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 report details the design, development and implementation of the Privacy Advisor in fulfilment of course TDT4290-Customer Driven Project. The "Privacy Advisor" software is 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 of using a Case Based Reasoning (CBR) agent as a core algorithm in this type of application. Thus this project implements CBR. SINTEF ICT has indicated the use of the project software as part of future research and therefore emphasis is 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 part of the course TDT4290 - Customer Driven Project and is 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 1.7.4). 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 comprises 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 4.5.3) is made, which henceforth serves as a guideline for implementing (Chapter 5.3.3) 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

Purpose

This chapter contains the project directive for the project ”Privacy Advisor” (the project) which is developed for SINTEF ICT (the customer) as a part of the course TDT4290 - Customer Driven Project at NTNU during the fall semester of 2011.

The project directive, or simply directive, is intended to provide a broad overview over the project, defining its objectives, scope, the responsibilities of project participants as well as a few core processes and routines related to project management, reporting and quality assurance. The directive is intended to serve as a guideline for project work and later on, project evaluation along with the requirements specification.

The project directive is intended as a dynamic document reflecting the nature of the project and its inherent uncertainties. If a major change of direction occurs during the project lifetime, the project directive is updated accordingly in agreement between the project team and the customer.

1.1 Mandate

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

1.1.1 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 Tøndel have 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 Tøndel 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.

1.2 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 is considered least important, as it is highly 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.

1.3 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.

1.4 Organization

Project management is based on 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 1.4. 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.

1.5 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.

1.6 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.

1.7 Quality Assurance

This section describes measures to be done to assure that the project is able to reach the quality level expected by establishing internal routines as well as reporting from the project group to ouside interests, that is, the customer and the project advisor.

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

Table 1.1: Responsibilities.

1.7.1 Status Reporting

External Reporting

The project manager is responsible for producing weekly status reports detailing the progress towards the objectives that are established in this and other planning documents. The status report forms the basis for discussions in weekly advisor meetings held on Wednesdays at 14.15 in ITS464 at NTNU and in on a semi-weekly basis in customer meetings with SINTEF.

Internal Reporting

For internal reporting, each team member is to keep a time sheet tracking the amount of time spent on different project activities. These time sheets are to be updated on a weekly basis so as to keep track of progress in accordance with the project plan. Furthermore, internal project team meetings are held prior to advisor and customer meetings in order to keep the status report up to date.

Templates

For the abovementioned reporting procedures, a set of templates have been worked out. The templates encourage reporting according to a particular standards, which makes document preparation and reading easier. Examples of these templates are those for meeting minutiae and status reports. All templates are stored in the document respository and are available for all project members.

Finally, a simple Java code template has been agreed on. This template illustrates key points from Sun Microsystems' coding standards. Strict adherence to a proper standard facilitates among other things the generation of JavaDoc documentation.

1.7.2 Response Timelines

To ensure an efficient decision making process and ensure that important tasks receive proper attention, some simple routines have been established:

Meetings: Meetings are to be called by e-mail 24 hours prior to the meeting. The invitation should include an agenda as well as any other documents relevant to the discussion.

Minutece of meeting: The project leader is responsible for assigning the task of taking minutes at meetings. The minutes are to be made available at the document repository within a 24 hours of the meeting.

1.7.3 Risk Management

A risk report is to be produced as a part of the project plan. The risks are to be identified by severity and probability as well as the activities that are touched by the risk. The risk reporting phase will also identify a group member responsible for managing and mitigating the particular risk, and serve as an input to planning so as to allow for a certain degree of slack.

1.7.4 Document Respository

Google Docs has been chosen to serve as repository for documents such as status reports, meeting minutiae, time sheets, and other frequently edited documents. Google Docs allows for simultaneous real-time editing and collaboration and has basic spreadsheet and word-processing functionality. All project interests, including customer and project advisor, are granted read-write access to the Google Docs repository to review plans, notes and comment on these.

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 simply 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.

2.1 Internet Privacy Technology

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

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

2.1.1 What is Internet Privacy?

Privacy is the means or ability of an individual to seclude himself or information about himself from third parties and selectively control what personal information[2] about him is to be available. When talking about *Internet privacy*, it is referred to the control over the type and amount of private information that is revealed under a transmission of data over the Internet, and hereunder, who has access to the information. The legal framework setting the boundaries of the service providers' priveliges with private information is very marginal, and typically, the best information available for users is through the provider's *privacy policy*.

The article "User Agents for Matching Privacy Policies with User Preferences"[3] describes the rationale for a system aiding users in making Internet privacy decisions in terms of a *privacy paradox* - the fact that while most users claim to put great emphasis on privacy, this is not mirrored in their actions.

Social networks such as Facebook provide a good illustration of this paradox and why it is important. Such services have grown to major prominence as a means of communication over the last half decade, and a common trait of these networks is the sharing of more private information than what was common on the "traditional" Internet. This is all well and fine, as long as the information is shared in the appropriate sphere, but there are obviously issues once your private images or derogatory comments appear in the public sphere, something that has been related through several news stories.

2.1.2 Current Privacy Technology

The disproportion between the average Internet user's concern for privacy and the actual control he has over private information referred to as the privacy paradox above, as well as the very fact that privacy is a *fundamental human right* is obviously something that calls for action. 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[4]. To aid users in understanding the implications of such policies and hence in making informed privacy related decisions, there is a need for tools providing users with information about policies in a format that is accessible to the user. As pointed out in the abovementioned paper, current technologies are often hard to configure and requires domain knowledge from the user or may be more concerned with hiding information[5], rather than the consequences of sharing information which is a common activity in online shopping and social networks.

[2]The term *personal information* will here refer to information such as name postal and e-mail address, financial information, social security number and so forth.

[3]Karin Bernsmed, Åsmund Ahlmann Nyre and Martin Gilje Jaatun: "User Agents for Matching Privacy Policies with User Preferences", SINTEF ICT.

[4]Users are in reality faced with a two cost-utility decisions: the first is wether or not to read the policy, which in most cases is clearly answered with a no. The second is given the lack of knowledge about the policy, wether or not to use the service.

[5]Ie. *anonymity*. While this certainly aids in protecting privacy, it also renders certain services such as social networks useless, since a sharing a certain level of private information is required for the service to have any meaning. As an example of this approach, see instance the Tor Project: https://www.torproject.org/.

The Platform for Privacy Preferences Project (P3P

As an aid in 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.

AT&T Privacy Bird

The AT&T *Privacy Bird* is mentioned in Tøndel et al. (2011) as an example of current P3P based 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.

Other Privacy Agents

Bernsmed et al. mentions a few other examples of available and suggested approaches to building privacy systems:

- **PIPWatch Toolbar** is a browser plugin that is based on users contributing privacy information about pages through the interface.
- **Privacy and Identity Mangament for Europe (PRIME)** is an EU project proposing that service providers add an extra layer for privacy sensitive transactions.
- **Collaborative Privacy Management**, in contrast to PRIME, proposes a user-centric arhcitecture that employs collaborative filetering.

2.1.3 How Machine Learning Can Improve Privacy Advice

[This section is to be expanded on]

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.

SINTEF suggests 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.

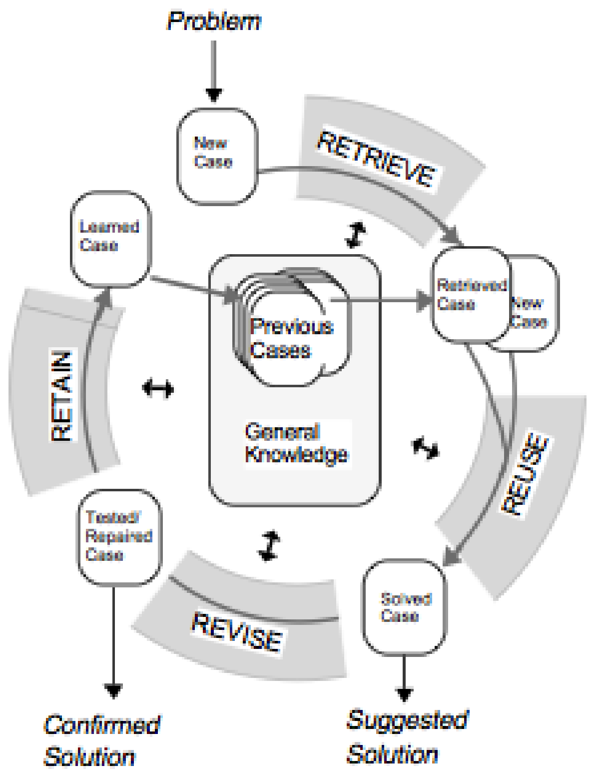


Figure 2.1: A Simplified CBR Cycle.

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 towards a scientific and statistical analysis type project, and while placing some emphasis on tuning the algorithms, this is not our primary focus[6]. Item 3 relies heavily on the effectiveness of the algorithm and may require exactly the type of statistical analysis previously mentioned. 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 stage implements the core CBR. In the second part, the focus is shifted to the community/CF portion. By 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 are 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. 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.

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 inherent 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

[6]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.

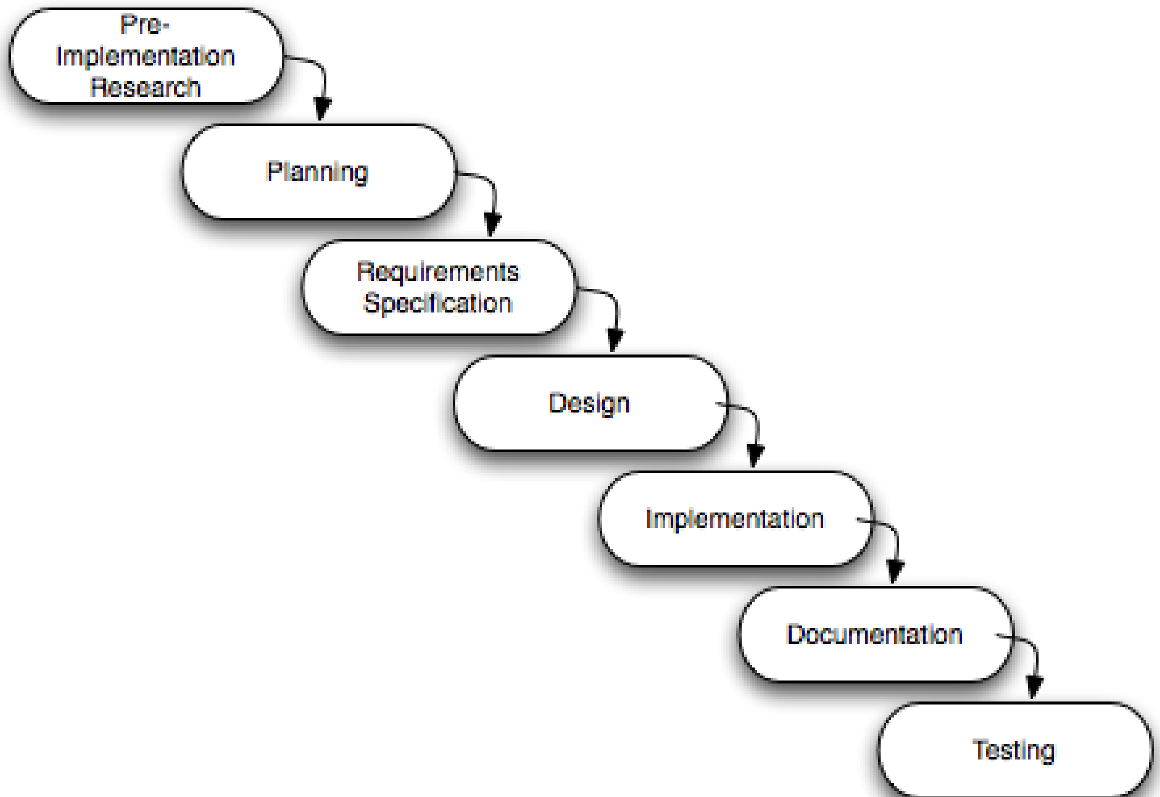


Figure 2.2: The Waterfall Model.

