Project Report
TDT4290 - Customer Driven Project
"Privacy Advisor"

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

NTNU, Fall 2011

Abstract

This report details the design, development and implementation of the "Privacy Advisor" software in fulfillment of course TDT4290-Customer Driven Project. Privacy Advisor 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 promising machine learning technique in the field of privacy advice, which is also what the Privacy Advisor is based on. SINTEF ICT has indicated the use of the project software as part of future research and therefore emphasis is placed on implementing a broad and modular testing 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 allows for network communication with a community server that could implement a collaborative filtering approach similar to the CBR.

Privacy Advisor

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 2.6) 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 of the project are; that is, given scarce resources, which objectives are prioritized. The project planning phase seeks to make more fine grained decisions 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 5.3.4) the software system.

Finally, Part III describes the software testing stage. It also contains a chapter evaluating the project as a whole and a summary.

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 provides 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 serves as a guideline for project work and later on, project evaluation along with the requirements specification.

The project directive is intended to be 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 decisions.

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 in terms of time and effort as privacy policies tend to be very long documents formulated in an inaccessible language. This has led to the suggestion that Internet privacy can be assisted by machine learning techniques, where a particular decision is based on the user's 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.

It is important to note that the success of the first objective is by far the most critical to this project. If the core framework and CBR engine is implemented in a robust and extensible fashio that caters for Objectives 2-4, reaching those objectives will be less work. In order to be able to proceed with objectives 3-4, a considerable success from the testing phase would also be required for the system to be viable, that is, it must actually provide good recommendations. Such testing may prove to require more resources than available to this project.

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 relativly 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 ??. 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.

Area	**Role**	**Description**	**Responsible**
Administrative	Project Manager	Customer relations Requirements specification Planning Meeting minutes 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.

1.5 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.

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 minutes 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.

Meeting minutess: 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 minutes, 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. The final piece of work in the preliminary study was to settle on the development and communication tools to be used for the project.

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.

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

[1] The system is outlined in broad terms in the article Inger Anne Tøndel, Åsmund Ahlmann Nyre, Karin Bernsmed. Learning Privacy Preferences. 2011 Sixth International Conference on Availability, Reliability and Security, Vienna, 22-26 Aug. 2011. (s. 621-626). Los Alamitos: IEEE Computer Society.

[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.

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.

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

[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/.

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.

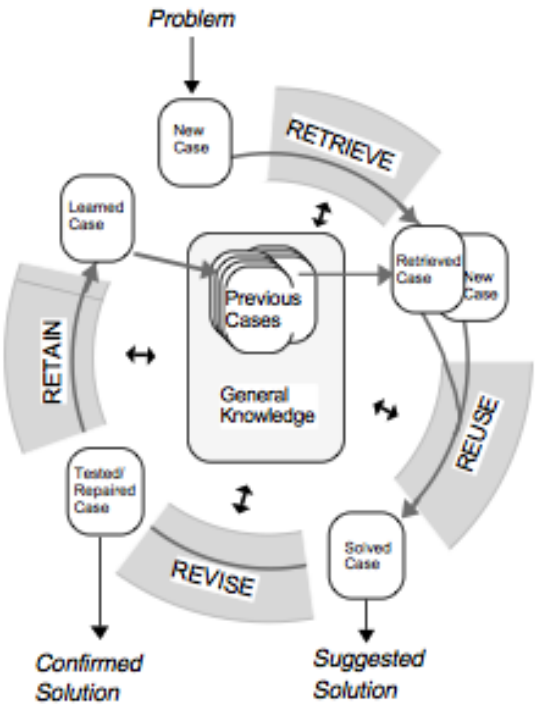

Figure 2.1: A Simplified CBR Cycle.

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 userï¿½s 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.

[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.

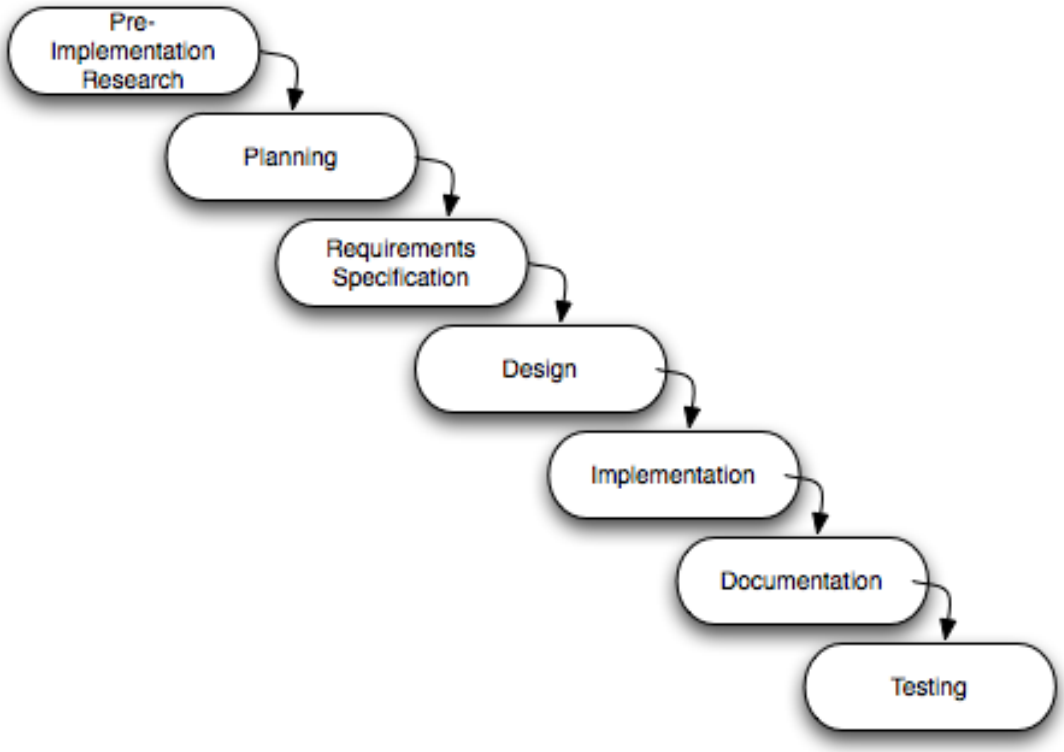

Figure 2.2: The Waterfall Model.

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 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.

Documentation/

2.5 Development Tools and Technologies

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

2.5.1 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.

2.5.2 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.

2.5.3 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.

2.5.4 Databases

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

For the collaborative filtering it was decided on doing further research along the lines of using either MySQL or CouchDB, but this work will be pushed forward in time as the key success factor for this project is the CBR engine. The collaborative filtering part will likely be worked on by spare resources in later project phases, and their background and competences will influence the choice of database platform.

2.5.5 Third Party Libraries

For developing the CBR as well as P3P parsing components of the Privacy Advisor, a decision had to be made regarding the usage of third party libraries, either for components or for the entire CBR system. Two options were considered with respect to the CBR system. The first was to use a full third party CBR system (jColibri). The alternative was to use a third party system for the retrieval component of the CBR system (i.e. a k Nearest Neighbors (kNN) implementation).

Third Party CBR System

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

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

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

Third Party k Nearest Neighbors Implementations

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

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

P3P/XML Parser

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

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

2.6 Standards

To achieve clean and reusable code, the project has adopted Oracle's Coding Conventions for the Java Programming Language[7]. This is mentioned in the requirements specification due to the high likelihood of the customer having to change the source code for later adaptations.

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

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 modelï¿½s 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.

Overview and Risk Management

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.

Table ?? contains a sample risk table depicting how risks are described, quantified and which actions are taken to mitigate the particular risk.

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 3.2: Risk characterization.

3.2.1 Discussion

As shown in Appendix A.3.2, three broad risk categories were identified:

1. **Technical**: These risks pertain mainly to the implementation; that critical success factors are not met by the implemented software.
2. **Communication**: These are risks related to miscommunication, either within the project team or between the customer and the team.
3. **Planning**: The final category has to do with planning and decisions made early on in the project phase.

Among the more critical risks identified were a potential failure to correctly parse policy documents as well as the risk of an improper choice of algorithms. Failure to properly parse P3P policies could impact progress as it would slow down and push the testing phase back

in time. This will also impact the second risk factor above, as an early testing would reveal the weaknesses of algorithms in time.

An important issue to remedy these problems is a modular design, that is that these two pieces of functionality are to be isolated in separate code modules. This implies that if the initial choice of algorithms made by the project team, while important, are not critical, as they can be replaced during later development stages.

Other risk factors identified include database and data storage issues, in particular pertaining to the collaborative filetering system, and disagreements within the project team. A full listing of the risk factors identified, and proposed measures to mitigate these is given in Appendix A.3.2.

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. The project plan was initially laid out based on *evidence based scheduling*, starting out with rough estimates of each particular task pertaining to each of the project phases.

The different components of the system were identified early in the process, and each componentsuch as core CBR and algorithms, networking, GUI and so forth were assigned as the responsibility of a team member. The responsible for each component then provides an estimate over the required time to design, implement and document the component.

These estimates will in turn be based on a breakdown of the required tasks for each system. These numbers are then aggregated to form the total plan.

3.4.1 Project Milestones

In the initial plan, four important milestones were laid down to provide a clear measure of progress. The milestones concern the key project phases related to designing, implementing and documenting the system.

1. **Requirements specification**: Week 38.
2. **Design**: Week 39.
3. **Implementation and documentation**: Week 42.
4. **Testing and evaluation**: Week 44.

A short span between the completion of the requirements specification and the design phases was set as these phases were deemed to be largely overlapping. Many requirements were deemed to be clear from the outset and could therefore be passed on to the design phase without waiting for the formal recognition by the customer. This improves project flow as several key activities occur at the same time.

Similar considerations were made with regard to the implementation and testing milestones. The preparatory steps for the testing work can be conducted in parallell with implementation.

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:

1. Past behavior using case based reasoning (CBR)
2. 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: ï¿½ Section 4.3 gives an overview over the system; its requirements and user characteristics. ï¿½ Section 4.4 presents four different use-case scenarios. ï¿½ 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.

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.

4.4.1 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.

4.4.2 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.

4.4.3 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.

4.4.4 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

1. The system should be able to parse a P3P file to instantiate the data as a privacy case/event/instance.
2. Based on past history (knowledge base), it should retrieve the cases most similar to the one presented.
3. Given the degree of similarity to past cases and the uniformity of action taken in the past, the system can either
   (a) Give the user a recommendation or
   (b) Pass the recommendation decision on to the community/CF system.
4. 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)
5. Update the database with the recommendation.
6. Allow the user to view the explanation for the recommendation.
7. Allow the user to overrule a recommendation.
8. When overruling a recommendation, the user must be allowed to explain why the decision is made, e.g. one time occurrence, permanent rule, etc.
9. Allow the user, if making a new general rule, to backtrack and alter previous cases

4.5.3 Non-Functional Requirements

1. Implementation
   (a) Code is written in Java following Sun Microsystems' conventions[8].
   (b) Third party libraries are to be documented with version numbers and to be included in the installation package.
2. Maintainability:
   (a) Code repositories and version control: github is used as code repository and for version control.
   (b) User docupmentation is to be produced.
   (c) A well documented API is to be designed
   (d) English (US) is to be used as language for naming convention for source code and filenames, and in code comments and documentation.
   (e) The code is to be designed in a modular fashion.
3. Performance:
   (a) 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.
4. Portability:
   (a) The testing/design system should be portable to any system with a JRE.
5. User interface:
   (a) Two UIs are to be implemented: A command line interface (CLI) as well as a GUI is to be designed using Java/swing.
   (b) These interfaces are meant to facilitate testing the model framework.

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

Design

Contents

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.

In implementing Privacy Advisor, a class structure is built around the CBR agent model discussed in the paper by Tøndel and Nyre. Given the broad structure in the CBR agent model, several details need to be fleshed out, including data structures for storing policies, databases, choosing actual algorithms, a user interface, and so forth. This chapter lays out the broad structure of the Privacy Advisor system. Implementation details are discussed in Chapter 5.3.4.

5.1 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

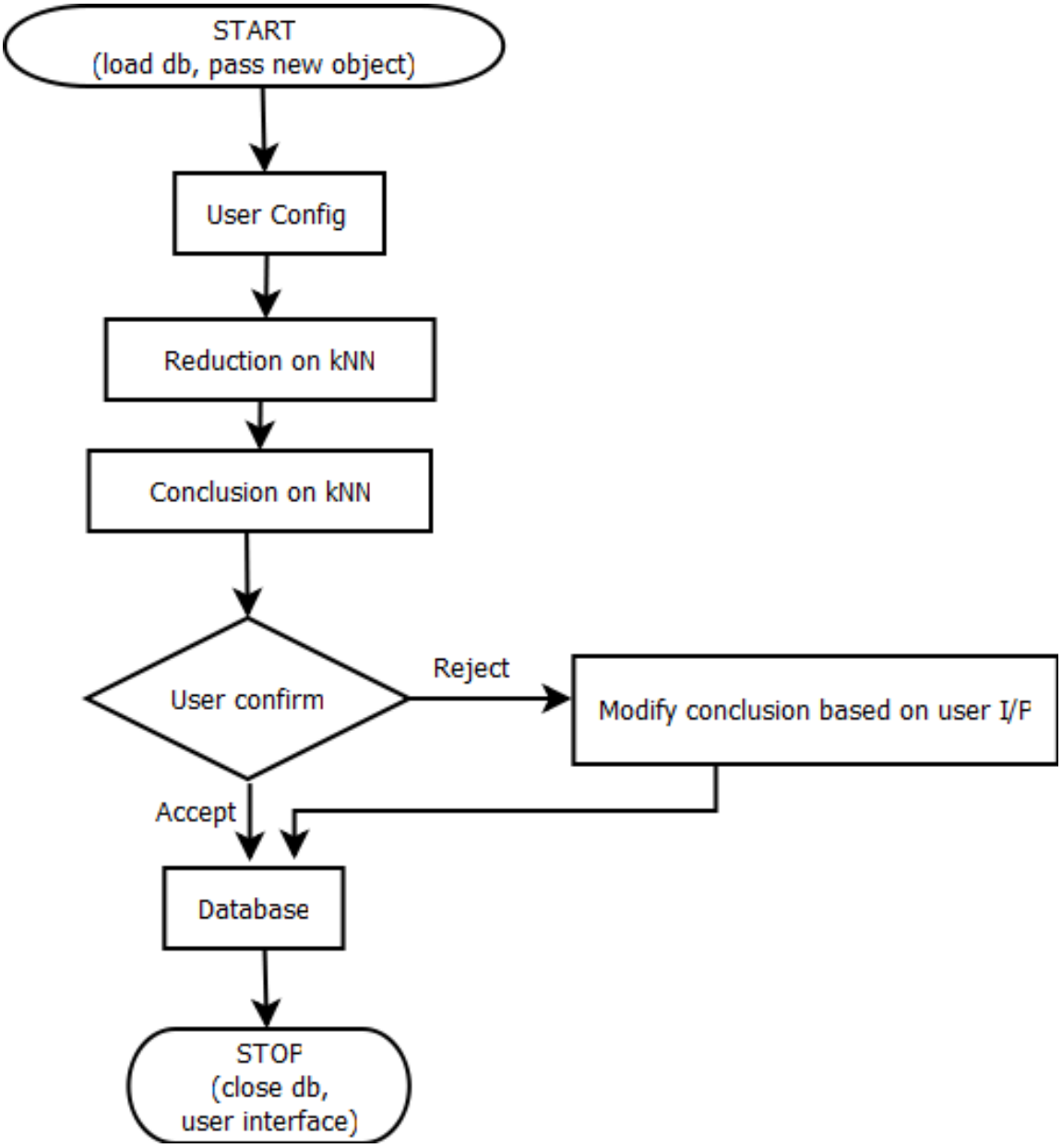

Figure 5.3: Program Flow.

how it interfaces with the local system.

