Cape Peninsula
University of Technology

THE CONFLICT OF INTEREST BETWEEN DATA SHARING AND DATA PRIVACY: A MIDDLEWARE APPROACH

by

KARABO OMPHILE MOLEMA

Thesis submitted in fulfilment of the requirements for the degree
Master of Technology: Information Technology
in the Faculty of Informatics & Design
at the Cape Peninsula University of Technology

Supervisor: Mr. T Makhurane
Co-supervisor: Prof Lutz Prechelt

Cape Town
March 2016

Declaration

I, Karabo Omphile Molema, declare that the contents of this dissertation/thesis represent my own unaided work, and that the dissertation/thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed Date

Abstract

People who are referred to as data owners in this study, use the Internet for various purposes and one of those is using online services like Gmail, Facebook, Twitter and so on. These online services are offered by organizations which are referred to as data controllers. When data owners use these service provided by data controllers they usually have to agree to the terms and conditions which gives data controllers indemnity against any privacy issues that may be raised by the data owner. Data controllers are then free to share that data with any other organizations, referred to as third parties. Though data controllers are protected from lawsuits it does not necessarily mean they are free of any act that may be considered a privacy violation by the data owner. This thesis aims to arrive at a design proposition using the design science research paradigm for a middleware extension, specifically focused on the Tomcat server which is a servlet engine running on the JVM. The design proposition proposes a client side annotation based API to be used by developers to specify classes which will carry data outside the scope of the data controller's system to a third party system, the specified classes will then have code weaved in that will communicate with a Privacy Engine component that will determine based on data owner's preferences if their data should be shared or not. The output of this study is a privacy enhancing platform that comprises of three components the client side annotation based API used by developers, an extension to Tomcat and finally a Privacy Engine.

KEYWORDS: data privacy, data sharing, middleware, privacy enhancing, PET

Acknowledgments

This section is to acknowledge the people who have been with me through every step of this long journey.

First of all I would like to give thanks to the almighty God for guiding me throw this journey secondly I would like to give thanks to my supervisor. Mr Temuso Makhurane for providing supervision of my thesis, and support whether it be emotional or mental.

Thirdly I would like to thank my co-supervisor Professor Lutz Prechelt for the constructive criticism and putting an emphasis on writing such that you get to the point. This was difficult to grasp at first but as time went it become easier and easier.

Finally I would like to thank my wife for being an understanding partner, in giving me time to work on my thesis when needed, and providing that emotional support especially during the tough times.

Table of Contents

List of Figures

List of Tables

Glossary

ACID	Atomicity, Consistency, Isolation, Durability
AMD	Advanced Micro Devices
AOP	Aspect Oriented Programming
API	Application Programming Interface
AST	Abstract Syntax Tree
BCEL	ByteCode Engineering Library
BDSG	Bundesdatenschutzgesetz
CGLIB	Code Generation Library
CPU	Central Processing Unit
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
CQRS	Command Query Responsibility Segregation
CQS	Command Query Separation
DSL	Domain Specific Language
EE	Enterprise Edition
EJB	Enterprise Java Beans
ERD	Entity Relational Diagram
ESB	Enterprise Service Bus
GPRS	General Packer Radio Service
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ICT	Information & Communication Technology
IDE	Integrated Development Environment
IDP	Internet Data Provider
IS	Information System
J2SE	Java 2 Platform Standard Edition
JAVA-RS	Java API for Restful Web Services
JBI	Java Business Integration
JMS	Java Messaging Service
JOOQ	Java Object-Oriented Querying
JPA	Java Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol

MVC	Model View Controller
NGO	Non Government Organisation
NIO	Non-Blocking IO
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
ORM	Object Relational Mapping
OSGi	Open Services Gateway Initiative
P3P	Platform for Privacy Preferences Project
PbD	Privacy by Design
PET	Privacy Enhancing Technology
REST	Representational State Transfer
SBT	Simple Build Tool
SDLC	Software Development Life Cycle
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSD	Solid State Drive
SQL	Structured Query Language
TCP	Transmission Control Protocol
UI	User Interface
URL	Uniform Resource Locator
WAR	Web Application ARchieve
XML	Extensible Markup Language

Chapter 1.

Introduction

One of the most important things that can be done with data is its analysis to derive useful information. This process is known as data mining whereby large data is extracted and processed, to generate new information for the purpose of making organisational decisions (Kantardzic cited Beynon-Davies 2004). This process can be used by various organisations in various sectors. According to Beynon-Davies (2004, p.548) some of the use cases for data mining can be the following:

- In the retail industry to determine the purchasing patterns of customers based on class, age and demographics. To determine what to sell, and where to sell it.
- In the insurance industry to determine pricing regime, by basing it on claims information
- In the finance industry to detect fraudulent transactions

1.1 Background to research problem

In this study the research problem stems from an ethical problem. The ethical problem can be described as follows. The problem involves three parties user X, organisation A and organisation B. User X uses services from organisation A, that happens to provide a social network service. Every individual who uses services from organisation A has the following scenarios true for them:

- Have read the notice and consent form (whether they actually read them or not the law assumes they have if they have been given a notice), and have accepted the terms and conditions.
- Have surrendered some of their personal information over to organisation A.

Going forward in this research user X would be referred to as a data subject, organisation A as a data controller, and organisation B as a third party. This scenario is quite common for social networking services like Facebook, Twitter and LinkedIn. A notice and consent option is what data controllers present to data subjects as an agreement between them. This agreement will inform the data subject of the terms and conditions of the service and what the data controller will do with the data subject's data.

Usually this notice and consent needs to be accepted in order to use the service provided by a data controller. Most data subjects don't simply read the notice and consent documents, and even if they were to read them It would take them a significant amount of time, if they started to make it an effort to read the various notice and consent forms they encounter (McDonald & Cranor 2008).

The nature of these kind of transactions becomes problematic when the data controller shares the data subject's information with third parties, as the data subject is excluded and might not even know that his data has been subjected to secondary usage. Later on this study the problematic nature of these transactions will be unpacked, with the aid of privacy theories from Solove and Nissenbaum.

1.2 Research Fields

This section gives a brief description of the fields of study. Those fields are Information Systems and Software Engineering.

1.2.1 Information Systems

Information Systems are socio-technical systems that are designed to collect, process, store and distribute information (Piccoli 2012). Further more Picolli (2012) defines Information Systems as either formal or informal. A formal Information Systems would be those systems that are designed specifically for an organisation, an organisation being an NGO, company or government department. Whereas examples of informal Information Systems are social networking sites like Facebook. For this study both types of Information Systems, informal and formal, are of importance.

1.2.2 Software Engineering

Software Engineering is a discipline that is concerned with the production of software, from the early stages of requirements gathering, to the maintenance phase of a system (Sommerville 2010, p.7). In this study there is a specific focus on the software construction aspects of Software Engineering. Specifically on middleware technologies.

1.3 Research problem

The purpose of middleware is to provide developers with higher level abstractions. The current crop of main stream middleware do not seem to provide a higher level abstraction that provide a standard and easier way to develop applications that mitigate against the conflict of interest between data sharing and data privacy.

1.3.1 Research question

1.3.1.1 Main Question

How can middleware provide developers with a platform that enables them to write privacy enhancing software faster and that can allow people to have more control and\or visibility over their data?

1.3.1.2 Sub Questions

1.How can technology enable people to have more control or visibility over their data?

2.How can middleware be extended to provide a privacy enhancing platform to developers?

3.How can a privacy enhancing API be designed?

1.4 Objectives and aims

The objectives and aims of this study is to provide developers with a privacy enhancing platform that enables easier development of privacy enhancing software, that eases the conflict of interest between data sharing and data privacy. Those objectives can be summarised as follows:

- Provide developers with a privacy enhancing API
- Extend middleware to cater for this API extension

1.5 Delineations

The scope of this study will be limited to the extension of Java based middleware specifically Apache Tomcat.

1.6 Outline

The work in this study is organised as follows:

Chapter 2, Data sharing and data privacy – This chapter will discuss the conflict of interest that may arise with the sharing of data subject's information by data controllers.

Chapter 3, Enterprise software development – This chapter highlights the process with which software for the enterprise is developed.

Chapter 4, Methodology – This chapter discusses the research methodology taken for this study which is Design Science

Chapter 5, Related work – This chapter discusses work that is related to this study, and can serve as design anchors

Chapter 6, Design – This chapter will detail the design decisions that went into constructing the artefact

Chapter 7, Discussion of results – This chapter is a discussion of the artefact that was built and the results achieved

Chapter 8, Conclusion – Conclusion to the study.

Chapter 2.

Data privacy and data sharing

This chapter will lay the foundation of what data privacy and data sharing is, in the context of this study. Through the laying of this foundation key anchors on the definition of privacy by Solove and Nissenbaum will be discussed. Then the impact of data sharing on data controllers and finally what is being done in the field regarding the improvement of the conflict between data sharing and data privacy.

2.1 Data Privacy

According to Wester (1987) privacy is the right of a person to be left alone. However as discussed by many others (Solove 2009; Solove 2002; Nissenbaum 2010) privacy is a difficult concept to define as it is different across cultures, and even within the same culture it changes with passing time (Solove 2009, pg. 50-66). Due to the fluidity of the concept Solove and Nissenbaum's theorisation of privacy will serve as anchors for this study in terms of providing a definition of what privacy is. An explanation of Solove (2009) and Nissenbaum (2010) is provided below.

Solove (2009; 2006) defines privacy as a multi-dimensional concept instead of a singular concept, as he feels that all other attempts to define privacy as a singular concept tend to be narrow and not widely applicable to most cases of privacy (Solove 2002, pg 4-5). The guiding principles by which privacy is defined as a multi-dimensional concept by Solove (2009;2002) is by the family of resemblances theory by Wittgenstein (1958, pg 66-67), which suggests that certain concepts might not have a single characteristic instead they draw from a common pool of similar elements.

Solove (2009, pg 102; 2006) outlines a taxonomy of privacy as can be seen by the image below found in Solove (2009, pg.104; 2006, pg.15). In this classification of the different facets of privacy what is of importance is various principles in the information processing and information dissemination group. However an outline of all the groups and sub-groups is provided below

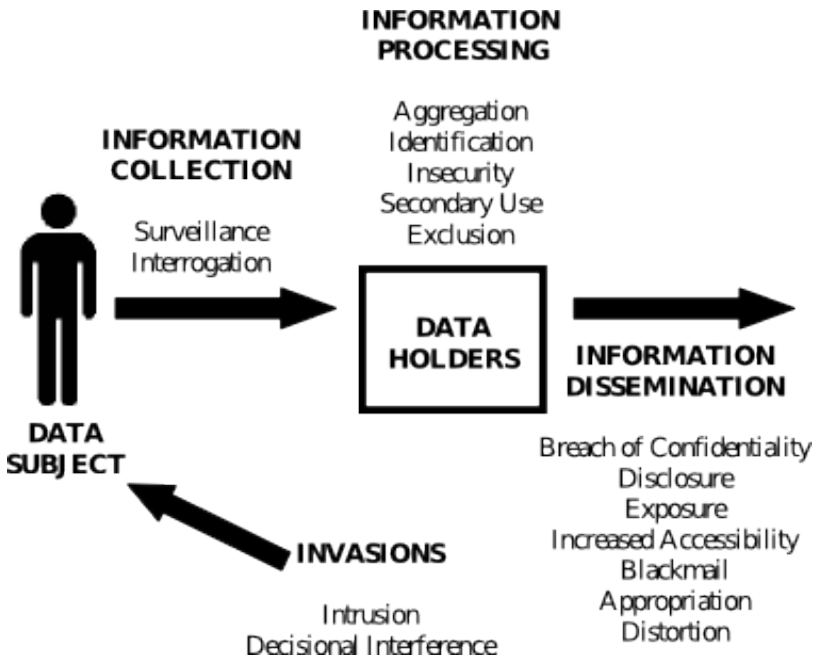

Figure 2.1: Privacy taxanomy by Solove (2009, pg 104)

The are four main groups that Solove (2009; 2006) categorises privacy into, they are;

- Information Collection
- Information Processing
- Information Dissemination
- Invasion

Information collection relates to how a person's information is collected, and this group is further split into surveillance, which is the watching or recording of a person's activities and interrogation, consists of various forms of probing and questioning, with varying degrees of coercion.

In Internet terms Google can qualify as undertaking surveillance activities whereby a data subject's search activities are recorded, and used to determine what content to advertise to them. This form of advertising is referred to as behavioural targeting (Penn 2011). Other big companies like Facebook are using data subject's recorded information to provide targeted advertisments to them, of which targeted advertisements bring in about 85 percent of Facebook's revenue (Andrews 2012). There are dedicated behavioural targeting platforms like Adroll, Retargeter and Fetchback.

The second group information processing is all about how a person's data is stored, processed and used. This group has five sub groups and they are

- Aggregation
- Identification
- Insecurity
- Secondary use-cases
- Exclusion

Aggregation is how different bits of information about a person is aggregated to form new kind of information about a person. An example of this is intelligence agencies aggregating information about a person from various sources from their telephone, banking, traffic and health records to derive new information. In this case to model their threat level to being involved in terrorist activities(Ackerman & Ball, 2013; De Goede, 2014).

Identification is linking information back to individuals to reveal their true identity. An example is the Truecaller mobile application that aims to have global telephone directory of everyone. The mechanisms by which Truecaller achieves this is by uploading the contact details of everyone on the application user's mobile. So one of the terms of having Truecaller installed on a mobile device, is to agree to hand over your contact list over to Truecaller (Truecaller.com n.d).

Insecurity is the carelessness in the handling of a person's data to guard against loss, theft and other malicious activities from 3rd parties. Sony has been guilty of this with the hacking incident of the Playstation network where many people's credit card details where stolen (Pepitone 2011). Secondary use is the usage of a person's data for another type of usage than the one originally intended for as companies cannot possibly know all the use cases of the data they collect. Finally exclusion is about failing to inform the user about other parties that may have their data.

The third group information dissemination is about how a person's information is being spread by the data controller. This group contains seven sub groups and they are;

- Breach of Confidentiality
- Disclosure
- Exposure
- Increased Accessibility
- Blackmail
- Appropriation
- Distortion

Breach of confidentiality is the breaking of a promise to keep a person's information private. Disclosure is the revealing of information that impacts the way other people see or judge a person's character. Exposure involves revealing another person's nudity or bodily functions. Increased accessibility is increasing the accessibility of a person's information which is what Truecaller does by making people's numbers readily available by aggregating them from there mobile application user base(Martienssen 2016).

Blackmail is the threat of disclosure of private information. Appropriation is the use of a person's identity to server the needs of another and finally distortion is the dissemination of false or misleading information about a person. The fourth and final group is invasion which is the invasion of people's private affairs and this group can be split into two sub groups. The first of these sub groups is intrusion which involve invasive actions which would disturb a person's peace. Lastly decisional interference involves the government's unsolicited involvement in a person's decisions regarding their private affairs(LAWRENCE ET AL. v. TEXAS, [2003]).

Nissenbaum (2009) views privacy as the flow of information that is bound by context, that is to say the privacy of information depends on the context by which that information is being shared. This theory Nissenbaum (2009) refers to as contextual information flow, and it is considering the context in which a person's information is being shared. Anyone would freely give out and discuss their health information with a health practitioner. However this very same information would be considered private for any other purpose, like being freely shared at their place of employment or being placed on an advertisement board.

Nissenbaum and Solove's core problem is how a data subject's data is handled by organisations. Though in some instances it may not be illegal the data subject may not appreciate how their data is being handled, like in the case of aggregation or secondary usage where there data is siphoned off to third parties which they would not agree with had they been given the choice.

When a data controller shares a data subject's data with third parties, this act may violate one of the privacy principles by Solove and Nissenbaum. As mentioned earlier the important groups in Solove's privacy taxonomy is the information processing and information dissemination groups. There implications in the data sharing process between data controller and third parties, is to be discussed and Nissenbaum's contextual privacy.

When analysing aggregation from the information processing group, it is not clear what bits of information may be held by a third party about a data subject, from which it would have obtained the data from various data controllers. A hypothetical example of this would be a butcher than obtains data from the bank, grocery store, and Google maps. Then proceeds to use this information to identify customers who buy a lot of meat from the grocery store, and have a high enough threshold of extra money to buy items at a higher price, and stay a long way from the grocery store. To warrant building a butchery close enough to them.

Some data subjects would welcome this as the butcher would make their lives easier, however others may feel uneasy with the butcher knowing the size of their pocket. On that chain of thought this speaks to identification and contextual privacy where a data subject may not want to certain information about them known by certain data controllers. Regarding secondary usage and exclusion, a data subject is generally not informed when his data is shared by the data controller, and the third party may use this data for what it was not originally collected for in the first place.

What can be discussed in the information dissemination group is disclosure and increased accessibility. Through supplementing the data they already have, Target which is a retail store in North America, was able to predict the pregnancy of their female customers. This had the unintended consequence of disclosing to a father of teenage daughter, that his daughter is pregnant before she had the chance to tell him (Hill 2012). With increased accessibility telemarketers get data subject's contact details from various data controllers and cold sell them products.

There are proposed laws that address the issues raised by Solove and Nissenbaum. The laws in question are the General Data Protection Regulation 2012 and the Protection of Personal Information Act 2013. The former is for the EU and the latter is for South Africa, which was inspired by the predecessor of the General Data Protection Regulation 2012 the EU Data Protection Directive.

The Protection of Personal Information Act 2013 has been signed into law on the 27th of November 2013 however only certain parts of the act are effective (Saica 2015; Kpmg.com n.d). The EU General Data Protection Regulation 2012 is scheduled to come into effect December 2017 (Rossi 2015). Some of the items both laws address is that data subjects should be able to revoke consent on the processing of their data, and the data controllers are not allowed to use the data subjects's data for any other purpose than the ones specified.

Data controllers currently deal with legal or ethical issues regarding data subject's data, with a mechanism called notice and consent. The fundamental idea with notice and consent is to notify a data subject of the terms and conditions, so that the data subject is aware of how the data controller intends to use their data, and also to comply with certain legal requirements. This is presumably ideal for data controllers since courts of law are of the view that when someone has been presented with a notice it is assumed the data subject has read the terms and condition even if they did not (Perillo et.al 1993 cited in Sloan & Warner, 2013).

The second part is the consent part whereby a user is presented with a take it or leave it scenario, were for them to proceed to use a service provided by the data controller, they have to agree to the terms and conditions of the data controller regarding the usage of their data. This of course as pointed out by McDonald & Cranor (2008) is infeasible for the average person to read each and every terms and conditions policy when accessing services online, and they reckon that it would take the average American 201 hours to read each and every privacy policy they encounter.

Graber (2002 cited by Cranor et.al, 2013) is of the view most people would not be even able to comprehend the details of the privacy policies. To compound matters Ur et.al (2013 cited by Cranor et.al, 2013) points out that though some data controllers provide multi-lingual interfaces they sometimes do not have accompanying terms and conditions in those target languages. So there is a lot of issues with notice and consent mechanisms from the infeasibility of reading them all to the comprehension required to understand them, and not being available in all supported languages that a data controller may provide on their online service.

Later on in the related works chapter the work by Birrell & Schneider (2014) will be discussed which aims to offer a solution that attempts to improve notice and consent mechanisms. The next section will discuss data sharing between data controllers and third parties, and the benefits both parties derive from the exercise.

2.2 Data sharing

Analysing customer or user data has been proven to be beneficial in understanding customer behaviour and generating new revenue streams (Tsiptsis & Chorianopoulos, 2010). Organisations like Amazon use customers shopping behaviour as a feeder into their recommendation engines, to recommend items to customers with similar tastes (Alag 2008). Having said that more information from external sources about current customers can improve the quality of data analysis.

U.P.M.C which is a non-profit organisation in western Pennsylvania in the USA that owns multiple hospitals with 12 billion dollars in profit for the 2015 fiscal year (Beckwith & Romoff 2015, pg.29). Has been supplementing their predictive forecasting information system, which has been predicting the health of their clients as to when they may have certain health problems, with supplementary data from Acxiom (Singer2014). Acxiom is an organisation that sells customer data that it obtains from publicly available sources (Singer 2012). Acxiom has been supplying U.P.M.C with supplementary information regarding their customer's online buying patterns and their food purchases.

Other data sharing examples is Google partnering up with Twitter to get access to instantaneous tweets through the Twitter firehose, which is an API that supplies streaming tweet data to third parties (Griffin 2015; Patel 2015). This supplementary data coming in from Twitter can come in various forms that Google can dictate, for example Google may be interested in filtering the streaming tweets by geographic location, that is the only data coming to them from the Twitter firehose is tweets about location specific events.

In that way Google is able to increase the precision of search queries pertaining to geographic locations. Search queries for London could be supplemented by data from Twitter about London to provide more detailed query results for London for example. These few cases highlight the importance of data to data controllers and the amount of money data about data subjects can contribute to the bottom line. More so that various studies have been carried out on the efficient mechanisms of selling data (Babaioff,2012; Li et.al, 2013; Ruiming et.al, 2013).

2.3 Enhancing data privacy

There are various approaches taken in the field that attempts to solve various forms of privacy issues in the computing sector. The approaches to be discussed are Privacy by Design and PET which stands for Privacy Enhancing Technologies.

Privacy by Design as outlined by Cavoukian (2010) is a set of seven principles aimed at embedding privacy concerns into every aspect of software development, such that the resulting software that is produced from such principles addresses privacy concerns. These ideas are espoused in article 23 of the General Data Protection Regulation 2013.The seven principles are

- Proactive not Reactive; Preventative not Remedial
- Privacy as the Default Setting
- Privacy Embedded into Design
- Full Functionality – Positive-Sum, not Zero-Sum
- End-to-End Security – Full Lifecycle Protection
- Visibility and Transparency – Keep it Open
- Respect for User Privacy – Keep it User-Centric

These seven principles are fashioned more like guidelines as they do not give specifics on how certain privacy problems should be tackled. As a result of this Privacy by Design can be interpreted as being vague and the implementer is required to retrofit and theorise the missing parts of the Privacy by Design framework into their specific use cases. Others have done this theorising by extending Privacy by Design to provide 8 strategies that can be used in the early stages of software development starting from the requirements gathering phase (Hoepman 2012).

Further more Gürses (2011) provides an approach which starts off with data minimization that outlines how to apply privacy by design in a project. This approach is supplemented by a case study on a E-Petition system. All in all even with inputs from Gürses (2011) and Hoepman (2012) PbD still lacks the specifics on how to actually implement it, therefore the uptake of the framework is estimated to be low.