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

Purpose

This chapter 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.

3.1 Project Phases

The choice of development model is detailed in Chapter1.7.4. Here the various phases of the project are described.

3.1.1 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.

3.1.2 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.

3.1.3 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.

3.1.4 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.

3.1.5 Implementation

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

3.1.6 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.

3.1.7 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.

3.2 Risk Report

The term 'risk' is usually 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.

The actual risks:

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).
Risk	Probability * Consequence
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.

Table 3.3: Changes in requirments

Risk item	1
Activity	All.
Risk Factor	The requirements might change.
Probability	2
Consequence	4
Risk	8
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
Risk	15
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
Risk	15
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
Risk	6
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
Risk	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
Risk	8
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
Risk	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
Risk	9
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
Risk	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.

3.3 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

3.4 Project Plan

As discussed in Section 3.1, 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.

3.4.1 Project Milestones

Time available

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

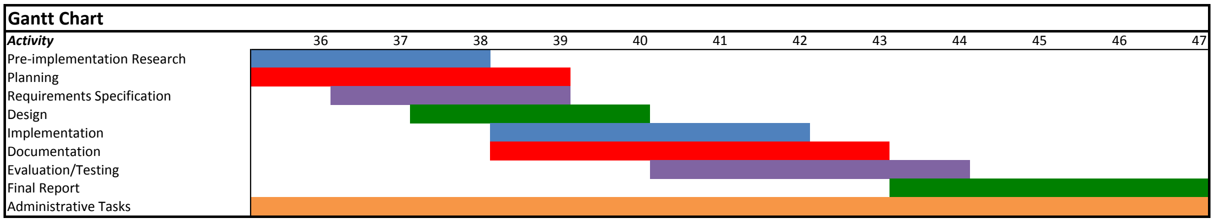
Planned Resource Allocation

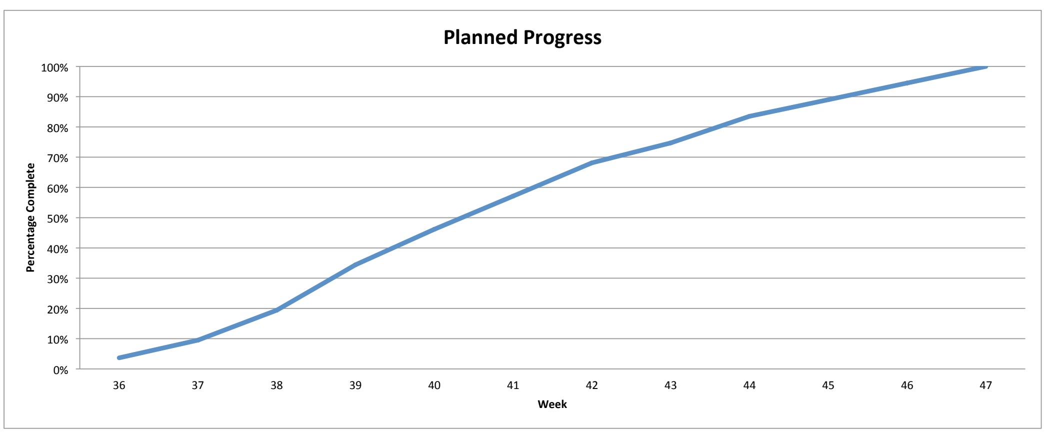
Activity	Percentage	Hours	Start Week	End Week
Pre-implementation Research	4%	100	36	38
Planning	4%	100	36	39
Requirements Specification	7%	150	37	39
Design	12%	275	38	40
Implementation	18%	400	39	42
Documentation	11%	250	39	43
Evaluation/Testing	13%	300	41	44
Final Report	18%	400	44	47
Administrative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

Purpose

This requirements specification has been prepared for and accepted by SINTEF ICT (the customer) it states 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, expected quality.

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. It could also serve as a basis for further development of the Privacy Advisor system.

4.1 Introduction

4.1.1 Background and Current Privacy Advising Software

SINTEF ICT is currently investigating new approaches to privacy protection of end-users. Tøndel 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 privacy protection, and more specifically 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 user's 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.

4.1.2 Scope

The primary aim of this project is the implementation of the core classification system described in Tøndel 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 for the purpose of more detailed research.

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 development can take two directions:

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.

While theoretically appealing, there is little empirical research documenting the applicability of CBR to the task at hand, which in turn implies that there is likely a large research work to be done for the system to be able to successfully predict user preferences. Based on this observation, the emphasis of this project will be on the first of the above directions, namely providing a research framework.

4.1.3 Overview

This document is organized as follows: Section 4.3 gives an overview over the system; its requirements and user characteristics. Section 4.4 presents four different use-case scenarios. Section 4.5 presents specific functional and non-functional requirements.

4.2 Overall Description

4.3 System Description

The overall structure of the system is detailed in Tøndel 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.

4.3.1 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.

4.3.2 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.

4.3.3 User Characteristic

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

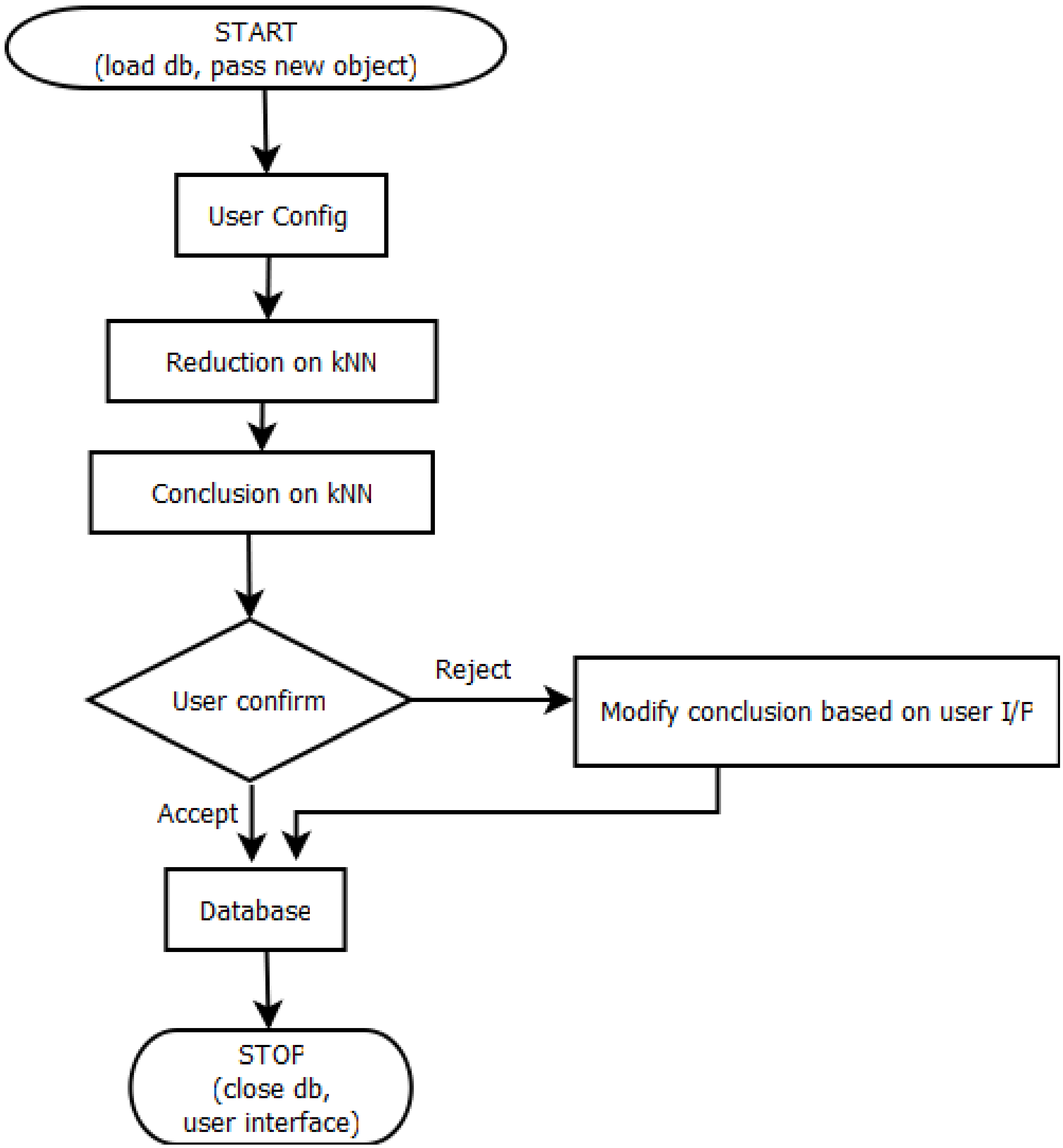


Figure 4.3: Program Flow.

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.

4.4 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 fetches the necessary data to do a local query. This query indicates, with sufficient confidence, that the site's 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.

4.5 Specific Requirements

4.5.1 Product perspective

As described in Section 4.1.2, 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.

4.5.2 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 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

4.5.3 Non-Functional Requirements

- Implementation
  - Code is written in Java following Sun Microsystems' conventions[7].
  - 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.

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

Design

Contents

Purpose

This chapter 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.

5.1 Development Tools and Technologies

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

5.1.1 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.

5.1.2 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. Javascript was used in small measure for the use of a CouchDB community server.