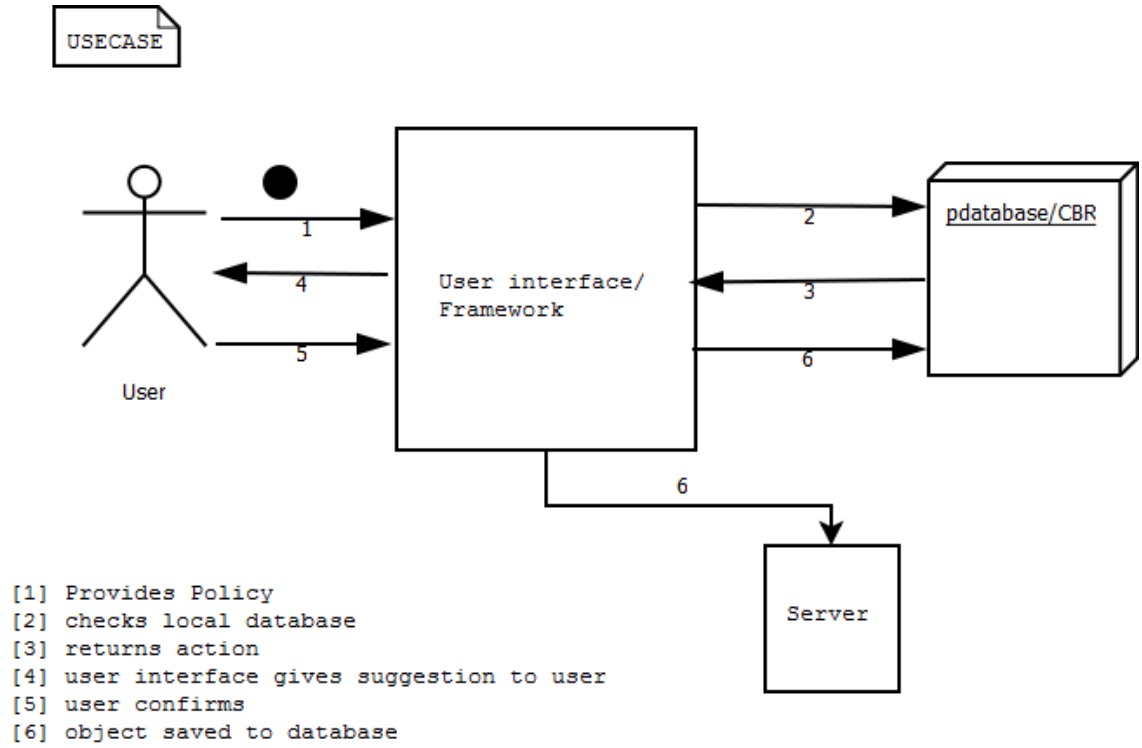

Figure 5.4: System Overview.

5.1.1 Program Flow

Figures 5.3 and 5.4 seek to provide a broad overview of the structure of the Privacy Advisor system. The program flow is given by Figure 5.3: a policy object is passed to the CBR system from the user, either through the CLI or GUI. The CBR does a lookup on similar cases in the local database and computes an initial recommendation based on the similar cases retrieved. If the confidence in this recommendation is above a given threshold, it provides the user with the advice and the background for the advice through the user interface. If the confidence is not sufficient, the system will query the community system for an advice, which then is combined with the initial recommendation for a final advice which is presented to the user. 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.

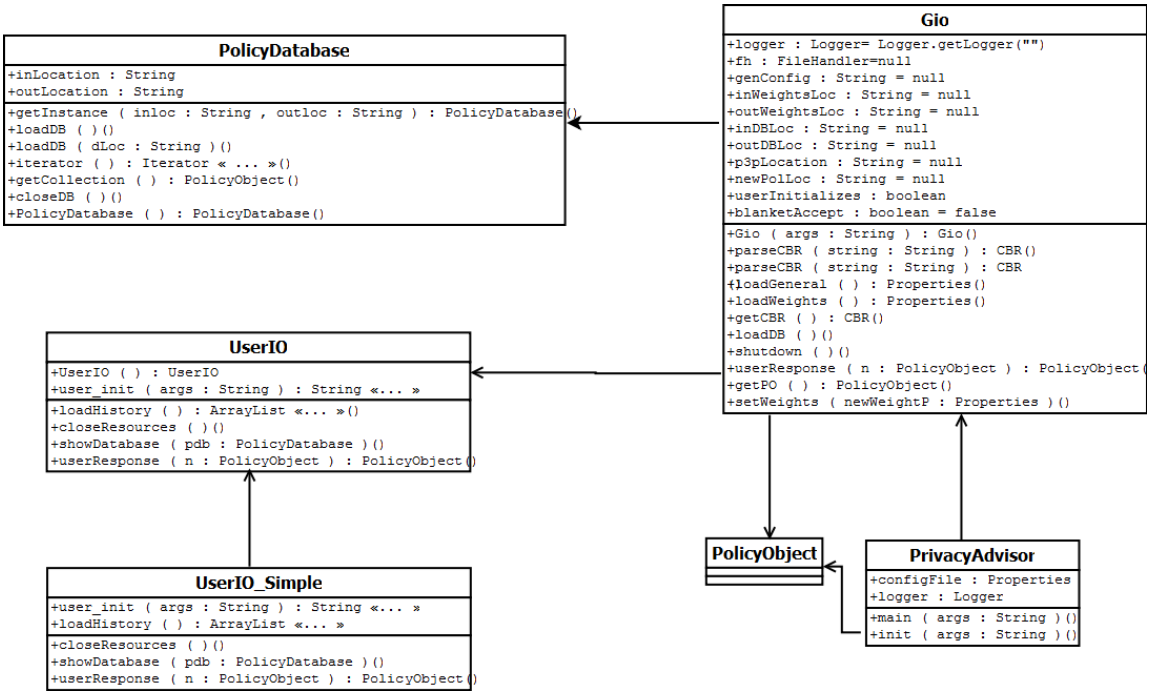

Figure 5.5: Input/output and user interfaces.

5.1.2 Top Level Structure

[Discuss figure 5.6.]

5.2 User Interfaces and Input/Output

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. The GIO class is shown in Figure 5.6.

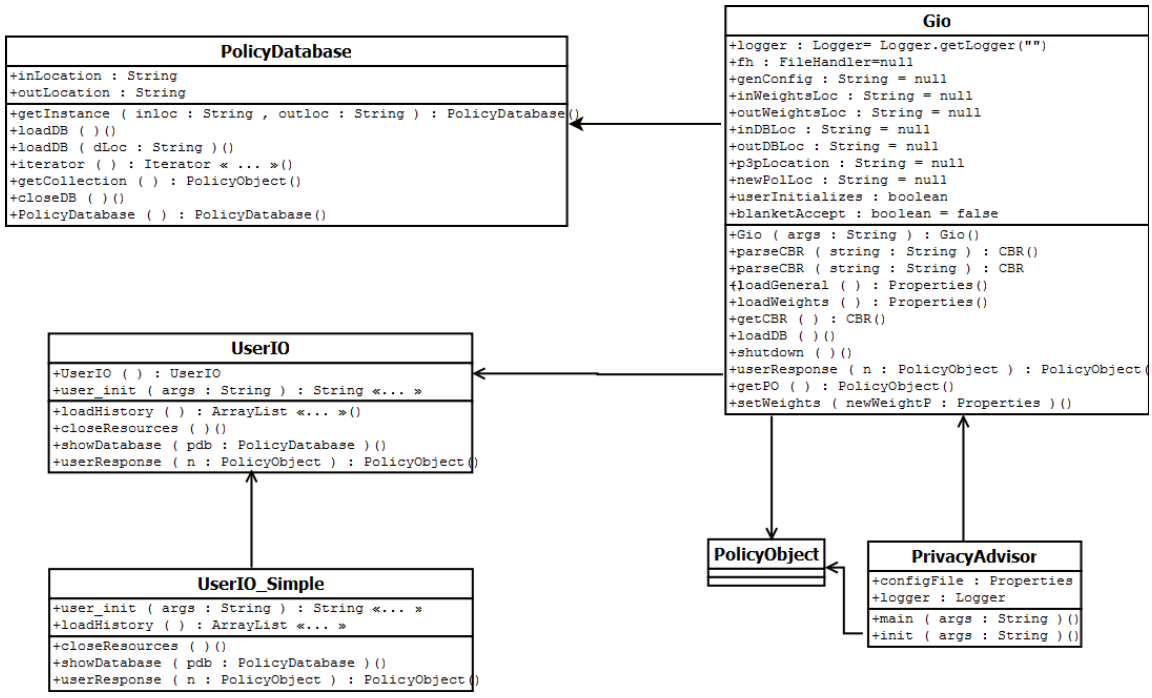

Figure 5.6: working of system.

Graphical User Interface

The Graphical User Interface is used to get a graphical overview over the database, as well as change configurations and run the framework itself. The database is shown in a tree structure so that one can easily click through it and have a look at it. Configurations,

reloading of the database and running the program can all be done from the PrivacyAdvisor drop-down menu.

After the program is run, a view of the new policy can be seen for reference in another view, also in a tree structure.

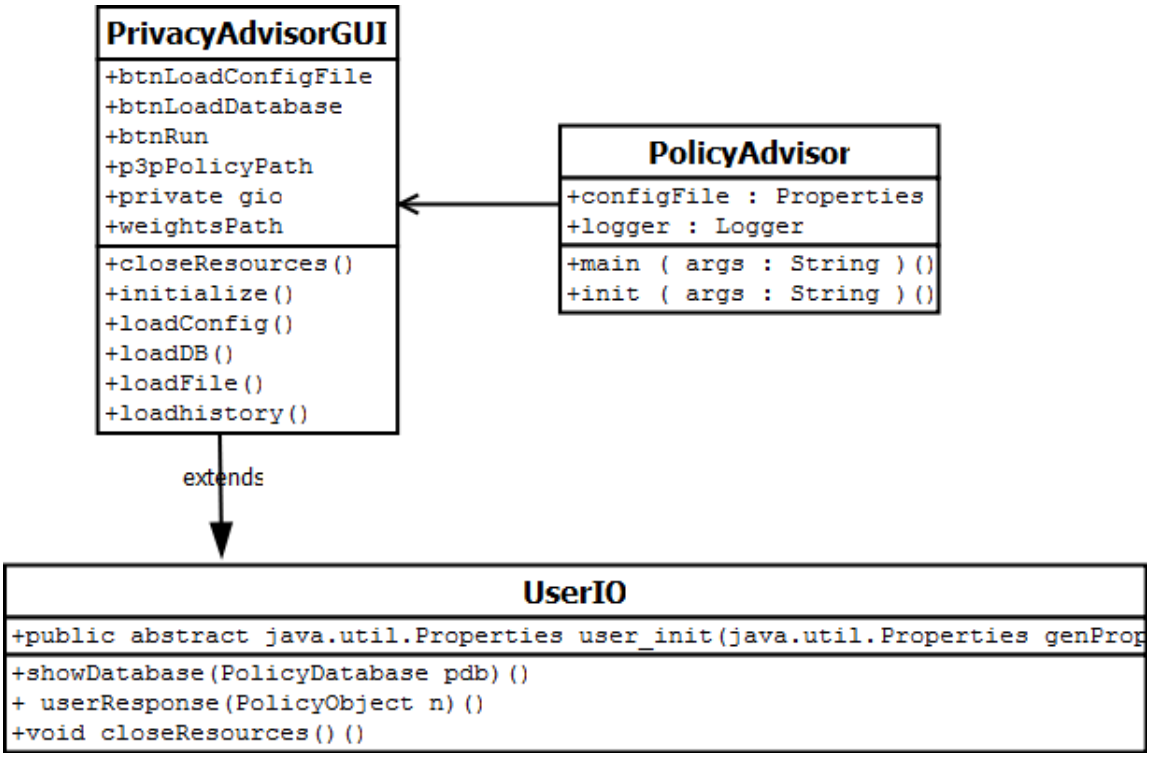

Figure 5.7: GUI interfaces.

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 ??.

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 6.1.2.

An overview of the CBR system is given in Figure 5.8...

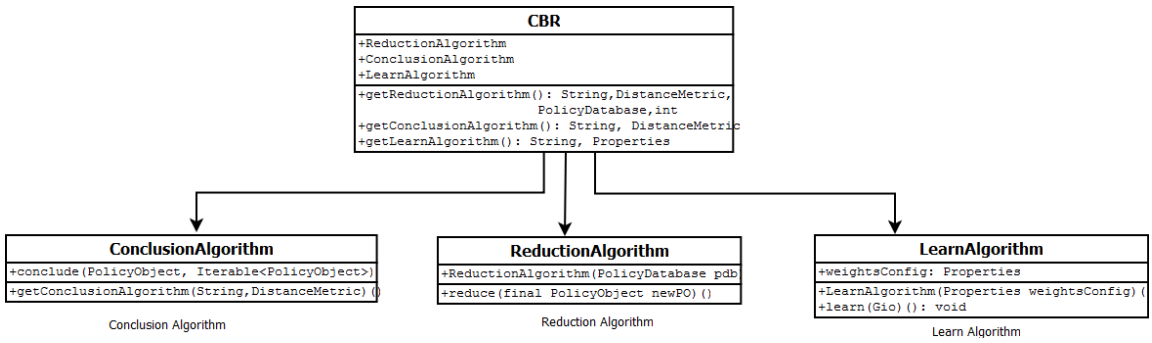

Figure 5.8: CBR System.

5.3 Data Objects and Storage

5.3.1 P3P Policy Objects

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

A P3P Policy Object consists of an `Action`, a `Context` and a list of different `Case` objects. The action object consists of the result from the comparison algorithm, stating if the policy is accepted as a good match, the reasons for this statement and with how much confidence this statement is correct.

The context objects holds most of the objects context, that is what domain the policy belongs to as well as when it was created, last accessed and when it will expire. The list of cases contains one case for each datatype within the policy. A datatype is what kind of information is collected, for example name or date of birth. Each case contains what the purpose for this information is, who are the recipients and the retention for this information. Each datatype has its own case as it simplified the comparison algorithm.

The last thing a policy object contains is a hashmap of the entity data. This is the data that is included at the beginning of every policy document and contains information about the company in question.

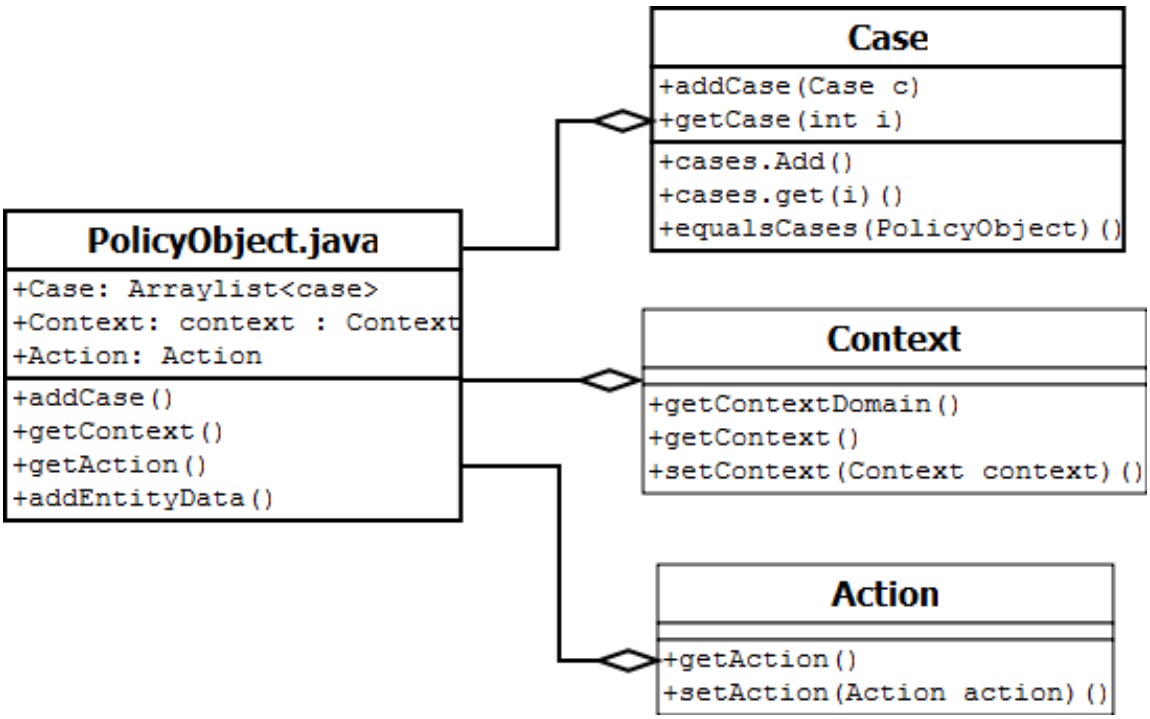

Figure 5.9: Policy Object.

5.3.2 P3P Policy Database

An overview of the Policy Database is given in Figure 5.10...

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 parameterless '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, 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 5.10...

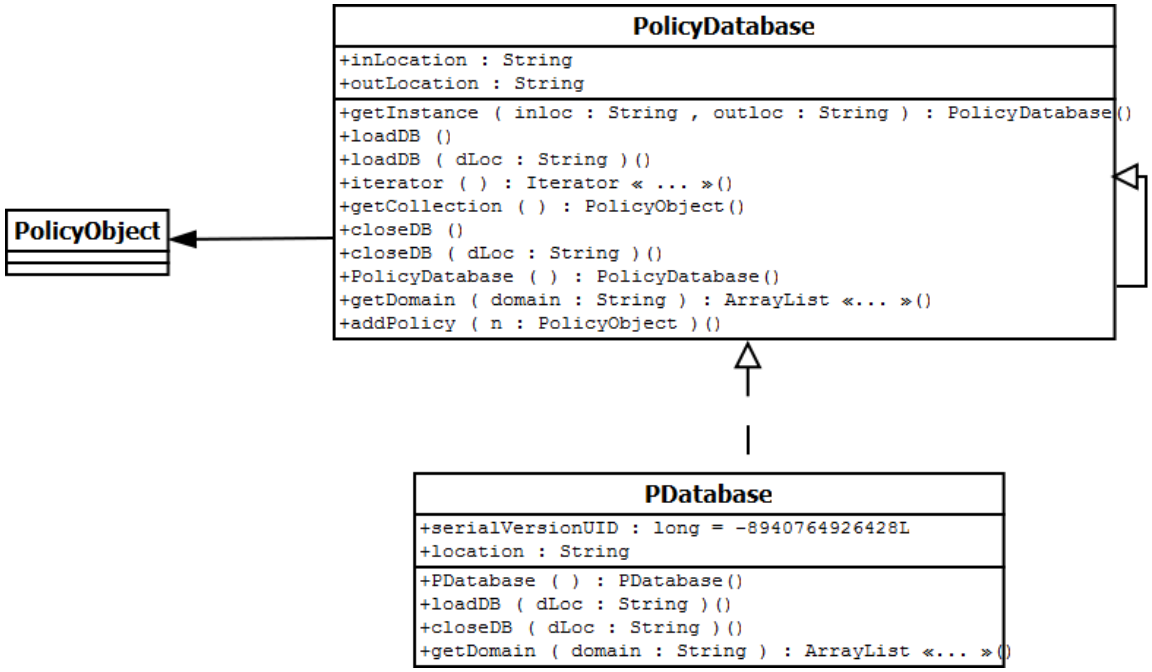

Figure 5.10: Policy Database.