Privacy Enhancing Technologies are set of technological solutions aimed at enhancing privacy for users. Various authors define PETs differently. Van Blarkom (2003) defines PET as a coherent set of ICT systems that protect data privacy by either reducing or eliminating the collection of personal information, or by preventing unnecessary processing of personal information, while maintaining the original function of an information system.

Diaz (2013) provides a more comprehensive definition, which builds upon Rubinstein (2011 cited in Diaz 2013). Rubinstein splits PET into three main categories which are listed and described below;

- Substitute PET – block or minimise data collection
- Complimentary privacy-friendly PETs – enhances notice and choice mechanisms
- Complimentary privacy-preserving PETs – Enables ad-targeting without allowing an ad network to track the user

From these three categories Diaz (2013) provides a definition for PET as heavily leaning on the first category, with some elements on the 3rd category. Further more Diaz (2013) gives various scenarios where PET are used to give a clearer picture of what constitutes PET, and in those scenarios elaborates on how a technology can be considered a PET and non-PET simply based on the context in which it is used.

Just like Diaz (2013) provides scenarios where PET are applicable, real life examples of PET will be shortly discussed. The first example is Tor which is pitched as a second generation Onion Routing system, that allows Internet users to browse the Internet anonymously (Dingledine et.al 2004). Dingledine et.al (2004) describes Tor as a low-latency anonymity system that requires no kernel modifications or special system privileges. It works on TCP based applications and connects a user to the Internet through a network Tor nodes which do most of the work in preserving the anonymity of Internet users.

The second example is P3P. According to Cranor et.al (2006) P3P short for Platform for Privacy Preferences Project, is a protocol that allows websites to specify through an XML based DSL what private information they will be collecting about their users. In turn the users can configure their own P3P configuration through a browser based GUI, Microsoft Internet Explorer is the only browser to have full support (Richmond 2010) and Firefox had taken the decision to remove it (Connor 2007). Then whenever a website loads on the user's machine where both parties have P3P XML configurations. Notes would be compared between the two parties. The website configuration will state the data it is interested in, and the user configuration will state the data that it is not willing to share, and the data it's willing to share the terms under which this should be done.

The aims of this study is to provide a privacy enhancing API to developers, to make the development of privacy enhanced software faster, and ultimately allow data subjects to have more control\visibility over the sharing of their data with third parties. The artefact to be produced by this study can be classified as a privacy enhancing technology, that satisfies the first and second categories of Rubinstein's (2011 cited in Diaz 2013) definition of PET. The next section will describe how enterprise software is developed from the tools, to the process and architectures with the aim of giving an idea of how such software that shares data subjects details are built.

Chapter 3.

Enterprise Software Development

This chapter will describe how formal and informal information systems as defined by Picolli (2012) are developed. The chapter will start of with an outline of the tools that are used and form part of the final system, following that the SDLC which provides a planning and execution framework for completing software projects. Finally the different architectures that can be utilised in the construction of an information system.

3.1 Enabling technologies

3.1.1 Relational databases

These are based on the relational model by (Codd E.F 1983). This class of database systems represent the industry standard. They are used in conjunction with structured query language SQL. There are a number of database products, the popular ones being Oracle, Microsoft SQL Server and DB2. Open source one's being MySQL, PostgreSQL, ApacheDB and FireBird

3.1.2 NoSQL databases

The term NoSQL databases refers to a class of databases that do not subscribe to the relational model (Codd E.F 1983). These database systems vary in their architectural structure, they range from document stores, key-value stores, column data stores, graph databases amongst others (Strauch 2011). NoSQL database have made headway into a number of organisation (MongoDB n.d). To give a few notable examples MongoDB (n.d) states that companies like SAP, MTV, Disnep, EA amongst others use their MongoDB NoSQL databases product.

3.1.3 NewSQL databases

Stonebraker et.al (2007) bemoaned the architectural staleness of modern relational database systems, and argues that most modern relational databases have retained the architectural structure of the 1970s, which were built during an era of limited memory and processing power. Throughout the years, the architecture was never really rethought, the vendors just simply added on to the old architecture (Stonebraker 2010).

As a result of this Stonebreaker proposed and developed a database system called VoltDB (Stonebraker & Weisberg 2013) which grew out of the H-Store research project (Stonebraker et.al 2007), which attempts to pave the way in redefining the general architectural approach of modern database systems. This attempt is characterised by the following characteristics:

- No disks or other persistent storage at all
- No multi-threading
- No locks
- No redo logs (Monash 2008).

These are some the changes that Stonebraker et.al (2013) advocates in terms of the implementation approaches taken towards the architectural structure of VoltDB. VoltDB is an in-memory database system optimised for OLTP workloads. VoltDB belongs to a class of relational databases called NewSQL databases, that aim to have the same performance characteristics as NoSQL databases while retaining ACID compliance that relational databases have (Stonebraker 2011). Other notable NewSQL systems are Google Spanner (Corbett et.al 2013), MemSQL (memsql.com 2015) and SAP HANA (Färber et.al 2012) amongst others.

3.1.4 Application Server

Application servers provide a platform for applications, that abstracts away underlying complexities. An example of such complexities would be things like resource management, in the sense that the application developer no longer needs to explicitly manage things like thread pools, database pools and so on. Such functionality is abstracted away and all the developer needs to know, is the API which enables them to utilise such features.

3.1.5 ESB

ESB is short for enterprise service bus. The words service bus are derived from the system bus or hardware bus in computer hardware systems. A System bus or Hardware bus is a communication pathway that connects two or more devices, whereby the devices are able to send messages to and receive messages from each other (Stallings 2010, pg 85).

Enterprise Service Bus (ESB) is a software system that is primarily concerned with providing an integration mechanism, for heterogeneous applications in an organisation or beyond the organisation (Mori 2014). This point is further clarified by Barry & Dick (2012, pg 62) by expanding on the usage scenarios for an ESB. This scenario used is whereby a data controller has information about a data subject, and that information needs to be updated in several heterogeneous systems.

This scenario presents a problem that vividly highlights the set of functionalities that an ESB brings to the table that is to provide a mechanism to integrate disparate systems. Without the presence of an ESB system, mechanisms like point-to-point integration will have to be utilised, which basically involves having endpoints flowing from each system to the other. This of course is not an ideal situation. It is evident that with such an approach, the number of custom point-to-point connections required, have a quadratic growth rate.

Apache ServiceMix can be considered a real-life example of an ESB. In 2007 ServiceMix graduated as a top-level Apache project. In the few years preceding that event the core developers of ServiceMix decided to extract the ServiceMix kernel, and use that kernel as its own project. This new project became known as the Apache karaf project. Apache karaf provides extensions to existing OSGi frameworks. Those frameworks are Eclipse Equinox and Apache Felix. In theory ServiceMix is an amalgamation of the following projects. These are used in order to form the singular software entity known as ServiceMix:

Apache Karaf

Apache Camel

Apache ActiveMQ

Apache CXF

3.2 SDLC

This section will discuss the SDLC, short for Software Development Life Cycle, and its two models, namely Waterfall and Scrum. The SDLC represents a series of steps that need to be taken in a software project in order to carry out a successful completion of the project. There are various definitions of what constitutes those steps.

Foster (2014) states that there are five key steps.

- Investigation and analysis
- Design (Modeling)
- Development (Construction)
- Implementation
- Management

However Dennis et.al (2012) consider the SDLC to contain four key phases, and they are planning, analysis, design and implementation. The first three steps of Foster (2014) correspond to the four phases described in Dennis et.al (2012). The 3rd and 4th phases in Foster (2014) are somewhat confusing as they seem to suggest they are one and the same, and there is no adequate explanation that clearly differentiates between them. For that reason Foster (2014)'s five phase definition will be preferred. The Waterfall model is pictured below:

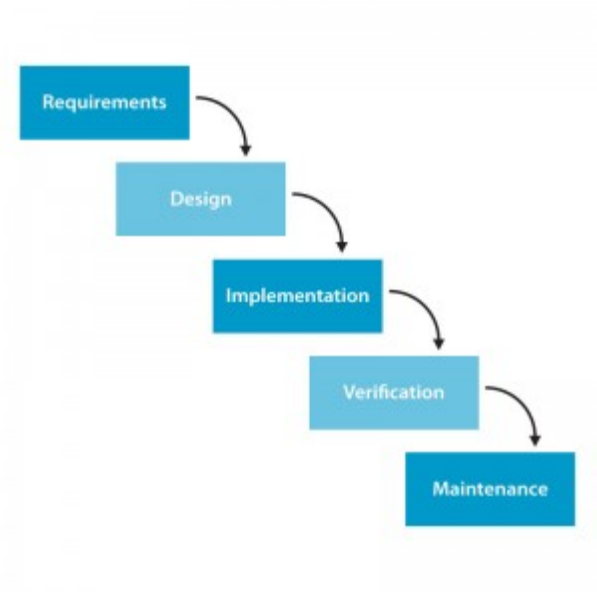

Figure 3.1: Waterfall method

As the diagram suggests the SDLC is represented as a process that flows from one phase to the next downstream, with the specific intent that going from one phase to the other is an irreversible process (Royce 1970). Royce (1970) states that the Waterfall process is well suited for projects where the requirements are well known upfront, and that one of the benefits of the process is that it produces a lot of documentation.

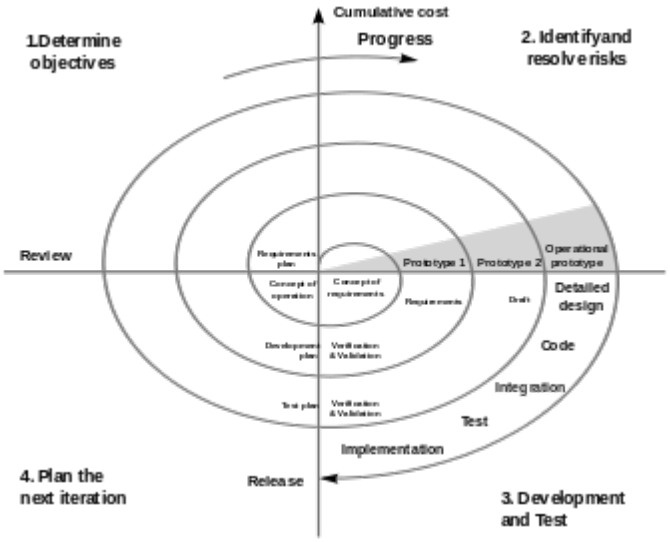

Figure 3.2: Spiral model

The diagram above represents the Spiral model which advocates for the construction of software in incremental steps whereby each increment goes through each of the quadrants in the diagram. The first quadrant is to determine the objectives of the iteration that is to be embarked on, the second quadrant is about identifying any risk. The third quadrant is about the development and testing of software components particular to the requirements of the iteration, then finally the last step is about planning the next iteration (Boehm 1988).

The third model is Scrum which is a process that forms part of the so called Agile methods. Schwaber (2004) and Rubin (2012) define Scrum as an iterative process which unlike the Waterfall method iterates through the various SDLC phases in one iteration and builds the system in incremental stages, somewhat similar to the Spiral model. However in Scrum the incremental stages are referred to as sprints.

The difference between the Spiral model and Scrum are the roles and ceremonial nature of Scrum. Scrum has distinctive roles that need to be filled in a project, these are the Scrum Master and the product owner. As can be seen in the diagram below the product owner is responsible for maintaining the product backlog which is a list of stories that need to be done. At the beginning of each sprint stories are taken from the product backlog and placed into the sprint backlog, and that is the work that will be carried out for that particular sprint. The duration of the sprint is recommended to be no more than 2 weeks, however various teams set their own standards.

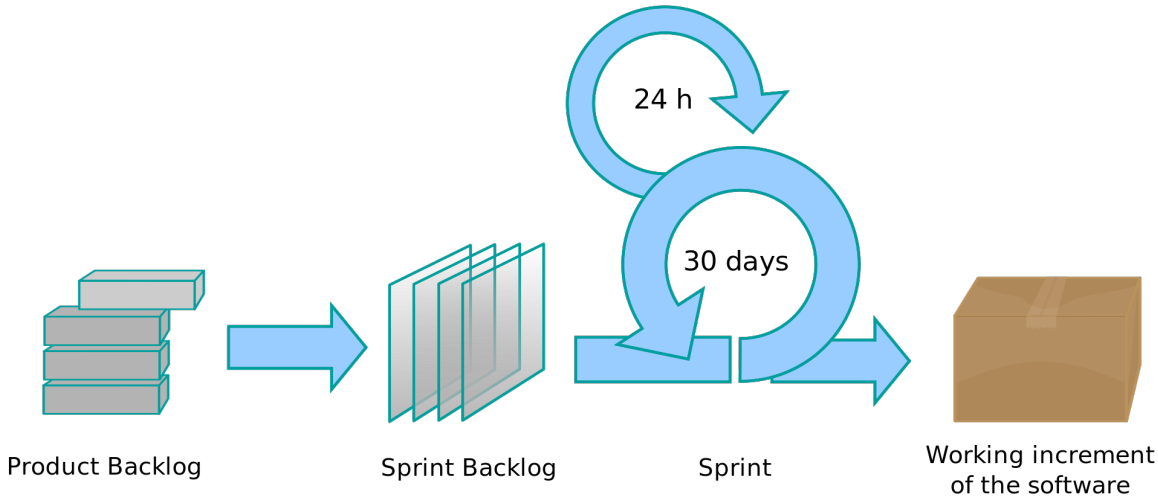

Figure 3.3: Scrum methodology

To retrofit the privacy by design principles outlined by Cavoukian (2010), on the various software development processes, would place additional burdens on project teams. Teams following the Waterfall process would most likely suffer the most given that Waterfall flows down and does not allow for going back a step in the process model. This would mean that the privacy requirements would have to be fully fleshed out in detail at the beginning. This might prove difficult to achieve given the fluidity of the definition of privacy (Solove 2009; Nissenbaum 2009).

The General Data Protection Regulation 2012 in Article 23 requires that data controllers prove that they do, somehow include privacy as a requirement in the construction of software, so the privacy by design guidelines will become more and more invaluable. The outcomes of this study will be beneficial to data controllers in helping them deal with the costs that may arise in the need to cater for privacy. The next sections will discuss the architectures employed in the construction of software.

3.3 Software Architecture

Bass et.al (2013) define software architecture as a set of structures that are necessary to reason about the system. With that in mind this section will detail software architectures that can be employed during the design and development phases of the SDLC phase as defined by Foster (2014). The architectures that will be discussed are the multi-tier architectures and CQRS.

3.2.2 Multi-tier architectures

Multi-tier architectures are based on a universal concept of layering found through out the computing body of knowledge. Layering is the concept of breaking up a complex system into discrete units called layers, whereby the layers will build on top of each other such that each proceeding layer will abstract the complexities of their respective domains and provide a simplified interface to upper layers (Fowler 2002).

A practical example of this is TCP which is a communication protocol that guarantees to send data reliably, which sits on top of another protocol called IP, which sends data unreliably (Fall & Stevens 2011). TCP has multiple layers, and each layer abstracts away certain complexities such that the proceeding layer can utilise the preceding layer seamlessly without worrying about the underlying complexities.

Sometimes this complexity that is intended to be abstracted away ends up propagating upwards into higher layers, an example would be an unplugged network cable which would result in the network messages not being sent, and TCP protocol being rendered unreliable in that instance. These phenomena are known as leaky abstractions and are discussed by Spolsky (2002). Figure 3.4 below shows a visual representation of the multi-tier architecture.

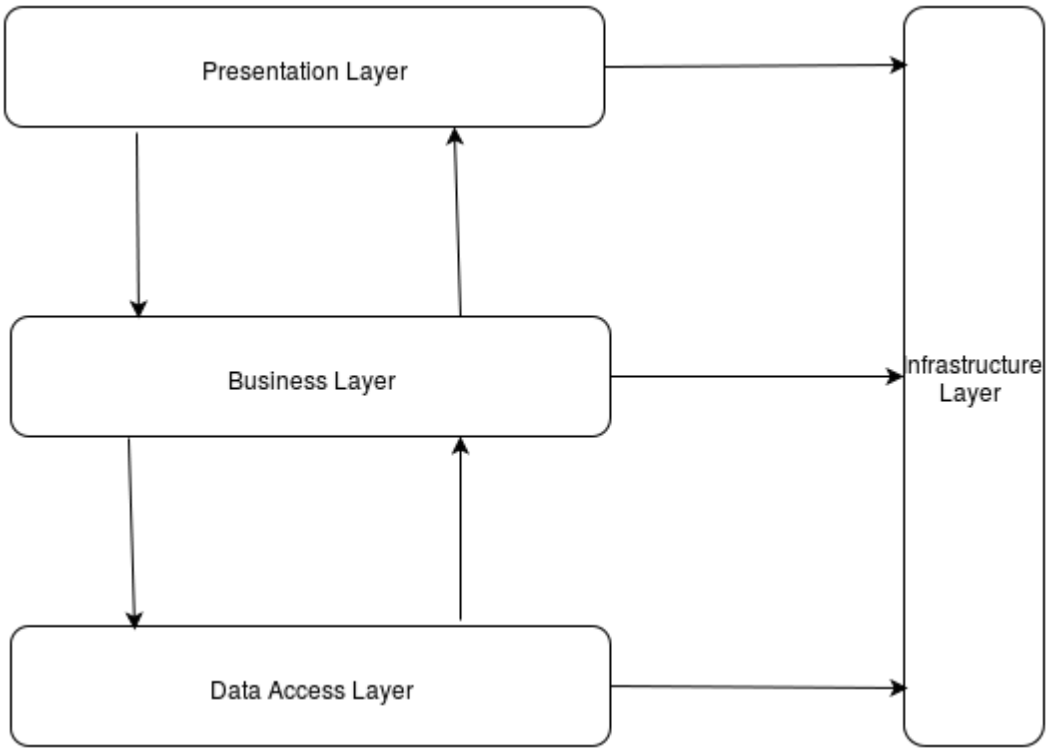

Figure 3.4: Graphical depiction of a multi-tiered architecture

The different parts of the diagram can be explained as follows:

- Presentation Layer: This layer is concerned with the code that handles the user interface.
- Business Layer: This layer is concerned with the business specific logic, that may be found in an application.
- Data Access Layer: This layer is concerned with code that deals with how data is accessed in the database, and any database specific method calls.
- Infrastructure Layer: This layer represents the platform and computing infrastructure that the application is running atop.

One of the ways that the multi-tier architecture can be implemented in Java is through the use of the spring framework. According to the creator of the spring framework, Rod Johnson, Spring was created to address the complexity of Java Enterprise development (Johnson cited in Walls 2011).The spring framework at its core is an IoC container or inversion of control in full. IoC is a design pattern which states that a class should not have direct access to the creation and management of its dependencies (Fowler 2005). Instead an outside entity should be the one in control of a class's dependencies.

In this way a class is not tightly coupled to its dependencies, in the sense that through IoC a class will be given its dependencies, and this situation allows for more testable code, since the dependencies of a class would not be managed by a class, they would be managed by the IOC container. The spring framework today contains a set of sub-projects, which address different areas of Enterprise development. Below is description of some of the projects.

Table 3.1: Spring framework sub projects

Project	Description
Spring Data JPA	Abstractions for dealing with JPA libraries through Spring
Spring Integration	Implementation of Enterprise integration patterns from (Hohpe & Woolf 2004)
Spring Security	Abstraction of the JAAS security framework and other security extensions (Jagielski and Nabrdalik, 2013).
Spring LDAP	Abstractions for dealing with LDAP access in a spring based application (Varanasi 2015)
Spring Batch	Batch processing framework for dealing with batch data in spring (Rao 2015)
Spring MVC	MVC framework for writing user interfaces (G 2014)

Spring can be used to implement the multi-tier architecture in the following manner. The **presentation layer** can use the Spring MVC sub-project to implement the user interface. However the developer is not only limited to Spring MVC, they can also use other UI frameworks like Vaadin (Holaň and Kvasnovský, 2013), GWT (Tacy 2013), IceFaces (Eschen 2009) amongst others.

The **data access layer** would use the Spring Data JPA project which provides an ORM API over an ORM framework like Hibernate, OpenJPA or EclipseLink (Keith & Schincariol 2013; Ottinger, Guruzu and Mak 2015; Pollack 2012). In addition Spring Data can be used on NoSQL databases like MongoDB, Neo4J and Hbase (Pollock 2012). The business and infrastructure layer would be left to the developer to implement.

3.2.3 CQRS

CQRS is an extension by Young (2010) of the CQS pattern by Meyer (1988). The central idea behind CQS is for methods that query for information not to change any state, and that any method that changes state should not return any results. Young (2010) extended this pattern by advocating for the separation of the command and query responsibilities into separate domains. That is the domain model for queries and commands should be different, in that way they can each be optimised differently according to their respective needs. The query part of the system can be configured to have increased memory and SSD for better read performance, and the command part of the system can be optimised for high write throughput.

Rajković et.al (2013) demonstrated this command and query optimisation, in the refactoring of a medical information system. this Medical information system initially had 384 database tables which needed to use data that was scattered in various data tables. This need resulted in many joins to retrieve the relevant data and caused performance problems. After the application of the CQRS pattern in some instances of system usage there was a performance increase of about 40%. Since in this particular case it was not clear which database system was in use, the performance of such separations could have possibly reached even higher performance figures by utilising specialised relational database systems like VoltDB, as is described by Stonebraker & Weisberg (2013).

In-memory databases can be used primarily as query databases in the CQRS pattern, this though can pose the issue of data loss since the data is maintained in volatile storage. This can simply be averted by using VoltDB's multi node data replication (Stonebraker & Weisberg 2013), in conjunction with event sourcing to replay command events to restore the state of the database (Betts et.al 2013, pg.118). Event sourcing is a mechanism whereby all command events that take place in the system are recorded such that replaying them will restore the state of the database to what it is supposed to be (Young 2010). CQRS at best is a software development framework and can be implemented without any frameworks, fortunately there is a Java framework called the axon framework that developers can use (Axonframework.org n.d).

This chapter gave an outline of the SDLC processes used in the development of software in particular the Waterfall model and Scrum. Then the different architectural approaches used in the construction of enterprise software systems were noted, with descriptive information about the various software components that can form part of the final enterprise software system. It is intended that the background given in this chapter and the previous one on privacy will serve as a firm foundation for the coming chapters.

In the next chapter studies related to this study will be discussed. These studies differ from each other, but this is to be expected given that the topic under consideration fuses together several very different topics. It is hoped that these studies will provide as input towards coming chapters that will discuss the design of the artefact.

Chapter 4.