5.1.3 Development Tools

Eclipse was used for all Java development, while standard text editors were used for Javascript and manual P3P policy editing. The CouchDB server was hosted on a VMware virtual machine, and was accessible via web interface on most standard web browsers.

5.1.4 Local Databases

For simplicity's sake, it was decided to use flat files to store privacy policy data between new problems in the CBR cycle. The provided class uses Java's `Serializable` interface. However, the output functionality is written in a generic fashion to simplify and allow use of database systems such as MySQL, CouchDB and so forth.

5.1.5 Non-local Database

The non-local database, hosting a broader selection of cases than the local history, has been demonstrated on a CouchDB server. Chosen for the ease of developement for those unfamilier with MySQL, CouchDB queries are written in Javascript, and all object are represented in JSON (Javascript Object Notation), making object transmission simple.

5.1.6 Third Party Libraries

For ease of development, multiple third party libraries were used, specifically for parsing, logging, and network communication. Additional third party libraries were considered, but not used for the CBR system, and k-nearest-neighbors implementation.

Third Party CBR System

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).

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.

Commandline Input Parser

A standard commandline input parser, provided by the Apache Commons, was used for parsing all commandline input. While the project's authors could have written their own parsing code, this simple task made quick by using a simple library.

Logging

An additional standard logging library was used for the generic Java logging functality, again from Apache Commons.

CouchDB and JSON

CouchDB and JSON functionality was implemented through the combination of gson, a Google-developed, free-use JSON parser for Java, and LightCouch, an open source library handling connections to CouchDB libraries. gson appears to be a widely used JSON parser, handling conversion of JSON to custom object and vice-versa, and is required for several different CouchDB libraries, including LightCouch. LightCouch is a lightweight library which holds the connection details for a given CouchDB server, and providing communication to said server. Several CouchDB libraries exist, but many are no longer active or functional projects. CouchLight was the simplest library found providing the requisite functionality.

5.2 Standards

To achieve clean and reusable code, the project has adopted Oracle's Coding Conventions for the Java Programming Language[8]. This is mentioned in the requirements specification due to the high likelihood of the customer having to change the source code for later adaptations.

5.3 Architecture

In implementing Privacy Advisor, a class structure is built around the CBR agent model discussed in Tøndel and Nyre. Refinements needed to be made include data structures for storing policies,

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

databases, choosing actual algorithms, and so forth. This section describes how the design in broad terms.

5.3.1 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.

Figures **??** and 5.5 illustrate 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.

CLI and I/O

Privacy Advisor can be run using either a command line interface (CLI) or a graphical user interface (GUI). Both the CLI and the GUI are built on top of a "General Input/Output" module, GIO. GIO creates the database objects and issues the proper commands to the CBR framework based on user input.

GUI

Weights and Configuration Files

As an addition to passing command line arguments, GIO also reads a text based configuration file containing CBR and database settings. The configuration files are detailed in the User Documentation in table 7.3.2.

CBR

Input from the UI is passed on to the CBR framework. CBR in turn references three other key modules, a *reduction* algorithm, a *conclusion* algorithm, and finally, a *learning* algorithm.

The reduction algorithm searches the database to find the most similar cases to the new case presented. The canonical reduction algorithm is k Nearest Neighbors, discussed in section 6.1.1. The conclusion algorithm looks at the set of cases returned by the reduction, and decides on the most appropriate action for the novel case. It also returns a measure of confidence in the conclusion reached.

Finally, a learning algorithm allows for automatically tuning the parameters used for distance calculations. This is discussed further in section 6.1.1.

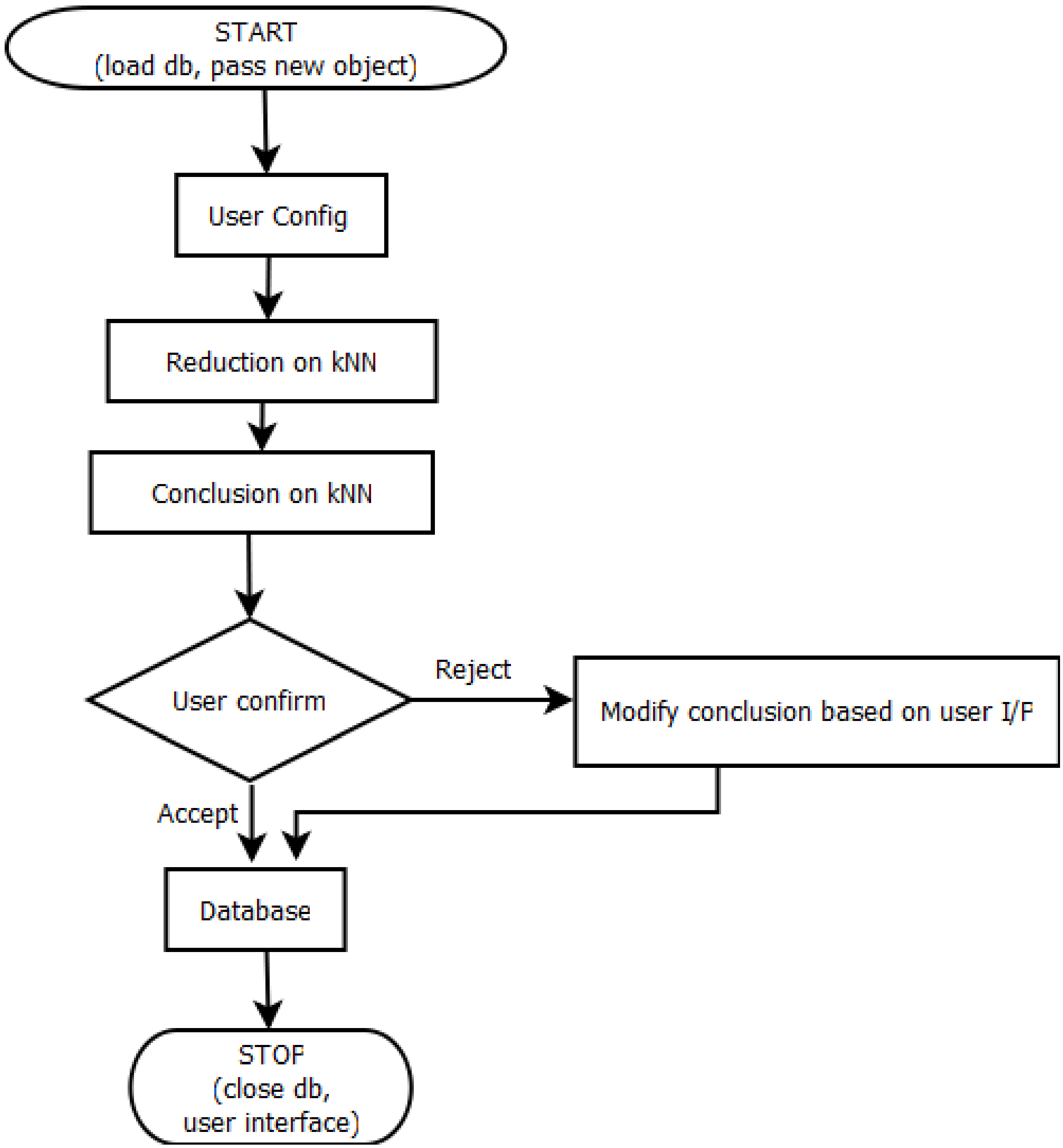


Figure 5.4: Program Flow.

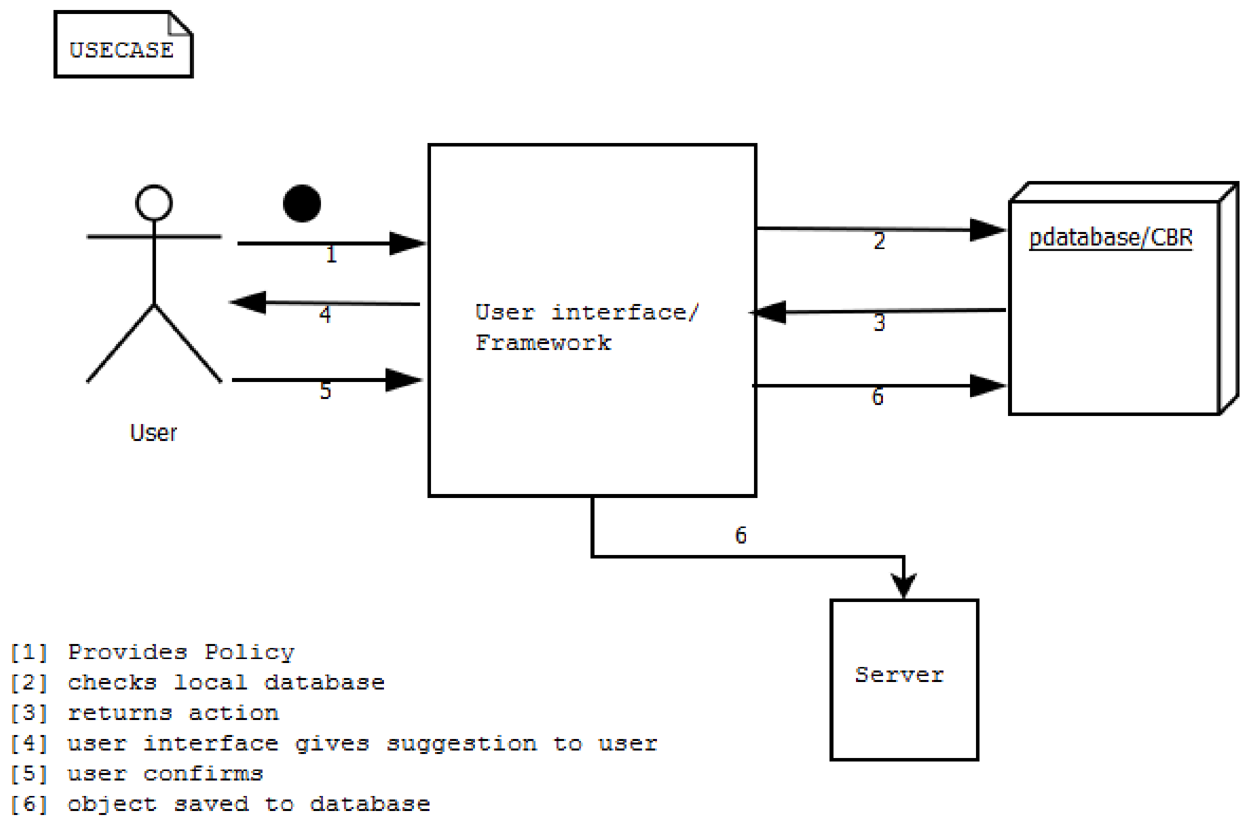


Figure 5.5: System Overview.