5.3.3 Interfaces

5.3.4 Community Server

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

In the first instance, the new policy under consideration is converted to JSON using GSON

(the Google Json libraries), and transmitted to the database, which parses the new policy and replies with a JSON encoded suggested Action.

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

Implementation

Contents

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 6.1.4.

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.

6.1.2 Learning algorithm

A learning algorithm is used to update the weights that are used to calculate the distance between policies. These weights are updated when the user overrides a reccomandation from the system. This way the system learns from the users actions, and the next time the system runs it would return with an answer that reflects the users actions more precisely. As an example, let's say that the P3P policy of website X have a field that says that they have the rights to give away the contact-information of their users. So the system reccomends the user to not accept this policy, but the user overrides the recommendation and accepts the policy. Then the learning algorithm is run, and the field(s) that are connected to contact-information are updated.

Implemented learning algorithm

The current learning algorithm implementation considers each possible weight field for every policy in the database and checks wether a case in the database contains that particular field. If a policy does indeed contain that field a counter is incremeted. Then it returns the proportion of policies containing the particular field. In figure 6.11 we have a simplified pseudocode of the learning algorithm. The key assumption behind this algorithm is that the most important fields are present in the majority of policies.

```
countYes=0
countNo=0
containsWeight = false
For weights in database
  For policies in database
    For cases in policy
      If (case contain weight)
        containsWeight=true
    If (containsWeight=true And policy is accepted)
      countYes++
    Else
      countNo++
newWeight=countYes/(counYes+countNo)
```

Figure 6.11: Pseudocode for the implemented learning algorithm.

Potential Improvements

Two important critiques can be raised against the current learning algorithm:

1. It does not update weights incrementally. This means that for small databases the weights may be subject to "jumps".
2. A large presence of a field may not indicate that it is more important.

6.1.3 Confidence Measure

The term confidence refers to the notion of certainty about a decision or a hypothesis. For Privacy Advisor it is important to attach a confidence number to a recommendation to give some idea of how certain the reasoning engine is in that it is the correct decision. Therefore, every time Privacy Advisor computes an advice, it also calculates the confidence in that advice. Confidence can be used to guide the learning of the program as well as its decision taking. The confidence measure is also used to determine which steps to take after the initial retrieval from the local database; given a sufficient confidence in the result, it will present an advice to the user. Otherwise it will query the community database.

The confidence measure implemented in Privacy Advisor is closely linked to the retrieval algorithm, that is, to the reduction kNN algorithm which picks out the k most similar cases to the one at hand and the similarity metric used to compare policies. Since the decision considered here is binary - the policy is either rejected or accepted - the k most similar cases can be split in two. If the conclusion is to accept the policy, one possible measure

of confidence could be the fraction of the k nearest neighbors in which the policy has been accepted. While there is a clear rationale for this approach it is flawed in the sense that it does not take into account that some of the k most similar policies may be *much* nearer than the others. To account for this, the confidence measure weights each of the neighbors by the inverse of its distance from the policy that is to be classified

6.1.4 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.

Existing 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.

6.1.5 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 $(A_i \wedge B_i)$.

A bitmap said to be *weighted* is when each of the values is multiplied by a 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 bitmap will be $[1 * w_a, 1*_b, 1*_c, 0 * w_d] = [1, 2, 4, 0]$. For the 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.

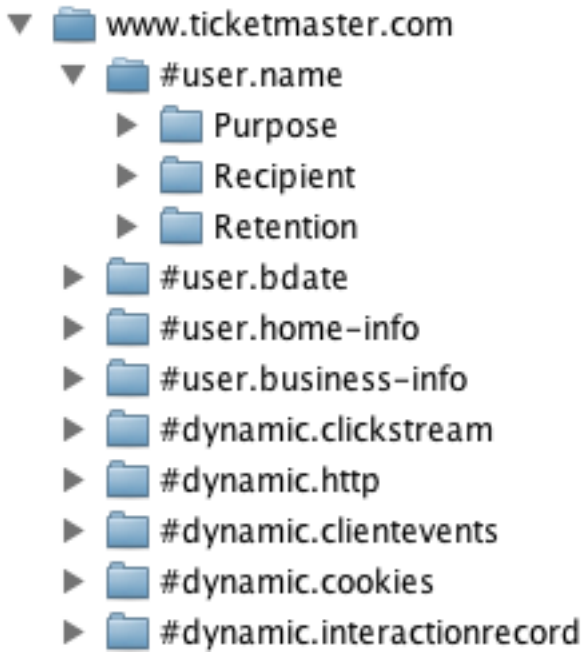

Figure 6.12: Sample P3P Policy as shown in Privacy Advisor.

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.

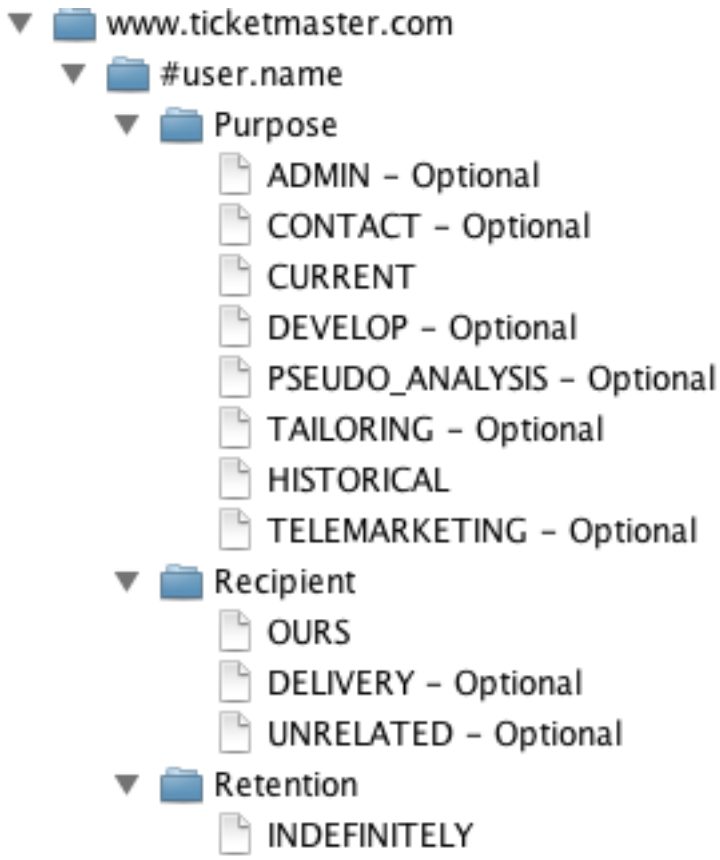

Figure 6.13: Indepth look at TicketMaster's policy regarding the user's name.

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. One of

the solutions are the 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 P3P Parsing

The `P3PParser` is used to parse a P3P policy from an XML document to a Java object that can be used by the rest of the framework. The parser is located in the `parser` group and takes in the path to the xml document as argument. It is based on `org.xml.sax`, an internal, sequential XML parser in Java.

What it does is sequentially go trough the document, and look at each tag, one by one, dealing only with the tags and attributes we are interested in. It collects information as it parses, and construct a `PolicyObject` and returns it. The parser also handles custom tags that can be added manually to P3P documents to pre-define action and context for the purpose of bulding a database.

6.3 Data Objects and Storage

For the Privacy Advisor CBR engine there are two central defining data storage; the first is the representation of a P3P policy, being the cases on which the CBR engine operates. The second is the knowledge base that holds the cases along with the recommended action. In the collaborative filtering augmented CBR model, two different knowledge bases are required, one local representing the knowledge about the user's previous actions, and one global with knowledge of all users.

6.3.1 P3P Policies

A P3P privacy policy is represented by the `PolicyObject` class. This class is detailed in Section 5.3.1. A sample view of a `PolicyObject` is shown in Figure 6.1.5.

6.3.2 Databases

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 several distinct bodies of code - the P3P parsing class, discussed in the Section 6.2, the internal Java representation of policies and the database classes holding the databases.

The extended CBR with collaborative filtering needs to support *two* databases - one local and one global global multi-user database that is accessed across a network whenever the local database proves incapable of providing a satisfactory solution to the problem.

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`).

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`).

6.4 User Interface

6.4.1 CLI

6.4.2 GUI

The GUI inherits from the `UserIO` class and implements a graphical user interface with the use of Java's own graphical library `SWING`. The main view is a split view of the database and the new policy that is being reviewed. Both shown via a tree view for easy overview.

Configure, reload database and run are the three options available to choose from the drop-down menu in the main view. Each run their own function to either start up the configure

editor, reload the database or run the framework. An exit option is also available.

The configuration editor is an abstract class which builds the appropriate textfields and checkboxes for editing, and then sends them back to `GIO` when done. This class is inherited in a subclass inside the `PrivacyAdviserGUI` class and called upon when needed.

Part III

Testing, Evaluation and Summary

Test plan

Contents

Test plan

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.

7.1 Test Methods

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

7.1.1 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.

7.1.2 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.

7.2 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.

7.3 Test case overview

This 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

This 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.

7.4 Test cases

See the appendix for the test cases.

7.5 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.

7.6 Test schedule

This is the table for when the UNIT tests are scheduled to start.

Test identifier	Execution date
UNIT-01	October 22nd
UNIT-02	October 24th
UNIT-03	October 26th
UNIT-04	October 29th
UNIT-05	October 29th
UNIT-06	October 29th
UNIT-07	November 12th

7.7 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. Sections 8.2 and 8.3 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 8.4 evaluates the course as a whole.

8.1 Software Evaluation

This section contains the project team's evaluation of the software system that has been produced.

8.1.1 Key Objectives

As discussed in Section 1.2, the main objectives of the project were:

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.

Objective 1 were considered the clearly most important for the project, being a prerequisite for the remaining objectives. The fuctional requirements section (Section 4.5.2) in requirements specification goes further in detail on what the first objective entails.

8.1.2 Evaluation

The project team realized quickly that there were insufficent resources to meet Objectives 3-4, so focus was shifted to the first objective while, there was planned for working towards the second objective at the end of the project period. With respect to the first objective, the *current Privacy Advisor system provides all the features listed in the requirements specification.* The second objective is partially realized. The core Privacy Advisor system has functionality extending the CBR for networking, and work on a collaborative filtering system has been started on a server machine at NTNU. The features supported by this system is however limited.

8.2 Tools

Referring to choices discussed in section 2.5, the key software tools and languages used are Java, Git, and Google Docs.

8.2.1 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.

8.2.2 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

There were no significant issues with Google Docs. It provided a good framework for sharing binary files such as spreadsheets, presentations and so forth which are not catered well for by most version control systems.

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.

8.3 Resource Use

Figures 8.16 and 8.15 illustrate the discrepancies between actual and planned time use on all activities, both cumulative over time and over each phase of the project. It is updated as by the end of the last week of the project.

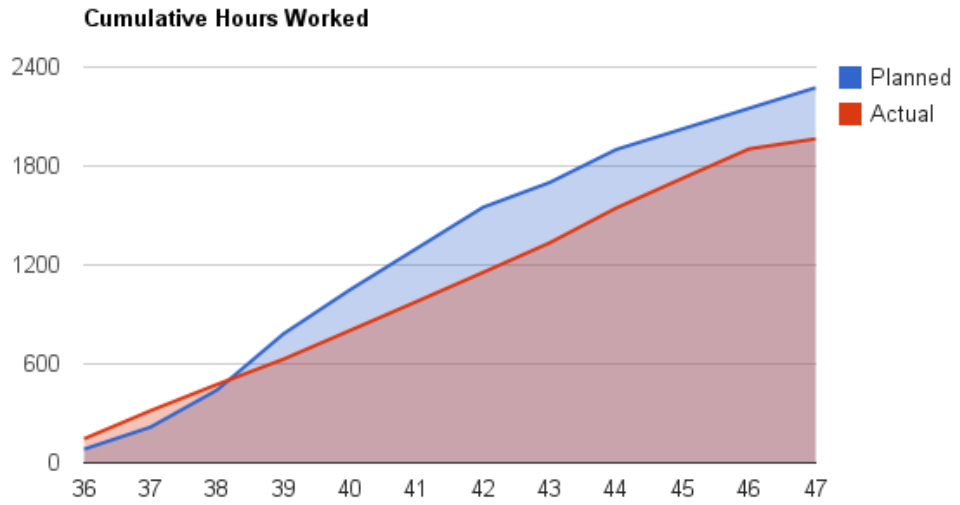

Figure 8.15: Actual vs. planned time. Cumulative over project span.

As shown in Figures 8.16 and refperActivityPct, there are a few important discrepancies between the project plan and how the project indeed turned out. Firstly, there was a shortage of approximately 300 hours. This was quite evenly spread among most team members, with the exception of the team members with administrative responsibilities who had a significantly higher workload. The uneven workload also reflects an uneven

background in academic writing which meant a large portion of the workload with this report was taken on by a few select team members.

Secondly, while the percentage workload between activities was quite well anticipated, there were some noticeable exceptions. One key pattern to notice is that while the design and implementation related activities required less time than initially planned, administrative work and planning required more work than planned. The overrun is most obvious for the administrative tasks part. One important lesson to take away from this is that a project team of a given size does need a major amount of administrative work. Another important reason for this overrun was initial problems with setting up the code repositories.

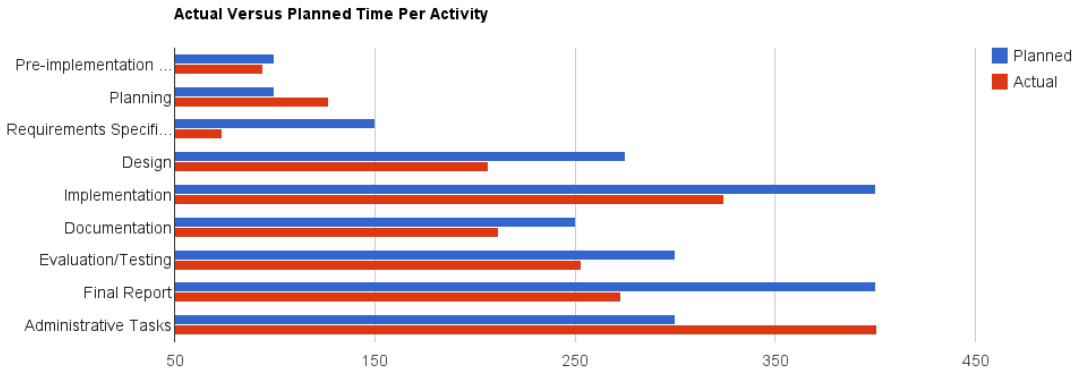

Figure 8.16: Actual vs. planned time. By project phase, in hours.

With respect to the relatively fewer hours spent on designing and implementing the system, a portion of this can be explained by the fact that a major portion of the design was clear from the beginning given that the system was to be based on the CBR framework. This and the fact that Java was decided on early on as programming language, severely limited the possible designs available. The implementation phase also required fewer hours than expected. This was in part due to the fact that the team did not realize all the objectives that were set (networking is only partially complete), and that work was divided well among team members. Interfaces were also built quite early on in the project so that adding on modules to existing code proved simple.