Related Work

The related work chapter serves to place the study in context in relation to the larger body of knowledge. Randolph (2009) provides methodological counsel on conducting a literature review, which is based on Cooper (1984, cited in Randolph 2009). Randolph (2009) provides a matrix that adequately maps the different steps in conducting a literature review as described by Cooper (1984, cited in Randolph 2009). This matrix serves as a guideline towards conducting the literature review. The steps are as follows;

1. Problem formulation
2. Data collection
3. Data evaluation
4. Analysis and interpretation
5. Public presentation

From these steps the first four were chosen. As the public presentation is somewhat irrelevant, as the work for the literature review is not for public presentation, but merely to serve as a chapter in this thesis on related work. The coming sections in this chapter will be expansions on the aforementioned steps, whereby the rationale for each step will be given and followed by the results for that particular step.

4.1 Problem Formulation

The overall aim of the study is to ease the conflict of interest between data sharing and data privacy. With that being said, this literature review can be best characterized as a systemic literature review with a methodological framework from Randolph (2009).The primary focus will be on the design propositions brought forward by the related studies, with the intention of extracting some of these design propositions, and incorporating them into the current study.

The guiding research questions are listed below for this literature review, and are based on the sub-questions of this study:

Table 4.1: Guiding research questions for the literature review

Questions
• How can technology enable people to have more control over or visibility of their data?
• How are developer middleware/platforms expanded
• How can a privacy enhancing API be designed

This chapter will be structured around addressing these questions. Firstly the focus will be on the various technologies in the field, that aim at improving the control or visibility of user data. Secondly, when considering how middleware used by developers in the construction of information systems is extended, the focus will be on Java middleware. Finally there will be a look at the state of the art in the design of developer API, with a focus on Domain Specific Languages.

4.2 Data Collection

The data was collected primarily from Google Scholar, and secondarily using major libraries. The technique employed in searching these sources was almost the same, and can be summarised as follows:

- Search for specific keywords pertaining to the literature review questions, the keywords are as follows: extending middleware, internal DSL, external DSL, language workbenches, privacy enhancing technologies, PET, privacy middleware, privacy technology
- Articles returned from the search query are then evaluated based on their title, abstract and content. The evaluation intends to find articles that have strong relevance to this study, and address the guiding research questions for the literature review.
- Of those articles that have been evaluated and subsequently choosen, their reference list is looked at for possible leads for more articles, of which those leads themselves are evaluated on the same basis as on the previous point.

The criterion used to keep or discard articles was their relevance to the research questions. From the articles collected certain themes presented themselves, which are listed below:

- Position in technological stacks
- State of research
- Domain Specific Languages

These three themes can be further described as follows: In terms of the 'position in technological stacks' theme, the core idea behind this is to discover where in the technological stack does a particular study find itself. In chapter 3 the multi tier architecture is explained, and it is described as having at three core layers that are responsible for the UI, where UI is the responsibility of the presentation layer. Then the business layer, in which the business logic resides. Then ultimately the data access layer. The infrastructure layer is not considered.

Having described the technical stack positioning, some of the literature either concentrated their effort on providing a technological solution that is aimed at improving the privacy by solely focusing on the data layer (Emekci et al 2005), or they focused on one of the other layers. Some studies provided solutions that were primarily focused on one layer, but touched on the other layers (Abour-tair 2006; Birell 2014; Anciaux 2012). The studies found in the literature that pertain to the first research question are viewed in relation to this framework pictured in figure 3.4, and it will be made explicit as to where exactly in the framework each study is situated.

The second theme relates to the state of the research in terms of whether it is just a proposal or there is an actual implementation of the proposed solution. Finally the third theme is concerned with addressing the last research question, which is about how the design of the privacy API can be approached. The constant theme that came up is that of Domain Specific Languages which will be explained in that respective section later. The next section will describe how the literature was evaluated, and then present the evaluation grouped by research question.

4.3 Data Evaluation

The major objective of the data evaluation is to extract the following from the papers, in order of importance;

- The artefact design
- The theories
- The methodology.

This section will discuss the design approaches taken in the literature and will be grouped by research question. Diagrams are provided as extra supplements to assist in the explanations of the design approach taken in the various studies. In addition for each diagram other related work may be discussed were relevant and contrasted not only with the diagrams in question but also with the broader aspects of the study.

4.3.1 Research Question 1:

The first research questions is about how technology can enable people to have more control over or visibility of their data. In addressing this question the studies will be grouped into three groups the presentation layer, application layer and the data layer. These three layers have the same names as the layers found in the n-tier architectures discussed in chapter 2. They also take on the same meaning and, for the purpose of this section, add on a little bit more. The relevance of each work/study to each layer will be assessed, and a consideration will be taken as to how each study relates to the developer and the users of the system.

The presentation layer will be represented by those studies that positioned most of their solutions on describing or the design of the user interface. That is, focus was on how the users, the people who give their information to systems, interact with a system that aims to provide them with more control over or visibility of the usage of their data. The application layer is for those systems that have positioned their system as a platform or framework, that can be incorporated into a larger information system that caters for privacy issues. Finally the data layer focuses on studies that aim to improve the privacy aspects of user information in information systems, through database extensions or add-ons.

4.3.1.1 Presentation Layer

Cranor et.al (2002) has put forward a P3P specification which aims to provide data subjects with an ability to specify their privacy preferences using their browser. Of the browsers in common use, Internet Explorer is the only browser with full support of P3P, Firefox has had the codebase associated with P3P removed (Connor 2007). The specified privacy preferences are stored by the in machine readable XML. This machine readable XML is used by the P3P engine in the browser to evaluate the data subject preferences against the website P3P policy XML. Each website can provide a P3P policy XML which is also in machine readable format which describes what data the website is going to collect about the data subject.

This P3P policy XML is programmed by the data controllers for their websites and it is not without problems, others have complained about its ambiguity and lack of guidelines for user agents (Schunter et.al 2002). In addition work by Leon et.al (2010) points to improper implementations of P3P by a majority of data controllers whose websites appear in the list of top websites that are implementing P3P.

Support for P3P has dwindled with Firefox removing it from their codebase (Connor 2007) and the only browser to support it, Internet Explorer, has it removed for version 11 of their browser on Windows 10 (msdn.microsoft.com n.d) and recommends website authors to not deploy P3P policies on their sites. This effectively is the death knell of P3P as there is now no browser that supports it.

4.3.1.2 Application Layer

The first diagram figure 4.1 is by Abour-tair (2006). Abour-tair (2006) proposed and implemented a privacy enhancing technology, that sits between the enterprise application's business logic, and the database. The implementation is a logical layer that is deployed within the target business logic as EJB's.

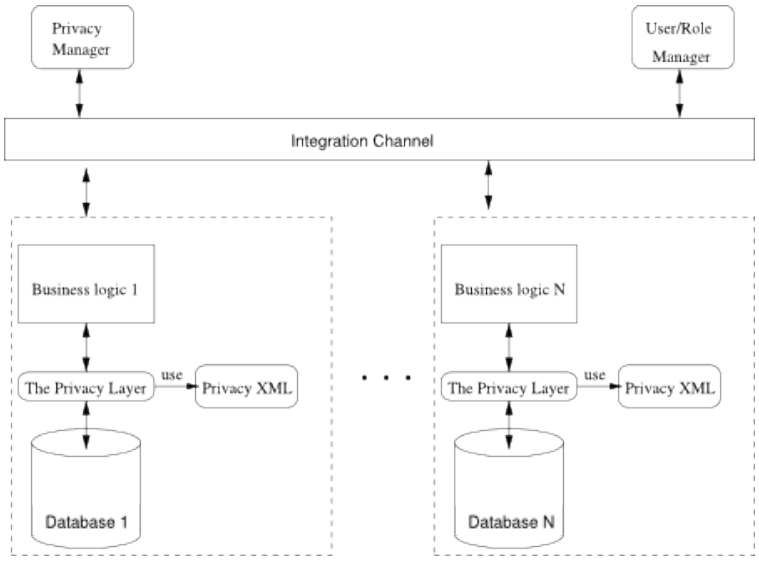

Figure 4.1: Privacy enhancing middleware by Abour-tair (2006)

The Privacy Layer acts as a regulator, of sorts, of the kind of data that can be accessed by the business logic. The Privacy Layer is guided by the Privacy XML, which is produced by the Privacy Manager. The Privacy Manager takes as input, Application ontology, and produces as output, Privacy XML. The inner workings of the Privacy Manager are as follows;

- Take Application ontology as input
- Combine Imported ontology and BDSG ontology
- Convert the combined ontologies into a Privacy XML

This process is deduced from the literature itself rather than from the second diagram that depicts the internal structure of the Privacy Manager. Ontologies form key components in the inner workings of the Privacy Manager. It is assumed that internally, the Privacy Manager does the automatic conversion of the ontologies into the Privacy XML.

It is unclear how this automatic conversion is achieved, since those details are not supplied. What is supplied is the information that OntoStudio was used to construct the custom ontology and the BDSG ontology. The construction of the custom ontology, seems to be on an application by application basis; even on a user by user basis. Since every user may have different privacy needs, it is assumed that this application is intended to cater for different users, by allowing them to control how their data is used. In addition, the BDSG ontology, is a conversion of German federal law, into an ontology based language, F-Logic to be more specific, that is used in the construction of a Privacy XML in conjunction with custom ontologies.

The second study looked at is by Jammalamadaka (2013) that proposes the iDataGuard middleware. IDataGuard is a middleware that aims to protect the security of user data and address the issue of heterogeneity in dealing with Internet Data Providers or IDPs in short. There are various IDPs on the Internet that users can utilise for data storage: Amazon S3, Google Drive and DropBox amongst others, they each have their own APIs and flow, therein comes the problem of heterogeneity. Jammalamadaka (2013) also states that even though a IDP may have good intentions, a user's data can be compromised by hackers or even by disgruntled employees of the IDP, a point that must be agreed with.

Jammalamadaka (2013) built iDataGuard using Java Standad Edition and its physical representation is a Java jar file. It can be simply included as a library in a Java project and be used, in fact a sample application is built into the study called FileSystem Backup. The iDataGuard is such that it abstracts away any complexity that arises as a result of each IDP having a different API from any other. In addition there is a Crypto module in the middleware that encrypts all user data before sending it to the IDP for storage.

Birrell & Schneider (2014) proposes Avenance tags to address the shortcomings of notice and consent, which is the primary method that is used by most service providers on the Internet to deal with privacy matters. Birrell & Schneider (2014) identifies five shortcomings that notice and consent has and they are expressiveness, scalability, transparency, user policy revision and enforcement.

Expressiveness is about the low level of expressiveness in notice and consent. The approach taken by service providers is to notify the user of the scope of usage that their information will have. by If service providers share information with third parties they would usually include this in the terms given to the users. The users are given a binary option of either accepting the terms or of not using the service if they have a problem with the terms and conditions. In some cases the service in question is an online email service like Gmail, that has some measure of importance and economic value, since a person would use gmail to apply for a job for example.

Scalability is about the impracticality of reading every notice and consent document, as already identified by McDonald & Cranor (2008).

Transparency is about the degree of clarity and detail on the actual usages of a data subject's data by the data controller.

User policy revision is about the static nature of notice and consent; if a user suddenly becomes uncomfortable with certain practices by the site they cannot necessarily retract consent.

Finally **enforcement** speaks of the mechanisms that exist in enforcing privacy policies; a service provider can transgress their own policies and there is no mechanism to deal with this. Although there are laws governing data privacy issues it is not clear how this author views these in terms of enforcement.

The technical design of Avenance tags is geared towards solving the five issues, already identified with notice and consent. Avenance tags as a solution are split into two parts 'avenance tags' and 'avenance ecosystem'. The 'avenance tags' part of the broader Avenance tags solution aims to address the expressiveness and transparency problems of notice and consent as identified by Birrell & Schneider (2014). The 'avenance tags' borrow design elements from P3P (Cranor 2002) with the usage of policy tags that are always attached to data.

These policy tags contain meta-information about the data which indicates the scope of usage on the data. The 'avenance tags' use a policy language created explicit for expressing privacy policy. This language should allow for the expression of fine-grained privacy directives from a user. For example, the user should be able to express their wish if they do not want to share information with gambling organisations for religious or moral reasons.

The issue with this approach by Birrell & Schneider (2014) is that the user would first need to have knowledge of the policy language used, and from the samples presented in the study, its effective usage would be limited to a select few, as the language is complex for the average user. If this language is intended for developers to use it would defeat the purpose of having a scalable solution, as user preferences would have to be well defined up front and developed and presented to the user in an easy understandable format, to which users can relate.

The 'avenance ecosystem' part of the broader Avenance tags is meant to address the issues of scalability, transparency, user policy revision and enforcement. The ecosystem consist of four actors;

- User Client
- Identity Providers
- Service Providers
- Auditing Party

The **User Client** represents the client code that collects and sends user data to the Identity Providers. The **Identity Providers** contain and store all the user information, in addition they also contain all the meta-information that may be needed to compute the results of the policy that has been specified using the policy language. The **Service Providers** are the organisations providing the service that users are accessing, examples of them are Google and Facebook. Finally the **Auditing Party** is responsible for enforcing the data policies, and they would have all the information pertaining to the usage of user data by the Service Provider. At this stage there is no real-life example of the Avenance tags in use, since work on them is ongoing.

4.3.1.3 Data Layer

Those studies that focus on building a database-centric privacy solution are associated with the Data layer. The first study is by Emekci et.al (2005) who proposes a relational database extension ABACUS that sits on top of the database, such that all calls to the database from external systems, have to pass through this proposed middleware. Below is a diagram depicting ABACUS middleware's architecture.

From the diagram, the ABACUS middleware is depicted as being an extra layer that executes within the same boundary as a relational database. When requests come in from external systems ABACUS will forward the calls via an SQL interface, to the relational database. In this diagram such an interaction is with a database, that has been configured to function as a OLAP data store.

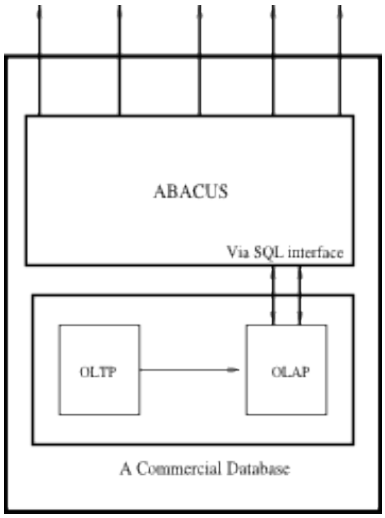

Figure 4.2 ABACUS middleware by Emekci et.al (2005)

This approach can be considered ideal, as the control of data is placed as close as possible to the data itself, and could make the enforcement of certain data laws, by an external party slightly easier. In this system it is possible to dictate to a database, what data is permitted to leave the boundaries of the database. Theoretically this ABACUS middleware can be configured by a regulatory body by remote control.

In practical terms, it would be complex to have ABACUS fully functional as specified in Emercke et.al (2005). One of those complex issues, would be to have the ABACUS middleware operational in different relational database platforms. The following are the more popular relational database platforms:

- MySQL
- PostgreSQL
- SQL Server (proprietary)
- Oracle (proprietary)
- DB2 (proprietary)

Porting ABACUS to all these platforms would prove challenging, in fact with some of the platforms it may be impossible, as some of the platforms are proprietary. In addition, the other shortcoming of ABACUS, is the non-support of non-relational databases such as NoSQL databases.

4.3.1.4 Multi Layered

This section presents a proposal by Anciaux (2012), called the Trusted Cell architecture. This proposal does not primarily focus on one universal layer, but on a multitude of layers, and even considers doing something at the hardware level. The Trusted Cell architecture proposes a client side reference monitor, that runs atop a secure hardware module. It is envisioned that this hardware module, would use technologies akin to the ARM TrustZone modules, that enforce security at the hardware level, by having a CPU core that is dedicated to such (ARM n.d).

Anciaux (2012) argues that since AMD is planning to incorporate ARM's TrustZone technology into their CPU (Computerworld 2012), devices with hardware level security enhancements, will be more prevalent. Anciaux (2012) views the Trusted Cell architecture as being comprised of these key attributes:

- An architecture abstracted as a Trusted Execution Environments
- Tamper resistant memory space, where cryptographic secrets are stored
- Mass storage
- Communication facilities

The list of key attributes for the Trusted Cell architecture, will assist in explaining the Trusted Cell architectural diagram, that has been extracted from Anciaux (2012).The diagram portrays our world view, in which the Trusted Cell architecture is prevalent. The vehicle in the diagram depicts a GPRS device, that presumably is part of the vehicle. This GPRS device would have a Trusted Cell module, thus allowing the user to dictate how their location data is shared. This of course is assuming that the GPRS would have its system software configured such that it satisfies one of the many privacy theories.

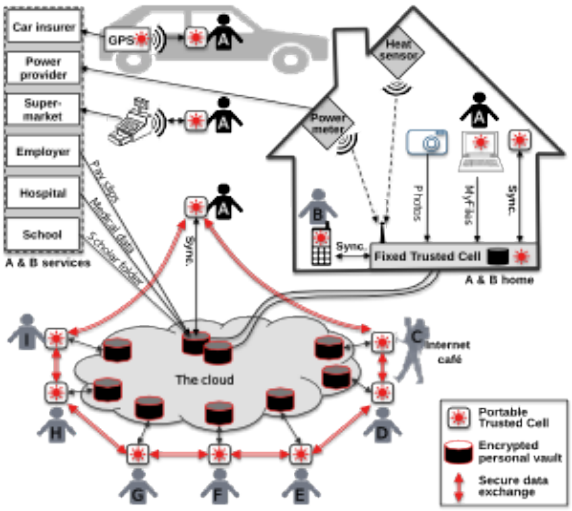

Figure 4.3: Trusted cell architecture by Anciaux (2012)

The diagram further shows the application of the Trusted Cell architecture to other platforms, such as:

- Power Meters
- Cellphones
- Heat Sensors
- Personal Computers

The Trusted Cell architecture, still needs to undergo further work, specifically in providing more detailed information, as to what would fully constitute a Trusted Cell architecture, and what would not. As it stands, from the four key attributes, it is highly likely that different vendors would implement the Trusted Cell architecture differently.

The communication attribute would be problematic as a communication standard has not been specified as part of what constitutes the Trusted Cell architecture. Different vendors might consider using REST endpoints, others might consider using sockets, whereas others might consider the usage of the Thrift protocol. At this high-level stage it is difficult to assess the impact this would have on developers or users

4.3.2 Research Question 2:

The second research question will be addressed by looking first at middleware extensions that have derived higher level products like the Carbon framework and Apache Karaf, and then at improvements that have been made on existing middleware solutions to either improve their performance or provide additional functionality.

4.3.2.1 Carbon Framework

Fremantle et.al (2010)'s proposition of a middleware building framework as is implemented, is very similar to the Apache Karaf project, and its derivatives. The diagram below gives a high-level overview of the Carbon Framework's architecture.

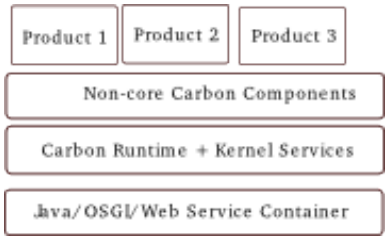

Figure 4.4: Carbon framework Freemantle et.al (2010)

A t the very foundation is an OSGi container that serves the need to have the Carbon Framework as a compliant component based platform. According to Sommerville (2011, p.453-454) component based software engineering is based on the idea that an application is composed of small discrete components. These serve as the individual parts that constitute the whole. OSGi containers are perfect platforms that can enable a component based application in the Java ecosystem.

At the next level in the stack, is the Carbon Runtime and Kernel Services. This is the level where the bulk of the carbon specific code would be. This code is intended to make the Carbon Framework archive its design intents. This is very similar to how Apache Karaf and the Virgo Web Server extend the base functionality of Apache Felix and the Eclipse Equinox OSGi containers respectively.

The next layer above the Carbon Runtime is the final layer that is part of the framework. The Non-core carbon components, represent the various libraries, frameworks and other servers. These are the dependencies that Carbon needs to archive SOA compliance. Other non-core carbon components could be web services platforms, message mediation platforms and so forth.

4.3.2.2 Apache Karaf

Karaf arose as part of a refactoring process of the servicemix ESB. It was decided while conducting refactorings for servicemix to spin off the kernel into a separate project and thus Karaf was born. Karaf adds additional behaviours atop an OSGi framework in a way similar to that used by the Carbon Framework, although the Karaf has a broader community by virtue of being part of the Apache foundation. The diagram below depicts the additional services provided by Karaf.

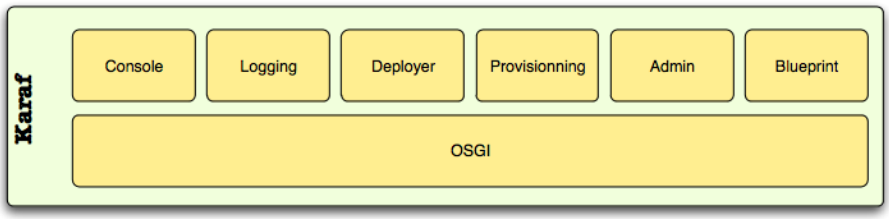

Figure 4.5: Karaf software architecture

4.3.2.3 Middleware Improvements

In other studies, work was embarked on that changed the internal code structure of middleware. Those studies are by Jayathilaka (2013) and Uralov (2013). Jayathilaka (2013) aimed to implement a more efficient mediation method for sending messages, for the Apache Synapse ESB. The extension was repackaged as WSO2 ESB.

According to Jayathilaka (2013) when it comes to the message mediation process, most ESBs carry out the mediation process in a similar fashion, which can be characterised as follows:

- Transport listener copies any incoming message to the input buffer
- The message is then taken from the input buffer and transformed into an XML object model
- Then the various mediators in the Mediation Engine will do the applicable processing on the transformed message
- Then the processed message is transformed into a serialized XML object model
- Finally the Transport Sender will pick up the message and send it to the relevant destination

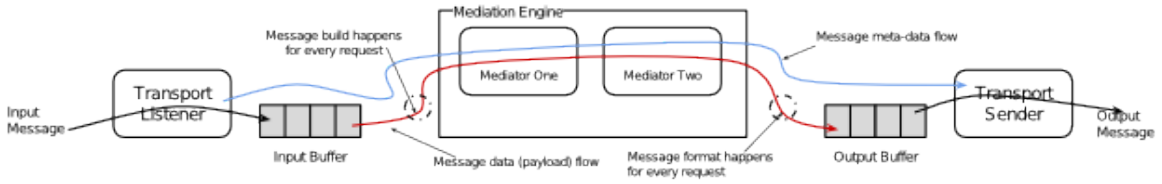

figure 4.6: 1.Improvements to Apache Synapse's message mediation process by Jayathilaka (2013)

Jayathilaka (2013) identified three scenarios which can be taken advantage of, where there is no need to follow the message mediation process meticulously all the time. Below is a diagram that illustrates one of the scenarios, whereby the steps to transform the message to an XML object model and a serialized XML object model, have been omitted thus omitting the mediation engine as well. This has the intended effect of placing the message directly from the input to the output buffer. Certain situations can allow for such flexibility, such as when all that is needed is to route the message based on the message headers.