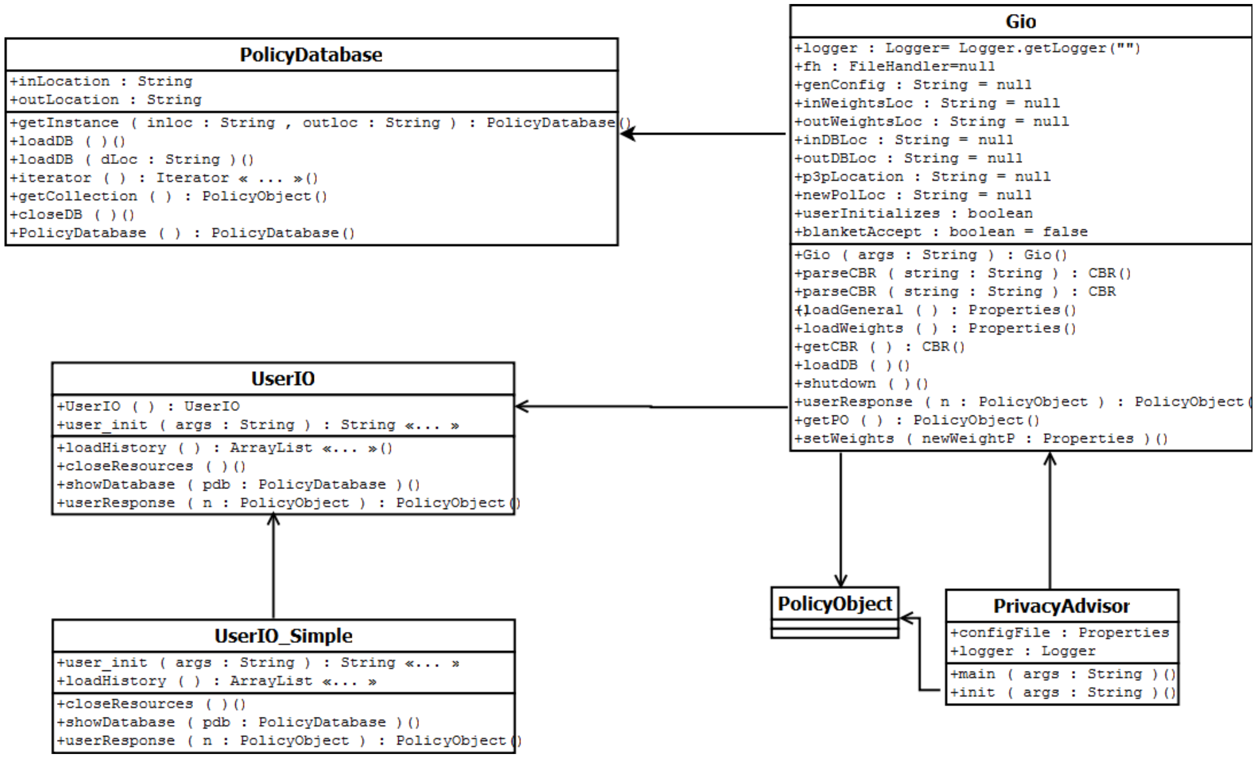


Figure 5.6: System UML.

An overview of the CBR system is given in Figure 5.7...

P3P Policy Objects

An overview of the Policy Object is given in Figure 5.8...

P3P Policy Database

The local case history, maintained in a local database, is stored via a concrete class implementing 'PolicyDatabase'. This abstract class details the required methods for a local policy database: a singleton constructor for the database object; a call to load the database once constructed, from disk; a method adding a single policy to the database; a method returning a Java iterator over the stored PolicyObjects, and a call to return all policies from a given domain.

In order to ensure consistency, the local policy database enforces singletonness. The database object itself is constructed without the actual history, requiring a seperate parameter-less 'loadDB' call on it to load policies from disk to the class, if necessary.

During the CBR cycle, it becomes necessary to check past cases for relevancy during the 'retrieve' phase. This is accomplished by using the standard java Iterator return by `getiterator()`. Finally,

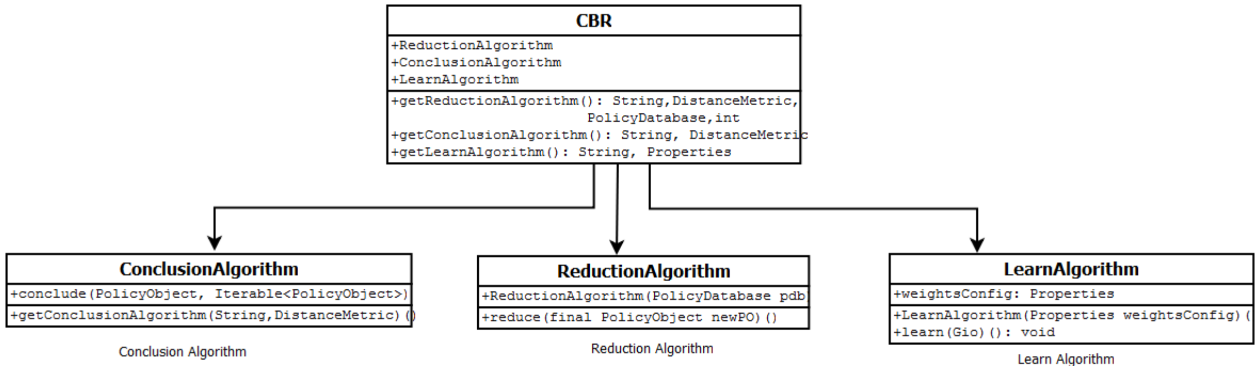


Figure 5.7: CBR System.

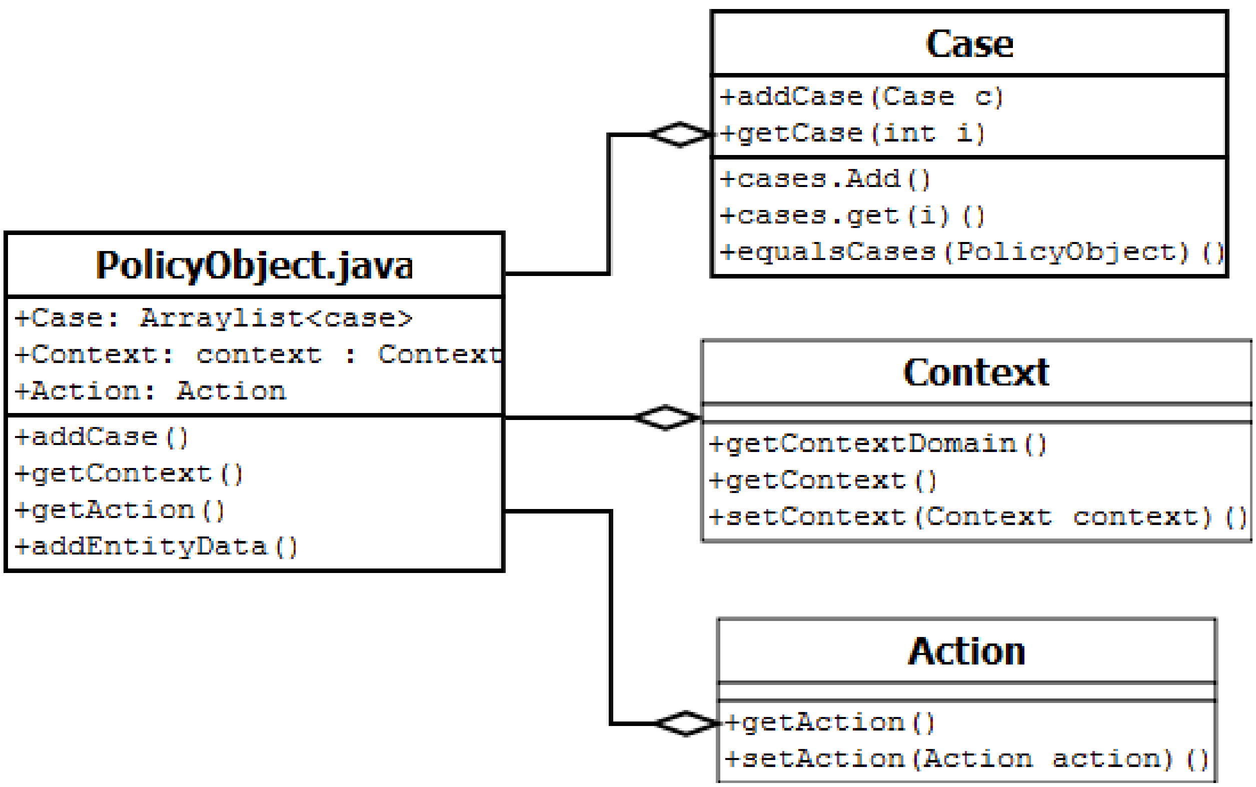


Figure 5.8: Policy Object.

the CBR cycle concludes by saving the new case (using 'addpolicy(newpolicy)'), and closing the database using 'closeDB()' (which is when the cases would be saved to disk).

An overview of the Policy Database is given in Figure 5.9...

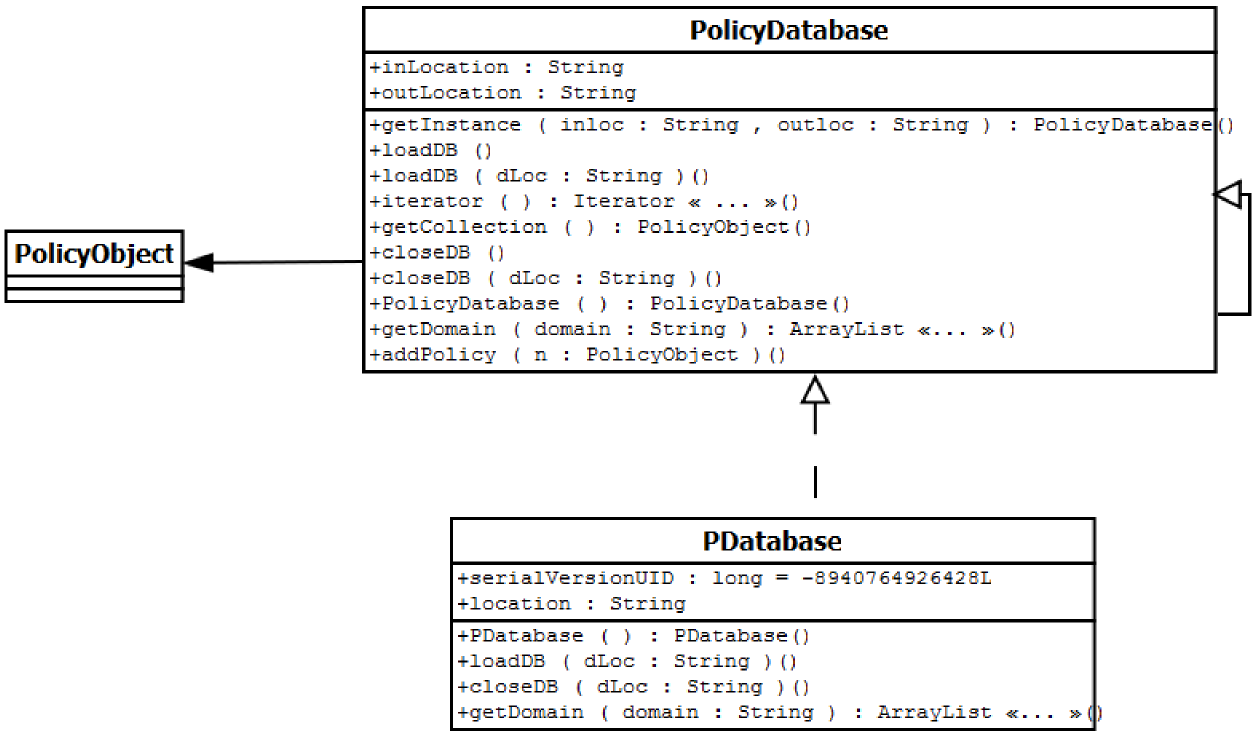


Figure 5.9: Policy Database.