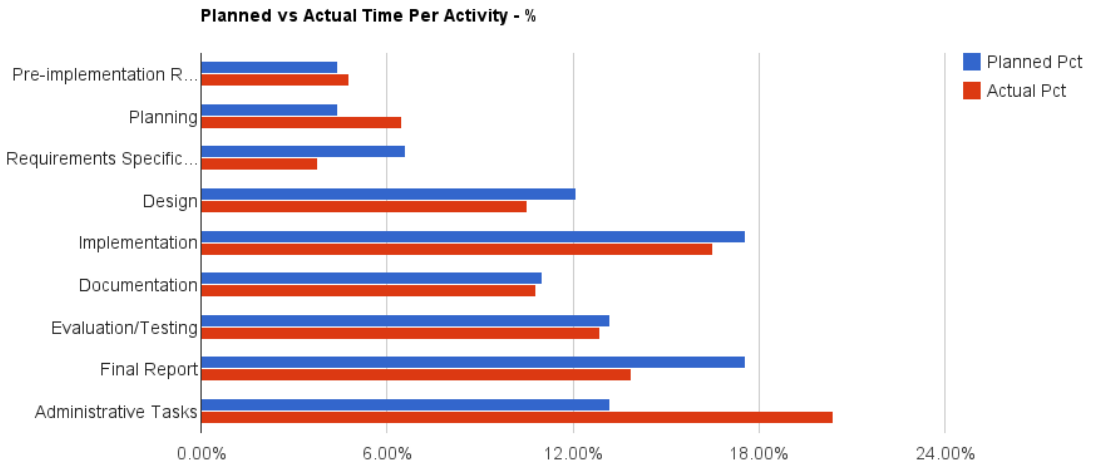

Figure 8.17: Actual vs. planned time. By project phase, in percentage of total time.

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

Conclusion

Contents

9.1 Summary and Final Remarks

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 remain work in progress, which is reflected by the system's user interface which is 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.

9.2 Future Development

To conclude this report, we look at the some important challenges that must be addressed in order to further the Privacy Advisor system to a end-user product.

9.2.1 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. This testing is a more involved type of testing than the standard unit testing that the Privacy Advisor system has been through at present.

For the system to actually be successful it has to provide accurate predictions of users' Internet decisions so that it can provide good privacy protection without being invasive in the users' day-to-day Internet activities. This type of testing will require putting together a group of test users and monitoring their activities over a *prolonged period of time*[9].

The project team feels that most important and time consuming part of future development lies in this area. Designing appropriate measures and testing routines for evaluating the performance of the CBR agent is probably the main obstacle on the way towards realizing the design.

9.2.2 Collaborative Filtering

One of the key improvements to the current Privacy Advisor system is the collaborative filtering/community portion of the system. While the interface for a networking system is well integrated in the CBR system, the current network resource has limited features. Furthermore, this component of the system will also require a prolonged period of testing, as a substantial amount of data.

9.2.3 Distance Metrics and Algorithms

9.2.4 The Current State of Internet Privacy Policies

Finally, some words about the current state of Internet privacy policies. While P3P represents an effort towards a machine readable standard for privacy policies, it is far from being adopted universally. At present several standards seem to coexist, so a finalized system would have to be able to handle several different standards.

Also posing significant problems to the effectiveness of privacy enhancement software is fact that there are little in the way of legal regulations of privacy protection online[10]. At present, businesses are free to arbitrarily change their privacy policies without noticing users, and very little effort has been made towards verification or enforcement of policies. This means

[9]The time aspect is stressed here as the system will need time to actually learn the user's preferences.

[10]See for instance: http://www.nytimes.com/2011/11/20/opinion/sunday/a-push-for-online-privacy.html?$_r = 1ref = opinion$.

that currently, the privacy policies cannot be the only input to a *truly* trustworthy privacy enhancement software.

Part IV

Appendices

Documentation

Contents

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

A.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[11].

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.

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

A.2 Installation

A.2.1 Compilation in Eclipse

The simplest way to make the program run as a stand-alone program on a Java Virtual Machine is to compile the source code to a runnable jar file. This section covers how this is done in the Eclipse environment.

Open Export Wizard

This can be done by right clicking the project in Eclipse, choosing `Export`, type in `jar` in the new window that opens up and choose **Runnable Jar File**, before clicking **Next**. These steps are shown in the two figures A.18 and A.19.

Select GUI or CLI

The next step is to select which class file is to be the main class[12]; in the **Launch configuration** dropdown select either `PrivacyAdvisor` or `PrivacyAdvisorGUI`.

`PrivacyAdvisor` is selected to compile the command line version, and the `PrivacyAdvisorGUI` is selected for command line. Then choose the export destination, and choose `Package requires libraries into generated JAR` as the library handling. Pressing finish to start exporting the program. These steps are shown in figure A.20.

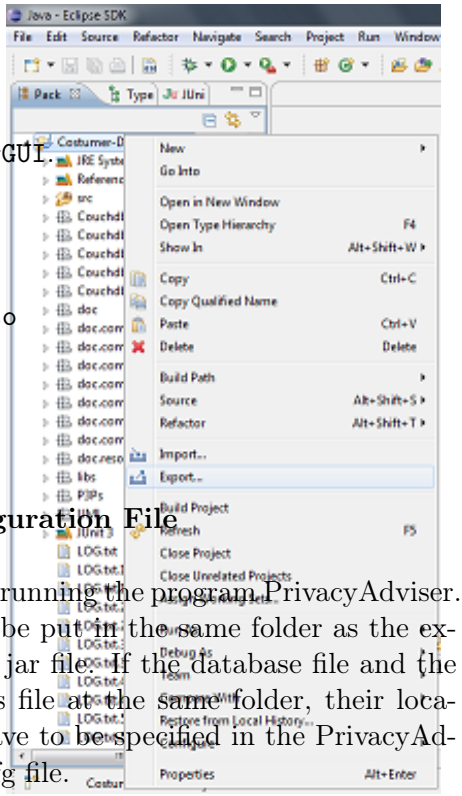

Figure A.18: First step: right click and choose "export".

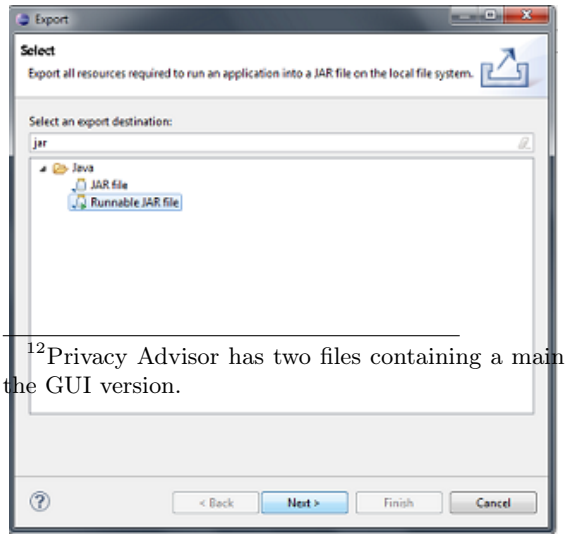

Figure A.19: Second step: choose runnable jar.

Configuration File

Before running the program PrivacyAdviser.cfg has to be put in the same folder as the exported jar file. If the database file and the weights file at the same folder, their location have to be specified in the PrivacyAdviser.cfg file.

[12] Privacy Advisor has two files containing a main method. One is for running the CLI version, one for the GUI version.

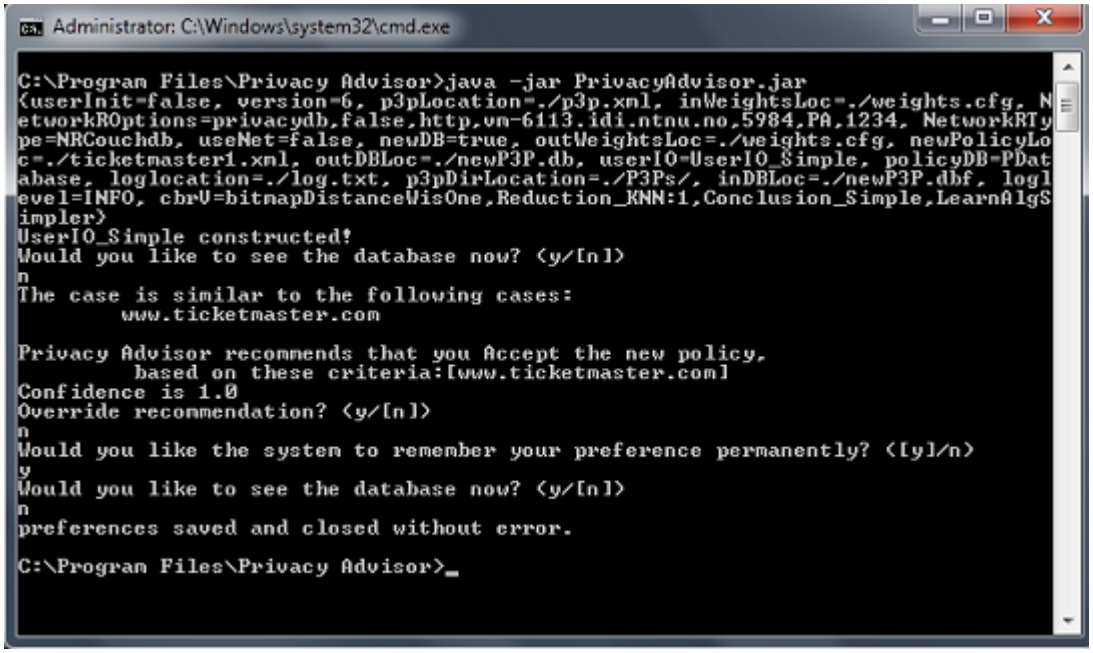

Figure A.21: Running the program.

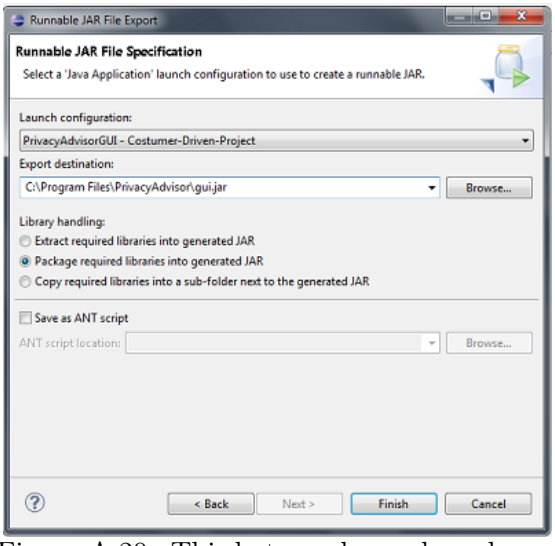

Figure A.20: Third step: choose launch config, destination and library handling.

Sample Run

To run the program, navigate to the folder where the jar file is stored and type the following command:

```
java -jar filename.jar
```

Now the program should start (even if it's the cli or gui version). This is shown in figure A.21.

A.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 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).

A.3 User Interfaces

The Privacy Adviser consists, as mentioned in section A.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 command line interface better suited for repetitive tasks, in particular for batch type testing. The command line interface operates according to the options given when starting the program, either as command line parameters or through the configuration file. This is explained in detail in section A.3.1.

A.3.1 Command Line Interface

By running the program from the command line options can be set directly as command line parameters when starting the program. If no parameters are given, the options set in the PrivacyAdviser.cfg will be used. Options added as command line parameters have priority over configuration file options.

As an example consider 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. If the user wants to store the changes to a new database file, the `outDBLoc` parameter needs to be set to a new file.

There are two ways to change this. The first is to set the `outDBLoc` parameter at the command line. This will override the config file for this particular run.

```
java -jar privacyAdvisor.jar -outDBLoc newDatabase.db
```

More options can be added in this manner, and for the options that are not specified, the default options in the config file will be used. For a complete list of the options see table ??.

The second approach is to change the outDBLoc in the configuration file directly.

A.3.2 Graphical User Interface

When using the graphical user interface, the left hand pane gives an overview of the all the policies and cases in the database as a tree-structure. The right hand pane gives a similar view of how the new policy. This can be seen in figure A.23. The GUI is started from the command line the same way the CLI is started.

Obtaining Advice for a Policy

When the GUI is stared, it automatically loads the database file, and displays the tree-structure. Pressing the `Run` button in the `Privacy Advisor` menu runs the CBR classifier and displays a message box containing an advice along with a measure of confidence and references to the sites on which the decision is based.

If we want to change the location of the database, the new policy or anything else we can press the `Configuration` button and

make the changes in the `Configuration Editor` window that shows up. This Configuration Editor window is showed in figure??.

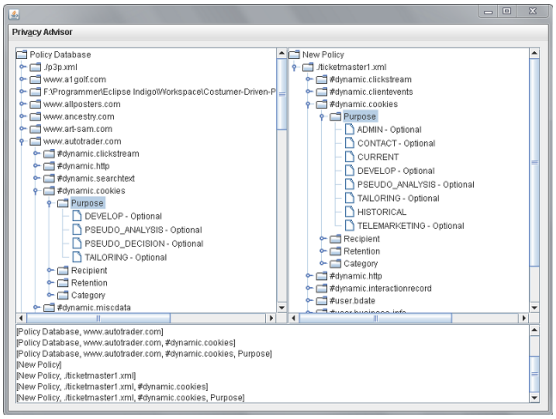

Figure A.22: The GUI.

Configuration Files

Table ?? gives an overview over the configuration file parameters.

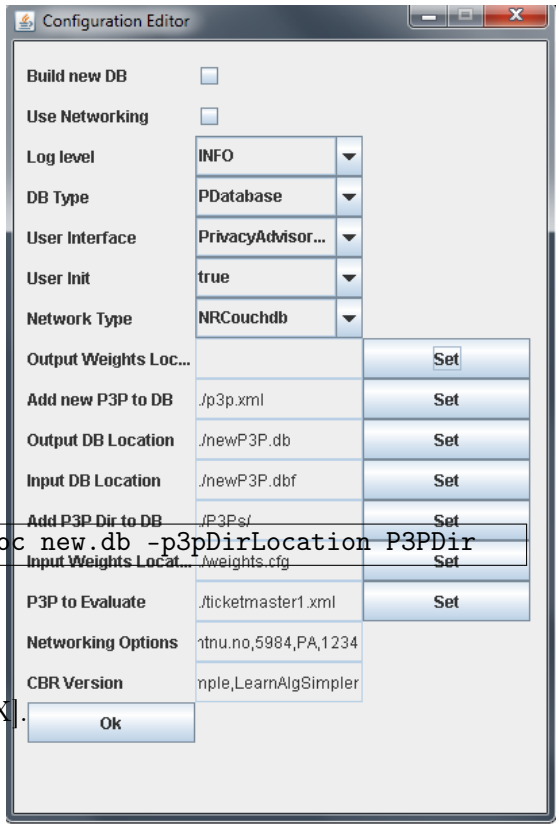

Figure A.23: The GUI.

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].

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 A.3: Configuration file parameters.

Loading and Viewing a Database

CLI: To view a database `p3pDB.db` privacy advisor can be called in the following fash-

ion:

```
PrivacyAdvisor -newDB false -outDBLoc new.db -inDBLoc p3pDB.db
```

Privacy Advisor will then display the following:

```
UserIO_Simple constructed!  Would you like to see the database now?  (y[n])
```

To view the database, confirm by pressing `y`.

GUI: To view a database from the GUI, open the configuration editor and uncheck the **Build New Database** item and navigate to the directory holding the new database. Select **Load Database** from the menu. The database can now be browsed in the left hand tree structure.

Risk Review

Contents

This chapter contains a listing of the a few risk factors identified during the project planning phase. The risks are categorized in three broader categories. Technical risk factors are problems occurring in the implementation of the software, i.e. important parts of the problem that are not functioning as indended. Communication issues pertain to problems communicating either within the project team or between the team and the customer. The last type of risks are those that have to do with planning and decisions made early on in the project.

The risk factors are quantifed as *probability* $\times$ *consequence* on a scale from 1(lowest) to 25 (highest).

A.4 Technical Issues

A.5 Communication Issues

A.6 Planning and Other Risks

Risk item	1
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. Focus on modular design.
Deadline	Implementation deadline.
Responsible	Einar Afiouni.

Table A.4: Problems with policy retrieving

Risk item	2
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 A.5: Problems with external storage

Risk item	3
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 A.6: Remote server problems

Risk item	4
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 A.7: 3rd party library problems.

Risk item	5
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 A.8: Misunderstandings between customer and project team.

Risk item	6
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.

Table A.9: Disagreements within the project team.

Risk item	7
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 A.10: Changes in requirments.

Risk item	8
Activity	Design, implementation.
Risk Factor	The implemented algorithms may not work as intended or not well suited for this project.
Probability	3
Consequence	5
Risk	15
Action taken	Research on similar algorithms and projects. Focus on modularity.
Deadline	N/A
Responsible	Dimitry Kongevold.

Table A.11: Poorly chosen algorithms

Risk item	9
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 A.12: Risk factor: Sickness.