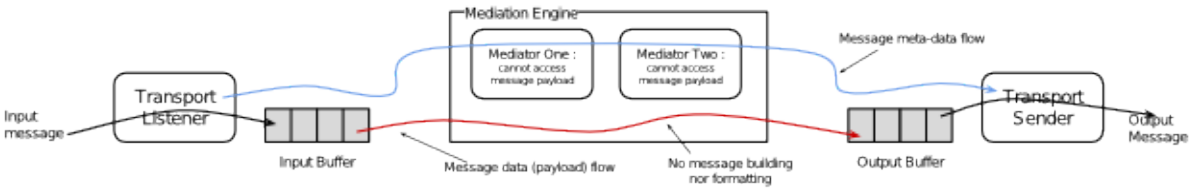

Figure 4.7: 2.Improvements to Apache Synapse's message mediation process by Jayathilaka (2013)

Uralov (2012) designed extensions for an ESB middleware platform, to be specific ServiceMix 4. This extension is concerned with enabling the ESB to support multi-tenancy in the cloud, and to have the ability to dynamically choose cloud data service providers. The method of extension is two-fold. First is to extend the parts of the middleware that communicate with the outside world, which can be seen in the diagram below as some of the bold parts, namely they are JMS, E-Mail and SOAP Over HTTP modules. Secondly is through the Tenant Router modules, where every outgoing message is decorated with a tenant context, so that in later processing stages, the message can be identified with the correct tenant.

Uralov (2012) utilises both JBI and OSGi in extending ServiceMix. JBI is used to extend the base functionalities of the outgoing JBI modules, such that they support multi-tenancy. The OSGi modules connect with a data source, that contains tenant specific information, and also provide a UI configuration layer.

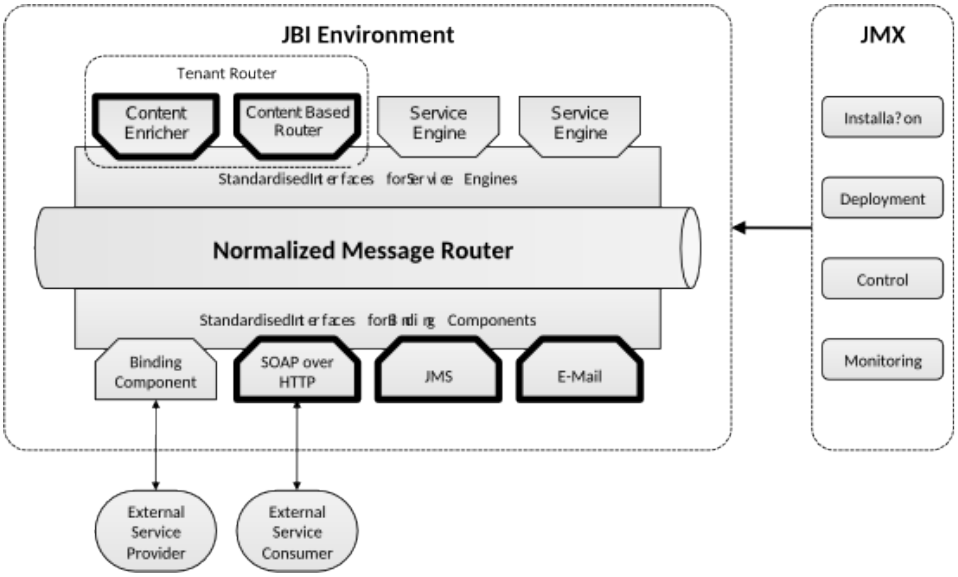

Figure 4.8: Improvements to ServiceMix by Uralov (2012)

What can be taken away from this section is the contrast between the approach taken by the Middleware Improvements and that taken by the combination of the Carbon Framework and Apache Karaf. The latter provide a clean extension to OSGi in that they provide a generic platform that goes over and above what OSGi frameworks support from the box.

The Middleware improvements looked at works that changed the internal structure of ServiceMix and Synapse ESB, which suggests a certain set of extra responsibilities. Every new release of the middleware will require the improvements to be ported, then the middleware to be recompiled and repacked. With the approach taken by the Carbon Framework and Karaf, new releases should not break the system since the extensions would have been programmed to an interface.

4.3.3 Research Question 3: Domain Specific Language

Designing any API is no easy feet (Tulach 2012) and therefore this section will look at the literature surrounding Domain Specific Languages. A DSL is specialised language optimised for a specific domain (Deursen 2000; Voelter et.al 2013, pg.28). The following are well known example of DSLs: SQL, CSS, Junit, Apache Camel, JOOQ. DSLs are usually not used to write full blown applications instead they are used for a very specific purposes a their utilisation is best described as such;

- SQL- Used to query data from a relational databases
- CSS – Used to define the styling characteristics of HTML elements
- JUnit – Used to denote and configure unit tests in Java
- Apache Camel – Used in JVM based languages, to implement EAI (Enterprise Integration) patterns
- JOOQ – Used as a way to access relational data from the database in Java. This is done in a way that is similar to that used to extract data using SQL.

More examples of the DSLs are given below, to enhance their focus.

Table 4.2: SQL sample

```
SELECT *
FROM accounts;

SELECT custName,custSurname,orderDate,productName,productPrices
FROM customer,order,order_line,product
WHERE customer.id = order.customerId
AND order.id = order_line.orderId
AND order_line.productId = product.id;
```

From the SQL script example on table 4.2. The first script is concerned with selecting all database rows from the accounts table. The second performs a JOIN operation on four tables in order to display a list of 1, customers, 2, their orders, 3, the products they ordered and 4, how much they cost. SQL is designed as a language to query relational databases.

Table 4.3: CSS sample

```
h1 {
 background-color: red;
}
p {
 background-color: green;
}
div {
 background-color: yellow;
}
```

The CSS on table 4.3, is concerned with setting the styling of certain HTML elements on a page. From this CSS code, the following can be deduced. The background colours of the <h1>,<p> and tags, where the style sheet is applied will have <h1>,<p> and tags set to red, green and yellow respectively.

Table 4.4: Junit sample

```
import org.junit.Test;
import static org.junit.Assert.*;

public class MyUnitTest {`
 @Test
 public void testConcatenate() {
 MyUnit myUnit = new MyUnit();
 String result = myUnit.concatenate("one", "two");
 assertEquals("onetwo", result);
 }
}
```

Junit is a unit testing framework for Java. Unit testing frameworks are used to test the functionality of Java classes, to assert if given a certain input they will give out a certain output. In the code example above the unit test is concerned with verifying if the concatenate method, from the MyUnit class concatenates strings.

The structure composition of the unit test, is firstly the unit test itself, which is a basic class definition. In this case that class definition is 'MyUnitTest'. Following that, is a set of methods in the class definition, some of which will be the actual unit test. In this case, the 'testConcatenate' method is a unit test, and it has been rightfully marked with the '@Test' annotation. In addition the method contains an 'assertEquals' method call. The 'assertEqual' is a unit testing prerequisite, as it is through 'asserts' that unit testing frameworks can report back on the state of a unit test, whether it has failed or succeeded.

Table 4.5: Apache Camel sample

```
public void configure() {
   from("direct:cafe")
      .split().method("orderSplitter").to("direct:drink");
   from("direct:drink").recipientList().method("drinkRouter");
   from("seda:coldDrinks?concurrentConsumers=2").to("bean:barista?
method=prepareColdDrink").to("direct:deliveries");
   from("seda:hotDrinks?concurrentConsumers=3").to("bean:barista?
method=prepareHotDrink").to("direct:deliveries");
   from("direct:deliveries")
      .aggregate(new CafeAggregationStrategy()).method("waiter",
"checkOrder").completionTimeout(5 * 1000L)
      .to("bean:waiter?method=prepareDelivery")
      .to("bean:waiter?method=deliverCafes");
}
```

On table 4.5 is an Apache Camel route, derived from the Cafe Shop example found on Camel.apache.org (n.d). From an abstract point of view, this example maps the process that a barrister would execute to take a customer order, right through to delivering the coffee to a customer. The example eloquently uses the Camel API to show off the enterprise integration patterns at play. The code has a human readable structure about it. This is due to the design principle behind Apache Camel. This is also known as a Fluent API, as described by Fowler (2010).

Table 4.6: JOOQ sample

```
DSLContext create = DSL.using(conn, SQLDialect.MYSQL);
Result<Record> result = create.select().from(AUTHOR).fetch();
```

Table 4.6 is a Java framework, which can be best described as a Fluent API, as defined by Fowler (2010). JOOQ is concerned with creating an internal DSL within Java, that best matches the syntactic structure of SQL. An explanation/description of an internal DSL follows.

4.3.3.1 Internal DSL

An internal DSL, is a DSL that is embedded within the host language (Ghost, 2010, p.42). The internal DSL uses a subset of the host language's features, so that the final result has the feel of a custom language as opposed to the host language (Fowler,2010, p.27). From the DSL examples given above, the following can be considered internal DSL, as they are embedded within a host language. Yet they have a certain feel to them, that closely resembles the domain they are trying to map.

- JOOQ
- Junit
- Apache Camel

Further more, Fowler (2010) defines a set of techniques, that can be utilised in Object-Oriented programming languages. These techniques are predominantly biased towards C# and Java, although some of the concepts can be generalised towards other languages like Ruby, C++, Scala and so forth. The techniques are Method chaining, Function Sequence, Nested Function and Annotation and are described below. Method chaining is the act of chaining together a set of methods, as a sequence, such that each subsequent method call acts on the result of the previous method call, achieving a result that looks something like this:

Table 4.7: Method chaining sample

```
computer().processor().cores(2)
.speed(2500).i386().disk()
.size(150).disk().size(75)
.speed(7200).sata().end();
```

As can be seen from table 4.7. Method chaining is intended for Object-Oriented languages, as the mechanism works by having the return value of the method call be a context object, that in turn provides access to other methods. In this case, these methods are used to configure a computer. Method chaining is usually terminated by a termination method, which in this case is the “end()” method. A function sequence is akin to method chaining. The difference between them is best illustrated in a code example on table 4.8:

Table 4.8: Function sequence sample

```
computer();
processor();
cores(2);
speed(2500);
i386();
disk();
size(150);
disk();
size(75);
speed(7200);
sata();
```

From the code example, it is clear that major difference between function sequence and method chaining is that there is no usage of a context object in the former.. Thus the flow of the DSL is no longer via an Object-Oriented mechanism, but through subsequent function calls. Since there will not be a context object to keep the state of the subsequently changing semantic model, the state would have to be maintained in a global context.

Table 4.9: Annotation sample

```
class PatientVisit...
@ValidRange(lower = 1, upper = 1000, units = Units.LB)
private Quantity weight;
@ValidRange(lower = 1, upper = 120, units = Units.IN)
private Quantity height;
```

Annotations are a Java language extension that allow the addition of metadata to a class or method at compile or runtime (Lindholm 2011, pg.67). Although Fowler (2010) lists Annotations as an internal DSL method, it is viewed in lesser light than other methods, and Fowler (2010) considers it as a way to implement a fragmentary DSL. From the code example above, the @ValidRange annotation. Communicates to the reader the intent.

4.3.3.2 External DSL

External DSLs, unlike internal DSLs are not tightly coupled to a specific host language's syntactic structure (Fowler 2010), therefore their design and ultimately their representation, need not be anything that resembles the host language that they might be attached to. From the DSLs mentioned previously, SQL and CSS can be considered examples of external DSLs.

The techniques that can utilised to implement external DSLs, will be outlined in relation to those used for internal DSLs, as described in that section. However they will not be drawn from the approaches taken by Fowler (2010), the reason being that the main techniques listed by Fowler are of a lower-level, in terms of implementation concerns, in that they are more directly applicable to the techniques used to build a generic purpose language. The techniques to be discussed are:

- Delimiter-Directed translation
- Syntax-Directed Translation
- Parser-Generator

All three of these methods can be found in various leading works on compiler design, such as Safonov (2010). Therefore this section would like to shine the light on language-oriented programming approaches, specifically on language workbenches. Language-Oriented programming as defined by Ward (1994) is a development approach whereby the development of a software system, at a very high level is split into two major parts. The first part is concerned with developing the software with a specific 'middle language' which is what Ward (1994) refers to as a DSL, and the second part is concerned with the development of the actual 'middle language' itself.

Language workbenches is a term coined by Fowler (2010) which describes a set of tools, that aim to provide a DSL development environment with a complete IDE feature set in the sense that there is auto-complete, syntax highlighting, and other post IntelliJ IDE features. Notable examples of language workbenches are Spoofax, Xtext and MPS. There are two broad approaches to language workbenches, textual or projectional. Spoofax and Xtext can be considered to take the textual approach (Erdweg 2013; Stoffel 2010; Visser 2010), and MPS the projectional approach (Voelter & Solomatov 2010).

The difference between projectional and textual language workbenches lies in how they approach the fundamental programming of DSLs. To explain their differences in depth, one has to first explain how a general purpose language is compiled into executable form. A simplistic explanation of the process that converts a general purpose language into executable form is this: the programmer writes programming language code to achieve some objective, this programming logic as seen by the user is considered to be the concrete syntax.

This concrete syntax, at compile time is converted by a parser into an abstract syntax tree. The abstract syntax tree, is a representation of the concrete syntax, that is understood by the computer. This abstract syntax tree is then converted, depending on the language, into byte code or machine code (Friedman & Wand 2008, pg.51-53). Having explained AST and concrete syntax, the next step is to explain how they fit into language workbenches.

Textual language workbenches like Spoofax and Xtext have a programming model that is based on a concrete syntax, the programming interface has a textual representation and its storage format is also textual (Visser 2010; Xtext 2012). Conversely a projectional language workbench like MPS, is the opposite, the interfaces that programmers deal with are just projections of the AST (Voelter & Solomatov 2010). In MPS the programmer has the option to modify the AST through other forms of projections such as diagrams, tables or mathematical notations (Stoffel 2010; Pech et.al 2013).

This chapter looked at studies related to this current study by using the research questions as a guide. Each of the research questions had its own section where related studies pertaining to that question were discussed, and it is hoped that those discussion will provide input to the design chapter which will detail the design of the artefact.

From the first question it is hoped that insight will be derived on things to consider, based on how others have approached the problem space at different layers. The second question gave insight into the different approaches in extending Java middleware and finally the third question discussed domain specific languages, which gave examples of various design approaches to designing APIs.

Chapter 5.

Methodology

This chapter will detail the research design and methodology employed for this study. This study will make use of the problem-solving design science research paradigm as outlined by Hevner et.al (2004). This chapter will start of by giving a brief overview of design science, followed by a description of the research model by Hevner et.al (2004) and how it relates to this study.

5.1 Design Science

Design science is a practice based research paradigm that has been developed for the information systems field as a result of the need to have the concept of 'bring back the artefact' in IS. The first mention of design science was the work by Nunamarker et.al (1991) which was influenced by Simon (1968) and Schon (1975). Nunamarker et.al (1991) provided a framework with which to carry out design science research with an emphasis on theory building, though not much details was provided on what this theory building should look like. Walls et.al (1992) provided a theory building approach in information systems with a focus on applied research. Other researchers have provided frameworks for conducting design science research (March & Smith 1995; Peffers et.al 2007).

5.2 Research Model

Below in figure 5.1, is a pictorial representation of the research model as described by Hevner et.al (2004). This model serves as a blueprint for design science research. This blueprint has served as a foundation for other studies (Speitkamp & Bichler, 2010), with each researcher modifying this model to suit the needs of their respective studies. Therefore in this study, the model will also be modified to suit the unique context of this study. A brief explanation of the various aspects of the model will be given.

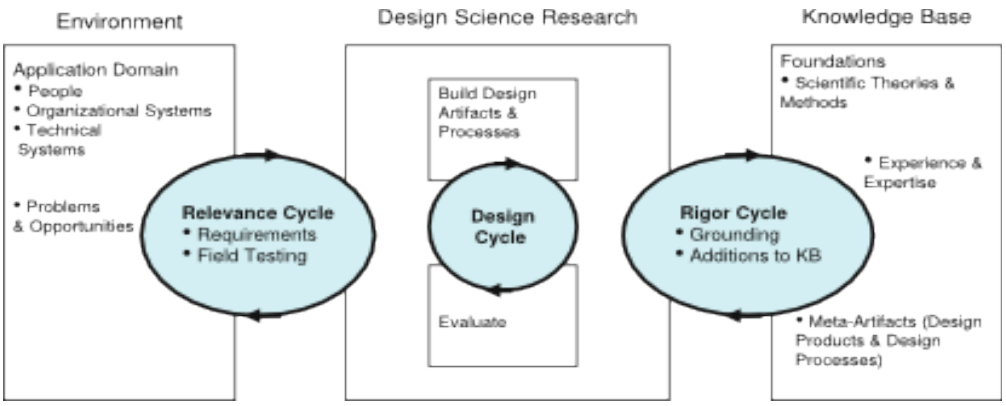

Figure 5.1: Design science research framework by Hevner (2004)

The model consists of three cycles, which serve the purpose of joining together the three different blocks. The three blocks are 'The Environment', 'Design Science Research' and the 'Knowledge Base'. The Environment represents the context in which the study is based, and this is where the research problems arise. The research problems are problems to do with people, organisations or technical systems. The Design Science Research block is concerned with the building of the artefact, of which there are two major parts, the building and design of the artefact, and the evaluation of the artefact. The Knowledge Base block represents the knowledge that lays the foundation of the study.

As can be seen on the diagram the blocks in the model are joined by the three cycles Relevance, Design and Rigour. These cycles are considered by Hevner & Chatterjee (2010) to be very important parts of design science research. They join together its different parts, from the research problem and questions, through the construction of the artefact to the contributions of the artefact. This model provides structure to a design science project.

5.2.1 Relevance Cycle

This cycle is concerned with the problem space in which the design science research project is situated. This is considered by prominent design science research evaluators to be the most important part of the design science research (Venable 2010). In relation to this study the relevance cycle takes cognisance of the research problem and the accompanying research questions.

5.2.2 Design Cycle

This cycle is concerned with the construction and evaluation of the artefact produced in the study. This cycle has two sub functions: design\build and evaluate. The evaluation of the artefact is rated as one of the more important criteria to be met (Venable 2010) in the guidelines provided by Hevner et.al (2004). The evaluation criteria will be guided by recommendations from Venable, Pries-Heje & Baskerville (2012). Venable, Pries-Heje & Baskerville (2012) provide a guideline for choosing evaluation methods in a design science research project which uses the framework developed by Pries-Heje & Baskerville (2008) as a base.

Table 5.1: Design science evaluation quadrant, adapted from Venable, Pries-Heje & Baskervilee(2012).

	Ex Ante	Ex Post
Naturalistic		
Artifical		

The framework provided by Venable, Pries-Heje & Baskerville (2012) is a quadrant which on the x-axis is the ex-ante and ex-post, which describe when an artefact is evaluated whether it is before it is built (ex-ante), or after it has been built (ex-post). Then the y-axis it is split into naturalistic or artificial. Naturalistic is when the artefact is being evaluated in a real life setting and artificial is when it's evaluation is not concrete, like a computer simulation or a laboratory experiment.

5.2.3 Rigour Cycle

This cycle is concerned with linking the design science project with the existing knowledge base. This knowledge base will be used to guide the construction of the artefact where applicable. Once the artefact is constructed it is hoped that the evaluation methods applied in the design cycle will yield worthwhile contributions to the body of knowledge.

Having explained the research model the next step is to discuss how this model relates to this study. In order to do that, a mapping mechanism is needed between the model and this study, and that mapping comes in the form of an eight point checklist provided by Hevner & Chatterjee (2010, pg.20). This checklist is argued to be more helpful than the original seven guidelines provided by Hevner et.al (2004).

The eight points in the checklist are as follows.

1. What is the research question (design requirements)?
2. What is the artefact? How is the artefact represented?
3. What design processes (search heuristics) will be used to build the artefact?
4. How are the artefact and the design processes grounded by the knowledge base? What, if any, theories support the artefact design and the design process?
5. What evaluations are performed during the internal design cycles? What design improvements are identified during each design cycle?
6. How is the artefact introduced into the application environment and how is it field tested? What metrics are used to demonstrate artefact utility and improvement over previous artefacts?
7. What new knowledge is added to the knowledge base and in what form (e.g., peer-reviewed literature, meta artefacts, new theory, new method)?
8. Has the research question been satisfactorily addressed?

Each of the checklists maps to a certain step in the Design Science Research model (Hevner & Chatterjee 2010, pg.20) of which the mapping shown pictorially on figure 5.2. A table 5.1 provides answers to the design science research checklists to give a better picture of the research design.

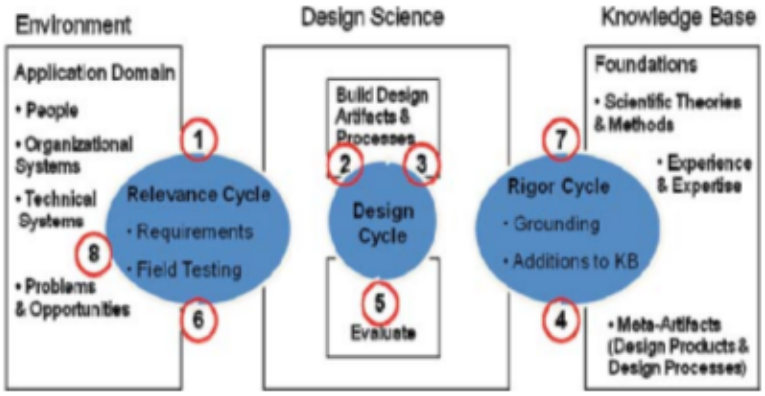

Figure 5.2: Mapping of Hevner design science framework to Hevner & Chatterjee (2010, pg.20)

Table 5.2: Answers to Hevner & Chatterjee checklist.