5.3.2 Interfaces

5.3.3 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

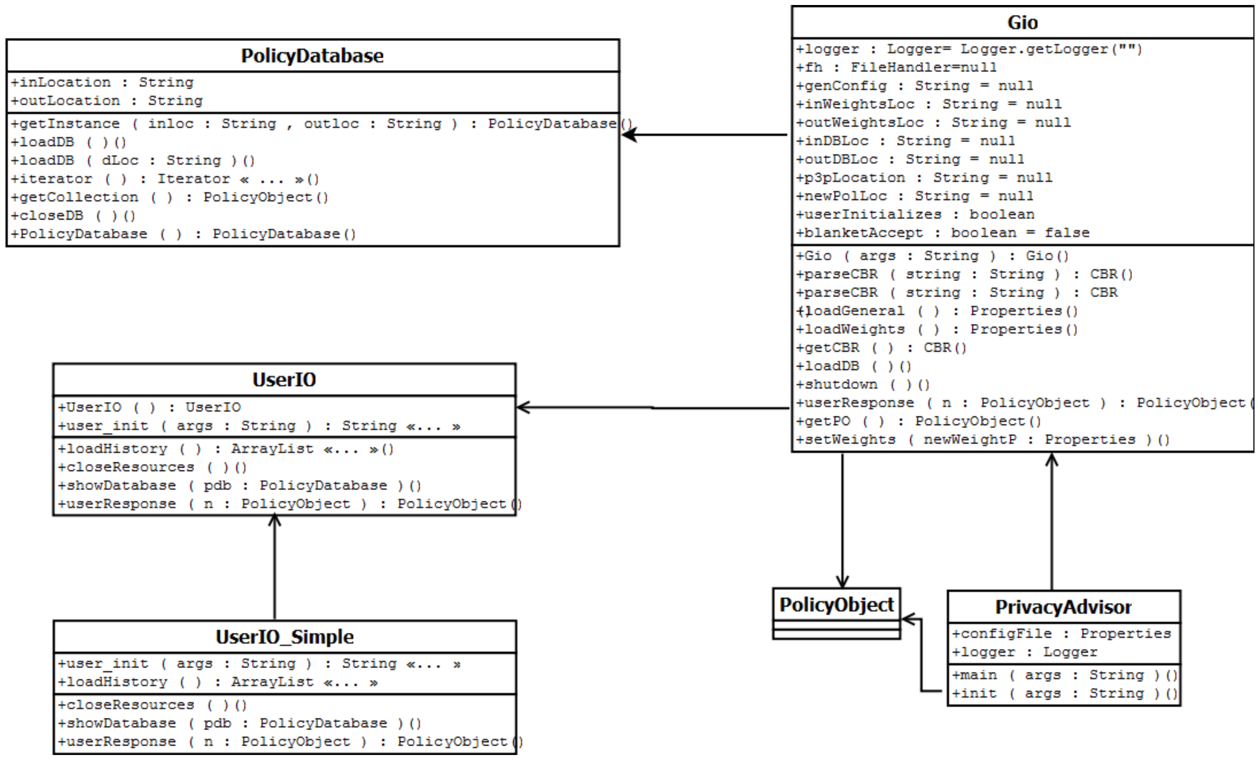


Figure 5.10: Input/output and user interfaces.

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.

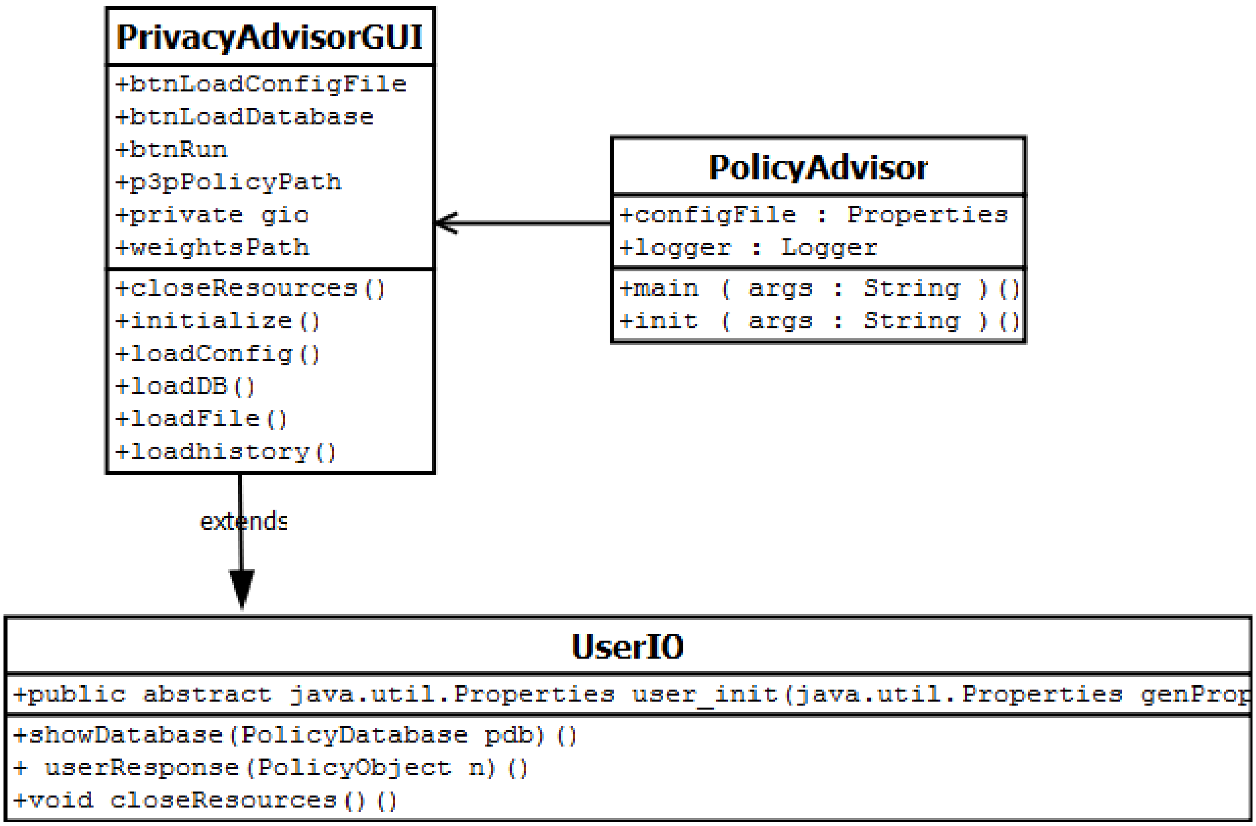


Figure 5.11: GUI interfaces.

Implementation

Contents

Purpose

This chapter explains the implementation phase of the project it provides 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 data structures that are used to represent policies and how comparisons are done on these data structures.

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 6.1.3.

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 Confidence Measure

6.1.3 Distance Measures

This section discusses the distance measures implemented for comparing two P3P policies. First, it gives a brief overview over the mathematical theory behind distance measures and looks at some standard distance measures. It then looks at what adaptations need to be made to the standard approaches for them to fit the domain of P3P policies.

Definition

Mathematically, a *metric* or *distance function* is a function that defines the *distance* between two objects in a set. 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 Eucldean, 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

- *Manhattan distance* is function computes the distance that would be travelled to get from one data point to the other if a grid-like path is followed.
- *Hamming distance* is defined as number of positions in which a source and target vector disagrees.
- *Levenshtein distance* is based on Hamming distance but adds also operands as insertion, deletion and substitution.
- *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 Advice

In the paper "Towards a Similarity Metric for Comparing Machine Readable Privacy Policies", 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 as 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 than between ours and unrelated.

6.1.4 Implementation

Bitmap Representation

A bitmap (or bitstring) is a way of representing a set of objects. It is a mapping from a set to a vector of fixed length, where each member of the set corresponds to a particular entity in the vector. For example over a language $L = \{a, b, c, d\}$ for a sets $\{a, b, c\}$ a bitmap representation will look like $[1, 1, 1, 0]$ where first integer represents a and last integer represents value d. This way it doesn't matter what order the values are arranged and how many values are so set $\{d, b\}$ over same language L will be $[0, 1, 0, 1]$ where first value still representing value a, or in this case, absence of the item a. Calculating intersections or unions over bitmaps A and B uses bitwise Boolean operators. 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. For the language L, consider the weights $w_a = 1, w_b = 2, w_c = 4, w_d = 3$. Since the bitmap representation of the set $\{a, b, c\}$ is $[1, 1, 1, 0]$, the weighed bit-map will be $[1 * w_a, 1*_b, 1*_c, 0 * w_d] = [1, 2, 4, 0]$. For another set $\{d, b\}$, it will be $[0, 2, 0, 3]$.

Privacy Policy Representation

This section describes the data structures used to represent policies. On the top level, `policyObjects` 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. The privacy policy of www.ticketmaster.com, as displayed in *Privacy Advisor* is shown in Figure 6.12. In the TicketMaster example, the data items concerned in the policy are the user's name, birth date, home and business contact information as well as a few "technical items". For every of these data items that being described in a policy, we create a `Case` object inside the `policyObject`. So in the TicketMaster example, `user.name` and `user.bdate` are cases.

