actions that can be taken to avoid this event occurring.
Deadline	An optional date set for taking precautions to deal with the risk.
Responsible	The group member responsible for the risk.

The actual risks:

Table 3.3: Changes in requirments

Risk item	1
Activity	All.
Risk Factor	The requirements might change.
Probability	2.
Consequence	4.
Action taken	Clarify the requirements and agree on deadlines for any changes that could happen.
Deadline	By acceptance of requirements specification.
Responsible	Ulf Nore, Customer.

Table 3.4: Poorly chosen algorithms

Risk item	2
Activity	Design, implementation.
Risk Factor	The implemented algorithms may not work as intended.
Probability	3.
Consequence	5.
Action taken	Research on similar algorithms and projects.
Deadline	N/A
Responsible	Dimitry Kongevold.

Table 3.5: Problems with policy retrieving

Risk item	3
Activity	Implementation.
Risk Factor	Problems with retrieving data from P3P policies.
Probability	3.
Consequence	5.
Action taken	Research into P3P, and cooperate with customer.
Deadline	Implementation deadline.
Responsible	Einar Afiouni.

Table 3.6: Problems with external storage

Risk item	4
Activity	Implementation.
Risk Factor	Problems with storing and/or retrieving data.
Probability	2.
Consequence	3.
Action taken	Look into several alternative knowledge base alternatives.
Deadline	End of design phase.
Responsible	Amanpreet Kaur.

Table 3.7: Remote server problems

Risk item	5
Activity	Implementation.
Risk Factor	Obtaining remote server space.
Probability	1
Consequence	3
Action taken	Ask IDI for virtual server.
Deadline	End of design phase.
Responsible	Nicholas.

Table 3.8: Sickness

Risk item	6
Activity	All.
Risk Factor	Unable to work due to sickness.
Probability	2.
Consequence	4.
Action taken	Plan with some degree of slack. Properly document work so that other members may take over.
Deadline	N/A
Responsible	Everyone in the group.

Table 3.9: 3rd party library problems

Risk item	7
Activity	Implementation, testing.
Risk Factor	3rd party code may be harmful or not work as intended.
Probability	1.
Consequence	3.
Action taken	Proper selection criteria and testing routines for selecting 3rd party code.
Deadline	End of design phase.
Responsible	The responsible for the functionality using 3rd party libraries.

Table 3.10: Misunderstandings.

Risk item	8
Activity	All
Risk Factor	Misunderstandings between customer and the group.
Probability	3
Consequence	3
Action taken	Proper reporting and documentation.
Deadline	N/A.
Responsible	Ulf Nore, Customer.

Table 3.11: Disagreements Within Project Team

Risk item	9
Activity	All
Risk Factor	Disagreements between team members on sharing of work, how to approach problems etc.
Probability	1.
Consequence	4.
Action taken	Attend seminars on group dynamics. Be quite specific on what's expected from the start. Assign leadership roles responsible for resolving conflicts.
Deadline	N/A
Responsible	All project members, project manager in particular.

3.4 Measurement of project effects

The primary objective of this project is to build a research prototype that allows for parsing P3P policies and provide advice using CBR given a particular knowledge base. The advice is based on:

- the user's previous actions
- community actions or what similar users have done
- context of use

3.5 Project Plan

As discussed in Section 3.2, the sequential part of the project is separated into six phases; pre-implementation research, requirement specification, design, implementation and documentation, evaluation, and report writing. The reporting started at the first day of the project and continues until project completion.

Implementation is scheduled to be complete at the end of week 42, which marks a shifting of focus to testing and evaluation. Some of the planning tools are outlined below.

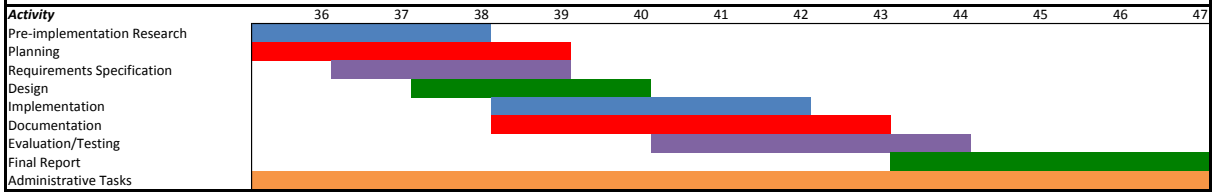
3.5.1 Project Milestones

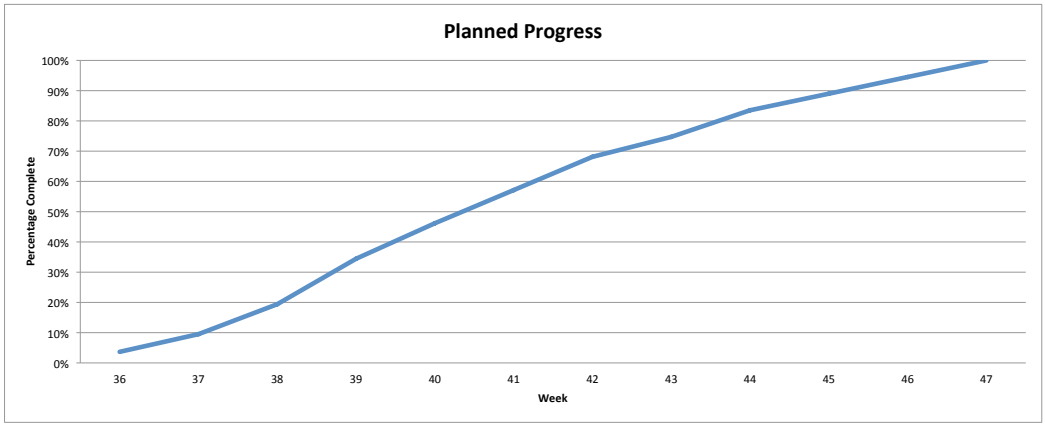
Time available	
Number of weeks	13
Group Members	7
Hours Per Week/Group Member	25
Total Hours Available	***2275***