Checklists	Answer

What is the research question (design requirements)?	• How can middleware provide Developers with a platform that enables them to write privacy enhancing software faster and that can allow people to have increased control over and\or visibility of their data?
	  • How can people have more control over or visibility of their data?
	  • How can middleware be extended to provide a privacy enhancing platform to developers?
	  • How can Developers write software faster

What is the artefact? How is the artefact represented?	The artefact is a middleware extension that is divided into three parts • Client side API • Tomcat server extension

	• OSGi based privacy engine The client side API and the Tomcat server are regular jars, and the privacy engine is packaged as an Apache Karaf feature.
What design processes (search heuristics) will be used to build the artefact?	An interative process that consisted in the following steps: • build • verify • literature search • Adapy
How are the artefact and the design processes grounded by the knowledge base? What, if any, theories support the artefact design and the design process?	The artefact is grounded in the knowledge base in the sense that the construction of the artefact draws on knowledge from the following sources • Java Specification • Java Servlet Specification • Javassist knowledge base • OSGi knowledge base • Karaf knowledge • Tomcat source code • Squeryl knowledge base • privacy literature
What evaluations are performed during the internal design cycles? What design improvements are identified during each design cycle?	The following methods are used to evaluate the artefact: • Unit tests • Debugging
How is the artefact introduced into the application environment and how is it field tested? What metrics are used to demonstrate artefact utility and improvement over previous artefacts?	The artefact will not be introduced into the application environment, that is to say it will not be used in a real life project, instead computer simulations will be used to evaluate the efficacy of the artefact. The evaluation metrics will be based on the following attributes • Practicality: How practical is the usage of the artefact in the field • Efficacy: How well does the artefact work • Design process: Evaluation of the

	construction process of the artefact
What new knowledge is added to the knowledge base and in what form (e.g., peer-reviewed literature, meta-artefacts, new theory, new method)?	To be discussed in the results chapter
Has the research question been satisfactorily addressed?	To be discussed in the results chapter

Having described the extent to which this study will aim to answer the checklist points identified by Hevner & Chaterfee (2010, pg.20), it is worthwhile noting that Venable (2010) conducted a study amongst top scholars who are reviewers and editors of top IS journals, and the general consensus amongst them is that the checklist points as outlined by Hevner & Chaterfee (2010, pg.20) should serve only as guidelines, as the feasibility of meeting all the criteria in a single study is very low.

The study revealed that two factors were of importance to the reviewers. Firstly, the study should be well placed in the field, in the sense that the problem the research is trying to solve should be of relevance. Secondly the by-product of the research will be more valued if the artefact produced is an instantiation of the proposed solution, that is to say the research produces some kind of artefact.

Chapter 6.

Design

As discussed in previous chapters the intent of this research is, through relevance and rigour, to arrive at a design proposition for a privacy enhancing artefact. It is intended that the ideal artefact should be able to function within currently used platforms in the Java middleware space. This chapter will discuss the software architecture for the design of this artefact with a focus on the various parts that make up the platform, the client library, the Tomcat extension and the privacy engine. The artefact will be dubbed the Lesie privacy enhancing platform, of which Lesie is a Tswana word for the Cape Fox.

6.1 Software Architecture

Software architecture is a set of structures that help in reasoning about a system (Bass et al 2012) and this section will present diagrams that lay out the rationale for the design of the artefact. The artefact as a whole consists of three major parts: the client library, the Tomcat extension and the privacy engine. The client library is meant to be used as a third party library. The client library will primarily provide an API used by developers to enhance the privacy aspects of their respective systems.

The Tomcat extension is a library that is meant to be bundled into the Tomcat lib folder which contains all the libraries used by Tomcat (Moodie & Mittal 2007, pg.30). The reason for this is that the Tomcat extension extends the functionality of the WebappClassLoader class which is found in the catalina.jar and is responsible for loading classes for a web application (Kurniawan & Deck 2004).

The extension weaves in code that communicates with the Privacy Engine to determine the course of action every time user data is shared. The Privacy Engine is the engine that is leveraged to enhance the privacy aspects of user data in information systems. The Privacy Engine is built atop Apache Karaf which is an OSGi distribution and it exposes its functionality through a rest API.

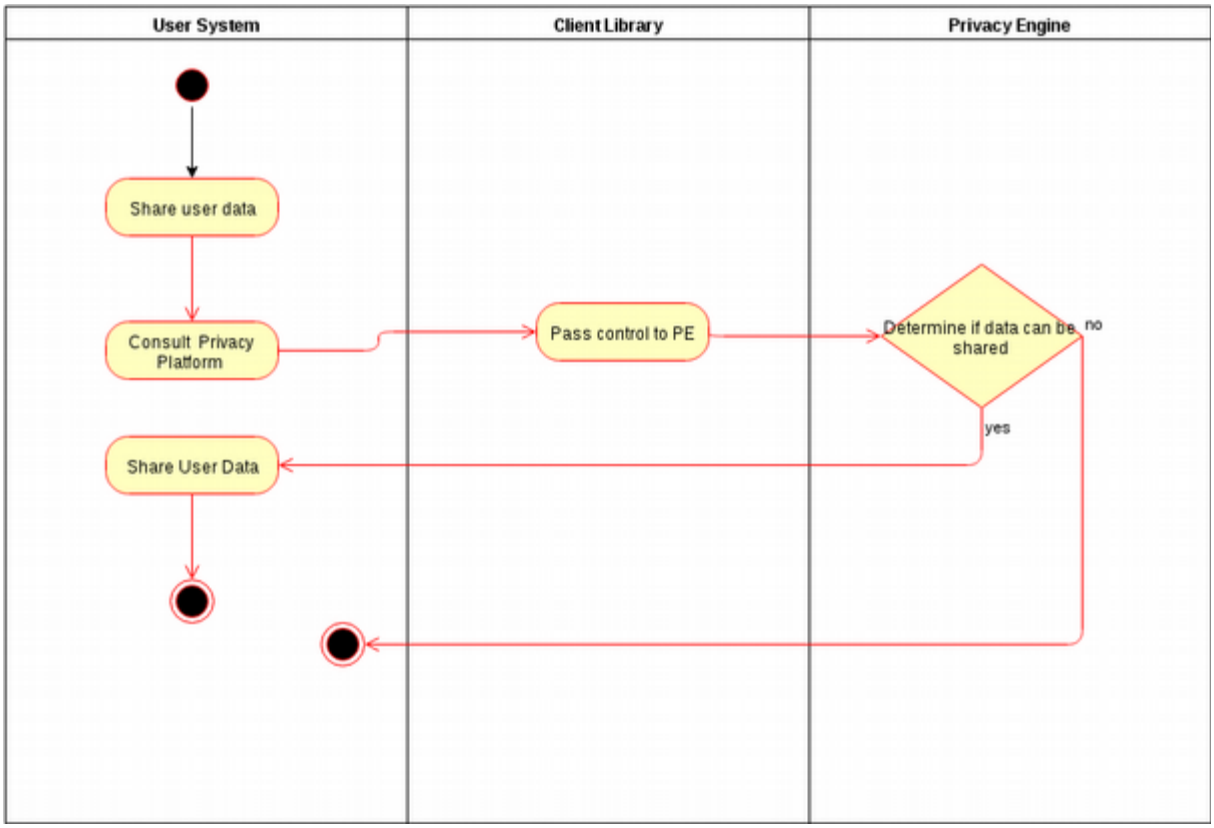

Figure 6.1: Activity diagram to depict the data sharing process in the Lesie platfom

The diagram on figure 6.1 indicates the logical flow when data is being shared by an application that utilises the Lesie privacy enhancing platform. When a data controller system (user system) initiates an action to share a data subject's data with a third party it will consult with the privacy engine on the correct course of action to take. This is achieved by utilising code in the client library that communicates with the privacy engine.

The deployment diagram in figure 6.2 depicts the placing of the various modules in the Lesie platform. The client system (data controller system) node depicts a system that is made up of various modules that are specific to the data controller's system. In addition to that the system has the client library as part of its system, which contains code that communicates with the privacy engine and then any API with which the developer can interact. In turn the privacy engine's modules are composed of modules that expose the rest API, perform CRUD operations on the database and provide rules for deciding when to share data.

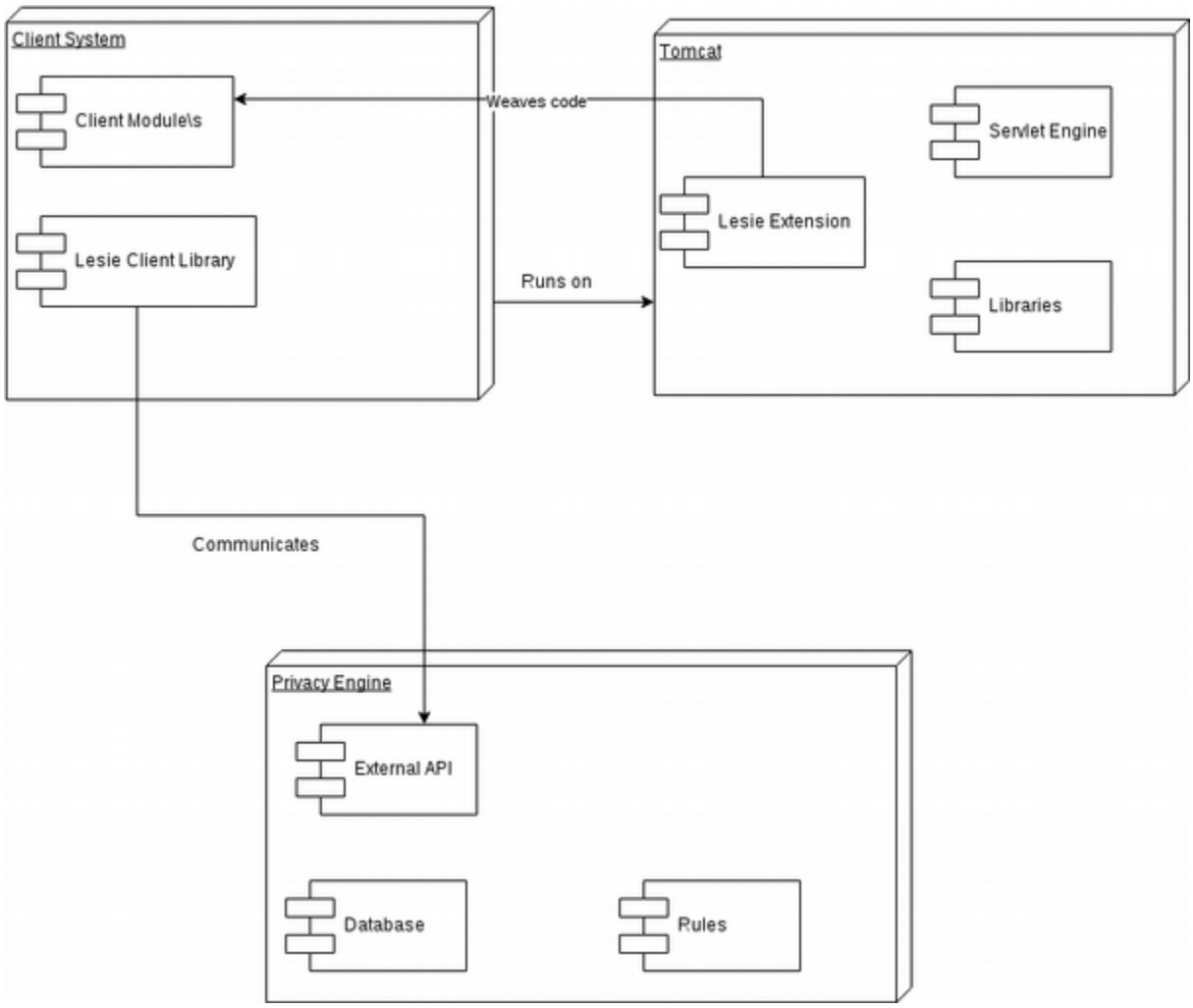

Figure 6.2: Lesie platform deployment diagram

The Tomcat node represents a Tomcat server instance with its various components, with a notable addition of the Lesie extension that extends the Tomcat web classloader to weave in code from the Lesie client library. The Tomcat extension also has its own internal copy of the Lesie client library. The sections to follow will give an in depth discussion of the various components that make up the artefact starting, with the Lesie client library.

6.2 Lesie Client Library

The client library is concerned with two major functionaries the first is to have an API that the developer can use to interface with the Lesie privacy enhancing platform. Secondly it provides code that communicates with the Privacy Engine. Regarding the first functionary, inspiration has been drawn from the previous chapter by modelling the API as a DSL, specifically as an internal DSL, that is annotation based.

Table 6.1 API annotation

```
@Gate
class UserDataSharingService...
@ExitPoint
private void shareUserData(User );
```

The above code on table 6.1 demonstrates an example of how the DSL can be used by a software developer to enhance the privacy aspects of applications. The main components behind this DSL are two annotations:

- @Gate
- @ExitPoint

These two annotations are meant to be used in conjunction with each other. The @Gate is a class level annotation that indicates classes, that move data out of the boundary of the data controller's system, into the boundary of an external system that belongs to third parties. The @ExitPoint is a method level annotation that indicates the methods that are responsible for moving the data subject's data. The @ExitPoint annotation can only be specified inside classes that have been annotated with the @Gate annotation, if this requirement is not met the code weaving will not work.

The second functionary of the client library is to provide code that communicates with the privacy engine. This code is embodied in the PrivacyEngineService and on figure 6.3 are code snippets from the canShare method.

```
public SharingResponse canShare(SharingRequest request) throws Exception {
    SharingResponse sharingResponse = new SharingResponse();
    HttpPost httpRequest = new HttpPost(PrivacyEngineService.PE_URL
            + PrivacyEngineService.PE_CAN_SHARE);
    httpRequest.addHeader("accept", "application/json");
    httpRequest.addHeader("content-type", "application/json");
    Gson gson = new Gson();
    String json = gson.toJson(request);
    StringEntity se = new StringEntity(json.toString());
    httpRequest.setEntity(se);
```

Figure 6.3: PrivacyEngineService canShare method part 1

The first thing the method does is declare the sharingResponse variable which will contain a response from the privacy engine. Following that it creates a http POST request object, and sets the header's content type to “application/json” to signify that the content inside the POST is in JSON format. The http POST request is also configured to accept JSON formatted response from the server. The code then proceeds to convert the request object into a JSON string and adds it to the http POST request.

```
  response = httpClient.execute(httpRequest);
    if (response.getStatusLine().getStatusCode() == 200) {
        BufferedReader br = new BufferedReader(
                new InputStreamReader(response.getEntity().getContent())
        );
        String brStr =br.readLine();
        sharingResponse = gson.fromJson(brStr, SharingResponse.class);
    } else {
        throw new SharingException();
    }
    return sharingResponse;
}
```

Figure 6.4: PrivacyEngineService canShare method part 2

This code on figure 6.4 communicates with the privacy engine by executing a http request against the privacy engine rest API. Following that a check is performed to see if the response from the privacy engine is successful, this is done by doing a conditional check to see if the status code is 200. If successful the contents of the response are read using a buffered reader, then demarshalled into a SharingResponse using the GSON library and finally return to callee. Alternatively, should the response be a failure then a SharingException is thrown.

This concludes the Lesie client library which has shown the annotations that determine which classes and methods are responsible for moving data in and out of the boundary of the data controller's systems, and snippets from the PrivacyEngineService which are responsible for communicating with the privacy engine. The next section will describe how the Tomcat extension weaves in code for the PrivacyEngineService on methods annotated with the @ExitPoint annotation.

6.3 Tomcat Extension

This section will detail the design of the Tomcat extension that enables code weaving into classes annotated with the annotations discussed in the previous section. What makes this code weaving possible is a concept known as AOP (Aspect Oriented Programming). AOP as a concept was pioneered by Kiczales (1996) and paraphrased by others (Laddad, 2003, pg 4) and according to them there are four key concepts in AOP and they are discussed below.

Table 6.1: AOP definitions

Concern	A concern can be defined as a matter of interest that is represented by application code in a computer program. It can be logging, security, business logic, transaction management and so forth.
Cross-cutting concern	A concern that cuts across various modules in a system for example logging and security. For example there can be two modules in a system one concerned with security and the other with storing data to the database. They each would need to log data thus making logging a concern that is needed into two modules, thus cross-cutting, so to speak.
Aspect	A unit of modularisation that is a representation of a cross-cutting concern. This can take the simplistic form of annotating a logging class with the @Aspect annotation for applications using AspectJ. This result in the concern being modularised and available for reuse as weavable code.
Aspect Weaver	A compiler like entity which composes the core and cross-cutting concerns into the final system in a process called weaving.

There are various libraries in Java which can enable AOP functionality. These libraries can be divided into low-level and high-level libraries, the reason for this distinction will be explained later. The libraries are shown in Table 6.2.

Table 6.2: AOP libraries

Low-level	• Java Dynamic Proxies • CGLib • Javassist • Apache BCEL
High-level	• AspectJ • Spring AOP

The split between high and low level libraries is based on the level of detail at which the developer has to operate. With the low-level frameworks the developer has to write more boilerplate code whereas with the higher level frameworks there is no need for verbosity. Their differences can also be characterised as follows;

- Low-level frameworks are either frameworks geared towards bytecode manipulation of compiled Java code, which is not exclusively about providing AOP like functionality, or they provide an AOP through the usage of proxies which are more limited than the higher level frameworks.
- High-level frameworks are frameworks that have been built from the ground-up with a specific focus on AOP.

Given this distinction between low and high level AOP frameworks it is best, for the artefact under consideration, to choose from the list of low-level frameworks, mainly due to the size of the high-level frameworks and the restrictions they would impose on the final solution. For example, with Spring AOP there would be a dependency on Spring and it would have to be bundled as part of the Tomcat installation that would have the Lesie platform extensions.

There is also the issue of the size of the extra memory space that would be used unnecessarily for the Spring framework, and there would most likely be other issues if any of the applications deployed were already using Spring. For example given the flat classpath hierarchy of Java, there is a chance that web applications might refer to the Spring classes loaded as part of Tomcat as opposed to the ones bundled with the application.

As for AspectJ is a rather impractical solution as the code to be weaved is either configured by XML or annotations (Laddad 2003), the impracticality with XML is that we do not necessarily know which class needs to have its code woven unto it when constructing the XML. The second approach of using annotations would require a two stage step: the first identifying suitable custom annotations, and second using one of the bytecode modification libraries to add the AspectJ annotations. This does not take in to account the fact that this might be too late in the class loading lifecycle and would render the added annotation irrelevant.

The next subsection will describe in detail the classloading mechanism and how it can be modified such that before a class is loaded into the JVM it is first modified using one of the low-level libraries. The low-level library to be utilised will be javassist for its ease of use. Javassist allows for the modification of bytecode by directly expressing the modification in Java code, as opposed to other bytecode modification libraries like CGLIB and Apache BCEL, which have an API very close to Java bytecode and require some knowledge about how Java byetecode works. In terms of Java proxies these libraries are too simplistic to be of any use.

6.3.1 Classloaders

This section describes the classloading mechanism in the JVM. Specifically, it describes how the process works for a normal JVM application and how it works in a servlet engine like Tomcat. It is through understanding the classloading mechanism that a better picture can be formed as to how the Tomcat extension intends to weave in code using Javassist.

According to Lindholm (2011, pg 337) the class loading process is split into three overarching parts and they are loading, linking and initializing. Loading is concerned with finding the binary representation of the class file and this can be from disk, memory or even across the network. The next stage is Linking which is concerned with verifying and preparing the class or interface so that it is in the correct state and is executable inside the JVM. Then finally the initialization phase involves executing the class's initialization method `<clinit>`. `<clinit>` is a special method in the JVM that is exclusively executed by the JVM, on behalf of the class being instantiated.

There are various delegation models in the JVM that deal with the way that classes are loaded into the JVM. The first one is the parent delegation model and is usually the most common one and is predominately found in J2SE environments (Panda et.al 2007, pg.397). The parent delegation model can be described as follows.

The current classloader is first interrogated to ascertain if its been registered as the initiating classloader. If this operation is successful the class is loaded from cache. If this is unsuccessful control is passed to the parent classloaders which would each try to load the class from cache. If the parent classloaders fails to load the class from their cache, only then will the current classloader try and load the class from disk for the first time. This will initiate the three step process of classloading described earlier.

The second classloading mechanism is that of a servlet engine, specifically that of Tomcat. The classloading mechanism is different to the stock standard parent delegation model used in JSE environments. This model first tries to load the class in the current web application classloader and only when it fails completely does it delegate to the upper classloaders in the chain (Mordani,2009, pg.125; Moodie & Mittal, 2007, pg.140).

Each web application has its own web application classloader which loads classes specific to a web application. The classes under the WEB-INF/classes directory and any libraries under the WEB-INF/lib folder are loaded. As can be seen pictorially in Figure 6.5. below the classloading mechanism under Tomcat is hierarchical, and flows such that the top level classloader is the bootstrap followed by the system, common and then finally the web application classloaders.

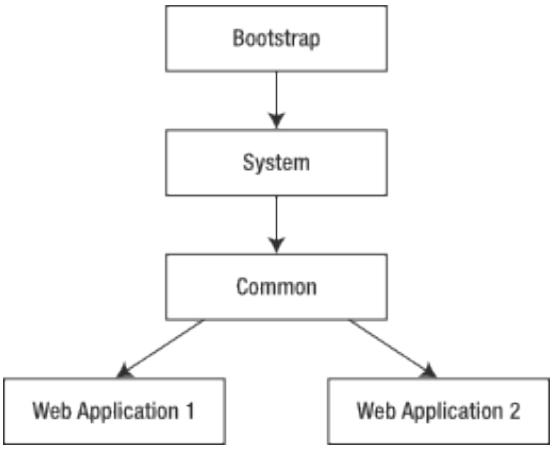

Figure 6.5: Tomcat classloader hierarchy

6.3.2 Implementation Details

The Tomcat extension library works by extending the main WebAppClassLoader which is responsible for loading classes in a web application (Kurniawan & Deck 2004). The extension modifies the loadClass method in the WebAppClassLoader to first interrogate every class being loaded by the classloader, if it has the @Gate and @ExitPoint annotations on class and method level respectively. Then if that condition is met the extension weaves in code using the Javassist library that communicates with the privacy engine.

The Tomcat extension library is a jar file that needs to be added to the Tomcat lib folder which contains a list of third party libraries used by Tomcat (Moodie & Mittal pg.30). This will ensure that the extended WebAppClassLoader is loaded into the classpath and is available for usage. Then a context would need to be configured to use the extended WebAppClassLoader. Each web application in Tomcat represents a context that is configured against a certain host (Vukotic & Goodwill 2011) and this context contains a loader. It is through this that a web application can be configured to use a custom classloader.

The classloader configuration is stored in a context.xml file and shown on figure 6.6:

```
<Context antiJARLocking="true" path="/">
 <Loader loaderClass="org.lesie.loader.web.TomcatLesieLoader"/>
</Context>
```

Figure 6.6 Context.xml configuration

This file can either be added to the $CATALINA_HOME/conf directory which will apply to all web applications running on that particular Tomcat instance, or alternatively the context.xml can be placed inside META-INF/ folder under the web application project and this will result in the extended classloader being applied to that particular web application.