Test cases

Table A.13: UNIT-01

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 the program runs without input
Pre-conditions	Code for input handling for all possible commands. Code for error handling for invalid inputs.
Execution steps	1. Run the program for every type of argument 2. Run the program without arguments
Expected results	1. The specified variable is set to the specified value 2. All the values are loaded from the config file

Table A.14: UNIT-02

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. Run a P3P xml in the P3P parser and print the parsed fields and their values to console 2. Manually compare the printed fields and values with the contents of the P3P xml
Expected results	1. The P3P xml is parsed successfully. It's content is printed to console 2. The printed output have the same fields, each having the same value as those in the xml

Table A.15: UNIT-03

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 and print policy A from the local database 4. Read and print policy A 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 results	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 and printed 4. Policy B is successfully read from the database file and printed 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

Table A.16: UNIT-04

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. Run the program using the GUI 2. Test every option in the menu bar 3. Test every button in the configuration menu 4. Test every scroll bar 5. Resize the window
Expected results	1. The program starts and loads the graphical user interface 2. The code connected to the option is executed 3. Every button is running the connected code 4. Every scroll bar scrolls through the list, up and down, from end to end, successfully 5. Window can be resized without having elements of the GUI overlapping. The elements and panes scales with the main window

Table A.17: UNIT-05

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 database file 2. Manually calculate and write down the distances between the single policy and each of the policies in the history 3. Run the distance algorithm on a single policy and the history and compare the distances that are returned by the algorithm with the manually calculated distances from step 2 4. Manually find the k policies with the lowest distances 5. Run the reduction algorithm with necessary input to find the k nearest policies and compare the k policies returned by the reduction algorithm with those found in step 4 6. Run the conclusion algorithm and verify the results returned by the algorithm

Continued on next page

Table A.17 – *Continued from previous page*

Item	**Description**
Expected results	1. The policies are added to the database file 2. The six distances are obtained 3. The algorithm returns the same distances as those found in step 2 4. The k policies are obtained 5. The algorithm returns the same k policies as those found in step 4 6. The algorithm is giving the correct recommendation and confidence value

Table A.18: UNIT-06

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. Make a set of policies and load the set into the history 2. Read the weights from the weights file 3. Run the classification and learning algorithms on the policy to be classified and the history, with the weights from step 2 4. Read the weights from the weights file 5. Compare the contents of the weights obtained in steps 2 and 4
Expected results	1. The policies are loaded into the history successfully 2. The weights are written down 3. The classification and learning algorithm runs successfully on the policy to be classified and the history 4. The weights are loaded 5. The weights loaded in step 4 are different from the weights written down in step 2

Table A.19: UNIT-07

Item	**Description**
Name	Interaction with community databas
Test identifier	UNIT-07
Person responsible	Henrik Knutsen
Feature(s) to be tested	That policies can be sent to and saved on the community database. That the community database returns a recommendation
Pre-conditions	A running local client and a server. Code for sending policies and must be implemented locally, and code for calculating and returning a recommendation must be implemented on the server.
Execution steps	1. Upload a policy to the community database 2. Check that the database returns a recommendation
Expected results	1. The policy to be classified is saved on the community database 2. The community database returns a recommendation based on the policy to be classified and the database history

Test execution

Table A.20: UNIT-01

Item	**Description**
Name	Command line interface (CLI) functionality
Test identifier	UNIT-01
Person responsible	Henrik Knutsen
Date of first execution Date of completion	October 24th November 16th
Execution steps	1. Run the program for every type of argument 2. Run the program without arguments

Continued on next page

Table A.20 – *Continued from previous page*

Item	Description
Steps executed	1. (a) Run with -logloc logTest.txt (b) Run with -loglevel ALL (c) Run with -inDBLoc database.db (d) Run with -outDBLoc database.db (e) Run with -inWeightsLoc testWeights.cfg (f) Run with -outWeightsLoc testWeights.cfg (g) Run with -newDB false (h) Run with -p3pLocation ticketmaster1.xml (i) Run with -p3pDirLocation P3P (j) Run with -blanketAccept true (k) Run with -newPolicyLoc test.xml (l) Run with -userInit true (m) Run with -userResponse accept,google,0.5,false (n) Run with -cbrV bitmapDistanceWisOne, Reduction_KNN ,Conclusion_Simple, LearnAlgSimpler (o) Run with -userIO UserIO_Simple (p) Run with -policyDB PDatabase (q) Run with -genConfig test.cfg (r) Run with -NetworkRType NRCouchdb (s) Run with -NetworkROptions privacydb, false, http, vm-6113.idi.ntnu.no, 5948, PA, 1234 (t) Run with -confidenceLevel 1.5 (u) Run with -useNet true 2. Run the program without arguments

Continued on next page

Table A.20 – *Continued from previous page*

Item	**Description**
Expected results	1. (a) Logfile is created at the specified filepath (b) loglevel is set to the specified value (c) inDBLoc is set to the specified value (d) outDBLoc is set to the specified value (e) inWeightsLoc is set to the specified filepath (f) outWeightsLoc is set to the specified filepath (g) newDB is set to false (h) p3pLocation is set to the specified file (i) p3pDirLocation is set to the specified filepath (j) recommendation is automatically accepted (k) newPolicyLoc is set to the specified filepath (l) userInit is set to the specified value (m) the specified response and context is appended to the policy (n) cbrV is set to the specified classes (o) userIO is set to the specified class (p) policyDB is set to the specified class (q) genConfig is set to the specified file. Values of the specified config file are loaded (r) NetworkRType is set to the specified class (s) NetworkROptions is set to the specified values (t) confidenceLevel is set to the specified value (u) useNet is set to the specified value 2. All the values are loaded from the config file

Continued on next page

Table A.20 – *Continued from previous page*

Item	Description
Step results	1. (a) Logfile is created at the specified filepath (b) loglevel is set to ALL (c) inDBLoc is set to database.db (d) outDBLoc is set to database.db (e) inWeightsLoc is set to testWeights.cfg (f) outWeightsLoc is set to testWeights.cfg (g) newDB is set to false (h) p3pLocation is set to ticketmaster1.xml (i) p3pDirLocation is set to P3P (j) Recommendation is automatically accepted (k) newPolicyLoc is set to test.xml (l) userInit is set to true (m) the specified response and context is appended to the policy (n) cbrV is set to bitmapDistanceWisOne, Reduction_KNN, Conclusion_Simple, LearnAlgSimpler (o) userIO is set to UserIO_Simple (p) policyDB is set to the specified class (q) genConfig is set to test.cfg. Values in test.cfg are loaded (r) NetworkRType is set to NRCouchdb (s) NetworkROptions is set to privacydb, false, http, vm-6113.idi.ntnu.no, 5984, PA, 1234 (t) confidenceLevel is set to 1.5, recommendation is gives from (default) community database (u) useNet is set to true, networking is enabled 2. All the values are loaded from the default config file

Continued on next page

Table A.20 – *Continued from previous page*

Item	**Description**
Test conclusion	1. (a) Success (b) Success (c) Success (d) Success (e) Success (f) Success (g) Success (h) Success (i) Success (j) Success (k) Success (l) Success (m) Success (n) Success (o) Success (p) Success (q) Success (r) Success (s) Success (t) Success (u) Success 2. Success Test passed
Comments	-

Table A.21: UNIT-02

Item	**Description**
Name	P3P parser
Test identifier	UNIT-02
Person responsible	Henrik Knutsen
Date of first execution Date of completion	October 24th November 7th
Execution steps	1. Run a P3P xml in the P3P parser and print the parsed fields and their values to console 2. Manually compare the printed fields and values with the contents of the P3P xml

Continued on next page

Table A.21 – *Continued from previous page*

Item	**Description**
Steps executed	1. Test for barnesandnoble.com (a) barnesandnoble.xml is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 1(a) 2. Test for daduru.com (a) daduru.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 2(a) 3. Test for ssa.gov (a) ssa.gov is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 3(a) 4. Test for toysrus.com (a) toysrus.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 4(a) 5. Test for gunbroker.com (a) gunbroker.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 5(a) 6. Test for latimes.com (a) latimes.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 6(a) 7. Test for planedesire.com (a) planedesire.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 7(a) 8. Test for yahoo.com (a) yahoo.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 8(a) 9. Test for nextel.com (a) nextel.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 9(a) 10. Test for ebay.xml (a) ebay.com is parsed and printed to console (b) Contents of the xml is compared with to what was printed in step 10(a)

Continued on next page

Table A.21 – *Continued from previous page*

Item	**Description**
Expected results	1. The P3P xml is parsed successfully. It's content is printed to console 2. The printed output have the same fields, each having the same value as those in the xml

Continued on next page

Table A.21 – *Continued from previous page*

Item	Description
Step results	1. Results for barnesandnoble.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 2. Results for daduru.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 3. Results for ssa.gov (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 4. Results for toysrus.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 5. Results for gunbroker.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 6. Results for latimes.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 7. Results for planedesire.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 8. Results for yahoo.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 9. Results for nextel.com (a) The P3P xml is parsed successfully. It's content is printed to console (b) The printed output have the same fields, each having the same values as those in the xml 10. Results for ebay.com (a) The P3P xml is parsed successfully. It's content is printed to console

Table A.21 – *Continued from previous page*

Item	**Description**
Test conclusion	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success 7. Success 8. Success 9. Success 10. Success Test passed
Comments	As mentioned in the test plan, it is not guaranteed that the parser is successfully parsing every possible field of every possible policy even though it has passed this test

Table A.22: UNIT-03

Item	**Description**
Name	Local database
Test identifier	UNIT-03
Person responsible	Henrik Knutsen
Date of first execution Date of completion	October 24th October 24th
Execution steps	1. Write policy A to the local database 2. Write policy B to the local database 3. Read and print policy A from the local database 4. Read and print policy A 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
Steps executed	1. Program is started writing policy A to an empty database 2. Accept recommendation and chose to save the new action and policy (policy B) 3. Print the new database after policy B was added (step 2) 4. Done in step 3 5. The contents of policy A that was written in step 1 is compared to what was printed of policy A in step 3 6. The contents of policy B that was written in step 2 is compared to what was printed of policy B in step 3

Continued on next page

Table A.22 – *Continued from previous page*

Item	**Description**
Expected results	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 and printed 4. Policy B is successfully read from the database file and printed 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
Step results	1. Policy A was successfully written to the database file 2. Policy B was successfully written to the database file 3. Database was successfully printed 4. Same as step 3 5. The contents of the loaded policy A and printed contents of policy A are identical 6. The contents of the loaded policy B and printed contents of policy B are identical
Test conclusion	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success Test passed
Comments	-

Table A.23: UNIT-04

Item	**Description**
Name	Graphical user interface (GUI) functionality
Test identifier	UNIT-04
Person responsible	Henrik Knutsen
Date of execution	October 29th
Date of completion	November 17th
Execution steps	1. Run the program using the GUI 2. Test every option in the menu bar 3. Test every button in the configuration menu 4. Test every scroll bar 5. Resize the window

Continued on next page

Table A.23 – *Continued from previous page*

Item	Description
Steps executed	1. Program started with graphical user interface 2. Chose every option in the menu bar (a) Clicked "Configuration" (b) Clicked "Reload Database" (c) Clicked "Run" (d) Clicked "Run", and then "Run" again after the algorithm has classified and stored the policy (e) Clicked "Exit" 3. Clicked every button in the configuration menu (a) Checked all the checkboxes individually (b) Opened the all the dropdown menus and selected an option (c) Chose a file for all the options with the "set" button (d) Clicked the "x" to close the configuration editor (e) Clicked the OK button 4. Used every scroll bar (a) Used scroll bar for scrolling up/down in database pane (b) Used scroll bar for scrolling left/right in database pane (c) Used scroll bar for scrolling up/down in new policy pane (d) Used scroll bar for scrolling left/right in new policy pane (e) Used scroll bar for scrolling up/down in output pane 5. Resized the windows (a) Attempted to resize the main window (b) Attempted to resize the configuration editor window

Continued on next page

Table A.23 – *Continued from previous page*

Item	**Description**
Expected results	1. The program starts and loads the graphical user interface 2. (a) Menu for changing configuration values is opened (b) The specified database is loaded and used instead of the database loaded on startup (c) The program gives a popup with a recommendation for the selected policy based on the history in the specified database (d) The program a warning message that there is no policy to classify. Algorithm is not run (e) Program terminates 3. (a) The "newDB" option is set to the specified option. The "useNet" option is set to the specified option. (b) All the dropdown menus can be opened, and an option and be chosen (c) File manager opens. The option is set to the selected file / folder (d) The configuration editor closes without saving changes (e) The configuration editor closes saving changes 4. (a) The scroll bar scrolls through the list, up and down, from end to end, successfully (b) The scroll bar scrolls through the list, up and down, from end to end, successfully (c) The scroll bar scrolls through the list, up and down, from end to end, successfully (d) The scroll bar scrolls through the list, up and down, from end to end, successfully (e) The scroll bar scrolls through the list, up and down, from end to end, successfully 5. (a) Window can be resized without having elements of the GUI overlapping. The elements and panes scales with the main window (b) Window can be resized without having elements of the GUI overlapping. The elements and panes scales with the main window

Continued on next page

Table A.23 – *Continued from previous page*

Item	Description
Step results	1. The program starts and loads the graphical user interface 2. (a) The mencu for changing configuration values is opened (b) The specified database is loaded (c) The program gives a recommendation for the selected policy based on history (d) The program a warning message that there is no policy to classify. Algorithm is not run (e) Program terminates 3. (a) The "newDB" option is set to the specified option. The "useNet" option is set to the specified option. (b) All the dropdown menus can be opened, and an option and be chosen (c) File manager opens. The option is set to the selected file / folder (d) The configuration editor closes without saving changes (e) The configuration editor closes saving changes 4. (a) The scroll bar scrolls from end to end without any error (b) The scroll bar scrolls from end to end without any error (c) The scroll bar scrolls from end to end without any error (d) The scroll bar scrolls from end to end without any error (e) The scroll bar scrolls from end to end without any error 5. (a) Window can be resized without having elements of the GUI overlapping. The elements and panes scales with the main window (b) Window can be resized without having elements of the GUI overlapping. The elements and panes scales with the main window

Continued on next page

Table A.23 – *Continued from previous page*

Item	**Description**
Test conclusion	1. Success 2. (a) Success (b) Success (c) Success (d) Success (e) Success 3. (a) Success (b) Success (c) Success (d) Success (e) Success 4. (a) Success (b) Success (c) Success (d) Success (e) Success 5. (a) Success (b) Success Test passed
Comments	-

Table A.24: UNIT-05

Item	**Description**
Name	Algorithm classification
Test identifier	UNIT-05
Person responsible	Henrik Knutsen & Dimitry Kongevold
Date of execution Date of completion	October 30th November 1st
Execution steps	1. Load a set of policies into the database file 2. Manually calculate and write down the distances between the single policy and each of the policies in the history 3. Run the distance algorithm on a single policy and the history and compare the distances that are returned by the algorithm with the manually calculated distances from step 2 4. Manually find the k policies with the lowest distances 5. Run the reduction algorithm with necessary input to find the k nearest policies and compare the k policies returned by the reduction algorithm with those found in step 4 6. Run the conclusion algorithm and verify the results returned by the algorithm

Continued on next page

Table A.24 – *Continued from previous page*

Item	**Description**
Steps executed	1. Created a test domain by loading six policies into the history 2. Distances between the policy to be classified and each of the six policies were calculated manually and the results were inserted into a table 3. A JUnit test testReduction_KNN was created. This test was used to assert that the six values returned by the algorithm are the same as those calculated manually in step 2 4. Found the k nearest policies from what was calculated in step 2 5. A JUnit test testReduction_KNN was created. This test was used to assert that the reduction algorithm returns the same k nearest policies as those that was found manually in step 4 6. Created a JUnit Conclusion_Simple. This test is checking if the algorithm is giving the correct recommendation based on the k nearest policies found in step 5, and that the confidence has the correct value
Expected results	1. The policies are added to the database file 2. The six distances are obtained 3. The algorithm returns the same distances as those found in step 2 4. The k policies are obtained 5. The algorithm returns the same k policies as those found in step 4 6. The algorithm is giving the correct recommendation and confidence value

Continued on next page

Table A.24 – *Continued from previous page*

Item	**Description**
Step results	1. The policies are added to the database file 2. The six distances are obtained 3. The JUnit is successful 4. The k policies are obtained 5. The JUnit test is successful 6. The JUnit test is successful
Test conclusion	1. Success 2. Success 3. Success 4. Success 5. Success 6. Success Test passed
Comments	As mentioned in the test plan, it is not guaranteed that the algorithm will classify correctly for every possible combination of policy and database history even though it has passed this test

Table A.25: UNIT-06

Item	**Description**
Name	Algorithm learning
Test identifier	UNIT-06
Person responsible	Henrik Knutsen & Neshahavan Karunakaran
Date of execution Date of completion	November 4th November 12th
Execution steps	1. Make a set of policies and load the set into the history 2. Read the weights from the weights file 3. Run the classification and learning algorithms on the policy to be classified and the history, with the weights from step 2 4. Read the weights from the weights file 5. Compare the contents of the weights obtained in steps 2 and 4
Steps executed	1. Program is started with a set of policies used to build a history 2. Contents of the weights file is written down 3. A JUnit test LearnAlgSimplerTest is created. The test runs the classification and learning algortihms on the policy to be classified and the history 4. The test loads the weights from the weights file 5. The test compares the weights loaded in step 4 with the values that was written down in step 2
Expected results	1. The policies are loaded into the history successfully 2. The weights are written down 3. The classification and learning algorithm runs successfully on the policy to be classified and the history 4. The weights are loaded 5. The weights loaded in step 4 are different from the weights written down in step 2

Continued on next page

Table A.25 – *Continued from previous page*

Item	**Description**
Step results	1. The policies are loaded into the history successfully 2. The weights are written down 3. The JUnit test runs the classification and learning algorithm successfully 4. The JUnit test loads the weights file successfully 5. The JUnit test confirms that the values in the weights file have changed
Test conclusion	1. Success 2. Success 3. Success 4. Success 5. Success Test passed
Comments	Some changes are needed for this test to run. These changes are mentioned in the test class LearnAlgSimplerTest.java

Table A.26: UNIT-07

Item	**Description**
Name	Interaction with community database
Test identifier	UNIT-07
Person responsible	Henrik Knutsen
Date of execution Date of completion	November 14th November 14th
Execution steps	1. Upload a policy to the community database 2. Check that the database returns a recommendation
Steps executed	1. Ran the program trying to classify a policy with a database history using networking. Chose to save the new policy 2. Checked that a recommendation is returned from the community database
Expected results	1. The policy to be classified is saved on the community database 2. The community database returns a recommendation based on the policy to be classified and the database history
Step results	1. The policy to be classified is saved on the community database with no loss of data 2. The community database returns a recommendation. See comments
Test conclusion	1. Success 2. Success. See comments Test passed

Continued on next page

Table A.26 – *Continued from previous page*

Item	**Description**
Comments	The community database returns a recommendation, but it returns the same recommendation every time for any combination of input. This is because this part is not yet implemented.