Planned Resource Allocation				
Activity	**Percentage**	**Hours**	**Start Week**	**End Week**
Pre-implementation Research	4%	100	36	38
Planning	4%	100	36	39
Requirements Specification	7%	150	37	39
Design	12%	275	38	40
Implementation	18%	400	39	42
Documentation	11%	250	39	43
Evaluation/Testing	13%	300	41	44
Final Report	18%	400	44	47
Administrative Tasks	13%	300	36	47
Total	**100%**	**2,275**		

Planned Resource Allocation												
Activity	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Pre-implementation Research	33	33	33	-	-	-	-	-	-	-	-	-
Planning	25	25	25	25	-	-	-	-	-	-	-	-
Requirements Specification	-	50	50	50	-	-	-	-	-	-	-	-
Design	-	-	92	92	92	-	-	-	-	-	-	-
Implementation	-	-	-	100	100	100	100	-	-	-	-	-
Documentation	-	-	-	50	50	50	50	50	-	-	-	-
Evaluation/Testing	-	-	-	-	-	75	75	75	75	-	-	-
Final Report	-	-	-	-	-	-	-	-	100	100	100	100
Administrative Tasks	25	25	25	25	25	25	25	25	25	25	25	25
Total	83	133	225	342	267	250	250	150	200	125	125	125
Cumulative Hours Worked	83	217	442	783	1,050	1,300	1,550	1,700	1,900	2,025	2,150	2,275
Estimated Completion	4%	10%	19%	34%	46%	57%	68%	75%	84%	89%	95%	100%

Gantt Chart





Part II

Design and Implementation Phase

Requirements Specification

Contents

4.1 Purpose

This requirements specification has been prepared for and accepted by SINTEF ICT (the customer) stating the requirements for the software system to be developed for the course TDT4290: Customer Driven Project. The requirements span two categories; functional requirements, describing the functionality the software needs to supply, and non-functional requirements, describing the development process.

The requirement specification, once accepted by the customer, will serve as a contract between the parties involved, being a guideline for design and implementation and the standard against which the product is evaluated.

4.2 Introduction

4.2.1 Background and Similar Software

SINTEF ICT is currently investigating new approaches to privacy protection of end-users. Tndel et al. (2011) proposes a specific agent design for a machine learning approach to advice users on privacy actions based on:

- Past behavior using case based reasoning (CBR)
- Similar users behavior in similar situations using collaborative filtering (CF)

While there are systems for aiding users in making privacy related decisions, the majority of these systems rely in a large extent on the user pre-specifying his preferences and being prompted with messages about where the policy of a given site conflicts with the users preferences. Our design aims at being low profile or non invasive, that is able to make sensible decisions with as little interference as possible, and at the same time, given as little feedback as possible, able to cater for the dynamic nature of both web sites privacy policies and user preferences with respect to privacy.

4.2.2 Scope

The primary aim of this project is the implementation of the core classification system described in Tndel et al. (2011) to allow for testing the applicability of the suggested approach to predicting privacy preferences. Since the software is intended to be a part of a research project, a design that allows for testing of various hypotheses and models is required. This implies a highly modular design where the various components of the core system can be replaced.

Furthermore, given the research nature of the project, less emphasis is placed on developing a complete stand-alone application. The core focus for our project will be on developing the underlying system and an interface for testing and parameter estimation. Hence two distinct systems are to be developed, both built around the same core functionality (I.E. the privacy advisor agent). Owing to the fact that this software is developed around a research project, the emphasis is on the first system:

1. A testing system that can be fed a knowledge base consisting of input-output mappings (P3p + context -¿ decision), and run interactive tests on a sample where the user is allowed to give feedback to the system and see the explanation for the recommendation. We envision a dual CLI/GUI (command line interface and graphical user interface) solution for this. In a final product, this testing system can also be used for the purpose of calibrating the model.

2. An end-user system that can run as a browser plug-in giving real time advice to the user as he browses the web.

4.2.3 Overview

This document is organized as follows: Section 4.4 gives an overview over the system; its requirements and user characteristics. Section 4.5 presents four different use-case scenarios. Section 4.6 presents specific functional and non-functional requirements.

4.3 Overall Description

4.4 System Description

The overall structure of the system is detailed in Tndel et al. (2011), and consists of the local CBR reasoning system, the remote/community collaborative filtering, both with their respective databases for storing information. This is in turn linked to an interface that is able to read and parse P3P policy files that are retrieved either from a local file (for the testing system) or by retrieving from the web.

Figure 1: End User System Flowchart

4.4.1 User Interface

Because of the research nature of the project, the customer considers the user interface to be of small importance. As the underlying algorithm/methodology is in an early development phase, the core focus is placed on producing a system for model testing and evaluation rather than an end user interface.

4.4.2 Hardware and Software Interface

Being written in Java, the software requires a local copy of the Java Runtime Environment (JRE) installed on the computer.

For the community functionality (collaborative filtering), a dedicated server running the filtering engine must also be available. Since this is basically a modified version of the local server, it has similar requirements, but as it presumably will hold a larger knowledge base, its hardware requirements will be greater, as both lookup time (computational demands) and storage demands will increase with the number of users. It may also require additional server/database software such as mySQL, CouchDB etc.

4.4.3 User Characteristic

For this project we distinguish between two groups constituting the users of the product.

Developers/Researchers

Firstly, developers/researchers that will be working on the testing and calibration of the underlying model and extending it to other policy types beside P3P etc. These users are the primary focus of our work. A research/developer is an expert user, and needs to be familiar with how privacy policies are coded in machine-readable form such as P3P, but also the software source codes in order to modify, extend and optimize the algorithms.

End Users

Secondly, the end user who will be using the software in the form of a browser plugin that provides advice with respect to the users behavior on the Internet. A key objective for the project is that the agent is to be able to make good decisions and require as little feedback as possible from the user. To the extent interaction is needed, it should be able to clearly state an explanation for its decisions and allow the user to override in a simple manner.

4.5 Use Cases

The first use case illustrates a research setting where calibration/testing interface allows the user to load in a dataset of P3P policies and test the performance of the underlying model.