The TomcatLesieLoader class extends the WebAppClassLoader class which is the default Tomcat classloader, and it is found in the catalina-X-X-X.jar library. The library version that is being extended is version 5.5.23. The extension looks like this:

```
public class TomcatLesieLoader extends WebappClassLoader{
```

Figure 6.7 TomcatLesieLoader class definition

Following that, the constructor initialises the main class that is responsible for the code weaving as follows:

```
public TomcatLesieLoader() {
 super();
 da = new DefaultAttacher();
}
```

Figure 6.8 TomcatLesieLoader constructor

Here 'da' is declared as a private class member of TomcatLesieLoader. Then the loadClass method is overridden such that it first performs the code weaving, if necessary, and then proceeds to load the class:

```
@Overridepublic Class loadClass(String name) throws ClassNotFoundException {
 try {
 da.weaveCodeToClass(name,this);
 } catch (Exception e) {
 log.log(Level.ALL,e.getMessage());
 }
 return super.loadClass(name);
}
```

Figure 6.9: TomcatLesieLoader loadClass method

The weaveCodeToClass method which is defined in the DefaultAttacher classis shown in figure 6.10

```
public void weaveCodeToClass(String name, ClassLoader cl) throws Exception {
 initClassPool(cl);
 CtClass lesieAnnotatedClass = classPool.get(name);
 weaveCode(lesieAnnotatedClass);
}
```

Figure 6.10: TomcatLesieLoader weaveCodeToClass method

In figure 6.10 the initClassPool initialises the Javasssist ClassPool object and inserts a classpath which is obtained from the current classloader, which is an instance of TomcatLesieLoader. Further more the weaveCode method is subdivided into sections that do the following:

- Checks if the class has @Gate and @ExitPoint annotations at class and method levels respectively.
- Weaves in the code that communicates with the Privacy Engine if the criteria is met for the previous point.

The code for the first criterion is in figure 6.11:

```
if (markCtClass.hasAnnotation(Gate.class)) {
 for (CtMethod ctMethod : markCtClass.getDeclaredMethods()) {
 if (ctMethod.hasAnnotation(ExitPoint.class)) {
```

Figure 6.11: Code to ascertain if class has @Gate and @ExitPoint annotations respectively

The code is self explanatory in that the markCtClass is of type CtClass which has methods hasAnnotation which return a true if a class has a specified annotation. In this case it is necessary to interrogate to see if they have the @Gate and if they do, then there is a further interrogation to determine if any of the methods have an @ExitPoint.

```
int sharingReqIndex = -1;
MethodInfo methodInfo = ctMethod.getMethodInfo();
LocalVariableAttribute table = (LocalVariableAttribute) methodInfo.getCodeAttribute().getAttribute(LocalVariableAttribute.tag);
CtClass[] parameterTypes = ctMethod.getParameterTypes();
for (int i = 0; i != parameterTypes.length; i++) {
    CtClass pType = parameterTypes[i];
    String className = pType.getName();
```

```
   if (className.equals(SharingRequest.class.getName())) {
       sharingReqIndex = i + 1;
    }
}
```

Figure 6.12: Code to get the index of a parameter of type SharingRequest

The ultimate aim of the code snippet above in Figure 6.12 is to determine the index of the SharingRequest object, which contains data used by the privacy engine, to determine whether to share a data subject's data or not. Finally the code responsible for the code weaving is represented, in Figure 6.13 below.

```
      if (sharingReqIndex > 0) {
         CtMethod genMethod = CtNewMethod.copy(ctMethod, ctMethod.getName(), markCtClass, null);
         String newMethodName = ctMethod.getName() + "$Impl";
         ctMethod.setName(newMethodName);
         //build method body
         StringBuffer body = new StringBuffer();
              body.append("{");              body.append("com.lesie.framework.service.PrivacyEngineService
 privacyEngineService = new com.lesie.framework.service.PrivacyEngineService();");
                                  body.append("com.lesie.framework.response.SharingResponse result =
 privacyEngineService.canShare($" + sharingReqIndex + ");");
         body.append("if(result.getR().get(\"code\").equals(\"PROCEED\")){" +
              newMethodName + "($$);"
              + "}else{System.out.println(\"Yeah I am printing\");}");
         body.append("}");

genMethod.setBody(body.toString());
```

```
        markCtClass.addMethod(genMethod);
        weaved = true;
      }
    }
}
markCtClass.toClass();
markCtClass.writeFile();
markCtClass.detach();
```

Figure 6.13: Code weaving code

The code in figure 6.13 creates a new method and gives it the name of the method being modified. Then that method is given a new name which is its current name affixed with a “$Impl”. Then in the newly created method a new instance of the PrivacyEngineService will be instantiated which will then make a call to the Privacy Engine with the SharingRequest object. This SharingRequest object is obtained by using the sharingRegIndex variable, which specified the index of the SharingRequest object in the original calling method. The index is used by Javassist, as Javassist has access to the parameters in the calling method, through a special parameter in the form of a $. Passing in a '$' followed by the sharingReqIndex will point to the correct SharingRequest object in the calling method.

6.4 Privacy Engine

The privacy engine is responsible for handling the decision whether to share a data subject's information with a third party or not. The privacy engine is able to interpret data subject access control for third parties and act upon the data subject's wishes. The privacy engine is built atop Apache Karaf which is an OSGi based distribution that can work atop any OSGi container. For this study it will be running atop the Apache Felix container.

The privacy engine will use the PostgreSQL 9 database to store its persistent data. Of course any type of relational database back end can be used in place of PostgreSQL. The Privacy Engine will need to store various data, particularly that data pertaining to what data can be shared with which third parties The database model will be explained later together with details of how it fits into supporting this requirement. The following technologies will be used in the construction of the Privacy Engine and will be explained briefly:

- OSGi
- Scala

OSGi is a Java modularity framework, its original design rationale was to provide a framework for embedded device manufactures that would allow them to easily make their devices integrate seamlessly with each other (OSGI Alliance 2012 pg.11). Its use has extended to other areas, in recent years. Middleware vendors have taken an interest in OSGi.

Middleware vendors use OSGi as a core component in their Middleware products, some have even rewritten their Middleware to take advantage of OSGi. Notable examples of Java middleware using OSGi as a building block are Oracle Glassfish (Glassfish, n.d), Apache ServiceMix (ServiceMix, n.d) and IBM Websphere (Krill, 2010). Figure 6.14 shows an architectural overview of OSGi.

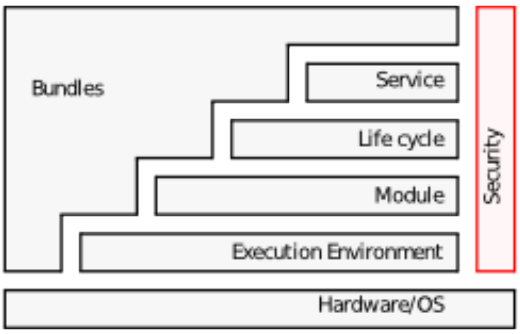

Figure 6.14: OSGi architecture (OSGi Aliance 2012, pg.12)

Just like Java EE, OSGi is a set of specifications which have different implementations. Notable examples being Apache Felix, Eclipse Equinox and Knopperfish. In turn, there is even higher level software which builds atop the OSGi infrastructure, in order to provide even more features than are specified in the OSGi specification. Apache Karaf sees itself as an OSGi distribution in the same way that Ubuntu, Fedora and SuSe see themselves as Linux distributions. Apache Karaf provides an extra set of features which simplify dealing with OSGi. Apache Karaf can sit atop any OSGi implementation.

Scala is multi-paradigm language that is fusion between functional and object-oriented programming that runs on the JVM (Odersky 2004). It was developed by Martin Odensky.

6.4.3 Implementation details

The implementation has a modular structure in the sense that it is composed of OSGi bundles, however not all the modules in the project are OSGi bundles. The modules that compose the Privacy Engine are:

- pe-api
- pe-assembly
- pe-model
- pe-distribution
- pe-policy

The project structure utilises both Maven and SBT build systems and each module uses one or the other of them. The main Privacy Engine root project encompasses all the Privacy Engine modules listed above and is a Maven project which has the pe-distribution and pe-asembly configured as sub-modules in its pom.xml. This effectively means that running a “clean compile” command on the main Privacy Engine project will propagate those commands to the pe-distribtion and pe-assembly projects. As for the pe-api, pe-policy and pe-model they are SBT projects and they are configured by a root level build.sbt file.

The pe-api module contains rest definitions which are declared in Scala using standard JAX-RS based annotations to declare restful webservices. CXF is the library utilised to enable rest capability in the module. CXF needs to be configured and this is achieved using blueprint XML configurations which are stored in the OSGI-INF/blueprint folder. Blueprint is a dependency injection solution for OSGi environments (OSGi Alliance 2010, pg.193). It is inspired by Spring dynamic modules which is an attempt to bring the Spring framework to the OSGi platform (Cogoluègnes 2010). Once the pe-api is deployed into an OSGi container which has blueprint and CXF services available, then the configured blueprint beans in rest.xml will be automatically constructed, and with that construction achieved the rest services will be created.

```
<cxf:bus id="cxfBus">
   <cxf:features>
      <cxf:logging/>
   </cxf:features>
</cxf:bus>


<jaxrs:server address="/api" id="restAPIManager">
   <jaxrs:serviceBeans>
      <ref component-id="apiService"/>
   </jaxrs:serviceBeans>
   <jaxrs:extensionMappings>
      <entry key="json" value="application/json"/>
      <entry key="xml" value="application/xml"/>
      <entry key="jsonp" value="application/x-javascript"/>
   </jaxrs:extensionMappings>
   <!-- custom providers -->
```

```
 <jaxrs:providers>
      <ref component-id="jsonProvider"/>
   </jaxrs:providers>
   <jaxrs:inInterceptors>
      <bean class="org.apache.cxf.jaxrs.provider.jsonp.JsonpInInterceptor"/>
   </jaxrs:inInterceptors>
   <jaxrs:outInterceptors>
      <bean class="org.apache.cxf.jaxrs.provider.jsonp.JsonpPreStreamInterceptor"/>
      <bean class="org.apache.cxf.jaxrs.provider.jsonp.JsonpPostStreamInterceptor"/>
   </jaxrs:outInterceptors>
</jaxrs:server>
```

Figure 6.15: Rest services blueprint configuration in pe-api OSGi bundle

The code in figure 6.15 is the blueprint configuration for the exposed rest service. The rest server is configured inside the <jaxrs:server> XML tags. It is configured to have root level mapping of “/api” and points to the “apiService” bean which is configured below. A provider is configured as well and it is the “jsonProvider” bean, also configured in the sample codebase below in figure 6.16. The “jsonProvider” bean uses a “scalaMapper” bean as its internal mapper. The reason for this is that without it the JacksonJsonProvider is unable to instantiate Scala based classes properly, since the JacksonJsonProvider is part of a Java based library.

```
<reference id="policyManager" interface="ac.za.cput.pe.policy.PolicyManager"/>
 <!-- Implementation of the rest service -->
 <bean id="apiService" class="ac.za.cput.pe.api.APIService">
    <property name="policyManager" ref="policyManager"/>
 </bean>
```

```
<bean id="jsonProvider" class="com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider">
   <property name="mapper" ref="scalaMapper"/>
</bean>
<bean id="scalaMapper" class="ac.za.cput.pe.api.mapper.ScalaMapper">
</bean>
```

Figure 6.16: PolicyManager blueprint bean

The APIService class which contains logic for the exposed canMove method that is a gateway into the privacy engine is laid out in figure 6.17:

```
@Path("/V1")
@Produces(Array("application/json"))
@WebService
class APIService {
  private var policyManager: PolicyManager = _
  @POST
  @Path("/canmove")
  @Consumes(Array(MediaType.APPLICATION_JSON))
  def canMove(request: SharingRequest): Either[Seq[ErrorResponse], SharingResponse] = {
    val validationResult = validateRequest(request)
    if (!validationResult)
      Left(Seq(ErrorCodes.ERR_SHARING_REQUEST_INVALID))
```

```
else {
    if (policyManager.validateHolderAndThirdParty(request.dataControllerKey.get,
     request.thirdPartyKey.get) == false) {
     Left(Seq(ErrorCodes.ERR_HOLDER_AND_THIRDPARTY_INVALID))
    } else {
     val auxillaryDatas: Seq[AuxillaryData] = request.data match {
      case Some(data) =>
       data.map(d => new AuxillaryData(d._1, d._2, 0)).toSeq
      case None =>
       Seq.empty
     }
                    val (halt, proceed) = policyManager.executeContextPolicies(request.contextKey.get,
request.dataSubjectKey.get,
      auxillaryDatas)
      .partition(_ == ContextStatus.HALT)
     if (halt.nonEmpty) {
      Left(halt.map(h => ErrorCodes.ERR_REQUEST_HALTED))
     }
     else {
      Right(SharingResponse("PROCEED", "PROCEED"))
     }
    }
   }
 }
```

Figure 6.17: APIService code

From the code above in figure 6.16 the canMove method is exposed as a rest service with URL mapping of “$LOCALHOST:{$CXF_PORT}/api/V1/canMove”. It is configured to accept JSON and its payload is automatically demarshalled into the SharingRequest object using the configured “jsonProvider” bean. Before control is passed into the policy managers to execute the policies configured, the request object is validated to make sure all the data required is supplied of which the data is:

- dataControllerKey – identifies organisation acting as data controller
- contextKey – identifies the context with which the data is being shared
- dataSubjectKey – identifies the data subject from whose data a sharing request is being made
- thirdPartyKey – represents the organisation acting as the third party
- data – supplementary data sent by the data controller to help in the processing to determine if data should be shared.

Depending on the result of the validation or policy manager, a response is returned wrapped in either a Left or Right object to signify failure or success respectively.

The second module pe-assembly is concerned with assembling modules into Karaf features. It achieves this by using a Karaf-Maven plugin which is activated when a Maven install command is executed against the module. The plugin will take all the dependencies listed in the dependencies section in a pom.xml and construct a features XML file based on that. This generated features XML file will be outputted into the target folder of the pe-assembly module, and will also be added to the local Maven repository on the development machine.

A features XML contains a list of OSGi bundles and their locations, in this case the pe-api, pe-policy and pe-model. These features XMLs can be deployed into Karaf and Karaf will automatically resolve these bundles using a Maven repository resolver. This features functionality is the extra set of features that Karaf provides over and above what a vanilla OSGi container, like Felix or Equinox, provides.

The pe-policy module contains code that determines the conditions under which to share the data subject's data. In this implementation there is only rudimentary code that accepts any request or rejects all requests. This code is represented in the AllowPolicyHandler.scala and the RejectPolicyHandler.scala files. See figure 6.18, the code either returns a “HALT” or “PROCEED” string to signify what a failure or success

```
class RejectPolicyHandler extends PolicyHandler {
  override def handle(data: Seq[AuxillaryData]): ContextState = {
    ContextStatus.HALT
  }
}
sealed class AllowPolicyHandler extends PolicyHandler {
  override def handle(data: Seq[AuxillaryData]): ContextState = {
    ContextStatus.PROCEED
  }
}
```

Figure 6.18: RejectPolicyHandler and AllowPolicyHandler code

```
sealed class PolicyManagerImpl extends PolicyManager {
  private var privacyEngineRepository: PrivacyEngineRepository = _
  private val policies = Map("ALLOW" -> new AllowPolicyHandler,
    "REJECT" -> new RejectPolicyHandler)
  override def validateHolderAndThirdParty(dataHolderKey: String, thirdPartyKey: String): Boolean ={
    privacyEngineRepository.organisationExist(dataHolderKey) &&
    privacyEngineRepository.organisationExist(thirdPartyKey)
  }
```

```
  override def executeContextPolicies(contextKey: String, ownerKey: String, data: Seq[AuxillaryData]):
Seq[ContextState] = {
   privacyEngineRepository.contextExist(contextKey) match {
    case true =>
     privacyEngineRepository.loadPoliciesForContextByOwner(contextKey, ownerKey)
      .map(p => executePolicy(p.name, data)).toList
    case false =>
     Seq.empty
   }
  }
```

Figure 6.19: PolicyManagerImpl code

The PolicyManagerImpl class in figure 6.19 is an implementation of the PolicyManager trait exposes the validateHolderAndThirdParty and executePolicies methods. It is configured as a blueprint bean in this module, and it is accessible by other modules by referencing it. This class in turn delegates some code to the privacyEngineRepository, which is part of the pe-model module. When the policyManger bean is configured it references the privacyEngineRepository bean, and it is injected using blueprint.

The exposed executePolicies method loads policies from the database by contextKey, then executes each policy in turn with the supplied auxiliary data and maps the result to the list, that is returned to the callee. See figure 6.20

```
  private def executePolicy(policyName: String, data: Seq[AuxillaryData]): ContextStatus.ContextState = {
   val policyNameUpper = policyName.toUpperCase
   policies(policyNameUpper) match {
    case handler: PolicyHandler =>
     handler.handle(data)
    case _ =>
```

```
      ContextStatus.UNDEFINED
    }
  }
  def setPrivacyEngineRepository(repo: PrivacyEngineRepository) =
    privacyEngineRepository = repo
}
```

Figure 6.20:executePolicy method

The code in the pe-policy module is not as expressive as the solution presented by Abour-Tair (2006) which allows for the usage of an ontology based language called F-Logic that can express German privacy laws amongst other things. In practical terms this ability to express privacy wishes through an ontology based language would not add much value as it is unlikely that the average person can write F-Logic to express their privacy wishes. However the infrastructure is there with the entire Lesie platform that is being presented in this study, to have custom handlers that process based on some complex logic.

The final OSGi module pe-model contains the domain model used in this study. The domain model is represented in Scala using the squeryl ORM library. Squeryl has a concept of Schemas where database tables are defined as attributes in a Schema object. The table attributes are mapped to a Scala class with attrubutes in the Scala class mapping to a corresping database column of the same name. This schema object is expressed in figure 6.21

```
object PeDB extends Schema {
  val organisations = table[Organisation]("organisation")
  val auxillaryData = table[AuxillaryData]("auxillary_data")
  val shares = table[Share]("share")
  val contexts = table[Context]("context")
  val dataOwners = table[DataOwner]("data_owner")
  val policyContexts = table[PolicyContext]("policy_context")
  val policies = table[Policy]("policy")
```

Figure 6.21: Mapping database tables to domain entities.

This code snippet in figure 6.21 declares squeryl database tables that map to a corresponding table in the database.

```
  val dataHolderToShare =
    oneToManyRelation(organisations, shares)
      .via((o, s) => o.id === s.dataHolderId)
  val thirdPartyToShare =
    oneToManyRelation(organisations, shares)
      .via((o, s) => o.id === s.thirdPartyId)
  val shareToAuxillaryData =
    oneToManyRelation(shares, auxillaryData)
      .via((s, a) => s.id === a.shareId)
  val contextToShare =
    oneToManyRelation(contexts, shares)
      .via((c, s) => c.id === s.contextId)
```

```
val contextToPolicyContext =
    oneToManyRelation(contexts, policyContexts)
      .via((c, pc) => c.id === pc.contextId)
  val policyToPolicyContext =
    oneToManyRelation(policies, policyContexts)
      .via((p, pc) => p.id === pc.policyId)
  val dataOwnerToPolicyContext =
    oneToManyRelation(dataOwners, policyContexts)
      .via((o, pc) => o.id === pc.dataOwnerId)
}
```

Figure 6.22: Code to define how the domain entities relate to each other

This code in figure 6.22 expresses the relationship that the tables have to each other. This relationship will be better visualised in an ERD diagram to follow later on figure 6.25. The code that follows afterwards is just a declaration of the domain model classes that map to database tables. Each of them will extend from the BaseEntity class that in turn extends the KeyedEntity class.

The KeyedEntity class is a squeryl class that designates a table that has a unique id key with its type designated by the type in square brackets, hence the extended class is written out as KeyedEntity[Long]. The BaseEntity declares the id type so that the domain classes that extend it do not have to redefine the id attribute each time. See figure 6.23

```
class BaseEntity extends KeyedEntity[Long] {
  val id: Long = 0
}
```

Figure 6.23: BaseEntity

```
sealed class Organisation(val name: String, val key: String) extends BaseEntity {

  lazy val thirdPartyShares: OneToMany[Share] = PeDB.thirdPartyToShare.left(this)

  lazy val dataHolderShares: OneToMany[Share] = PeDB.dataHolderToShare.left(this)

}

sealed class Share(val date: LocalDate, val status: String, val dataHolderId: Long, val thirdPartyId: Long, val
contextId: Long)

  extends BaseEntity {

  lazy val dataHolder: ManyToOne[Organisation] = PeDB.dataHolderToShare.right(this)

  lazy val thirdParty: ManyToOne[Organisation] = PeDB.thirdPartyToShare.right(this)

  lazy val context: ManyToOne[Context] = PeDB.contextToShare.right(this)

}



sealed class AuxillaryData(val key: String, val value: String, val shareId: Long) extends


BaseEntity {

  lazy val share: ManyToOne[Share] = PeDB.shareToAuxillaryData.right(this)

}

sealed class Context(val key: String, val name: String) extends BaseEntity {

  lazy val shares: OneToMany[Share] = PeDB.contextToShare.left(this)

  lazy val policyContexts: OneToMany[PolicyContext] = PeDB.contextToPolicyContext.left(this)

}

sealed class PolicyContext(val contextId: Long, val policyId: Long, val dataOwnerId: Long)

  extends BaseEntity {

  lazy val context: ManyToOne[Context] = PeDB.contextToPolicyContext.right(this)

  lazy val policy: ManyToOne[Policy] = PeDB.policyToPolicyContext.right(this)

  lazy val dataOwner: ManyToOne[DataOwner] = PeDB.dataOwnerToPolicyContext.right(this)

}
```

```
sealed class DataOwner(val name: String, val surname: String, val key: String) extends BaseEntity {

  lazy val policyContexts: OneToMany[PolicyContext] = PeDB.dataOwnerToPolicyContext.left(this)

}


sealed class Policy(val name: String) extends BaseEntity {

  lazy val policyContexts: OneToMany[PolicyContext] = PeDB.policyToPolicyContext.left(this)

}
```

Figure 6.24: Domain entities definition

Having listed the code that describes the domain models and their relationship to each other in figure 22,23 and 24 respectively. Figure 6.25 depicts these relationships in an entity relationship diagram (ERD) that depicts this pictorially. The ERD expresses a structure that allows a data subject to dictate a data sharing policy for a specific data sharing context.