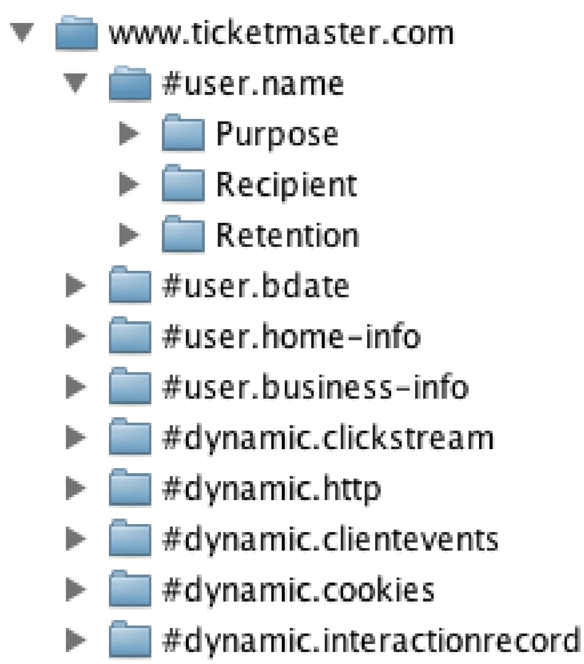


Figure 6.12: Sample P3P Policy as shown in Privacy Advisor.

Each statement contains of four fields: the type of data collected, purpose, recipient and retention. Purpose, recipient and retention can contain one or many given values, the combination of what 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. Figure 6.13 shows TicketMaster's policy with regard to the user's name. In this case we see that TicketMaster stores data for several purposes, both administrative and marketing for an indefinite time period. The statement also reveals that data is shared with unrelated third parties.

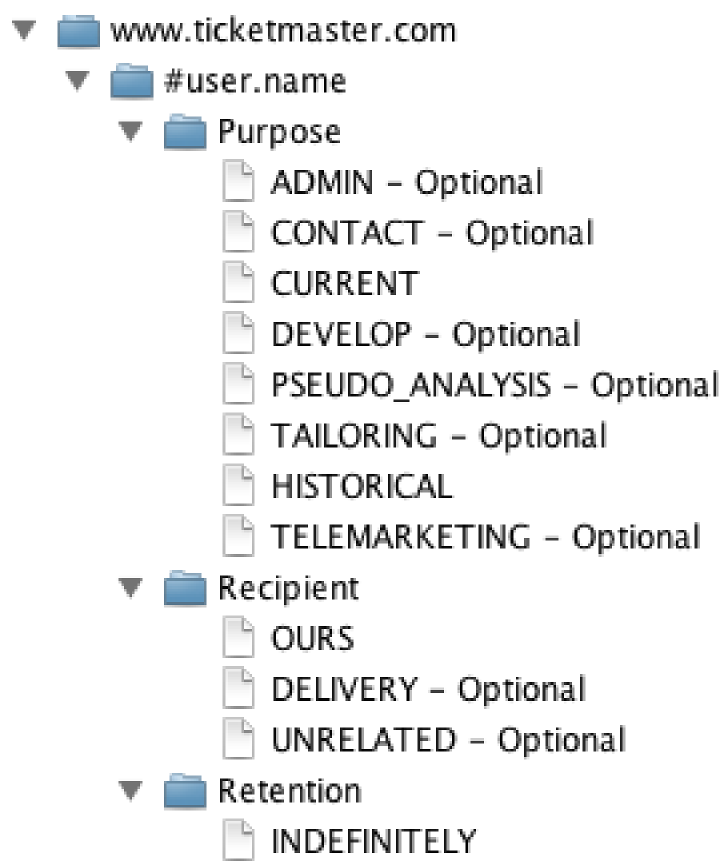


Figure 6.13: Indepth look at TicketMaster's policy regarding the user's name.

We then see that a `Case` 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. Since data-type is a string value the bitmap representation would be impossible. The distance between data-types can be calculated by either string comparison or ontological tree. The Hamming distance for weighed bitmap is easy to implement it can be represented by Boolean expressions. With creation of bitmaps is a fast algorithm with a linear run-time. The easy way of predicting the result and fast run-time is the strengths of this implementation.

Data-type-string similarity

The way data type is structured in P3P policies makes it possible to calculate a data-type distance by just parsing data-type-string. Every data-type string has to start with one of the four previously mentioned fields, then after a dot a sub-field is followed after other dot a sub-sub-field. Let us say we have your simplified ontological tree just within the syntax of data-type-string with an "invisible" root node over those four fields. This way when we have two data-type-strings for comparison we can easily count number of ancestors from string A to string B. For instance String A is "user.home-info.postal" and B is "user.bdate". Number of nodes we need to take from stringA to stringB is 3.

The weakness is that without weights it is more of string comparison than ontological tree.

It is possible to create a weight system so this algorithm will work like a full ontological tree. For example if each node had a value/weight it would be possible to simply take difference between StringA's 1st 2nd and 3rd field and respectfully StringB's fields.

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

The global similarity

The global similarity part was not as easy as creating a bitmap. The number of cases a policy can be 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 policyB.

```
Sum=0
For caseA in PolicyA
  Min=infinity
  For caseB in PolicyB
    Dist=Compare(caseA, caseB)
    If dist<min then
      Min=dist
  Sum=sum+min
Distance=sum
```

Figure 6.14: Algorithm for similarity computation.

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 two 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 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 will 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

As the CBR model requires, data must be stored in a persistent fashion between cycles, and thus, while the program is not running. Furthermore, the ability to load raw cases (P3P policies, in this case), is needed not only to initialize a database, especially in a repeatable fashion, but also for establishing the new case under consideration. These requirements are handled by two distinct bodies of code- the P3P parsing class, and the policy database class. Furthermore, there is also the need for a centralized, multi-user database, accessed not locally but across a network, when the local database proves incapable of providing a satisfactory solution to the problem.

6.2.1 P3P Parsing

6.2.2 Local Database

The provided local database class (`PDatabase`) is located in the `datastorage` group, and implements the abstract model `PolicyDatabase`. It enforces the singleton model, and is based on `java.io.serializable` for saving objects on disk, and stores objects in a private `ArrayList<PolicyObject>`. The methods function precisely as laid out in the Design section, and offer an example of how a PolicyDatabase should function.

Please note the lack of direct access to the internal object storage- thus enabling the use of MySQL, or some other data storage format for not only saving the case history between CBR cycles, but also during the cycle itself, provided that a policy can be added and an iterator provided over the dataset. As with all of the modular classes (algorithms, policy databases, network resources, and user interfaces), the class used to load, interface with, and save the local database can easily be switched out by setting the appropriate option to the qualified class name in either the configuration file or on command line (in this case, the option is `policyDB`).

6.2.3 Remote Databases

Access to a community server or network resource is contingent on two things- a remote server, and the class used to interface with said server. In the framework provided, the two classes provide access to a distinct dataset- in fact, while it is possible to create a local substitute providing identical functionality to the existing PolicyDatabase and CBR algorithms, the Java NetworkR class providing the abstract implementation does not provide access to the same case history. Responsibility for providing a reasonable solution in this combined 'retrieve-reuse' phase (the execution of algorithms in a similar model to those used in the local CBR) is intended to reside on the server side, as the resources consumed in transmitting the much larger community case history to a local client are far more significant than those required to run the query locally, and provided only a suggested solution.

Provided in the code is an implementation using a CouchDB database, and a class. The Java code uses several third party libraries (CouchLight and gson, described above) to communicate with a CouchDB database located on a virtual machine provided for the project by NTNU IDI.

CouchDB server

The virtual machine hosting the CouchDB server is located at vm-6113.idi.ntnu.no, with the server using port 5984 for communication. All test queries and preliminary code was developed using a set of tools known as `couchapp` in javascript. The queries can be broken down into two distinct parts: the map-reduce, and the view.

In CouchDB, a map-reduce operates on a dataset to produce a direct report of the result, with minimal formatting (formatting is possible, but not used in this case). A map-reduce is composed of a 'map' javascript function, which is applied to every document in the database and can be used to reveal the useful attributes of the document (e.g., the value of an attribute or sum of attributes). This resulting list can be sorted, range-limited, etc. The reduce function is applied to initial results of the associated map function (required), and combines the results in a recursive fashion- the function will be applied to the initial results, then the combined results of the first set of reduce operations, then the second, and so on.

A view formats the results of a map-reduce operation (offering an easy way to switch between, say, http, xml, and json results, depending on the view applied to a given map-reduce). The code provided includes a test view that simulates the results of a query- it sends a valid, JSON-encoded Action object that carries the server's suggested solution to the client. A proposal for a map-reduce based on the same algorithms presented above (the k-nearest-neighbors algorithm and conclusion_simple) is included, but not complete.

Java Connector Class

The remote database class provided is entitled `NRCouchdb`. This class implements the abstract NetworkR class detailed above. The server instance specific options necessary to provide location, port, password, etc, to the class are pulled from the `NetworkROptions` option, valid at command line as well as in the configuration file. A replacement class should pull any necessary options from a comma-separated list at the same location.

As with all of the modular classes (algorithms, policy databases, network resources, and user interfaces), the fallback resource (usually on a remote server) can easily be switched out by setting the appropriate option to the qualified class name in either the configuration file or on command line (in this case, the option is `NetworkRType`).

6.3 User Interface

6.3.1 CLI

6.3.2 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[9]. 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.

7.2 Installation

7.2.1 Local Installation

To make the program run outside of the Eclipse environment it have to be exported as a runnable jar file. This can be done by right clicking the project in eclipse, choose `Export`, type in `jar` in the

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

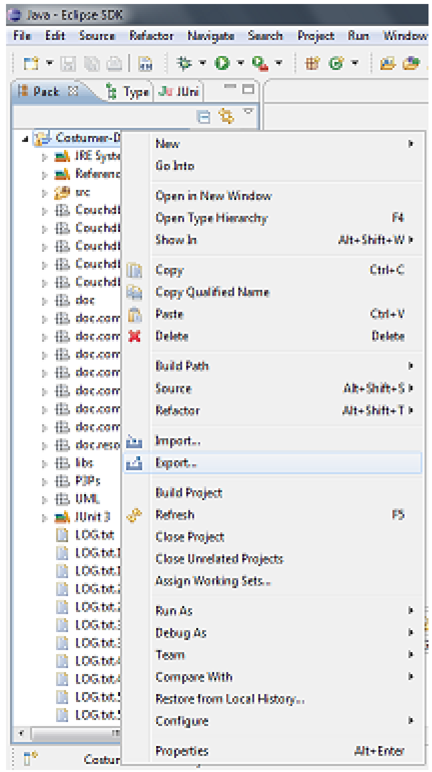


Figure 7.15: First step: right click and choose ”export”.

new window that opens up and choose `Runnable Jar File`, before clicking `next`. These steps are showed in the two figures 7.15 and 7.16.