Templates

A.6.1 Status Reports

Status Report

Customer Driven Project - Group 4

Week:
Prepared by:

Item	Task	Status	Responsible	Deadline
1				
2				
3				
4				
5				
6				
7				
8				
9				

Figure A.24: Status Report Template.

A.6.2 Meeting Notes

Meeting Minutes

Customer Driven Project - Group 4

"Learning Privacy Preferences"

Date/Time	
Purpose	
In attendance	
Notes taker	

Agenda

Topic	Discussion	Action	Responsible
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Figure A.25: Meeting Report Template.

A.6.3 Time Reporting

Week / Activity	Planning	Pre-Implementation Research	Requirement Specification	Design	Implementation	Evaluation/Testing	Documentation	Reporting	Adminstrative Tasks	**Total**
36										
37										
38										
39										
40										
41										
42										
43										
44										
45										
46										
47										
48										
49										
Total hrs										
Pct.										

Figure A.26: Time Report Template.

A.6.4 Test Plan

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.

Figure A.27: Test Plan Template.

A.6.5 Java documentation

```
//note double ** in all block comments

/**
 * One sentence describing what is here and what it does.
 *
 * @author author1
 * @author author2
 *
 */


//NB- all imports above class definition
import java.math.*; //purpose of import


/**
 * Paragraph describing purpose of class in detail- what it contains, stores etc.
 * @version versionnum //NB (use date.version- eg DDMMYY.1, etc)
 * @author author1
 */
//NB class defintion- one class per file please, unless anonymous
public class TemplateClass {

	//NB all class variables here, grouped by type.
	//NB only define one per line, with associated comment of purpose

	public int status; //global status tracker, 0 if done
	private int internalStatus; //internal status tracker, 1 unless error

	private String name; //input name


	//NB all constructors here

	/**
	 * Paragraph laying out constructor details, etc.
	 * Constructor can be auto-generated by Source->Generate Constructor...
	 *
	 *
	 * @param status default starting status, should be 1
	 * @param internalStatus default internal status, needs to be 1
	 * @param name  random string
	 *
	 */
	public TemplateClass(int status, int internalStatus, String name) {
		super();
		this.status = status;
		this.internalStatus = internalStatus;
		this.name = name;
	}



	//NB begin methods here
	//NB only the PrivacyAdvisor should have a main method...
	/**
	 * description of method
	 *
	 * @param a input int for method...
	 * @return and garbled string
	 */
	public String getResult(int a)
	{
		internalStatus = 1;
		status = 1;
		String temp = name.replace((char)a, (char)status);
		status = 0;
		return temp;

	}
}
```

Figure A.28: Javadoc Template.

Javadoc

A.7 Main

A.7.1 Gio

Full name: `public class Gio`

Package com.kpro.main

Inherits Object

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public Gio(String[] args) throws Exception Constructor fo gio class. There should only be one. Consider this a singleton instance to call I/O messages on. Constructs and parses command line arguements as well.

Parameter String[] args

Exceptions Exception — Mostly from loadWeights, but should also happen for loadFromConfig

---

public Gio(String[] args, UserIO ui) throws Exception A constructor permitting a user interface class to launch everything and be in control.

Parameter	String[] args	any commandline arguements
	UserIO ui	the known UserIO object
Exceptions	Exception	Mostly from loadWeights, but should also happen for loadFromConfig

---

Methods

public void configUI( ) call the user interface's general configuration method if the userInit option is true, and a user interface exists

---

public void setGenProps(Properties genProps)

Parameter Properties genProps

---

public Properties loadFromConfig(String fileLoc) Loads the general configuration file, either from provided string, or default location (./PrivacyAdviser.cfg)

Return properties object corresponding to given configuration file

Parameter String fileLoc

---

public Properties loadWeights( ) throws Exception Loads the weights configuration file, from the provided location

Return properties object corresponding to given configuration file

Exceptions	Exception	if there's an issue reading the file (if it doesn't exist, or has an IO error)

---

public Logger startLogger(String logLoc, String logLevel) startLogger initializes and returns a file at logLoc with the results of logging at level logLevel.

Return Logger object to log to.

Parameter	String logLoc	location of the output log file- a string
	String logLevel	logging level (is parsed by level.parse())

---

public void loadDB( ) Loads the case history into cache. This is where the background database chosen.

---

public PolicyDatabase getPDB( ) returns the only policy database

Return the policy database

---

public void shutdown( ) closes resources and write everything to file

---

public PolicyObject userResponse(PolicyObject n) Generates handles response. This is were we would pass stuff to cli or gui, etc

Return the policyObjected as accepted by user (potentially modified

Parameter	PolicyObject n	the processed policy object

---

public void loadPO( ) returns the policy object from the policyObject option

Return the policy object to be processed

---

public PolicyObject getPO( )

---

public boolean isBuilding( ) returns the true if it should only build

Return true if a CBR should NOT be run

---

public void setWeights(Properties newWeightP) saves the new weights to a buffer variable before writing in the shutdown call

Parameter Properties newWeightP the new weights file to save

---

public CBR getCBR( ) throws Exception returns the CBR to use

Return the cbr to use

Exceptions Exception

---

public Properties getWeights( ) returns the originally imported set of weights

Return the weights for policy attributes

---

public void showDatabase( ) shows the database on the user interface, if the user interface exists and no user response is specied and there is no 'blanketAccept' option.

---

public boolean fileExists(String filepath) GUI classes should use this to ensure the user passes valid files to load.

Return true if the file exists, else false

Parameter String filepath path of the file to check

---

public NetworkR getNR( )

---

public double getConfLevel( ) gets the confidence level threshold from the configuration

Return the confidence threshold

A.7.2 PrivacyAdviser

Full name: `public class PrivacyAdviser`

Package com.kpro.main

Inherits Object

Main class.

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public PrivacyAdviser( )

Methods

public static void main(String[] args) throws Exception Program in following sequence- init, load stuff for cbr, run cbr, shutdown. should alter for more flexible cbr

options (different algorithms, selected by switch statement on ReduceChoice, ConclusionChoice, LearnChoice, etc, once those are in the config file/cli

Author ngerstle

Parameter String[] args — accepts optional command line arguments, including location of general config file (default pwd)

Exceptions Exception

---

public static void init(String[] args) throws Exception Initializes the program-loads general configuration, starts logger, loads weights, loads database

Author ngerstle

Parameter String[] args — accepts optional command line arguments, including location of general config file (default pwd)

Exceptions Exception

A.8 Algorithm

A.8.1 CBR

Full name: `public class CBR`

Package com.kpro.algorithm

Inherits Object

Case based reason. This is a working CBR class that handles process flow between init and shutdown. Should be easy to extend and overload various features, but should work for most cases as is.

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public CBR(Gio theIO, Properties weightsConfig, ReductionAlgorithm reduceAlg, ConclusionAlgorithm conclusAlg, LearnAlgorithm learnAlg) Our generic constructor.

Author ngerstle

Parameter	Gio theIO	the GIO instance
	Properties weightsConfig	the weights to use for distance metric & learning algorithm
	ReductionAlgorithm reduceAlg	the retrival algorithm- produce relevent cases from history
	ConclusionAlgorithm conclusAlg	the 'reuse' algorithm- produces a solution from relevant cases
	LearnAlgorithm learnAlg	the retain algorithm- modifies the 'weightsConfig' so the distance metric is more accurate

---

public CBR(Gio theIO) constructor that so we can call cbr.parse(string)

Author ngerstle

Parameter Gio theIO

Methods

public void run(PolicyObject newpol) runs through CBR with selected algorithms

Author ngerstle

Parameter PolicyObject newpol

---

public CBR parse(String string) throws Exception Parses the CBR option to create the class and instantiate correct algorithms.

Return the CBR defined by the input string

Parameter	String string	the string from either configuration file or commandline

Exceptions Exception

A.8.2 ReductionAlgorithm

Full name: `public abstract class ReductionAlgorithm`

Package com.kpro.algorithm

Inherits Object

The abstract class for implementing reduction algorithms, like Knearestneighbors. ReductionAlgorithm objects store the database, and reduce the set of polices to only the relevent policies (one or more). May include 'Conclusion'/'Summary' algorithms in the future.

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public ReductionAlgorithm(PolicyDatabase pdb, String[] extraArgs) Constructor for a reductionAlgorithm

Parameter PolicyDatabase pdb
String[] extraArgs

Methods

public abstract ArrayList reduce(PolicyObject newPO) the reduce call. returns an arraylist of policies in the policydatabase relevent to newPO

Author ngerstle

Return a modified newpol

Parameter PolicyObject newPO the new policy to consider- it shouldn't change within the algorithm

A.8.3 Reduction_KNN

Full name: `public class Reduction_KNN`

Package com.kpro.algorithm

Inherits Object←ReductionAlgorithm

A k-nearest-neighbors algorithm class. create it and call run on it to get the nearest k neighbors to the object passed to run().

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
PolicyDatabase pdb	ReductionAlgorithm
ArrayList reduce(PolicyObject)	ReductionAlgorithm
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public Reduction_KNN(DistanceMetric distanceMetric, PolicyDatabase database, String[] extraArgs) creates a kNearestNeighbors algorithm to use

Author ngerstle, ulfnore

Parameter	DistanceMetric distanceMetric	the class defining distance between objects
	PolicyDatabase database	the database of objects to operate on
	String[] extraArgs	from the config file

Methods

public ArrayList reduce(PolicyObject newPO) the method that returns the closest k objects to the parameter. works by sorting elements by distance from passed object, and passing the first k elements.

Author ngerstle

Return ArrayList<PolicyObject> an arraylist of size k of the nearest neighbors

Parameter	PolicyObject newPO	the new PolicyObject the thing to find the neighbors of

---

A.8.4 DistanceMetric

Full name: `public abstract class DistanceMetric`

Package com.kpro.algorithm

Inherits Object

An abstract Distance metric class. A DistanceMetric interface has to contain 3 methods method for calculation of distance between Recipients, Purposes and Retentions between cases and distance for data-type string

Author dimitryk

Version 160911.1

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Methods

public abstract double getTotalDistance(PolicyObject a, PolicyObject b)
Calculates total distance between two policies

Author dimitryk

Return double 0 if cases are similar and positive integer if they are not

Parameter	PolicyObject a	input PolicyObject
	PolicyObject b	input PolicyObject

A.8.5 Bitmapwithdata

Full name: `public class Bitmapwithdata`

Package com.kpro.algorithm

Inherits Object←DistanceMetric

A distance metric that calculates distance based on weighed union of a bit map interception

Author dimitryk

Version 240911.01

Inheritancetable

Element	Inherited from
double getTotalDistance(PolicyObject, PolicyObject)	DistanceMetric
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Methods

public double getTotalDistance(PolicyObject a, PolicyObject b) Initializes weights and returns the distance between two PolicyObjects.

Return the distance between the two policy objects

Parameter	PolicyObject a	the 1st policy object
	PolicyObject b	the second policy object

A.8.6 bitmapDistanceWisOne

Full name: `public class bitmapDistanceWisOne`

Package com.kpro.algorithm

Inherits Object←DistanceMetric

A distance metric that calculates distance based on weighed union of a bit map interception

Author dimitryk

Version 240911.01

Inheritancetable

Element	Inherited from
double getTotalDistance(PolicyObject, PolicyObject)	DistanceMetric
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Methods

public double getTotalDistance(PolicyObject a, PolicyObject b) Initializes weights (to one) and returns the distance between two PolicyObjects.

Return the distance between the two policy objects

Parameter		
	PolicyObject a	the 1st policy object
	PolicyObject b	the second policy object

A.8.7 bitmapDistance

Full name: `public class bitmapDistance`

Package com.kpro.algorithm

Inherits Object←DistanceMetric

A distance metric that calculates distance based on weighed union of a bit map interception

Author dimitryk

Version 240911.01

Inheritancetable

Element	Inherited from
double getTotalDistance(PolicyObject, PolicyObject)	DistanceMetric
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public bitmapDistance(Properties weights, String[] extraArgs)

Parameter Properties weights
String[] extraArgs

Methods

public double getTotalDistance(PolicyObject a, PolicyObject b) Initializes weights and returns the distance between two PolicyObjects.

Return the distance between the two policy objects

Parameter	PolicyObject a	the 1st policy object
	PolicyObject b	the second policy object

A.8.8 ConclusionAlgorithm

Full name: `public abstract class ConclusionAlgorithm`

Package com.kpro.algorithm

Inherits Object

abstract class for all conclusion classes (they take the new policy and a reduction of the history versus new policy), and return an Action. May extend ReductionAlgorithm in the future (return a modified np, instead of an action). May be used instead of a ReductionAlgorithm. call with: Action a = (new ConclusionAlgorithm()).conclude(newpol,theIO.getPDB());

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public ConclusionAlgorithm(DistanceMetric dm, String[] extraArgs) ConclustionAlgorithm constructor

Parameter DistanceMetric dm
String[] extraArgs

Methods

public abstract Action conclude(PolicyObject np, Iterable knearestns)
Provides an action recommendation for np based on the given set of objects

Return a recommended Action

Parameter	PolicyObject np	the new policy
	Iterable knearestns	a set of relevant policies

A.8.9 Conclusion_Simple

Full name: `public class Conclusion_Simple`

Package com.kpro.algorithm

Inherits Object←ConclusionAlgorithm

a very simple conclusion class. result is based on the closest objects only, as determined by the sum of inverse distances of the accepted versus rejected policies. confidences is the ratio of sum inverse distances of the chosen decision, versus the sum of all inverse distances.

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
DistanceMetric distanceMetric	ConclusionAlgorithm
Action conclude(PolicyObject, Iterable)	ConclusionAlgorithm
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public Conclusion_Simple(DistanceMetric dm, String[] extraArgs)

Parameter DistanceMetric dm
String[] extraArgs

Methods

public Action conclude(PolicyObject np, Iterable releventSet) makes a decision on the reduced set. This class creates two lists, one for accepted policies and one for rejected. Assuming there are policies in both (easy decision otherwise), whether the policy is accepted or not will depend on the difference between the sum of inverse distances of the list items (excluding zero-distances), with the smaller sum indicating the more relevent decision.

Author ngerstle

Return an arraylist of {Action a, double Confidence)

Parameter	PolicyObject np	the object under consideration
	Iterable releventSet	the reduced set of neighbors

A.8.10 LearnAlgorithm

Full name: `public abstract class LearnAlgorithm`

Package com.kpro.algorithm

Inherits Object

An abstract class covering all learning algorithms. The learning algorithm alters the weights configuration after examining the current database after the addition of a new policy.

Author ngerstle

Version 29.09.11

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public LearnAlgorithm(Properties weightsConfig, String[] extraArgs) Constructor for a learning algorithm. accepts a weights configuration file.

Parameter	Properties weightsConfig	the weights configuration file
	String[] extraArgs	extra arguments defined in configuration file

Methods

public void learn(Gio theIO) runs the learning algorithm, and puts the results in the newWeight buffer in theIO(Gio)

Author ngerstle

Parameter Gio theIO

A.8.11 Learn_Constant

Full name: `public class Learn_Constant`

Package com.kpro.algorithm

Inherits Object←LearnAlgorithm

The simplest implementation of learnAlgorithm, this literally does nothing, and thus doesn't actually learn.

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
Properties applyML(Gio)	LearnAlgorithm
void learn(Gio)	LearnAlgorithm
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public Learn_Constant(Properties weightsConfig, String[] extraArgs) constructor

Author ngerstle, ernie

Parameter	Properties weightsConfig	the weights configuration file
	String[] extraArgs	from the config file

A.8.12 LearnAlgBasic

Full name: `public class LearnAlgBasic`

Package com.kpro.algorithm

Inherits Object←LearnAlgorithm

A very slow learning algorithm that goes through every case in every PolicyObject in the whole database.