The last three use cases illustrate the potential application of the system as a browser plugin that runs in the background monitoring the users activities and the web sites he is visiting. As previously stressed, the success of testing according to the first use case determines the extent to which the system described in cases 2-4 is implemented.

Case 1: Research/calibration

In this case a researcher wants to test the properties of the underlying model. Using the Calibration GUI, he imports 50 P3P policies that are parsed. Further he designates that 40 of these are to be stored immediately in the knowledge base along with a corresponding action for each policy. The user now specifies the distance metric he wants to apply to each of the different components. Finally, he can either set the (importance) weights assigned to each of the policy components, or he can load the weights from a flat text file. Now that the configuration is complete, the ten policies withheld earlier from the sample can be classified. For each of the ten policies, the user can choose either to accept, or reject, and provide a reason for his rejection before proceeding to the next policy.

Case 2: End user local query, recommendation accepted, site rejected

A user visits a previously unvisited website. The privacy agent tries to retrieve machine-readable privacy information from the site. When the policy is obtained it is parsed and a context object, consisting of the policy, domain, time of visit, and other contextual information, is created. The context object is compared to the local database for similar contexts. Since the user has visited sites with a similar policy previously, the comparison succeeds and the site is blocked based on data from the local database. The user agrees with this decision and navigates away from the site.

Case 3: End user local query, site approved by recommender, recommendation accepted

A user visits a previously unvisited website. As before, the system the system fetches the necessary data to do a local query. This query indicates, with sufficient confidence, that the sites policy is acceptable. The user is then allowed to continue browsing with no intervention from the Privacy Agent.

Case 4: End user global query, recommendation overridden

As before, but in this case, no sufficiently similar cases are found locally. In this case the system will query the global server for similar users that have visited the same site to base its decision on this. In this case, site is blocked, but the user disagrees. He selects an override feature and gives a reason for why he overrides.

4.6 Specific Requirements

4.6.1 Product perspective

As described in Section 4.2.2, as the main goal of the project is to develop a testing framework for the core reasoning system. The secondary goal is to implement a user interface that can work as a stand-alone application to allow for actual user testing.

4.6.2 Functional Requirements

- The system should be able to parse a P3P file to instantiate the data as a privacy case/event/instance.
- Based on past history (knowledge base), it should retrieve the cases most similar to the one presented.
- Given the degree of similarity to past cases and the uniformity of action taken in the past, the system can either
  - Give the user a recommendation a recommendation or
  - Pass the recommendation decision on to the community/CF system.
- If passed on to the CF, the system will query a server for the most similar users and use the data on their decisions in similar cases to make a recommendation (along with local/CBR recommendation)
- Update the database with the recommendation.
- Allow the user to view the explanation for the recommendation
- Allow the user to overrule a recommendation.
- When overruling a recommendation, the user must be allowed to explain why the decision is made, e.g. one time occurrence, permanent rule, etc.
- Allow the user, if making a new general rule, to backtrack and alter previous cases

4.6.3 Non-Functional Requirements

- Implementation
  - Code is written in Java following Sun Microsystems conventions[3].
  - Third party libraries are to be documented with version numbers and to be included in the installation package.
- Maintainability:
  - Code repositories and version control: github is used as code repository and for version control.
  - User documentation is to be produced.
  - A well documented API is to be designed
  - English (US) is to be used as language for naming convention for source code and filenames, and in code comments and documentation.
  - The code is to be designed in a modular fashion.
- Performance:
  - For the final end-user product that will run as a browser plug-in, performance will be important, as the program should not be seen as a nuisance in getting work done.
- Portability:
  - The testing/design system should be portable to any system with a JRE.
- User interface:
  - Two UIs are to be implemented: A command line interface (CLI) as well as a GUI is to be designed using Java/swing.
  - These interfaces are meant to facilitate testing the model framework.

[3] http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

Design

Contents

5.1 Purpose

This document describes the design phase of the program, where the program architecture is established. Several critical decisions are made in this phase and the design and architecture decisions impacts the way the implementation phase proceeds as it defines how the final software system is decomposed into modules, and how these modules behave and interact with each other. Details regarding programming languages, algorithms, data structures, coding standards and other software engineering features must also be established prior to proceeding from this phase.

5.2 Development Tools and Technologies

This section details some of the choices that were made regarding development tools and technologies for this project.

5.2.1 Documents and Source Code Repositories

For software projects of a certain scale, version control is an important technology that allows project members to work simultaneously against the same files without causing inconsistencies. Version control systems also allow for comparison with older versions, tracking changes and restoring previous copies in the case of errors.

For source code repositories and version control, **Git** was selected. Git is a distributed, open source version control system that is available for all platforms. Git is also used for hosting the LaTeXdocuments that comprise this report. For other documents such as meeting minutes, agendas, status reports, time reporting and certain planning documents, Google Docs is used.

5.2.2 Programming Languages

Java is chosen as the primary programming language for implementing the majority of the code. Java is an object oriented programming language providing a level of abstraction appropriate for the task at hand in addition to a rich set of libraries, including the SWING library for GUI programming, and several libraries for networking. It also simplifies writing a browser plugin as major browsers such as Google Chrome and Firefox employ Javascript as scripting language for plugin.

5.2.3 Databases

For testing purposes, it has been decided on using flat file storage of privacy policy data using Java's `Serializable` interface. However, the output functionality is to be written in a generic fashion to simplify use of database systems such as MySQL, CouchDB and so forth.

5.2.4 Third Party Libraries

For developing the CBR as well as P3P parsing components of the Privacy Advisor, a decision had to be made regarding the usage of third party libraries, either for components

or for the entire CBR system. Two options were considered with respect to the CBR system. The first was to use a full third party CBR system (jColibri). The alternative was to use a third party system for the retrieval component of the CBR system (i.e. a k Nearest Neighbors (kNN) implementation).

Third Party CBR System

The customer (SINTEF ICT) suggested looking into an open source CBR library developed at the Universidad Complutense de Madrid.

jColibri is a CBR system that has been under development for well over 10 years and is a very comprehensive system allowing for database interfaces and several other features, and is according to the customer, a popular choice in academia for CBR projects. It is also written in Java, which of course makes interfacing it simple from our own Java project.