Then we have to choose which classfile we want to be the main class; in the `Launch configuration` dropdown menu we can choose between the two classes `PrivacyAdviser` and `PrivacyAdvisorGUI`. The first one is choosen if we want to run the program by using the command line, and the second one if we want to use the GUI. Then choose the export destination, and choose `Package requires libraries into generated JAR` as the library handling. Pressing finish now should start exporting the program. These steps are shown in figure 7.17.

Before running the program we have to put the PrivacyAdviser.cfg file in the same folder as the exported jar file. And if the database file and the weights file isn't at the same folder, their location have to be specified in the PrivacyAdviser.cfg file. To run the program, open the command prompt and locate to the folder where the jar file is stored. Then write the command `java -jar filename.jar`, and now the program should start (even if it's the cli or gui version). This is shown in figure 7.18.

7.2.2 Server Installation

The CouchDB server is a default Ubuntu installation on a VMware virtual machine provided by NTNU IDI, with CouchDB installed using the command `sudo apt-get install couchdb`. The

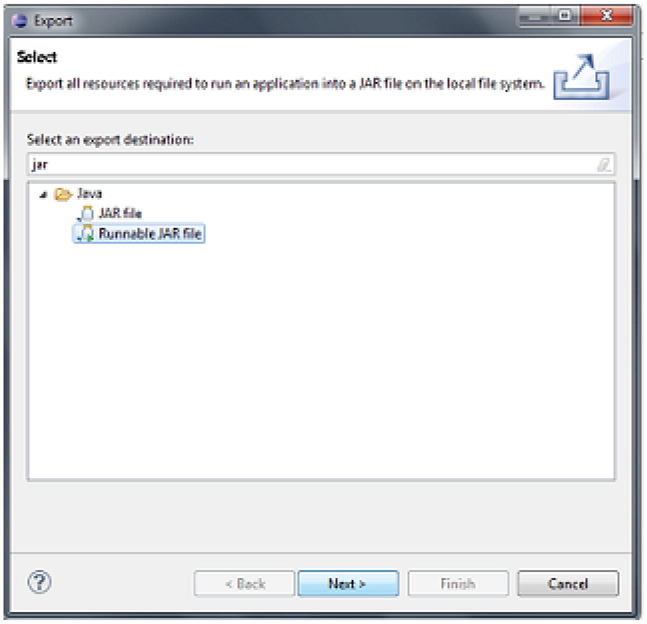


Figure 7.16: Second step: choose runnable jar.

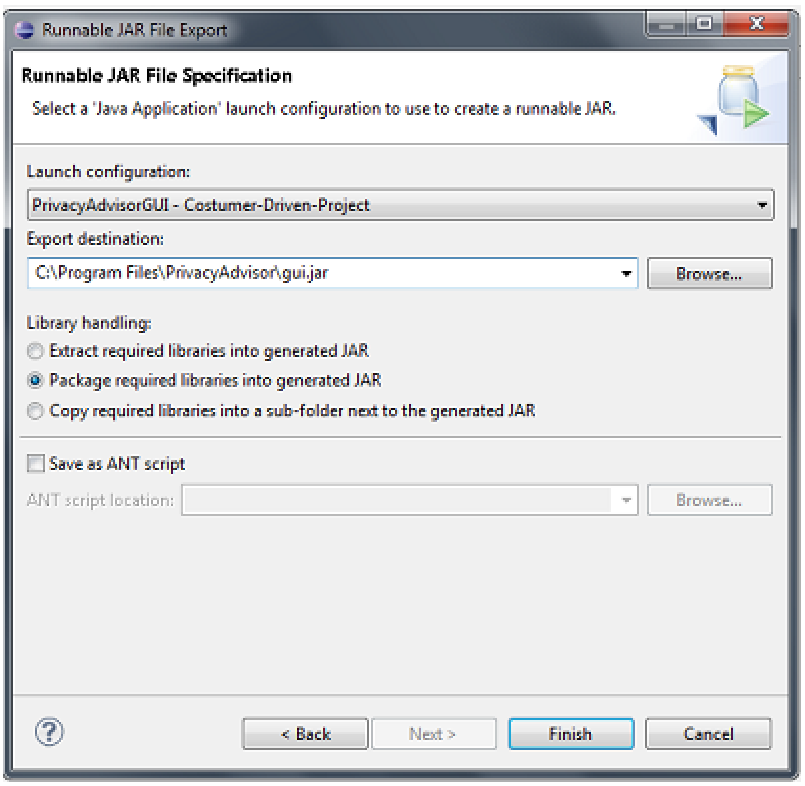


Figure 7.17: Third step: choose launch config, destination and library handling.

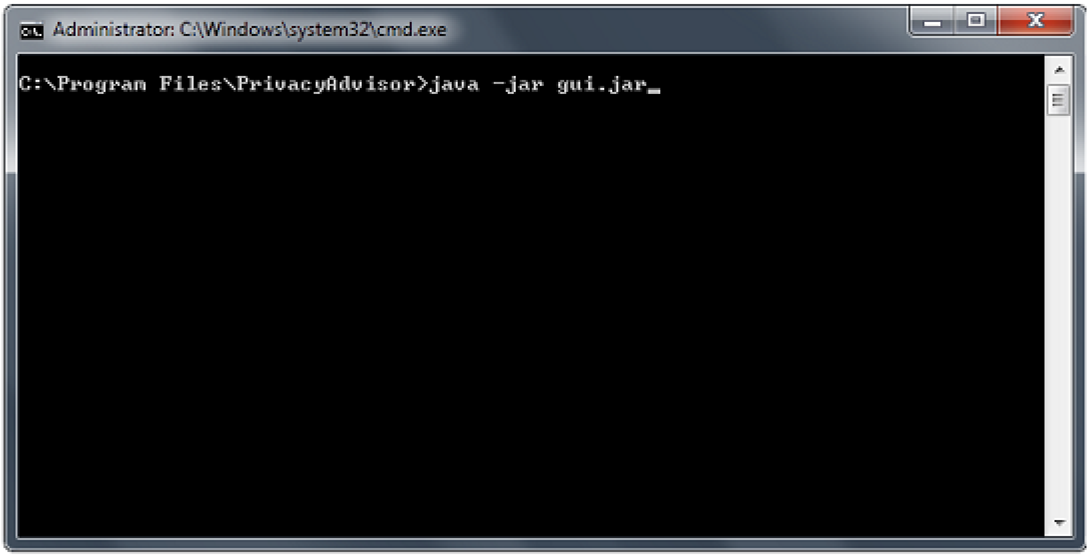


Figure 7.18: Run the program.

database server was configured via futon (the default web interface, located at localhost:5984_util) to provide a database entitled `privacydb`, with a non-administration user (details noted in default program configuration file).

7.3 User Interfaces

The Privacy Adviser consist, as mentioned in section 7.2.1, of two different user interfaces. The advantage of the GUI is that it's easier to see the contents in the database before and after the program runs. The advantage of using the command line interface is that we can work much faster than by using the graphical user interface. But the best part of using the command line interface is that we can add options when calling (starting) the program, this is explained in detail in section 7.3.2.

7.3.1 Graphical User Interface

7.3.2 Command Line Interface

By running the program from the command line we can add options (parameters) directly when starting up the program. If we run the program normally, as described in section 7.2.1, the options in the PrivacyAdviser.cfg will be used. But when we add options in the command line when starting the program, the added options will override the ones in the config file. As an example we can take the location of the database. By default the options in PrivacyAdviser.cfg is set to replace the old database with the new database when the program exits. There are two ways to change this. If we never or rarly want the new database to replace the old one we can just change the `outDBLoc` option to be different from the `inDBLoc` option in the config file. But if we want to change the location/name just once, we can add an option in the command line like this: `java -jar privacyAdvisor.jar -outDBLoc newDatabase.db` We can add more than one option in this manner, and for the options that we don't specify, the options in the config file will be used. For a complete list of the options see table 7.3.2.

Configuration Files

Table 7.3.2 gives an overview over the configuration file parameters.

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.

Building a Database

CLI: To build a new database from a directory `P3PDir` holding P3P files, Privacy Advisor can be called from the command line in the following fashion: `PrivacyAdvisor -newDB true -outDBLoc new.db -p3pDirLocation P3PDir`.

GUI: To build a new database in the similar fashion using the graphical user interface, the configuration window can be set up similarly to that illustrated in figure[XXXX].

Loading and Viewing a Database

Parsing a P3P Policy

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.
- **Learning of our algorithm:** 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.
- **Interface testing:** 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 cases

See the appendix for the test cases.

8.5 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.

8.6 Test schedule

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

8.7 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

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?

The first section evaluates the software system in terms of the objectives and requirements worked out prior to design and implementation. Section 9.2 evaluates the project process and group dynamics in terms of how well team members have cooperated in reaching the objectives and the experiences made throughout the process. Sections 9.3 and 9.4 evaluate the appropriateness of the tools used in developing the system and the resources committed to the project in manhours relating to the original project plan. Finally, section **??** evaluates the course as a whole.

9.1 Software Evaluation

9.2 Group Dynamics and Organization

9.2.1 Internal Dynamics

The project team was slightly heterogenous, with two international students and the remaining five Norwegian. For this reason, English was chosen as the offical language of the project. This may to some extent have impacted the degree of participation of some team members during meetings, and discussions came to be dominated by a minority.

9.2.2 External Interests

The key stakeholders in the project aside from the project team were the SINTEF (the customer) and the advisor.

Customer

The main pillar in communication with the customer were status meetings on a semi-weekly basis. In addition to these meetings, status reports were e-mailed to the customer on a weekly basis and the customer had full access to all planning documents through an online repository.

All in all, communication with the customer has been very effective. Once the priorities and ambitions of the project were settled (i.e. pursuing the construction of a CBR system for testing), the team and the customers expectations have been very much alined and no misunderstandings worth mentioning have occurred. The customer has also been very helpful in reading over phase documents and providing feedback.

Project Advisor

The project team has had weekly meetings with the advisor providing status updates and laying out planned work for the upcoming week every wednesday. The advisor has also reviewed the report on two occasions.

The feedback on the project report and phase documents has been very through and helpful. He has also

9.3 Tools

Referring to choices discussed in section 5.1, the key software tools and languages used are Java, Git, and Google Docs.

9.3.1 Programming and Implementation

Java

The choice of programming language for the project was very much up to the project team as the customer had no particular preference, and the project did not require anything beyond what is catered for by most general purpose languages. **Java** was therefore selected selected as implementation language based on the following points:

- All team members had some previous experience in Java programming from previous coursework.
- Good documentation and a large amount of third party libraries available on the web.