The significance of having a domain model that can store the wishes of the data subject's preferences, regarding the sharing of their data by data controllers, addresses the group of information processing concerns raised by Solove (2009, pg.104) in his privacy taxonomy. In a data sharing context between the data controller and third parties, the ERD in figure 6.25 is explained.

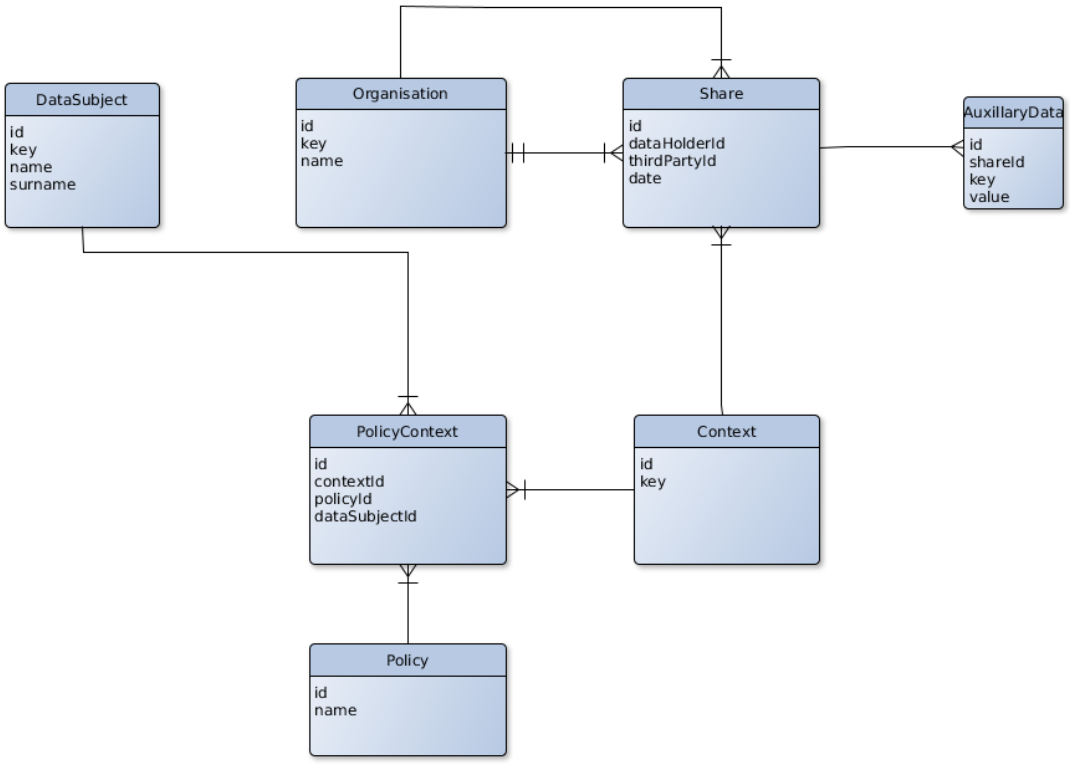

Figure 6.25: Privacy engine ERD

DataSubject – Stores the data subject's data

Policy – Stores the different types of policies available in the privacy engine. These are the policy handlers available in the pe-policy module, currently AllowPolicyHandler and RejectPolicyHandler.

Context – Represents the context with which data can be shared. e.g marketing, analytics, fraud etc.

PolicyContext – This is a joining entity, that joins the Policy, Context and Owner entities together. It effectively represents the data subject's wishes regarding the sharing of their data for a certain context, that is, whether to reject or allow in this current state.

Organisation – This represents an organisation which can either be a data controller or a third party depending on which side of the relationship it assumes in the Share entity.

Share – This represents the sharing activities that took place in the privacy engine. Data to this table is inserted whenever there is a sharing request.

AuxillaryData – This represents additional data that has been sent as part of the request to share data. This data will be used by the policy handlers to determine whether to share data or not.

Code that is executed against the ERD, and has received mention in the section discussing the pe-policy is based in the PrivaceEngineRepository class, is listed in Figure 6.26:

```
sealed class PrivacyEngineRepositoryImpl(val session: Session) extends PrivacyEngineRepository {
  override def contextExist(key: String) =
    using(session){
      contexts.where(_.key === key).nonEmpty
    }
  override def organisationExist(key: String) =
    using(session) {
      organisations.where(_.key === key).nonEmpty
    }
  override def loadPoliciesForContextByOwner(contextKey: String, ownerKey: String) =
    using(session) {
      val q = from(PeDB.contexts, PeDB.policies, PeDB.dataOwners, PeDB.policyContexts)((c, p, o, pc) =>
        where(c.key === contextKey and o.key === ownerKey
          and c.id === pc.contextId and p.id === pc.policyId
        and o.id === pc.dataOwnerId).
          select(p)
          orderBy (p.id)
      )
      val result = q.toList
      result
    }
```

Figure 6.26: PrivacyEngineRepositoryImpl code

The class exposes the contextExist and organisationExist methods which query the database to ascertain if a match exists, where the supplied parameter key matches any data in the key column. Then finally it exposes the loadPoliciesForContextByOwner which loads policies configured for a data subject for a certain context. In all these methods the database is accessed via the squeryl API which on closer observation looks very similar to SQL. This is an example of an internal DSL discussed in the previous chapter in Section 4.3.3.1.

The last module is pe-distribution which is a Maven project and its main purpose is to create a custom Karaf distribution that has the following features installed and active at start up:

- standard
- ssh
- management
- configuration
- http
- CXF
- pe-assembly

The pe-assembly is the feature that is generated by the pe-assembly module and the CXF feature is the module needed in order to have the rest endpoints enabled for the pe-api module.

In conclusion the Lesie platform is made up of three components. The client-api and the Tomcat extension, which are both jar files, and thirdly the privacy engine, which is a custom Karaf distribution with Lesie specific modules deployed on it, such that it can complete the privacy enhancing features of the Lesie platform.

Chapter 7.

Discussion of results

This chapter will detail the results obtained from this research endeavour. The first section will discuss the lessons learnt while building and designing the artefact, mostly around how the code evolved throughout the research. Following that an evaluation of the artefact's efficacy in the form of unit tests carried out to test individual classes, then a test on multiple Tomcat version, to ascertain how well the artefact remains functional across Tomcat versions.

The development process required to develop all three major components of artefacts will be discussed. In particular how debugging and deployment was approached for all three components. Following the discussions on the evaluations the contributions of this research will be weighed in and finally future directions.

7.1 Challenges and lessons learnt

The codebase has gone through various changes throughout this study. The initial design remained relatively the same, with the client library communicating with a privacy engine using code that has been weaved in. What has changed is the mechanisms to achieve this. The initial design was to use Java sockets through the Java NIO library (Hitchens 2012). With this approach the PrivacyEngineService in the client library would have selectors listening on multiple channels for incoming data, and equally there would be similar mechanisms on the privacy engine side that handles incoming data from the client library.

This approach proved more complex from a coding perspective than setting up rest service endpoints as there were low-level constructs to deal with. The REST approach was ultimately favoured over a socket based approach using the Java NIO library, as with the NIO library there were low level networking constructs to deal with, and with REST there is no low level networking contstructs. Another major shift in the structure of the codebase was a switch from Java to Scala for the privacy engine.

Initially the privacy engine went by the name LesieS the S denoted the server side processing part of the Lesie platform. The name change to privacy engine seemed appropriate as that is what the server side is basically doing by being the component that determines how data is shared.

The switch to Scala proved to be beneficial as more was achieved with significantly smaller codebase. The configuration is carried out using SBT as opposed to Maven and is generally less verbose, for example there is only one build.sbt file to manage all the OSGi modules, as opposed to having a pom.xml per module when using Maven. Configuring OSGi bundles in Maven was error prone and difficult to trace down errors, as Maven did not provided detailed error reporting and tended contain unnecessary information. SBT was better in that regard and using the sbt-osgi plugin was a seamless process.

The other front is the amount of hibernate boilerplate code that was eliminated with the switch to squeryl. Compared to the schema definition found in the previous chapter in figure 6.21 up until figure 6.24. Each domain entity needs to have its own class file and needed to be annotated accordingly with JPA specific annotations to signify that the class is an entity class and on attributes to identify the relational model. There was also a persistance.xml to configure the behaviour of hibernate on whether to create new database tables at startup, or how to connect to the database and which database type to generate SQL for.

At the start of the project karaf was on version 2.3.1 and there was quite a bit of boilerplate code involved in setting up a custom distribution. Version 3.0 addressed this by making it possible to specify features to load at application startup by just adding a dependency to a karaf feature in the project pom.xml file, and adding it to the configuration section of the karaf maven plugin as shown below on figure 7.1:

```
<plugin>
   <groupId>org.apache.karaf.tooling</groupId>
   <artifactId>karaf-maven-plugin</artifactId>
   <extensions>true</extensions>
   <version>${karaf.version}</version>
   <configuration>
      <!-- no startupFeatures -->
      <bootFeatures>
         <feature>standard</feature>
         <feature>ssh</feature>
         <feature>management</feature>
         <feature>config</feature>
         <feature>http</feature>
         <feature>cxf</feature>
         <feature>pe-assembly</feature>
         <feature>jndi</feature>
         <feature>transaction</feature>
      </bootFeatures>
   </configuration>
</plugin>
```

Figure 7.1: Custom karaf boot features configuration with Maven plugin

Highlighted in red is the feature that has been generated by the pe-assembly module. This concludes the section on the lessons learnt throughout this study, the next section will discuss the evaluation of the artefact in terms of its efficacy and practicality.

7.2 Evaluation

This section will discuss the evaluation of the artefact in terms of its efficacy, practicality and the development process in terms of the debugging, testing and deployment process. First a look at the unit tests used for the privacy engine and the client library. Then a web application that was used to test the effifacy of the platform, with all the components that make up Lesie working together.

The pe-model module's PrivacyEngineRepositoryImpl class was unit tested using the scalatest library, the unit tests are as follows on figure 7.2:

```
class PrivacyEngineRepositorySpec extends FlatSpec with Matchers with BeforeAndAfter{
    var session: Session = _
    var privacyEngineRepo: PrivacyEngineRepositoryImpl = _
    before{
      session = PESessionFactory.createSession
      privacyEngineRepo = new PrivacyEngineRepositoryImpl(session)
    }
    it should "return false if a context does not exist" in {
      val result = privacyEngineRepo.contextExist("none")
      assert(result == false)
    }
```

```
it should "return true if a context exist" in {
    val result = privacyEngineRepo.contextExist("ct1")
    assert(result == true)
  }
it should "return false if an organisation does not exist" in {
    val result = privacyEngineRepo.organisationExist("none")
    assert(result == false)
  }
  it should "return true if an organisation exist" in {
    val result = privacyEngineRepo.organisationExist("e123")
    assert(result == true)
  }
  it should "return a list of policies by context for a data subject" in {
    val results = privacyEngineRepo.loadPoliciesForContextByOwner("ct1", "ck1")
    assert(results.nonEmpty)
  }
}
```

Figure 7.2: Unit tests for the PrivacyEngineRepositoryImpl class

Exposed methods on the PrivacyEngineRepositoryImpl were unit tested, the contextExist method was tested by doing a positive and negative test. This is achieved by passing in a non existing context key and an existing context key respectively. The same is also done for the organisationExist method. Finally the loadPoliciesForContextByOwner is tested by doing a positive case that loads a sequence of policies for a certain context for a data subject. The second unit test is for the PrivacyEngineService class found in the client library. The unit test is as follows on figure 7.3

```
@Testpublic void testCanShare(){
    SharingRequest request = new SharingRequest("e123", "ck1", "w123", "ct2",
            new HashMap<String, String>());
    SharingResponse result = null;
    try {
        result = privacyEngineService.canShare(request);
    } catch (Exception e) {
        e.printStackTrace();
    }
    Assert.assertEquals("Not a success", "PROCEED", result.getR().get("status"));
}
```

Figure 7.3: Unit test to test the PrivacyEnginService

The unit test creates a SharingRequest object and passes it into the canShare method. This method underneath does a call to the privacy engine and determines the data sharing eligibility of the request. The test data supplied corresponds to test data in the privacy engine database and therefore will return a PROCEED to signify that the data sharing request may proceed. At this point the test runs successfully provided that the privacy engine is running, the URL to the privacy engine has been hard coded.

The testing approach for the Tomcat extension had to be different given the inability to test the TomcatLesieLoader class in isolation, since its parent class WebAppClassLoader is tightly coupled to its Tomcat codebase. The tests where instead carried out using a web application that makes use of the Lesie platform. The web application tests out all three components of the Lesie platform. The web application has a servlet that is shown pictorially on figure 7.4:

```
public class ExampleServlet extends javax.servlet.http.HttpServlet {

                protected void doGet(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response) throws javax.servlet.ServletException, IOException {

    SharingRequest sharingRequest = new SharingRequest("w123", "ck1", "e123", "ct1", null);

    ShareService shareService = new ShareService();

    shareService.shareData(sharingRequest, "extra");

  }

}
```

Figure 7.4: ExampleServlet code

The servlet creates a SharingRequest object with data that represents a relationship in the privacy engine, that allows for the sharing of a data subject's information from a data controller to a third party. Then this is passed into the shareData method in the ShareService class. This ShareService class is annotated with the @Gate and @ExitPoint annotation, the shareData method has one of its parameters as a SharingRequest type and the other as a String, this is required since when the code is weaved there is an explicit search for a parameter of type SharingRequest. This requirement may be a downside for developers as it puts a constraint on methods annotated with @ExitPoint picture below on figure 7.5.

```
@Gatepublic class ShareService {
  @ExitPoint
  public void shareData(SharingRequest sharingRequest, String otherStuff) {
    doSomething();
  }
  private void doSomething(){
    System.out.println("doSomething");
  }
}
```

Figure 7.5: ShareService code

The web application includes the client library as part of its third party libraries. The Tomcat utilised has the lesie-loader.jar and lesie-api.jar added to its lib folder so that the TomcatLesieLoader and annotations are available in the Tomcat classpath. In addition TomcatLesieLoader has been configured to be the main classloader for the web application through a context.xml placed in the META-INF directory of the web application. The context.xml looks as such on figure 7.6:

```
<Context antiJARLocking="true" path="/">

   <Loader loaderClass="org.lesie.loader.web.TomcatLesieLoader"/>

</Context>
```

Figure 7.6: Context.xml

The client library was tested using unit tests and it enabled it to be developed in isolation up until the end stages. The privacy engine was mostly developed in isolation as well. Testing for the privacy engine included the use of unit tests for individual class testing, and the entire privacy engine component was tested using the postman application, which is a chrome application that has handy tools for working with rest endpoints.

The development process for the client library was straight forward. The component is a Maven project and was built using the “clean install” Maven commands. This created the jar files, and for testing the PrivacyEngineService code, the unit test was run from within the IDE.

The Tomcat extension was also a Maven project and had the client library setup as a third party library project, therefore the build step was setup such that the client library builds before the Tomcat extension so that any new code is included in the Tomcat extension jar file. After building the jar file the next step would be to copy the jar file into the Tomcat lib folder. This step was automated by using a shell script that uses the 'mv' shell command to move the jar into the Tomcat lib folder.

Following that the sample web application was built and it would pull in the latest client library and it was also placed into the webapps folder in Tomcat. Then finally Tomcat would be started. Only then can any new code changes in the Tomcat extension be tested or debugged. This process is cumbersome and there are to many manual steps. This is not as straight forward as the development of Java EE applications whereby a WAR file is automatically deployed by the IDE into the application server and everything is taken care of. This process could be improved in future to reduce the number of manual steps the developer has to carry out.

Practical aspects were considered in the usage of the Lesie privacy platform. The efficacy of this Tomcat extension was tested by running the example web application against various versions of Tomcat. In total the example web applications was run on a total of ten versions of Tomcat. Four versions were from the 6.x.xx series a further for against the 7.x.xx series, and finally the last two versions against the 8.x.xx series. The versions per release series where chosen based on the gap between there releases, which was usually a year. The results are as follows:

Table 7.1: Lesie platform functional results on various versions of Tomcat

Tomcat version	Status
6.0.36	Working
6.0.37	Working
6.0.43	Working
6.0.44	Working
7.0.47	Working
7.0.57	Working
7.0.61	Working
7.0.65	Working
8.0.15	Not Working
8.0.28	Not Working

In about 80% of the cases the Lesie platform performed as expected. The remaining 20% failed as a result of code changes that had taken place in the WebAppClassLoader. In the TomcatLesieLoader a reference is made to an attribute resourceEntries that is found in the WebAppClassLoader. It seems that in the 8.x.xx series there is some code changes that has seen the attribute removed from the WebAppClassLoader. This indicates the slight fragility inherent in tight coupling to Tomcat internal code. In future this can be mitigated against by following a defensive programming approach and pegging the development of the framework against that of Tomcat.

7.3 Contributions

This section will detail the contributions of this study and how they relate to the main question in section 1.3.1.1 and its correspnding sub-questions in section 1.3.1.2. In the introduction the problem of data controllers sharing data subject's data was laid out, and with further engagement with the literature it was found that various people are trying to address this issue. In this study we approached this problem from the perspective of the developer and aimed to develop an artefact that can be used by developers to address this problem. This study laid a foundation for a privacy enhancing platform that is developer focused, and targets mainstream platforms like Java specifically the Tomcat servlet engine, which is widely used in the development of information systems.

This study described an approach that can be taken in constructing a privacy enhancing platform, that uses AOP techniques to minimize the amount of boilerplate code that a developer would have to write, in order to build in privacy enhacing features into software systems. These AOP techniques are implemented by extending the Tomcat server.

An artefact was developed demonstrating this and is distributed as a set of Jar files, specifically the client library and the middleware extension. The privacy engine is distrubuted as a stand alone server that is a custom karaf distribution. In addition to the executable binaries that consist of the Lesie plaform, there is source code that describe how this platform is constructed with the full versioning history in Git.

These contributions map to the main question in section 1.3.1.1 in the following way. The contributions clearly provide a platform, that is an extension of middleware used by developers in building software systems, that enables developers to write privacy enhanced software that can ultimately give data subject more control of their data. The sub-questions are discussed below

The mapping of the sub-questions is as follows for the various questions:

How can technology enable people to have more control or visibility over their data?

This question was not sufficently answered by the main empirical activities of the research as the research only showed how technology that enables such an activity to take place.

How can middleware be extended to provide a privacy enhancing platform to developers?

This question was sufficently answered as the Tomcat server's default classloader was extended, which enabled it to weave in code to classes that have been annoted with certain annotations.

How can a privacy enhancing API be designed?

This question was answered as an API was constructed and guided by theory from Fowler(2010) specifially annotations which are a type of internal DSL.

7.4 Future direction

The current solutions has drawbacks in the sense that it is limited to the JVM platform and it is currently only supported in the Tomcat application server. The client library and Tomcat extension can be ported to other platforms specifically the .Net platform. Since the Tomcat extension makes use of AOP techniques there are AOP libraries available in the .Net platform. They are discussed by Groves (2013). The Tomcat extension can also be extended to other types application servers namely Glassfish, JBoss and even having a standard Java SE classloader, for those applications that run independently of an application server.

The next thing would be to build more advanced policy handlers as the ones presented in the study either reject or accept a data sharing request. These advanced policy handlers can be inspired by studies discussed in the literature (Abou-Tair 2006; Birrell & Schneider 2014). Secondly is to build a user interface that data subjects can use to interact with the privacy engine and specify their data sharing preferences.

Chapter 8.

Conclusion

In this study we set out to answer the question of how middleware can be extended to provide a privacy enhancing platform to developers. The aim was to extend Tomcat to provide this privacy enhancing platform and client library that developers would use to interact with the platform.

Due to multifaceted nature of privacy. Theories from Solove and Nissenbaum were used as anchors to ground the study. Solove provided a privacy taxonomy which was very helpful in explaining some positions regarding the problem identified in the study. Nissenbaum provided the contextual privacy theory which was used to further give weight to arguments made in the study surrounding the conflict of interest between data sharing and data privacy.

The various approaches that technology was being used in the fight against privacy violations as identified by Solove and Nissenbaum were identified and discussed. Further more studies were identified that were discussing a privacy enhancing solution, and their designs were analysed in order to provide input to the artefact being developed in this study. Studies that were extending middleware and the development of domain specific languages were looked at, in order to also provide input in the construction of the artefact.

The tools and systems used by developers to construct information systems were discussed with a hope of giving an understanding on how these systems that collect data from data subjects are built. This was done to place the artefact that was to be built in context of these tools and systems, so that it is known where in the development stack the artefact fits in.

The outcome of this study was a privacy preserving platform dubbed Lesie privacy enhancing platform, that allows developers to write privacy enhanced software by simply adding annotations at the relevant places. Then another component in the artefact at load time will weave in code that communicates with the privacy engine using AOP techniques.

It is hoped this study has laid a foundation that can be carried forward in the construction of privacy enhancing platforms aimed at mainstream development platforms. The privacy identified by Solove and Nissenbaum and discussed in this study are to some varying degrees mitigated by the Lesie platform in theory. It is hoped future studies can test this in a real environment as opposed to an artificial one, to get the extend at which this is achieved.

References

[no author]. (2013). GlassFish Server Open Source Edition Application Development Guide. 4th ed. Oracle. https://glassfish.java.net/docs/4.0/application-development-guide.pdf 28 March 2015.

Abou-Tair, Dhiah el Diehn I. (2006). A Framework Ensuring Privacy in a Distributed Environment. In Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services. PST '06. New York, NY, USA: ACM, pp. 47:1–47:6. Available at: http://doi.acm.org/10.1145/1501434.1501490.

Ackerman, S. and Ball, J. (2013). NSA loophole allows warrantless search for US citizens' emails and phone calls. [online] the Guardian. Available at: https://www.theguardian.com/world/2013/aug/09/nsa-loophole-warrantless-searches-email-calls [Accessed 26 Jun. 2016].

Alag, S.(2008).Collective Intelligence in Action: Manning Pubs Co Series

Anciaux, N., Bouganim, L. & Nquyen, B. (2012). Trusted cells: A sea change for personal data services., (August). Available at: https://pure.itu.dk/ws/files/39499980/ITUTR2012158pdf.pdf [Accessed October 19, 2014].

Andrews, L.(2012). 'Facebook is Using You',New York Times ,4 February,viewed 11 July 2015,http://www.nytimes.com/2012/02/05/opinion/sunday/facebook-is-using-you.html?_r=1&pagewanted=all

ARM. (n.d). TrustZone. [ONLINE] Available at:http://www.arm.com/products/processors/technologies/trustzone/index.php. [Accessed 24 November 14].