However, its comprehensiveness also means that it takes more reading to understand and properly apply to the project at hand, and due to its size and poor documentation, jColibri was ultimately deemed unfit for the Privacy Advisor project. Due to the limited time resources available to this project, the risks associated with spending a large amount of time on a third party library that eventually would not be running was to high.

Third Party k Nearest Neighbors Implementations

Since kNN is a standard classification algorithm, there are several open source implementations available. Limiting the search space to Java implementations, a library called The Java Machine Learning Library (JavaML) was the primary candidate, as it provided a clean and simple interface and allowed for extracting confidence measures.

The problem with this library relates to the nature of distance metrics used in classifying privacy policies which is compositional in a way that is non-trivial to handle in JavaML. Furthermore, JavaML seems to operate only on arrays of floating point numbers, which means the distance metric must be defined in two stages; first mapping from policy domain to real numbers, then in terms of a metric on real vectors.

P3P/XML Parser

Looking for XML parsers on the Java platform, we found out that there are two different types of XML parsers we could use, the first being a DOM Parsers and the second one being a sequential access parser. The difference being that DOM parsers operate on the

document as a whole, while sequential access parsers operates on each piece of the XML document sequentually.

We ended up using SAXParser, an internal sequential access parsers in Java. The task from here was to implement it, making the policy as an object with the fields of our choosing. It works by sequentially going through all elements of the XML document, and with easy string comparison, checking if the element is of the wanted ones.

5.3 Standards

To achieve clean and reusable code, the project has adopted Oracle's Coding Conventions for the Java Programming Language[4]. This is mentioned in the requirements specification due to the high likelihood of the customer having to change the source code for later adaptations.

5.4 Architecture

In implementing Privacy Advisor, a class structure is built around the CBR agent model discussed in Tndel and Nyre. Refinements needed to be made include data structures for storing policies, databases, choosing actual algorithms, and so forth. This section describes how the design in broad terms.

5.4.1 Design Overview

This section describes the architecture of the local CBR based system. The next section gives an overview over the design of the server component using collaborative filtering and how it interfaces with the local system.

Figure 5.3 illustrates the broad structure of the system; a policy object is passed to the CBR system from the user via the user interface. The CBR does a lookup on similar cases in the local database and makes a recommendation based on the similar cases and provides the user with the advice and the background for the advice through the user interface. The user can then give a feedback on the advice choosing to accept or reject it. The user feedback is then stored back to the database.

[4]http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

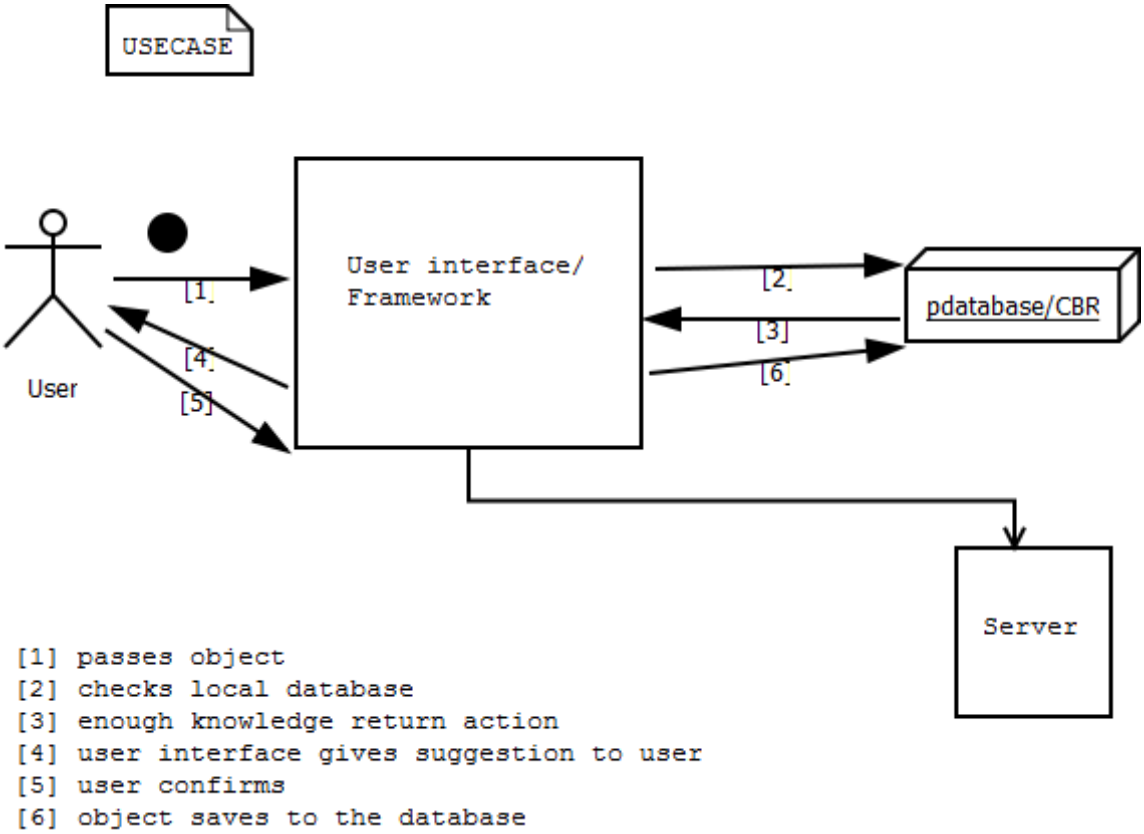


Figure 5.3: System Overview.