Author Nesha

Inheritancetable

Element	Inherited from
Properties applyML(Gio)	LearnAlgorithm
void learn(Gio)	LearnAlgorithm
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public LearnAlgBasic(Properties weightsConfig, String[] extraArgs) constructor

Parameter Properties weightsConfig the weights
String[] extraArgs from the config

A.8.13 LearnAlgStand

Full name: `public class LearnAlgStand`

Package com.kpro.algorithm

Inherits Object←LearnAlgorithm

An learning algorithm that extends LearnAlgorithm

Author Nesha

Inheritancetable

Element	Inherited from
Properties applyML(Gio)	LearnAlgorithm
void learn(Gio)	LearnAlgorithm
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public LearnAlgStand(Properties weightsConfig, String[] extraArgs) constructor

Parameter	Properties weightsConfig	the weights
	String[] extraArgs	from the config

A.9 Datastorage

A.9.1 PolicyDatabase

Full name: `public abstract class PolicyDatabase`

Package com.kpro.datastorage

Inherits Object

Implements Iterable

This is abstract class for databases. All policy databases must implement this interface. Auto-generated via eclipse from PDatabase.java, then made abstract. Does enforce singleton-ness here. need to do it in each subclass.

Author ngerstle

Version 29.09.11.1

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Methods

public void loadDB( ) Loads database from a file dLoc

Author ngerstle

---

public abstract void loadDB(String dLoc) Loads database from a file dLoc we have this public just in case we want to be able to load a db from a file other than where we plan to store it.

Author ngerstle

Parameter String dLoc the inlocation of the database file on disk

---

public void addPolicy(PolicyObject n) Adds a PolicyObject to the database

Parameter PolicyObject n the policy to add to the database

---

public Iterator iterator( ) provides an iterator over the database's PolicyObjects

Return an iterator over the internal collection

---

public void closeDB( ) calls closeDB(PolicyDatabase.outlocation). needed due to lack of a good destructor system, as far as java goes (according to google)

---

public abstract void closeDB(String dLoc) should implement writing the information contained by the PolicyDatabase to a file/disk

Author ngerstle

Parameter String dLoc the inlocation to save the data

---

public abstract ArrayList getDomain(String domain) This class should return all policys for the given domain.

Author ngerstle

Parameter String domain the domain to look for

---

public String toString( ) Standard toString method

Author ulfnore

A.9.2 PDatabase

Full name: `public class PDatabase`

Package com.kpro.datastorage

Inherits Object←PolicyDatabase

Implements Serializable

This singleton class will store all past P3P/contexts instances in a hashmap, and is saved on disk via java.io.serializable.

Author Nicholas Gerstle, Henrik Knutsen, Aman Kaur

Version 1.0

Inheritancetable

Element	Inherited from
PolicyDatabase i	PolicyDatabase
Collection idb	PolicyDatabase
String inLocation	PolicyDatabase
String outLocation	PolicyDatabase
void addPolicy(PolicyObject)	PolicyDatabase
void closeDB( )	PolicyDatabase
void closeDB(String)	PolicyDatabase
ArrayList getDomain(String)	PolicyDatabase
Iterator iterator( )	PolicyDatabase
void loadDB( )	PolicyDatabase
void loadDB(String)	PolicyDatabase
String toString( )	PolicyDatabase
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object

void wait( )	Object

Methods

public void loadDB(String dLoc) See PolicyDatabase.java. implements loading/closing the db via a serialize PDatabase object. WILL overwrite existing database if called more than once.

Author ngerstle

See also PolicyDatabase#closeDB()

Parameter String dLoc the location of the database file on disk

---

public void closeDB(String dLoc) See PolicyDatabase.java. implements loading/closing the db via a serialize PDatabase object.

See also PolicyDatabase#closeDB()

Parameter String dLoc

---

public ArrayList getDomain(String domain) returns a list of all policies from a given domain.

Author ngerstle

See also PolicyDatabase#getDomain()

Return an arraylist of policies from given domain

Parameter String domain

---

public static PolicyDatabase getInstance(String inloc, String outloc) returns an instance of the database.

Parameter	String inloc	where the database should be loaded from (on disc)
	String outloc	where the database should be saved to (on disc)

A.9.3 NRCouchdb

Full name: `public class NRCouchdb`

Package com.kpro.datastorage

Inherits Object←NetworkR

The network resource class for working with a couchDB server cross network at the specified location.

Author ngerstle

Version 17.10.11.1

Inheritancetable

Element	Inherited from
void disconnect( )	NetworkR
String getInfo( )	NetworkR
void parseNOptions(String)	NetworkR
Action reqAct(PolicyObject)	NetworkR
void saveObj(PolicyObject)	NetworkR
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Fields

public final String type

---

public final String LVquerybase Couchlight object- holds connection to database

---

Construktors

public NRCouchdb(String options) The view, and the list to pass through when requesting: append jsonated object as key/value

Parameter String options

Methods

public Action reqAct(PolicyObject a)

Parameter PolicyObject a

---

public void saveObj(PolicyObject a) Saves the policy object to the database

Return void

Parameter PolicyObject a the PolicyObject to save

---

public void disconnect( ) closes/deletes any remaing resources

Return void

A.9.4 NetworkR

Full name: `public abstract class NetworkR`

Package com.kpro.datastorage

Inherits Object

Network resources class. Abstract. See javadocs for what contains what. Should allow request for an action (if local knowledge is insufficient), as well as saving to the community database.

Author ngerstle

Version 17.10.11.1

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public NetworkR(String options) Constructor- should accept as minimum the location of the networked database.

Parameter String options

Methods

public abstract Action reqAct(PolicyObject a) accepts a PolicyObject, and returns the remote suggested action for it (the suggestion from the server).

Return the action to give the policy object.

Parameter	PolicyObject a	the PolicyObject to obtain a response for.

---

public abstract void saveObj(PolicyObject a) sends a PolicyObject to the server. It needs to have complete Action attached.

Parameter	PolicyObject a	the policy to upload remotely.

---

public abstract void disconnect( ) closes whatever resources were opened during initialization by the constructor

---

public String getInfo( ) returns the server type and location.

Return String "A "+ typeOfServer + " database located at "+ locationOfServer

---

A.10 Dataobjects

A.10.1 PolicyObject

Full name: `public class PolicyObject`

Package com.kpro.dataobjects

Inherits Object

Implements Serializable
Iterable

The PolicyObject class acts to hold all the relevent information for a given policy. This informations is broken into 'cases' (see the 'case' class- different items, indexed by datatype), the context (see the 'context' class- holds information that applies to whole policy), and action (see the 'action' class- holds the action taken).

Author ernie, ngerstle

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public PolicyObject( ) This is the constructor The constructor initializes the variables within the class when you make a new instance of it

Author ernie

Methods

public Context getContext( )

---

public void setContext(Context context)

Parameter Context context

---

public Action getAction( )

---

public PolicyObject setAction(Action action)

Parameter Action action

---

public void addCase(Case c) Adds a case to the policy

Author ernie

Parameter Case c input Case

---

public void addEntityData(String key, String value) Adds a data to the entity hashmap of the policy

Author ernie

Parameter String key input String
String value input String

---

public Case getCase(int i) Returns a specific case for the policy

Author ernie

Return Case

Parameter int i input int

---

public ArrayList getCases( ) Returns all cases for the policy

Author ernie

Return ArrayList<Case>

---

public HashMap getEntities( ) Returns the entity for the policy

Author ernie

Return HashMap<String, String>

---

public String getEntity(String key) Returns the entity data for a specific key

Author ernie

Parameter String key input String

---

public String getContextDomain( ) returns domain of policy (URL) as string.

Author ngerstle

Return the domain/url from which the policy came.

---

public String toString( ) This is based on the debug_print

Author ulfnore

---

public boolean equalsCases(PolicyObject newpol) A simple true/false check to see if policies are identical- if all the strings inside them are, then the policies are.

Return boolean; true if getContextDomains are equal and getCases() are equal (two string comparisons) else false.

Parameter PolicyObject newpol the policy to compare 'this' to

---

public Iterator iterator( )

A.10.2 Context

Full name: `public class Context`

Package com.kpro.dataobjects

Inherits Object

Holds domain (from p3p), time, and other contextual information that applies to an entire p3p policy.

Author ngerstle, iernie

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public Context(Date accessTime, Date creationTime, String domain)

Parameter Date accessTime
Date creationTime
String domain

Methods

public Date getAccessTime( ) Gets the date of when the object was last accessed

Return Date

---

public void setAccessTime(Date accessTime) Sets the date of when the object was last accessed

Parameter Date accessTime

---

public Date getCreationTime( ) Gets the date of when the object was created

Return Date

---

public void setCreationTime(Date creationTime) Sets the date of when the object was created

Parameter Date creationTime

---

public Date getExpiryDate( ) Gets the date of when the object is to expire

Return Date

---

public void setExpiryDate(Date expiryDate) Sets the date of when the object is to expire

Parameter Date expiryDate

---

public String getDomain( ) Gets the specific domain for this context

Return String

---

public void setDomain(String domain) Sets the specific domain for this contact

Parameter String domain

A.10.3 Action

Full name: `public class Action`

Package com.kpro.dataobjects

Inherits Object

holds results of a algorithmic comparison- t/f on approve, with the nearest neighbor, as well as a string & enum for exception, if it is one

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public Action( )

---

public Action(boolean accept, ArrayList domains, double confidence, boolean override)

Parameter boolean accept
ArrayList domains
double confidence
boolean override

Methods

public ArrayList getReason( )

---

public void setReason(ArrayList reason)

Parameter ArrayList reason

---

public String getAcceptedStr( ) converts the internal accept/reject values to a String

Return a boolean that can be sent to the user with a accept/reject

---

public boolean getAccepted( ) Returns true if the action was accepted, and false otherwise.

Return boolean

---

public void setAccepted(boolean accept) Sets the accepted state of the action.

Parameter boolean accept

---

public ArrayList getReasons( )

Return an arraylist that verbalizes why the policy was accepted or rejected

---

public boolean isOverridden( ) Returns true if the action is manually overridden.

Return boolean

---

public void setConfidence(double confidence) Sets the confidence, with checks on the value: if confidence = abs(input) if 1>=input>=-1, else negative infinity

Parameter double confidence

---

public double getConfidence( )

---

public Action parse(String optionValue) Parse a comma-seperated string into an Action. The string needs to have four comma-seperated tokens (thus three commas) and no spaces. The format is accept,domains,confidence,override where accept is 'accept' if accept==true, or anything else if accept!=true; domains is a semi-colon seperated string list of domains (no commas, no spaces, etc), eg 'www.google.com;www.yahoo.com;domain3;domain4' ; confidence is a double that is the confidence (parsed by parseDouble), and override is a boolean (parsed by parseBoolean).

Return an Action parsed from above

Parameter	String optionValue	the option string- see above. must have 3 commas, no spaces

---

public Action setOverride(boolean b)

Parameter boolean b

A.10.4 Case

Full name: `public class Case`

Package com.kpro.dataobjects

Inherits Object

Implements Comparable

A class that contains a single datatype from a PolicyObject.

Author ernie

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public Case( )

---

public Case(ArrayList purpose, ArrayList retention, ArrayList recipients, ArrayList categories, String datatype)

Parameter ArrayList purpose
ArrayList retention
ArrayList recipients
ArrayList categories
String datatype

Methods

public void addPurpose(Purpose p) Adds a purpose to the case

Parameter Purpose p

---

public void addRetention(Retention t) Adds a retention to the case

Parameter Retention t

---

public void addRecipient(Recipient r) Adds a recipient to the case

Parameter Recipient r

---

public void addCategory(Category c) Adds a category to the case

Parameter Category c

---

public void setDataType(String s) Sets the datatype to the case

Parameter String s

---

public ArrayList getPurposes( ) Gets the purposes

Return ArrayList<Purpose>

---

public Purpose getPurpose(int i) Returns the ith purpose

Return Purpose

Parameter int i

---

public ArrayList getRetentions( ) Gets the retentions

Return ArrayList<Retention>

---

public Retention getRetention(int i) Returns the ith retention

Return Retention

Parameter int i

---

public ArrayList getRecipients( ) Gets the recipients

Return ArrayList<Recipient>

---

public Recipient getRecipient(int i) Returns the ith recipient

Return Recipient

Parameter int i

---

public ArrayList getCategories( ) Gets the categories

Return ArrayList<Category>

---

public Category getCategory(int i) Returns the ith category

Return Category

Parameter int i

---

public String getDataType( ) Returns the datatype

Return String

---

public String toString( ) Based on debug.print

Author ulfnore

---

public int compareTo(Object o) to allow comparision of cases, primarily for white/blacklisting.

Author ngerstle

Return -1 if this > other, 0 if equal, else 1

Parameter Object o

A.10.5 Retention

Full name: `public final class Retention`

Package com.kpro.dataobjects

Inherits Object←Enum

The retentions that a case can contain. See P3P specs for more info.

Author ernie

Inheritancetable

Element	Inherited from
Object clone( )	Enum
int compareTo(Enum)	Enum
int compareTo(Object)	Enum
boolean equals(Object)	Enum
void finalize( )	Enum
Class getDeclaringClass( )	Enum
int hashCode( )	Enum
String name( )	Enum
int ordinal( )	Enum
String toString( )	Enum
Enum valueOf(Class, String)	Enum
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Fields

public static final Retention NO_RETENTION

---

public static final Retention STATED_PURPOSE

---

public static final Retention LEGAL_REQUIREMENT

---

public static final Retention BUSINESS_PRACTICES

---

public static final Retention INDEFINITELY

Methods

public static Retention[] values( )

---

public static Retention valueOf(String name)

Parameter String name

A.10.6 Recipient

Full name: `public final class Recipient`

Package com.kpro.dataobjects

Inherits Object←Enum

The recipients that a case can contain. See P3P specs for more info.

Author ernie

Inheritancetable

Element	Inherited from
Object clone( )	Enum
int compareTo(Enum)	Enum
int compareTo(Object)	Enum
boolean equals(Object)	Enum
void finalize( )	Enum
Class getDeclaringClass( )	Enum
int hashCode( )	Enum
String name( )	Enum
int ordinal( )	Enum
String toString( )	Enum
Enum valueOf(Class, String)	Enum
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Fields

public static final Recipient OURS

---

public static final Recipient DELIVERY

---

public static final Recipient SAME

---

public static final Recipient OTHER_RECIPIENT

---

public static final Recipient UNRELATED

---

public static final Recipient PUBLIC

Methods

public static Recipient[] values( )

---

public static Recipient valueOf(String name)

Parameter String name

---

public void setOptional( )

---

public boolean isOptional( )

A.10.7 Purpose

Full name: `public final class Purpose`

Package com.kpro.dataobjects

Inherits Object←Enum

The purposes that a case can contain. See P3P specs for more info.

Author ernie

Inheritancetable

Element	Inherited from
Object clone( )	Enum
int compareTo(Enum)	Enum
int compareTo(Object)	Enum
boolean equals(Object)	Enum
void finalize( )	Enum
Class getDeclaringClass( )	Enum
int hashCode( )	Enum
String name( )	Enum
int ordinal( )	Enum
String toString( )	Enum
Enum valueOf(Class, String)	Enum
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Fields

public static final Purpose CURRENT

---

public static final Purpose ADMIN

---

public static final Purpose DEVELOP

---

public static final Purpose TAILORING

---

public static final Purpose PSEUDO_ANALYSIS

---

public static final Purpose PSEUDO_DECISION

---

public static final Purpose INDIVIDUAL_ANALYSIS

---

public static final Purpose INDIVIDUAL_DECISION

---

public static final Purpose CONTACT

---

public static final Purpose HISTORICAL

---

public static final Purpose TELEMARKETING

---

public static final Purpose OTHER_PURPOSE

---

public static final Purpose CUSTOMIZATION

Methods

public static Purpose[] values( )

---

public static Purpose valueOf(String name)

Parameter String name

---

public void setOptional( )

---

public boolean isOptional( )

A.10.8 Category

Full name: `public final class Category`

Package com.kpro.dataobjects

Inherits Object←Enum

The categories that a case can contain. See P3P specs for more info.

Author ernie

Inheritancetable

Element	Inherited from
Object clone( )	Enum
int compareTo(Enum)	Enum
int compareTo(Object)	Enum
boolean equals(Object)	Enum
void finalize( )	Enum
Class getDeclaringClass( )	Enum
int hashCode( )	Enum
String name( )	Enum
int ordinal( )	Enum
String toString( )	Enum
Enum valueOf(Class, String)	Enum
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Fields

public static final Category PHYSICAL

---

public static final Category ONLINE

---

public static final Category UNIQUEID

---

public static final Category PURCHASE

---

public static final Category FINANCIAL

---

public static final Category COMPUTER

---

public static final Category NAVIGATION

---

public static final Category INTERACTIVE

---

public static final Category DEMOGRAPHIC

---

public static final Category CONTENT

---

public static final Category STATE

---

public static final Category POLITICAL

---

public static final Category HEALTH

---

public static final Category PREFERENCE

---

public static final Category LOCATION

---

public static final Category GOVERNMENT

---

public static final Category OTHER_CATEGORY

Methods

public static Category[] values( )

---

public static Category valueOf(String name)

Parameter String name

A.11 Parser

A.11.1 P3PParser

Full name: `public class P3PParser`

Package com.kpro.parser

Inherits Object

Parser that parses a P3P policy and makes it into a PolicyObject

Author ernie

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public P3PParser( )

Methods

public PolicyObject parse(String p3p) Parses a P3P Policy by URL

Author ernie

Return Parsed P3P Policy as PolicyObject

Parameter String p3p input String

A.12 UI

A.12.1 PrivacyAdvisorGUI

Full name: `public class PrivacyAdvisorGUI`

Package com.kpro.ui

Inherits Object←UserIO

Privacy Advisor GUI to run on top of

Author ulfnore

Inheritancetable

Element	Inherited from
void closeResources( )	UserIO
ArrayList loadHistory( )	UserIO
void showDatabase(PolicyDatabase)	UserIO
PolicyObject userResponse(PolicyObject)	UserIO
void user_init(Properties)	UserIO
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public PrivacyAdvisorGUI( ) Default no-arg constructor

Author ulfnore

Methods

public static void main(String[] args) Launch the application.