Axonframework.org, (n.d.). AxonFramework. [online] Available at: http://www.axonframework.org [Accessed 15 Nov. 2015].

Babaioff, M., Kleinberg, R. and Leme, R. (2012). Optimal Mechanisms for Selling Information. [online] Arxiv.org. Available at: http://arxiv.org/abs/1204.5519v1 [Accessed 19 Jul. 2015].

Barry, D. and Dick, D. (2013). Web services, service-oriented architectures, and cloud computing. Waltham, MA: Morgan Kaufmann.

Bass, L., Clements, P. and Kazman, R. (2013). Software architecture in practice. Upper Saddle River, NJ:

Addison-Wesley.

Beckwith, N. and Romoff, J. (2015). Transforming Health Care THE YEAR IN REVIEW FISCAL YEAR 2015. 1st ed. [ebook] Available at: http://www.upmc.com/about/finances/Documents/2015-annual-report-print.pdf [Accessed 28 Oct. 2015].

Betts, D., Dominguez, J., Melnik, G., Simonazzi, F., & Subramanian, M. (2013). Exploring CQRS and Event Sourcing: A Journey into High Scalability, Availability, and Maintainability with Windows Azure (1st ed.). Microsoft patterns & practices.

Beynon-Davies, P. (2004). *Database systems*. Basingstoke: Macmillan.

Birrell, E., & Schneider, F. B. (2014). Fine-Grained User Privacy from Avenance Tags⋆. Retrieved from http://www.cs.cornell.edu/fbs/publications/avenanceHotPET.pdf [Accessed 31 Aug. 2015].

Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61–72.

Camel.apache.org, (n.d.). *Apache Camel: Cafe Example*. [online] Available at: http://camel.apache.org [Accessed 13 Dec. 2014].

Cavoukian, A. (2010). *Privacy by Design The 7 Foundational Principles Implementation and Mapping of Fair Information Practices*. Available at: https://www.ipc.on.ca/images/Resources/pbd-implement-7found-principles.pdf [Accessed 18 Jul. 2015].

Codd, E.F. (1983). A relational model of data for large shared data banks.

Cogoluègnes`, A., Templier, T. and Piper, A. (2011). Spring dynamic modules in action. Greenwich [Conn.]: Manning.

computerworld. (2012). AMD adds ARM processor as it looks beyond x86. [ONLINE] Available at:http://www.computerworld.com/article/2504305/computer-hardware/amd-adds-arm-processor-as-it-looks-beyond-x86.html. [Accessed 24 November 14].

Connor, M. (2007). 225287 – Remove p3p from the default build. [online] Bugzilla.mozilla.org. Available at: https://bugzilla.mozilla.org/show_bug.cgi?id=225287 [Accessed 26 Jul. 2015].

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., … Woodford, D. (2013). Spanner: Google&Rsquo;s Globally Distributed Database. ACM Trans. Comput. Syst., 31(3), 8:1–8:22.

doi:10.1145/2491245

Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M. and Reagle, J. (2002). The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. [online] W3.org. Available at: http://www.w3.org/TR/2002/REC-P3P-20020416/ [Accessed 1 Nov. 2015].

Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J., Dobbs, B., Egelman, S., Hogben, G., Humphrey, J., Schunter, M., Stampley, D. and Wenning, R. (2006). The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. [online] W3.org. Available at: http://www.w3.org/TR/P3P11/ [Accessed 26 Jul. 2015].

Cranor, L. F., Idouchi, K., Leon, P. G., Sleeper, M., & Ur, B. (2013). Are They Actually Any Different ? Comparing Thousands of Financial Institutions ' Privacy Practices. The Twelfth Workshop on the Economics of Information SeCurity (WEIS 2013) , June 11–12, 2013, Washington, DC.

De Goede, M. (2014). The Politics of Privacy in the Age of Preemptive Security. Int Polit Sociol, 8(1), pp.100-104.

Dennis, A., Wixom, B. and Tegarden, D. (2012). Systems analysis design, UML version 2.0. Hoboken, NJ: John Wiley & Sons.

Diaz, C., Tene, O. and Guerses, S. (2013). Hero or Villain: The Data Controller in Privacy Law and Technologies. [online] Ssrn.com. Available at: http://ssrn.com/abstract=2321480 [Accessed 26 Jul. 2015].

Dingledine, R., Mathewson, N., & Syverson, P. (2004). Tor: The Second-generation Onion Router. In Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13 (p. 21). Berkeley, CA, USA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1251375.1251396

Emekci, F., Agrawal, D. & El Abbadi, A., (2005). ABACUS: A Distributed Middleware for Privacy Preserving Data Sharing Across Private Data Warehouses. In Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware. Middleware '05. New York, NY, USA: Springer-Verlag New York, Inc., pp. 21–41. Available at: http://dl.acm.org/citation.cfm?id=1515890.1515892.

Erdweg, S.; Storm, T.; Völter, M.; Boersma, M.; Bosman, R.; Cook, W.; Gerritsen, A.; Hulshout, A.; Kelly, S.; Loh, A.; Konat, G. D. P.; Molina, P.; Palatnik, M.; Pohjonen, R.; Schindler, E.; Schindler, K.; Solmi, R.; Vergu, V. A.; Visser, E.; Vlist, K.; Wachsmuth, G. H. & Woning, J. (2013), The State of the Art in Language Workbenches, *in Martin Erwig; Richard F. Paige & Eric Wyk*, ed., 'Software Language Engineering' , Springer International Publishing, , pp. 197--217 .

Eschen, R. (2009). ICEfaces 1.8. Birmingham, UK: Packt Pub.

Fall, K. and Stevens, W. (2011). TCP/IP illustrated. Upper Saddle River, NJ: Addison-Wesley.

Färber, F., Cha, S. K., Primsch, J., Bornhövd, C., Sigg, S., & Lehner, W. (2012). SAP HANA database: data management for modern business applications. ACM Sigmod Record, 40(4), 45–51.

Fremantle, P. et al., (2010). Carbon: Towards a Server Building Framework for SOA Platform. In Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing. MW4SOC '10. New York, NY, USA: ACM, pp. 7–12. Available at: http://doi.acm.org/10.1145/1890912.1890914.

Friedman, D. and Wand, M. (2008). Essentials of programming languages. Cambridge, MA: MIT Press.

Foster, E. (2014). Software Engineering: A Methodical Approach. Apress.

Fowler, M. (2002). Patterns of enterprise application architecture. Boston: Addison-Wesley.

Fowler, M. (2005). InversionOfControl. [online] martinfowler.com. Available at: http://martinfowler.com/bliki/InversionOfControl.html [Accessed 10 Aug. 2015].

Fowler, M. (2010). Domain-specific languages. Upper Saddle River, NJ: Addison-Wesley Professional.

G, A. (2014). Spring MVC beginner's guide. Birmingham, UK: Packt Pub.

General Data Protection Regulation 2012.European Union . Available at http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf [Accessed 28 October 2015]

Griffin, A. (2015). Think before you tweet - it's going straight to Google. [online] The Independent. Available at: http://www.independent.co.uk/life-style/gadgets-and-tech/news/twitter-and-google-team-up-so-tweets-now-go-straight-into-google-search-results-10262417.html [Accessed 28 Feb. 2016].

Groves, M. (2013). AOP in .NET. Shelter Island, NY: Manning Publications Co.

Gürses S, Troncoso C, Diaz C. (2011). 'Engineering privacy by design'. Paper presented at the Fourth Conference on Computers, Privacy and Data Protection, held 25–7 January 2011, Brussels.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. MIS

Q., 28(1), 75–105. Retrieved from http://dl.acm.org/citation.cfm?id=2017212.2017217

Hevner, A. and Chatterjee, S. (2010). Design research in information systems. New York: Springer.

Hill, K. (2012). How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did. [online] Forbes.com. Available at: http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/ [Accessed 18 Nov. 2015].

Hitchens, R. (2002). Java NIO. Beijing: O'Reilly.

Hoepman, J. (2012). Privacy Design Strategies. [online] Arxiv.org. Available at: http://arxiv.org/abs/1210.6621v2 [Accessed 19 Jul. 2015].

Hohpe, G. and Woolf, B. (2004). Enterprise integration patterns. Boston: Addison-Wesley.

Holaň, J. and Kvasnovský, O. (2013). Vaadin 7 cookbook. Birmingham, UK: packt pub.

Jagielski, P. and Nabrdalik, J. (2013). Instant Spring Security starter. Birmingham: Packt Pub.

Jammalamadaka, R.C. et al., (2013). A middleware approach for outsourcing data securely. Computers {&} Security, 32, pp.252–266. Available at: http://dx.doi.org/10.1016/j.cose.2012.07.005 .

Jayathilaka, H., Fernando, P., Fremantle, P., Indrasiri, K., Abeyruwan, D., Kamburugamuve, S., … Perera, S. (2013). Improved Server Architecture for Highly Efficient Message Mediation. Proceedings of International Conference on Information Integration and Web-Based Applications & Services - IIWAS ’13, 418–427. doi:10.1145/2539150.2539167

Keith, M. and Schincariol, M. (2013). Pro JPA 2. Berkeley, Calif.: Apress.

Kiczales, G. (1996). Aspect-oriented programming. CSUR, 28(4es): 154-es.

Kpmg.com, (n.d.). Protection of Personal Information Bill (POPI) | KPMG | ZA. [online] Available at: http://www.kpmg.com/za/en/issuesandinsights/articlespublications/protection-of-personal-information-bill/pages/default.aspx [Accessed 18 Nov. 2015].

Kurniawan, B. and Deck, P. (2004). How Tomcat works. [Vancouver]: [Brainy Software Corp.].

Laddad, R. (2003). AspectJ in action. Greenwich: Manning.

LAWRENCE ET AL. v. TEXAS [2003]539 U.S. 558 No. 02-102 (CERTIORARI TO THE COURT OF APPEALS OF TEXAS, FOURTEENTH DISTRICT).

Leon, P. G., Cranor, L. F., McDonald, A. M., & McGuire, R. (2010). Token Attempt: The Misrepresentation of Website Privacy Policies Through the Misuse of P3P Compact Policy Tokens. In Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society (pp. 93–104). New York, NY, USA: ACM. doi:10.1145/1866919.1866932

Li, C., Li, D. Y., Miklau, G., & Suciu, D. (2013). A Theory of Pricing Private Data. In Proceedings of the 16th International Conference on Database Theory (pp. 33–44). New York, NY, USA: ACM. doi:10.1145/2448496.2448502

Lindholm, T., Yellin, F., Bracha, G. & Buckley, A. (2011). The Java ® Virtual Machine Specification Java SE 7 Edition. Oracle.

Martienssen, T. (2016). Putting names to your phone numbers - BBC News. [online] BBC News. Available at: http://www.bbc.com/news/business-36015547 [Accessed 26 Jun. 2016].

McDonald, A. and Cranor, L. (2008). The Cost of Reading Privacy Policies. I/S: A Journal of Law and Policy for the Information Society, [online] 4(3), pp.540-565. Available at: http://lorrie.cranor.org/pubs/readingPolicyCost-authorDraft.pdf [Accessed 19 Jul. 2015].

Memsql.com, (2015). MemSQL: The Fastest In-Memory Database. [online] Available at: http://www.memsql.com/ [Accessed 10 Aug. 2015].

Meyer, B. (1988). Object-oriented software construction. New York: Prentice-Hall.

Monash, C. (2008). Mike Stonebraker calls for the complete destruction of the old DBMS order | DBMS 2 : DataBase Management System Services. [online] Dbms2.com. Available at: http://www.dbms2.com/2008/02/18/mike-stonebraker-calls-for-the-complete-destruction-of-the-old-dbms-order/ [Accessed 9 Aug. 2015].

Moodie, M. & Mittal, K. (2007). Pro Apache Tomcat 6. Berkeley, CA: Apress.

Mordani, R. (2009). Java TM Servlet Specification. 3rd ed. Oracle.

Mori, K. (2014). Concept-oriented research and development in information technology. Hoboken, N.J.: J. Wiley

& Sons.

Msdn.microsoft.com, (n.d.). P3P is no longer supported (Windows). [online] Available at: https://msdn.microsoft.com/en-us/library/mt146424%28v=vs.85%29.aspx [Accessed 1 Nov. 2015].

Nguyen, T. (2011). Building .NET Enterprise Applications with Patterns: Part I - Introducing Enterprise Development. [online] C-sharpcorner.com. Available at: http://www.c-sharpcorner.com/uploadfile/b3e929/building-net-enterprise-applications-with-patterns-part-i-introducing-enterprise-development/ [Accessed 2 Aug. 2015].

Nissenbaum, H. (2009). Privacy in context. Stanford, Calif.: Stanford Law Books.

Nissenbaum, H. (2010). A Contextual Approach to Privacy Online. Daedalus, 140(4), pp.32-48.

Odersky, M., Altherr, P., Cremet, V., Emir, B., Maneth, S., Micheloud, S., … Zenger, M. (2004). An overview of the Scala programming language.

OSGi Alliance (2010). Osgi Service Platform Enterprise Specification The Osgi Alliance Release 4, Version 4.2. 1st ed. aQute Publishing. Available at https://osgi.org/download/r4v42/r4.enterprise.pdf [Accessed 14 Oct. 2015].

OSGi Alliance (2012). The OSGi Alliance OSGi Core Release 5. Available at https://osgi.org/download/r5/osgi.core-5.0.0.pdf [Accessed 10 Sep. 2015]

Ottinger, J., Guruzu, S. and Mak, G. (2015). Hibernate Recipes. Berkeley, CA: Apress.

Panda, D., Rahman, R. & Lane, D. (2007). EJB 3 in action. Greenwich, CT: Manning Publications Co.

Patel, N. (2015). Everything You Need To Know About The Google-Twitter Partnership. [online] Search Engine Land. Available at: http://searchengineland.com/everything-need-know-google-twitter-partnership-216892 [Accessed 28 Feb. 2016].

Penn, J., 2011. Behavioral Advertising: The Cryptic Hunter and Gatherer of the Internet. *Fed. Comm. LJ*, *64*, p.599.

Pepitone, J. (2011). Massive hack blows crater in Sony brand. [online] CNNMoney. Available at: http://money.cnn.com/2011/05/10/technology/sony_hack_fallout/?iid=EL [Accessed 28 Oct. 2015].

Pech, V., Shatalin, A., & Voelter, M. (2013). JetBrains MPS as a tool for extending Java. Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform Virtual Machines, Languages, and Tools - PPPJ '13, 165. doi:10.1145/2500828.2500846Pech, V., Shatalin, A., & Voelter, M. (2013). JetBrains MPS as a tool for extending Java. Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform Virtual Machines, Languages, and Tools - PPPJ '13, 165. doi:10.1145/2500828.2500846

Piccoli, G. (2012). Information systems for managers. Hoboken, NJ: Wiley.

Pollack, M. (2012). Spring Data. Sebastopol, CA: O'Reilly.

Pries-Heje, J., & Baskerville, R. (2008). The Design Theory Nexus. MIS Q., 32(4), 731–755. Retrieved from http://dl.acm.org/citation.cfm?id=2017399.2017405

Protection of Personal Information Act 2013.South Africa. Cape Town: Government Gazette. Available at http://www.justice.gov.za/legislation/acts/2013-004.pdf [Accessed 28 October 2015]

Rajković, P., Janković, D., & Milenković, A. (2013). Using CQRS Pattern for Improving Performances in Medical Information Systems.

Rao, P. (2015). Spring Batch Essentials. Packt Publishing.

Richmond, R. (2010). A Loophole Big Enough for a Cookie to Fit Through. [online] Bits Blog. Available at: http://bits.blogs.nytimes.com/2010/09/17/a-loophole-big-enough-for-a-cookie-to-fit-through/?_r=0 [Accessed 26 Jul. 2015].

Royce, W. W. (1970). Managing the development of large software systems. In proceedings of IEEE WESCON (Vol. 26, pp. 328–388).

Rossi, B. (2015). New EU data law's go-live date finally revealed – and why its costs will run into the billions | Information Age. [online] Information-age.com. Available at: http://www.information-age.com/technology/information-management/123459991/new-eu-data-laws-go-live-date-finally-revealed-and-why-its-costs-will-run-billions [Accessed 28 Oct. 2015].

Rubin, K. (2012). Essential Scrum. Upper Saddle River, NJ: Addison-Wesley.

Rubinstein, I. (2011). Regulating Privacy by Design. Berkeley Technology Law Journal, 26(3).

Ruiming, T., Dongxu, S., Bressan, S., & Valduriez, P. (2013). What you Pay for is What you Get. In H. Decker, L. Lhotska, & S. Link (Eds.), DEXA'2013: 24th International Conference on Database and Expert Systems Applications (pp. 395–409). Prague, Czech Republic: Springer. Retrieved from http://hal-lirmm.ccsd.cnrs.fr/lirmm-00831864

Saica, (2015). Protection of Personal Information Act. [online] Available at: https://www.saica.co.za/Technical/LegalandGovernance/Legislation/ProtectionofPersonalInformationAct/tabid/3335/language/en-ZA/Default.aspx [Accessed 28 Oct. 2015].

Safonov, V. (2010). Trustworthy compilers. Hoboken, N.J.: John Wiley & Sons.

Schunter, M., Van Herreweghen, E., & Waidner, M. (2002). Expressive privacy promises - How to improve the Platform for Privacy Preferences (P3P).

Schwaber, K. (2004). Agile project management with Scrum. Redmond, Wash.: Microsoft Press.

Singer, N. (2014). When a Health Plan Knows How You Shop. NYTimes. [online] Available at: http://www.nytimes.com/2014/06/29/technology/when-a-health-plan-knows-how-you-shop.html?_r=0 [Accessed 19 July 2015].

Singer, N. (2012). You for Sale: Mapping, and Sharing, the Consumer Genome. NYTimes. [online] Available at: http://www.nytimes.com/2012/06/17/technology/acxiom-the-quiet-giant-of-consumer-database-marketing.html?_r=1&ref=technology&pagewanted=all [Accessed 19 July 2015].

Sloan, R. and Warner, R. (2013). Beyond Notice and Choice: Privacy, Norms, and Consent. SSRN Journal.

Solove, D. (2002). Conceptualizing Privacy. California Law Review, 90(4), p.1087.

Solove, D. (2006). A Taxonomy of Privacy. University of Pennsylvania Law Review, 154(3), p.477.

Solove, D. (2009). Understanding privacy. Cambridge, Mass.: Harvard University Press.fsl

Sommerville, I. (2010). Software engineering. Boston: Pearson.

Speitkamp, B. and Bichler, M. (2010). A Mathematical Programming Approach for Server Consolidation Problems in Virtualized Data Centers. IEEE Transactions on Services Computing, 3(4), pp.266-278.

Spolsky, J. (2002). The Law of Leaky Abstractions - Joel on Software. [online] Joelonsoftware.com. Available at:

http://www.joelonsoftware.com/articles/LeakyAbstractions.html [Accessed 10 Aug. 2015].

Stallings, W. (2010). Computer organization and architecture. Upper Saddle River, NJ: Prentice Hall.

Stoffel, R., (2010). Comparing Language Workbenches. In MSE-seminar: Program Analysis and Transformation, University of Applied Sciences Rapperswil (HSR), Switzerland. pp. 18–24.

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., & Helland, P. (2007). The end of an architectural era:(it's time for a complete rewrite). In Proceedings of the 33rd international conference on Very large data bases (pp. 1150–1160).

Stonebraker, M. (2010). SQL Databases V. NoSQL Databases. Commun. ACM, 53(4), 10–11. doi:10.1145/1721654.1721659

Stonebraker, M. (2011). New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps. [online] Cacm.acm.org. Available at: http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext [Accessed 10 Aug. 2015].

Stonebraker, M. and Weisberg, A. (2013). The VoltDB Main Memory DBMS. Data Engineering, [online] 36(2), p.21. Available at: http://sites.computer.org/debull/A13june/issue1.htm [Accessed 9 Aug. 2015].

Strauch,C.(2011).NoSQL Databases,https://oak.cs.ucla.edu/cs144/handouts/nosqldbs.pdf, viewed 10 July 2012

Tacy, A. (2013). GWT in action. Shelter Island, NY: Manning Publications.

Truecaller.com, n.d. Phone Number Search | Truecaller. [online] Available at: https://www.truecaller.com [Accessed 28 Feb. 2016].

Tsiptsis, K & Chorianopoulos A.(2010).Data mining techniques in CRM: inside customer segmentation. Wiley

Tulach, J. (2012). Practical API design. [New York]: Apress.

Uralov, M. (2012). Extending an open source enterprise service bus for dynamic discovery and selection of cloud data hosting solutions based on WS-policy, (3347). Retrieved from http://elib.uni-stuttgart.de/opus/volltexte/2012/8043/Uralov, M. (2012). Extending an open source enterprise service bus for dynamic discovery and selection of cloud data hosting solutions based on WS-policy, (3347). Retrieved from http://elib.uni-stuttgart.de/opus/volltexte/2012/8043/

Varanasi, B. (2015). Practical Spring LDAP. Apress.

Van Blarkom, G. W., Borking, J. J., & Olk, J. G. E. (2003). Handbook of privacy and privacy-enhancing technologies. Privacy Incorporated Software Agent (PISA) Consortium, The Hague.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. Sigplan Notices, 35(6), 26–36.

Venable, J. R. (2010). Design Science Research Post Hevner Et Al.: Criteria, Standards, Guidelines, and Expectations. In Proceedings of the 5th International Conference on Global Perspectives on Design Science Research (pp. 109–123). Berlin, Heidelberg: Springer-Verlag. Doi:10.1007/978-3-642-13335-0_8

Venable, J., Pries-Heje, J. and Baskerville, R. (2012). A Comprehensive Framework for Evaluation in Design Science Research. Lecture Notes in Computer Science, pp.423-438.

Voelter, M. & Solomatov, K. (2010). Language modularization and composition with projectional language workbenches illustrated with MPS. Software Language Engineering, SLE. Available at: http://www.researchgate.net/publication/228572379_Language_Modularization_and_Composition_with_Projectional_Language_Workbenches_illustrated_with_MPS/file/50463523c4589162e3.pdf [Accessed October 12, 2014].

Voelter, Markus et.al., (2013). DSL engineering : designing, implementing and using domain-specific languages, S.l: CreateSpace Independent Publishing Platform.

Vukotic, A. and Goodwill, J. (2011). Apache Tomcat 7. Berkeley, CA: Apress.

Walls, J., Widmeyer, G. and El Sawy, O. (1992). Building an Information System Design Theory for Vigilant EIS. Information Systems Research, 3(1), pp.36-59.

Walls, C. (2011). Spring in action. Shelter Island, NY: Manning.