5.4.2 Detailed Design

Dependencies

User Interface

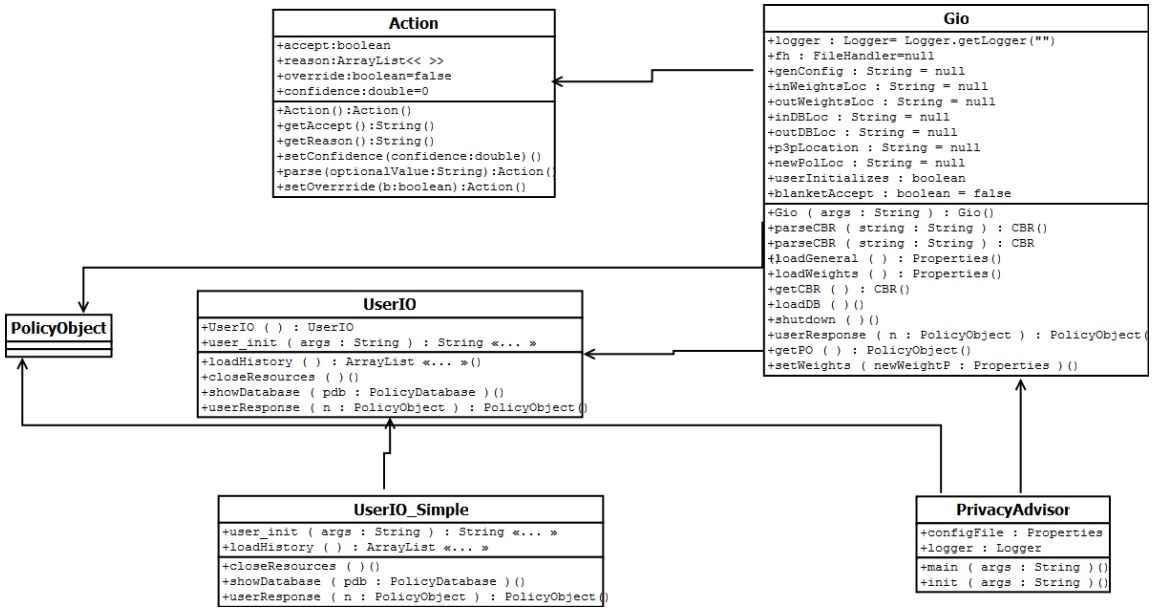


Figure 5.4: Input/output and user interfaces.

5.4.3 Community Server

The community knowledge repository is implemented using a public CouchDB (no-SQL server), which is accessed using standard Java to JSON java libraries. The client program (end-user java application) communicates with the server at two points- when the application has insufficient knowledge, or confidence in its knowledge, to make a suggestion as to the acceptance of a new P3P policy; and after the user has confirmed or overridden the policy.

In the first instance, the new policy under consideration is converted to JSON using GSON (the Google Json libraries), and transmitted to the database, which parses the new policy and replies with a JSON encoded suggested Action.

In the second instance, the final policy (including the action taken on it) is sent to the CouchDB server, and the server proceeds to store the object. On the database end, there

are two essential interfaces (beyond any standard initialization and shutdown procedures). As seem above, these two interfaces are the suggestion provider, which includes a query to find the most similar policies and actions on them by the community, and a interface to simply save the new policy to the appropriate database. The database is easily replaceable, requiring only the construction of a new class implementing 'NetworkR', the abstract class detailing the methods called by the PrivacyAdvisor framework. The selection between available 'NetworkR' implementations in made by setting the 'NetworkRType' configuration variable during initialization to the full classname.

Implementation

Contents

Purpose

This document explains the implementation phase of the project providing a more detailed description of particular key details that were not decided on in the design phase. This relates in particular to choices regarding the particular CBR algorithms (k-Nearest Neighbors) and the similarity measures describing how "equal" two cases are. It also details the datastructures that are used to represent policies and how comparisons are done on these datastructures.

6.1 Algorithms

6.1.1 K-Nearest Neighbors

K-Nearest Neighbors (k-NN) is a *lazy, non-parametric* algorithm for classifying objects based on the classification of the nearest examples in a given feature space. k-NN is one of the simplest machine learning algorithms as it decides the classification based on a majority vote of, that is, an object is classified according to the most common classification of its k nearest neighbors. The most critical component for the success of the k-NN algorithms is the definition of distance. This is discussed in Section 6.2.2.

For testing purposes, we have implemented a very simple kNN that sorts the example set (knowledge base) by distance from the object to be classified and returns the k nearest objects. This is obviously not an optimal approach being $O(nlg(n))$ where n denotes the size of the knowledge base. This is not problematic for a small scale application such as ours where the knowledge contains less than 200 objects. If the system is to be scaled up, it would require a new kNN implementation, of which there are several available.

Learning algorithms

The learning algorithm updates the weighs that each property field is assigned in computing distances. The learning algorithm goes through the policies in the database and computes updated weights. The implemented learning algorithm is a simple one that for each weight goes through every policy and checks if it have been accepted or not. Then it returns the number of accepts divided by the number of policies for each weight.

By updating the weights like this we make sure that the system to some degree learns what the user wants. Thus, hopefully the system would be able to make a different conclusion next time if the user didn't agree on the conclusion the system came up with this time.

6.1.2 Distance Measures

Definition

Mathematically, a *metric* or *distance function* is a function defines the *distance* between two objects in a set are, that is, it defines a notion of how far apart two objects are. In a purely mathematical sense, a distance function defined over a set X, $X \times X \longrightarrow \mathbb{R}$ that is required to obey the conditions of non-negativity, symmetry, sub-additivity (the triangle inequality) and identity of indiscernibles.

Some examples of commonly used metrics are the Euclidian, Mahalanobis, and the Manhattan distance measures. These along with a few others are defined in the next section. These metrics have all in common that $\mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$, which in the case of comparing privacy policies and corresponding context information, is problematic as these, in their raw form contain large amounts of textual data. Two remedies could be proposed for this situation:

1. Provide a function to map privacy objects (P3P policies and context info) to real vectors.
2. Define a new metric that operates directly on privacy objects.