Author ulfnore

Parameter String[] args

---

public void user_init(Properties genProps) Called from GIO. Takes default properties file as argument.

Parameter Properties genProps

---

public ArrayList loadHistory( )

---

public PolicyObject userResponse(PolicyObject n) Shows recommendation and prompts for user action

Needs improvement to allow for giving reasons as for why recommendation is not accepted.

Author ulfnore

Parameter PolicyObject n

---

public void closeResources( )

---

public void showDatabase(PolicyDatabase pdb)

Parameter PolicyDatabase pdb

A.12.2 UserIO

Full name: `public abstract class UserIO`

Package com.kpro.ui

Inherits Object

The UserIO provides an abstract model of all the methods a user interface method must implement. There are five essential portions: construction of the user interface if necessary, via the object constructor; user reconfiguration (of the same options found in configuration file or on the commandline); the ability to display the database and all loaded policies; user revision, in which the suggested solution is provided to the user so the user can accept or reject it; and shutdown/deconstruction of any resources needed for the interface.

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public UserIO( )

Methods

public abstract void user_init(Properties genProps) returns a modified Properties to use init on.

Return the values to initialize the program with- same as usual args from main

Parameter Properties genProps the default values for all commandline arguments

---

public abstract void showDatabase(PolicyDatabase pdb) display the contents of the database

Author ngerstle

Parameter PolicyDatabase pdb the database to display

---

public abstract ArrayList loadHistory( ) gets any policies not already provided for the history

Author ngerstle

Ddeprecated

Return an arraylist of policy objects to be added to history prior to the CBR run.

---

public abstract PolicyObject userResponse(PolicyObject n) Displays recommended action for policyObject, and returns used accept verion- same thing if no change, or altered if user disagrees.

Author ngerstle

Return the policy given

Parameter PolicyObject n the policy display

---

public abstract void closeResources( ) closes all resources used by UserIO - windows, files, streams, etc

Author ngerstle

A.12.3 UserIO_Simple

Full name: `public class UserIO_Simple`

Package com.kpro.ui

Inherits Object←UserIO

This is a very simple commandline version of a user interface. It doesn't permit user configuration (of running options), it has a ugly database display, but it works.

Author ngerstle
ulfnore

Inheritancetable

Element	Inherited from
void closeResources( )	UserIO
ArrayList loadHistory( )	UserIO
void showDatabase(PolicyDatabase)	UserIO
PolicyObject userResponse(PolicyObject)	UserIO
void user_init(Properties)	UserIO
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public UserIO_Simple( )

Methods

public void showDatabase(PolicyDatabase pdb) does nothing

See also UserIO#showDatabase(PolicyDatabase)

Parameter PolicyDatabase pdb

---

public ArrayList loadHistory( ) does nothing

See also UserIO#loadHistory()

---

public PolicyObject userResponse(PolicyObject n) A super simple, static user display of the result on command line. does not wait for user response

Author ngerstle
ulfnore

See also UserIO#userResponse(PolicyObject)

Return the policy given

Parameter PolicyObject n the policy display

---

public void closeResources( ) nothing to close

Author ngerstle

---

public void user_init(Properties genProps) This user interface doesn't actually let the user reconfigure anything. Would do so through the reference to Gio if necessary.

Parameter Properties genProps

A.12.4 IO_Listing

Full name: `public final class IO_Listing`

Package com.kpro.ui

Inherits Object←Enum

Author ulfnore

Inheritancetable

Element	Inherited from
Object clone( )	Enum
int compareTo(Enum)	Enum
int compareTo(Object)	Enum
boolean equals(Object)	Enum
void finalize( )	Enum
Class getDeclaringClass( )	Enum
int hashCode( )	Enum
String name( )	Enum
int ordinal( )	Enum
String toString( )	Enum
Enum valueOf(Class, String)	Enum
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Fields

public static final IO_Listing UserIO_Simple

---

public static final IO_Listing PrivacyAdvisorGUI

Methods

public static IO_Listing[] values( )

---

public static IO_Listing valueOf(String name)

Parameter String name

A.13 Test

A.13.1 testConclusion_Simple

Full name: `public class testConclusion_Simple`

Package com.kpro.test

Inherits Object←Assert←TestCase

Inheritancetable

Element	Inherited from
int countTestCases( )	TestCase
TestResult createResult( )	TestCase
String getName( )	TestCase
TestResult run( )	TestCase
void run(TestResult)	TestCase
void runBare( )	TestCase
void runTest( )	TestCase
void setName(String)	TestCase
void setUp( )	TestCase
void tearDown( )	TestCase
String toString( )	TestCase
void assertEquals(String, Object, Object)	Assert
void assertEquals(Object, Object)	Assert
void assertEquals(String, String, String)	Assert
void assertEquals(String, String)	Assert
void assertEquals(String, double, double, double)	Assert
void assertEquals(double, double, double)	Assert
void assertEquals(String, float, float, float)	Assert
void assertEquals(float, float, float)	Assert
void assertEquals(String, long, long)	Assert
void assertEquals(long, long)	Assert
void assertEquals(String, boolean, boolean)	Assert
void assertEquals(boolean, boolean)	Assert
void assertEquals(String, byte, byte)	Assert
void assertEquals(byte, byte)	Assert
void assertEquals(String, char, char)	Assert
void assertEquals(char, char)	Assert
void assertEquals(String, short, short)	Assert
void assertEquals(short, short)	Assert

void assertEquals(String, int, int)	Assert
void assertEquals(int, int)	Assert
void assertFalse(String, boolean)	Assert
void assertFalse(boolean)	Assert
void assertNotNull(Object)	Assert
void assertNotNull(String, Object)	Assert
void assertNotSame(String, Object, Object)	Assert
void assertNotSame(Object, Object)	Assert
void assertNull(Object)	Assert
void assertNull(String, Object)	Assert
void assertSame(String, Object, Object)	Assert
void assertSame(Object, Object)	Assert
void assertTrue(String, boolean)	Assert
void assertTrue(boolean)	Assert
void fail(String)	Assert
void fail( )	Assert
void failNotEquals(String, Object, Object)	Assert
void failNotSame(String, Object, Object)	Assert
void failSame(String)	Assert
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public testConclusion_Simple( )

Methods

public void testConclusion( )

A.13.2 testBitmap

Full name: `public class testBitmap`

Package com.kpro.test

Inherits Object←Assert←TestCase

Inheritancetable

Element	Inherited from
int countTestCases( )	TestCase
TestResult createResult( )	TestCase
String getName( )	TestCase
TestResult run( )	TestCase
void run(TestResult)	TestCase
void runBare( )	TestCase
void runTest( )	TestCase
void setName(String)	TestCase
void setUp( )	TestCase
void tearDown( )	TestCase
String toString( )	TestCase
void assertEquals(String, Object, Object)	Assert
void assertEquals(Object, Object)	Assert
void assertEquals(String, String, String)	Assert
void assertEquals(String, String)	Assert
void assertEquals(String, double, double, double)	Assert
void assertEquals(double, double, double)	Assert
void assertEquals(String, float, float, float)	Assert
void assertEquals(float, float, float)	Assert
void assertEquals(String, long, long)	Assert
void assertEquals(long, long)	Assert
void assertEquals(String, boolean, boolean)	Assert
void assertEquals(boolean, boolean)	Assert
void assertEquals(String, byte, byte)	Assert
void assertEquals(byte, byte)	Assert
void assertEquals(String, char, char)	Assert
void assertEquals(char, char)	Assert
void assertEquals(String, short, short)	Assert
void assertEquals(short, short)	Assert

void assertEquals(String, int, int)	Assert
void assertEquals(int, int)	Assert
void assertFalse(String, boolean)	Assert
void assertFalse(boolean)	Assert
void assertNotNull(Object)	Assert
void assertNotNull(String, Object)	Assert
void assertNotSame(String, Object, Object)	Assert
void assertNotSame(Object, Object)	Assert
void assertNull(Object)	Assert
void assertNull(String, Object)	Assert
void assertSame(String, Object, Object)	Assert
void assertSame(Object, Object)	Assert
void assertTrue(String, boolean)	Assert
void assertTrue(boolean)	Assert
void fail(String)	Assert
void fail( )	Assert
void failNotEquals(String, Object, Object)	Assert
void failNotSame(String, Object, Object)	Assert
void failSame(String)	Assert
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public testBitmap( )

Methods

public void testDistance( )

A.13.3 LearnAlgSimplerTest

Full name: `public class LearnAlgSimplerTest`

Package com.kpro.test

Inherits Object←Assert←TestCase

This class is a junit test class to test the class LearnAlgSimpler For this junit test to work correctly some changes have to be made in LearnAlgSimpler.java These are: -comment out "extends LearnAlgorithm" -comment out the constructor -make the applyML method take in a Properties and a PolicyDatabase, and make the method public, like this: public Properties applyML(Properties prop, PolicyDatabase pd) -change the two first lines in applyML to: Properties weights = prop; pdb = pd;

Author Nesha

Inheritancetable

Element	Inherited from
int countTestCases( )	TestCase
TestResult createResult( )	TestCase
String getName( )	TestCase
TestResult run( )	TestCase
void run(TestResult)	TestCase
void runBare( )	TestCase
void runTest( )	TestCase
void setName(String)	TestCase
void setUp( )	TestCase
void tearDown( )	TestCase
String toString( )	TestCase
void assertEquals(String, Object, Object)	Assert
void assertEquals(Object, Object)	Assert
void assertEquals(String, String, String)	Assert
void assertEquals(String, String)	Assert
void assertEquals(String, double, double, double)	Assert
void assertEquals(double, double, double)	Assert
void assertEquals(String, float, float, float)	Assert
void assertEquals(float, float, float)	Assert
void assertEquals(String, long, long)	Assert
void assertEquals(long, long)	Assert

void assertEquals(String, boolean, boolean)	Assert
void assertEquals(boolean, boolean)	Assert
void assertEquals(String, byte, byte)	Assert
void assertEquals(byte, byte)	Assert
void assertEquals(String, char, char)	Assert
void assertEquals(char, char)	Assert
void assertEquals(String, short, short)	Assert
void assertEquals(short, short)	Assert
void assertEquals(String, int, int)	Assert
void assertEquals(int, int)	Assert
void assertFalse(String, boolean)	Assert
void assertFalse(boolean)	Assert
void assertNotNull(Object)	Assert
void assertNotNull(String, Object)	Assert
void assertNotSame(String, Object, Object)	Assert
void assertNotSame(Object, Object)	Assert
void assertNull(Object)	Assert
void assertNull(String, Object)	Assert
void assertSame(String, Object, Object)	Assert
void assertSame(Object, Object)	Assert
void assertTrue(String, boolean)	Assert
void assertTrue(boolean)	Assert
void fail(String)	Assert
void fail( )	Assert
void failNotEquals(String, Object, Object)	Assert
void failNotSame(String, Object, Object)	Assert
void failSame(String)	Assert
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public LearnAlgSimplerTest( )

Methods

public void testApplyML( ) The actual testing is done here

Some lines are commented out because it shows error messages when the LearnAlgSimpler.java isn't as it should be when this junit test class is run (see the description of this class).

Uncomment the commented lines before running the junit test

A.14 Sample

A.14.1 TemplateClass

Full name: `public class TemplateClass`

Package com.kpro.sample

Inherits Object

Paragraph describing purpose of class in detail- what it contains, stores etc.

Author author1

Version versionnum //NB (use date.version- eg DDMMYY.1, etc)

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Fields

public int status

Construktors

public TemplateClass(int status, int internalStatus, String name) Paragraph laying out constructor details, etc. Constructor can be auto-generated by Source->Generate

Constructor...

Author author1

Parameter		
	int status	default starting status, should be 1
	int internalStatus	default internal status, needs to be 1
	String name	random string

Methods

public String getResult(int a) description of method

Author author1

Return and garbled string

Parameter		
	int a	input int for method...

---

public void setInternalStatus(int internalStatus) description of method

Author author1

Return void

Parameter int internalStatus

---

public int getInternalStatus( ) description of method

Return absolute value of the internal status times pi

---

public void setA(ArrayList a)

Parameter ArrayList a

---

public ArrayList getA( )

A.14.2 SerializationDemo

Full name: `public class SerializationDemo`

Package com.kpro.sample

Inherits Object

Author Aman This class shows the serializaton process It creates 2 objects Obj1 and Obj2 and saves them as 2 serialized objects (to the outputstream in a file)

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public SerializationDemo( )

Methods

public static void main(String[] args)

Parameter String[] args

A.14.3 readWeightConfig

Full name: `public class readWeightConfig`

Package com.kpro.sample

Inherits Object

A class that reads the weights from the weights.cfg file, and returns an array with values. The returned array have values that are arranged so that they correspond to the values as they are written in the config file. That means that if the first uncommented line in the config file is Recipient.OUR, then the first value in the returned array from the recipientWeight() method would be the value for Recipient.OUR.

Author Nesha

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public readWeightConfig( )

Methods

public int[] recipientWeight( ) reads the weights of the recipients and returns them as a list

Author Nesha

Return an array of the weights of the recipients

---

public int[] retentionWeight( ) reads the weights of the retentions and returns them as a list

Author Nesha

Return an array of the weights of the retentions

---

public int[] purposeWeight( ) reads the weights of the purposes and returns them as a list

Author Nesha

Return an array of the weights of the purposes

A.14.4 msgBox

Full name: `public class msgBox`

Package com.kpro.sample

Inherits Object

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public msgBox( )

Methods

public static void main(String[] args) throws IOException

Parameter String[] args

Exceptions IOException

A.14.5 jlistest

Full name: `public class jlistest`

Package com.kpro.sample

Inherits Object

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public jlistest( ) Create the application.

Methods

public static void main(String[] args) Launch the application.

Parameter String[] args

A.14.6 jfilechoosertest

Full name: `public class jfilechoosertest`

Package com.kpro.sample

Inherits Object

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public jfilechoosertest( )

Methods

public static void main(String[] args)

Parameter String[] args

A.14.7 GioTest

Full name: `public class GioTest`

Package com.kpro.sample

Inherits Object

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public GioTest( )

Methods

public static void main(String[] args)

Parameter String[] args

A.14.8 dummy

Full name: `public class dummy`

Package com.kpro.sample

Inherits Object

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public dummy( )

Methods

public static void main(String[] args)

Parameter String[] args

A.14.9 distanceMetricTest

Full name: `public class distanceMetricTest`

Package com.kpro.sample

Inherits Object←DistanceMetric

Inheritancetable

Element	Inherited from
double getTotalDistance(PolicyObject, PolicyObject)	DistanceMetric
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public distanceMetricTest(Properties weightsConfig)

Parameter Properties weightsConfig

Methods

public double getDistResip(Case a, Case b)

Parameter Case a
Case b

---

public double getDistReten(Case a, Case b)

Parameter Case a
Case b

---

public double getDistPurpose(Case a, Case b)

Parameter Case a
Case b

---

public double getWeigth(Recipient r) Just used some random return numbers, should be changed something meaningful later

Parameter Recipient r

---

public double getWeigth(Retention r)

Parameter Retention r

---

public double getWeigth(Purpose r)

Parameter Purpose r

---

public double getTotalDistance(PolicyObject a, PolicyObject b)

Parameter PolicyObject a
PolicyObject b

A.14.10 DeserializationDemo

Full name: `public class DeserializationDemo`

Package com.kpro.sample

Inherits Object

Author Aman This class reads the serialized objects from file system and displaying it

Inheritancetable

Element	Inherited from
Object clone( )	Object
boolean equals(Object)	Object
void finalize( )	Object
Class getClass( )	Object
int hashCode( )	Object
void notify( )	Object
void notifyAll( )	Object
String toString( )	Object
void wait(long)	Object
void wait(long, int)	Object
void wait( )	Object

Construktors

public DeserializationDemo( )

Methods

public static void main(String[] args)

Parameter String[] args

Bibliography

[1] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.

[2] Inger Anne Tøndel & Åsmund Ahlmann Nyre *Towards a similarity metric for comparing machine-readable privacy policies*. Sintef ICT, Trondheim, Norway, 2011.

[3] Inger Anne Tøndel & Åsmund Ahlmann Nyre & Karin Bernsmed, *Learning Privacy Preferences*. Sintef ICT, Trondheim, Norway, 2011.