While all team members did have a Java programming background, there were clearly vast differences in skill levels. This was the cause of some frustration and extra work. In retrospect, more time should have been spent on clarifying background and interests of each team member so as to better allocate workload. Some will also argue that a more "light-weight" programming language, such as Python, would have been a better choice, even despite the lack of previous experience.

Git/GitHub

Git and GitHub were chosen as version control system (VCS) and code repository. Several team members had only marginal experience with VCSs and did not commit to learning this in a serious manner, which led to somewhat slow progress in the start. However, once these problems were resolved, it can be agreed that Git has served its purpose well.

9.3.2 Reporting and Organizational Tools

Google Docs was used as repository for all "temporary" documents such as agendas, status reports, drafts and so forth. All "major" documents were typeset in LaTeXand stored at the GitHub repository.

Google Docs

LaTeX

Being the de-facto typesetting system for academic writing, and the only one that some group members had experience with, LaTeXwas chosen for writing the final report as well as all phase documents. As with GitHub, though to a lesser extent, there was a lack of commitment to learning by certain group members, causing frustration and extra work on a later stage in the project.

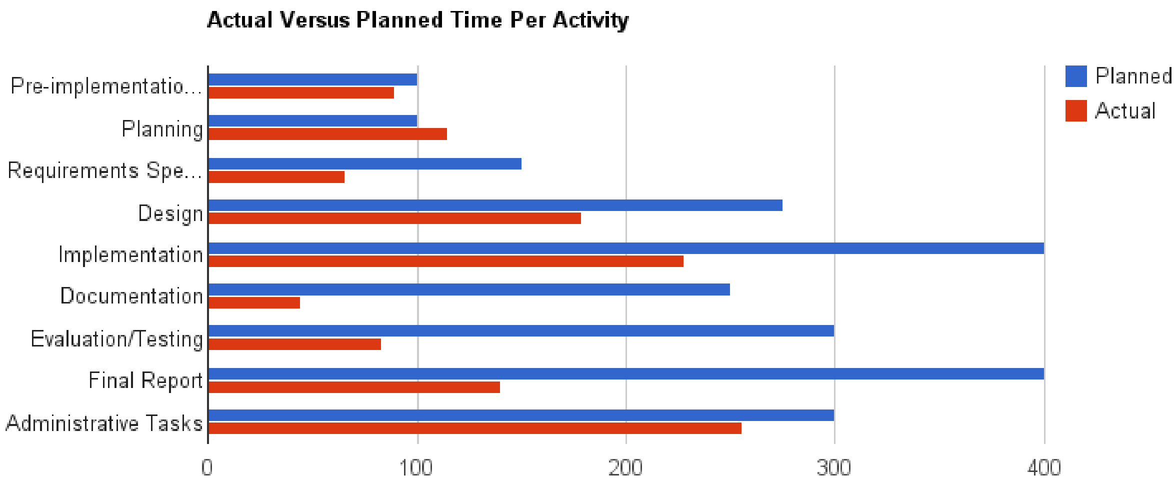


Figure 9.19: Actual vs. planned time. By project phase.

9.4 Resource Use

Figures 9.19 and 9.20 illustrate the discrepancies between actual and planned time use on all activities, both cumulative over time and over each phase of the project.

9.5 Risks that Occurred

Referring to the discussion in section 3.2, we can state in retrospect that few of the risks we anticipated occurred, at least with major consequences. Some problems occurred building a P3P parser (risk item 3), causing some initial delays as it took longer time than anticipated to get the system fully working. This problem was in part due to some unclear notions in the specification of the dataobjects storing P3P objects.

Furthermore, the project has not proceeded without any conflicts. While roles and responsibilities were agreed on early on in the project phase, it turned out that these assignments did not distribute the workload very evenly.

9.6 Course Evaluation - TDT4290

Finally, we include some notes on how well the objectives of the course *TDT4290 - Customer Driven Project* itself have been met.

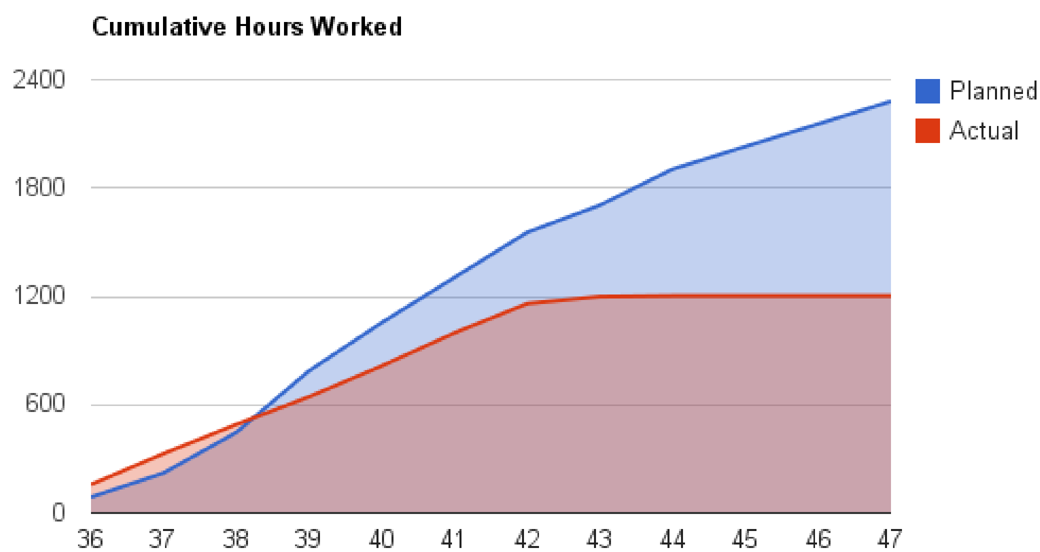


Figure 9.20: Actual vs. planned time. Cumulative.

Conclusion

This project has designed and implemented a privacy agent based on the research ideas set forth by SINTEF. The Privacy Advisor software as it is today, is a case based reasoning engine that can provide advice with regard to P3P privacy policies given knowledge of previous user actions. Both the idea and the software are still work in progress, which is reflected by the system's user interface which are oriented towards testing. It is built in a modular fashion, so that the knowledge base, the retrieval algorithm and the similarity metrics, which are likely to be the parts of the system that require most of the tweaking, can be substituted without affecting the remainder of the software.

10.1 Future Development

10.1.1 Testing

For the Privacy Advisor system to be truly valuable, it has to be made available to end users; the most likely realization being in the form of a web-browser plugin. For this to occur, much testing is still required.

10.1.2 Networking

10.1.3 Distance Metrics

Part IV

Appendices

Test Plan

Test cases

Item	Description
Name	Command line interface (CLI) functionality
Test identifier	UNIT-01
Person responsible	Henrik Knutsen
Feature(s) to be tested	That all possible commands are working correctly when using the CLI. That input with incompatible commands does not run.
Pre-conditions	Code for input handling for all possible commands. Code for error handling for invalid inputs.
Execution steps	1. Start the program with all possible commands. Run once for each command. 2. Start the program with all possible combinations of incompatible commands. Run once for each combination of incompatible commands.
Expected result	1. The program starts successfully, performing the action connected to the command. 2. The program aborts startup. Gives error warning: Invalid arguments.

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 their 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	1. All required fields and values are obtained and written down. 2. The P3P XML is parsed successfully. Its content is printed to console. 3. The results are identical. The the two outputs on paper and in console have the same fields, with 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. 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. 5. Compare the written policy A and the read policy A. 6. Compare the written policy B and the read policy B.
Expected result	1. Policy A is successfully written to the database file. 2. Policy B is successfully written to the database file. 3. Policy A is successfully read from the database file. 4. Policy B is successfully read from the database file. 5. The written policy A and the read policy A are identical. They both have the same fields, with the same values. 6. The written policy B and the read policy B are identical. They both have the same fields, with the same values.

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

Item	Description
Name	Algorithm classification
Test identifier	UNIT-05
Person responsible	Henrik Knutsen, Dimitry Kongevold
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 distance algorithm on the single policy and the history. 3. Manually calculate and write down the distances between the single policy and each of the policies in the history. 4. Compare the distances that are returned by the algorithm with the manually calculated distances. 5. Run the reduction algorithm with input to find the k nearest policies. 6. Manually find the k policies with the lowest distance. 7. Compare the policies returned by the reduction algorithm and those found manually. 8. Run the conclusion algorithm.
Expected result	1. The policies are successfully stored in the database. 2. The distance algorithm runs successfully for the chosen input, and returns the distances between the single policy and each of the policies in the history. 3. The distance between the single policy and the policies in the database are found and written down. 4. The distances returned by the algorithm, and the respective distances calculated manually are identical. 5. The reduction algorithm runs successfully for the chosen input and returns the k nearest policies. 6. The k nearest policies are found and written down. 7. The k policies returned by the reduction algorithm and the k policies found manually are the same policies. 8. The conclusion algorithm runs successfully.

Item	Description
Name	Algorithm learning
Test identifier	UNIT-06
Person responsible	Henrik Knutsen, Neshahavan Karunakaran
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. Code for writing to the database. Algorithms for classification and learning must be implemented.
Execution steps	1. Obtain and write down 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. Accept the recommendation. 5. Choose to save the new policy to history. 6. Obtain and write down the contents of the weights file. 7. Compare the contents of the weights files obtained in steps 1. and 6.
Expected result	1. The contents of the weights file is obtained and written down. 2. All the chosen policies are successfully stored in the database. 3. The algorithm classifies the new policy, gives a recommendation on what action to take, and asks the user if the recommendation is accepted. 4. The user is asked whether to save the new policy to history or not. 5. The new policy is successfully written to the database. 6. The contents of the weights file is obtained and written down. 7. At least one of the fields in the weights files obtained from steps 1. and 6. 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 policy A from the local client to the server. Print the contents of policy A. 4. Receive packet A at the (virtual) server. Print the contents of policy A. 5. Compare the sent policy A and the received policy A. 6. Send policy B from the (virtual) server to the client. Print the contents of policy B. 7. Receive policy B at the local client. Print the contents of policy B. 8. Compare the sent policy B and the received policy B.
Expected result	1. The program starts successfully. 2. The (virtual) server starts successfully. 3. Policy A is sent to the server. Its contents are printed to console. 4. Policy A is received at the (virtual) server. Its contents are printed to console. 5. The contents of the sent policy A and the received policy A are identical. 6. Policy B is sent to the local client. Its contents are printed to console. 7. Policy B is received at the local client. Its contents are printed to console. 8. The contents of the sent policy B and the received policy B are identical.