Common Metrics

- The*Manhattan distance* function computes the distance that would be traveled to get from one data point to the other if a grid-like path is followed. It can be written as where d is the dimensionality of the data objects.
- The *Hamming distance* is defined as number of positions in which a source and target vector disagrees. If data are bitstrings then Hamming distance can be calculated as.
- *Levenshtein distance* is based on Hamming distance but adds also operands as insertion, deletion and substitution. And can be written...
- *Ontology distances* are more based on compute semantic similarity of objects rather than their textual representation. For example distance between apple and orange is less then between apple and house. To calculate this distance you need some sort of logical tool like ontological tree. Where every leaf has a logical ancestor for example ancestor for apple will be fruit.

Customer Advise

In the paper "Towards a Similarity Metric for Comparing Machine Readable Privacy Policies", the some of the problems of defining a similarity metric for privacy policies is discussed. A key topic is how the calculation of similarity between online 3P3 policies can be subdivided in two parts, *local similarity* and *global similarity*. This way known metrics such Levenshtein distance can be applied to local distance. And for global similarity we can calculate a simple or weighted average of local distances, where the second one allows for amplifying the importance of particular attributes.

Another important topic is how system designers can apply domain knowledge to improve distance calculation. For example, for the recipient field identifying who data is shared with,

there are certain values revealing that significantly more private information is exposed than others. Having private information retained by the website (recipient = "ours") is in a sense less critical than it being given away to third parties for commercial purposes (recipient = "other) or being public (recipient= "public"). So distance between unrelated or public is less then between ours and unrelated.

6.1.3 Implementation

Bitmap Representation

A bitmap (or bit string) is a way of representing a set of objects. It simply translates values from a set to a vector of fixed length, where each value has a specific place. For example over language $L = \{a, b, c, d\}$ for a sets $\{a, b, c\}$ a bit-map will look like [1110] where 1st integer represents a and last integer represents value d. This way it doesnt matter what order the values are arranged and how many values are so set $\{d, b\}$ over same langue L will be [0101] where 1st value still representing value a, or more correctly said, absence of value a. Calculating intersections or unions over bitmaps A and B uses bitwise Boolean operators. Where union can be easily written as $(A_i \vee B_i)$ where i is the position in the vector, and the intersection $(A_i \wedge B_i)$.

Weighed bitmap is when each of the values is multiplied by its corresponding weight. Let us say on the language L the weights are $a = 1b = 2c = 4d = 3$, since bit-map over set $\{a, b, c\}$ is [1110] the weighed bit-map will be $[1 * w_a 1 *_b 1 *_c 0 * w_d] = [1240]$. And for set $\{d, b\}$ it will be [0203].

Privacy Policy Representation

This section describes the data structures used to represent policies. On the top level, `policyObject`s contain P3P policy and context information (URL + time etc.). A P3P policy can have a varying number of *statements*, some of them greatly differ from each other, and others are very similar. Each statement is build of four fields: data-collected, purpose, recipient and retention. Purpose, recipient and retention can contain one or many given values, the combination of witch describe how the given data-collected may be used in the future. Data-collected is divided in four major fields: dynamic, user, third-party and business. Many different data-types can be collected in one statement.

For every data object that being described in a statement, we create a `Case`. A `Case` in a `policyObject` corresponds to a statement in P3P policy. It contains the purpose, recipient and retention, but a Case can only have one unique data type; this results that one statement can be translated to many cases. `policyObject` has a list of all its `Case`s

and additional information that we find useful like time of the visit, location of the domain, and action decided upon by the CBR system or the user. Based on SINTEFs proposal the two levels of similarity, local and global are accounted for in implementation.

Bitmap Distance

The distance function is to be a highly modular component of the CBR system. The default distance function implemented uses the bitmap data structure mentioned above, and is henceforth referred to as "Bitmap Distance". For the local similarity Bitmap Distance generates bitmaps for fields, retention, purpose, and recipient. This eliminates the problem that these fields can have a differing number of attributes. Prior to computing the Hamming distance the bit-map is transformed into weighed bit-map by multiplying every attribute by its weight.

The weakness is that without weights its more string comparison then ontological tree.

We decided that weighting data-types values was outside of scope of this project. We simply gave static values to 1st nodes from the root(the four main fields) based on our knowledge and interpretation of private policy.

It is possible to create a weight system so this algorithm will work like a full ontological tree. For example if we add a value for each possible node value/weight so we can simply take difference between StringAs 1st 2nd and 3rd field and respectfully StringBs fields. If the field 1 in A same as Bs then distance is of course 0. But this way we can say that some of the fields or sub-fields based on the same depth in your tree can have same values and distance between them will be 0. For example if we want to say that bdate and name are equally important for user we can give then same value making difference between them 0, but if we want to increase distance between name and home-info we have give them values that have a greater difference like 1 for name and 10 for home-info, creating distance 9 between them.

Weights are the key to learning and adjusting this algorithm. They are greatly used in our implementation of Hamming distance and can give great results in data-type analysis.

The global similarity part was not as easy as creating a bitmap. The number of cases a policy can is undefined, and the similarity of those cases can differ a great deal. Your solution was sum of minimal distance of each Case to a case in the other policy. For instance a caseA from policyA have distance 2, 3, 1 to cases in policyB that mean the distance caseA will have distance 1 to policy.

Sum=0 For caseA in PolicyA.Cases Min=infinity For caseB in PolicyB.Cases Dist=Compare caseA to caseB If dist¡min then Min=dist End End Sum=sum+min End

By choosing minimal distance between cases we guaranty that if two cases are identical the distance between them will be 0. But it also creates a problem, consider to policies policyA with caseA1 caseA2 caseA3 and policyB with caseB1 caseB2 where every case in A has minimal distance with caseB1. This way the properties of caseB2 will go unnoticed in the sum of distances between cases. We solved this problem by running algorithm twice but changing places of policy A and B the 2nd time and simply summing results.

The weakness of computing this way is that some of the distances will be counted twice, but user-privacy-safety wise is better to count some cases of minimal distance twice then leave a dangerous or most distant case out.

There is some variations in this method. You can use maximum/minimum distance between cases or average of the sum. We choose minimum because this way we always try to find a best match between cases and policies. With an algorithm that will minimize error from computing twice, from a to b and b to a, minimum distance with give the best results. But in total picture if you use same algorithm that considers every case for every policy in your database the results will be proportional.

6.2 Data Storage

6.2.1 Flat File Storage

6.2.2 Databases

6.3 User Interface

6.3.1 Model View Controller Architecture

[Describe theoretically]

6.3.2 Privacy Advisor Interface

[Describe choices that have been made and reasons for differences from standard architecture]

6.3.3 CLI

6.3.4 GUI

Documentation

Contents

This section provides documentation for the Privacy Advisor system. It gives an overview over the available documentation of source code, instructions for how to compile and install the system, and how it is used via. its GUI and command line interfaces.

7.1 Source Code Documentation

The source code is documented using JavaDoc which is a tool that generates documentation in HTML format based on source code comments in Java, and is a standard part of the Java SDK. The JavaDoc for the Privacy Advisor system follows Sun Microsystems' style guide for writing JavaDoc comments[5]. Source code documentation plays an important role in this project, as it is an early software prototype to be used in research which means that the code is then likely to be modified. The aim of the source code documentation is to supplement UML design documents to facilitate future development.

[5]See: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

7.2 Installation

7.2.1 Local Installation

7.2.2 Server Installation

7.3 User Interfaces

7.3.1 Graphical User Interface

7.3.2 Command Line Interface

Configuration Files

Table 7.3.2 gives an overview over the configuration file parameters.

Building a Database

Loading and Viewing a Database

Parsing a P3P Policy

Item	Datatype	Description
loglocation	string/filepath	where the log is written to. can't be changed once the UI is called.
loglevel	string/logging level	what level to log at.
inDBLoc	string/filepath	where to read the past history from.
outDBLoc	string/filepath	where to write DB- defaults to where it reads from.
inWeightsLoc	string/filepath	where to read the weights config file from.
outWeightsLoc	string/filepath	where to write DB- defaults to where it reads from.
newDB	string/boolean	are we overwriting/ignoring an old database.
p3pLocation	string/filepath	a p3p to be added to the history.
p3pDirLocation	string/FOLDERpath	a folder of p3ps to be added to the history.
blanketAccept	string/boolean	accept the advisers recommendation.
newPolicyLoc	string/filepath	the new policy to be parsed.
userInit	string/boolean	true if some initialization occurs via the user interface.
userResponse	string/action	the response to the suggestion, if know beforehand.
cbrV	string/CBR	parses for algorithms, etc to use. See CBR:parse(String).
userIO	string/UIO	the user interface to use. see Gio:selectUI.
policyDB	string/policyDB	select the database type. see Gio:selectPDB .
genConfig	string/filepath	load an alternate configuration file.
networkRType	string/classname	the name of a networkR class.
networkROptions	string/commasepoptions	the options necessary for the above networkR class.
confidenceLevel	string/double	the confidence level at which the algorithm trusts itself; if below this, it uses the server's suggestion.
useNet	string/boolean	whether to activate network functionality.

Table 7.12: Configuration file parameters.

Part III

Testing, Evaluation and Summary

Test Plan

Contents

Test plan overview

This is the test plan for the "Privacy Advisor" application requested by SINTEF ICT. This test plan is based on IEEE829-1998, the IEEE standard for software test documentation, with some adaptions to fit this project better. The purpose of testing is find bugs and errors and correct them, and to make sure the program is working as expected. The purpose of this test plan is to make sure the tests will be executed as planned, and that they are well documented.

8.1 Test Methods

There are two main types of software testing: black-box testing and white-box testing.

8.1.1 Black-box testing

This is a method that test the functionality of an application. For this type of testing, knowledge about the application's code and structure is not required. The test cases are based on external descriptions of the software, e.g. specifications, functional requirements or designs. Black-box tests are usually functional, but they can also be non-functional. This type of testing can be applied to all levels of software testing.

8.1.2 White-box testing

This method is used for testing internal structures of an application. For white-box testing it is required to have both knowledge about the code and structure of the application, as well as knowledge about programming to design the test cases. This type of testing is normally done at unit level where it can test paths within a unit or paths between units, but it can also be used at integration and system levels of testing. This method can uncover many errors and problems, but it is not a good test method for finding out whether the program is fulfilling the requirements or not.

8.2 Testing approach

Our main focus will be on white-box testing. This is a program that is going to be used for research, which means that black-box testing will not be very useful as the client want to work on and test algorithms themselves. Our main task is to deliver a good framework with the necessary tools, and a working, learning algorithm so that further testing can be done with ease. Since one of the system requirements is high modularity, it will be a goal to have the tests be as little dependent on other modules as possible. There will be no training needs, as the testers are also involved in the programming.

What will be tested:

- **Unit testing:** This will be used for testing the functionality of the modules, so that we can ensure that they are working as intended.
- **Interface testing:** As our algorithm is based on case-based reasoning (CBR), it will be important to test that it is learning from new data.

What will not be tested:

- **Usability testing:** This program is intended for further research by the client, and not for use by customers. Since we are not delivering a program ready for users, there is no need to perform end-user tests to see how users interact with the program, and whether the product is accepted by users or not.
- **Learning of our algorithm:** For the same reasons we will not perform any tests on the quality of the GUIs. The GUIs that will be included is there to make testing easier for the client, not to provide the best possible interaction with end-users.
- **Run time:** We will not do designated tests for checking and optimizing the run time. This is because the main classification algorithm can be easily changed. Run time will only be looked into if the program is very slow even for small data sets.

8.3 Test case overview

Under is the test cases and their identifiers. The identifiers are named UNIT-XX, where XX is the number of the test case.

Unit tests:

- UNIT-01: Command line interface (CLI) functionality
- UNIT-02: P3P parser
- UNIT-03: Local database
- UNIT-04: Graphical user interface (GUI) functionality
- UNIT-05: Algorithm classification
- UNIT-06: Algorithm learning
- UNIT-07: Packet passing through network to community database

Under is the test case template for the unit tests

Item	Description
Name	The name of the test
Test identifier	The identifier of the test
Person responsible	The person responsible for making sure the test is executed correctly and on time.
Feature(s) to be tested	What kind of functionality that is being tested.
Pre-conditions	What code and environment that has to be in place before the test can be executed.
Execution steps	Stepwise explanation of how to perform the test.
Expected result	The expected output/result for the test to be successful.

8.4 Test pass / fail criteria

A test is passed if the given execution steps and input produce the expected results. If they do not, the test is failed.

8.5 Test scheudule

Under is the table with the scheduled execution dates of the unit tests.

Test identifier	Execution date
UNIT-01	Saturday 22.10
UNIT-02	Saturday 22.10
UNIT-03	To be determined
UNIT-04	To be determined
UNIT-05	To be determined
UNIT-06	To be determined
UNIT-07	To be determined

8.6 Risks and contingencies

For some of the tests we will not be able to test every possible input / output. So there is a chance a test will pass for all the combinations we will be testing for a specific test case, but still fail at some later point for some other combination. This can be a problem for the tests UNIT-02 P3P parser, and UNIT-05 Algorithm classification.

The P3P policies have a huge variety in which elements they contain, and certain elements have a N-to-1 relation, so it will be impossible to test if everything is parsed correctly for every possible P3P policy. The best way to prevent this is to handpick a set of policies that have as different content as possible, so that as many of the extremes as possible will be covered.

We got the same issue for testing the algorithm classification. There is simply too many combinations of learning base and input to cover everything. So again we have to do our best with regards to also covering as many extremes as possible.

Project Evaluation

Contents

Project Evaluation

The final phase of the project consists of evaluating the actual outcome of the project work and the processes that have led to the outcome. It seeks to answer questions such as:

- Has the project acheived its major objectives?
- In what areas could a better result have been achieved?
- What could be done differently to acheive a better result?
- How well did planned resource use reflect actual resource use?

9.1 Software Evaluation

9.2 Organization and Group Dynamics

9.3 Tools

9.3.1 Programming and Implementation

9.3.2 Reporting and Organizational Tools

9.4 Resource Use

Conclusion

10.1 Introduction

Part IV

Appendices

Test cases

A.1 Test cases

Item	Description
Name	Command line interface (CLI) functionality
Test identifier	UNIT-01
Person responsible	Henrik Knutsen
Feature(s) to be tested	All possible commands when running the program with the CLI. That input with incompatible commands does not run.
Pre-conditions	Code for input handling for all possible commands. Error handling for invalid inputs.
Execution steps	1. Test all the commands one by one. 2. Test the combinations of invalid inputs.
Expected result	1. The program runs using the input variables. 2. Error warning: Invalid arguments. Abort startup.

Item	Description
Name	P3P parser
Test identifier	UNIT-02
Person responsible	Henrik Knutsen
Feature(s) to be tested	That the P3P parser correctly parses all the required fields and their values.
Pre-conditions	The P3P parser must be implemented. Need to have P3P policies with a wide range of cases.
Execution steps	1. Manually go through a P3P XML and obtain all the required fields and the values. 2. Run the same P3P XML in the P3P parser and print the parsed elements and their values to console. 3. Compare the results from the two parsing methods.
Expected result	The two parsing methods give identical output. They must both have the same fields, each containing the same values

Item	Description
Name	Local database
Test identifier	UNIT-03
Person responsible	Henrik Knutsen
Feature(s) to be tested	Writing to and reading from the local database. That the serialization of the database is working.
Pre-conditions	Code for writing to and reading from the database file must be implemented. Need to have two different P3P policies.
Execution steps	1. Write policy A to the local database. 2. Write policy B to the local database. 3. Read policy A from the local database. 4. Read policy B from the local database.
Expected result	The written policy A and the read policy A must be identical. The written policy B and the read policy B must be identical.

Item	Description
Name	Graphical user interface (GUI) functionality
Test identifier	UNIT-04
Person responsible	Henrik Knutsen
Feature(s) to be tested	That all the interactable elements, buttons, lists etc., is working as intended.
Pre-conditions	GUI with all the necessary listeners must be implemented. Code for running the program with the GUI must be implemented.
Execution steps	1. Run the program using the GUI. 2. Test all the interactable elements.
Expected result	All the interactable elements is triggering the right methods when used.

Item	Description
Name	Algorithm classification
Test identifier	UNIT-05
Person responsible	Henrik Knutsen
Feature(s) to be tested	That the k-nearest neighbor algorithm bases its decision on the k most similar policies
Pre-conditions	Code for reading from the weights file must be implemented. A working k-nearest neighbor algorithm that uses the weights must be implemented. Need one policy to test on, and a set of policies to be used as history.
Execution steps	1. Load a set of policies into the history. 2. Run the k-nn algorithm on the policy to be classified and the history. 3. Manually go through the policies and verify the output of the algorithm.
Expected result	The algorithm finds the most similar policy.

Item	Description
Name	Algorithm learning
Test identifier	UNIT-06
Person responsible	Henrik Knutsen
Feature(s) to be tested	That the weights file is updated when a new policy is added to history.
Pre-conditions	Code for reading from and writing to the weights file must be implemented. Algorithms for classification and learning must be implemented.
Execution steps	1. Get the contents of the weights file. 2. Load a set of policies into the history. 3. Run the classification algorithm on the single policy and the history. 4. Choose to store the new policy, the context and the action. 5. Get the contents of the weights file. 6. Compare the contents of the weights files obtained in steps 1. and 5.
Expected result	The two weights files obtained in steps 1. and 5. are different

Item	Description
Name	Packet passing through network to community database
Test identifier	UNIT-07
Person responsible	Henrik Knutsen
Feature(s) to be tested	That packets can be sent between the client program and the community database.
Pre-conditions	A running local client. A (virtual) server. Code for sending and receiving packets must be implemented.
Execution steps	1. Start the program locally. 2. Start the (virtual) server. 3. Send packet A from the local client. 4. Receive packet A at the (virtual) server. 5. Send packet B from the (virtual) server. 6. Receive packet B at the local client.
Expected result	The received packet A is identical to the sent packet A. The received packet B is identical to the sent packet B.

Templates

A.1.1 Status Reports

A.1.2 Meeting Notes

A.1.3 Testing

A.1.4 Time Reporting