A Decentralised Semantic Architecture for Social Networking Platforms

A thesis submitted for the degree of

Doctor of philosophy

By

Yasir Iqbal

Brunel Business School

Brunel University London

October 2018

Abstract

Social networking platforms (SNPs) are complex distributed software applications exhibiting many challenges related to data portability. Since existing platforms are propriety in design, users cannot easily share their data with other SNPs, however decentralisation of social networking platforms can provide a solution to this problem. There is a difference of opinion, the way the research and developer communities have pursued this issue. Existing approaches used in decentralisation provide limited structural detail and lack in providing a systematic framework of design activities. There is a need for an architectural framework based on standardised software architectural principles and technologies to guide the design and development of decentralised social networking platforms in order to improve the level of both data portability and interoperability.

The main aim of this research is to develop an architectural solution to achieve data portability among SNPs via decentralisation. Existing proposed decentralised platforms are based on a distributed structure and are mainly for a specific aspect such as access control or security and privacy. In addition to this, existing approaches lack in practicality due to underdeveloped and non-standardised design. To solve these issues a new architectural framework is needed, which can provide design and development guidelines for the decentralised social networking platform.

The goal of this thesis is to study, design and develop an architectural framework for social networking platforms that can incorporate the requirements of the decentralisation, to make portability possible. The synergies between the software engineering principles and social web technologies are investigated to create a standard approach. The proposed architecture is based on component-based software development (CBSD) and aspect-oriented software development (AOSD), a unified approach known as CAM (Component Aspect Model). The foundations of the proposed architecture are based on decentralised social networking architecture (DSNA), architectural style which is derived from CAM. Components and aspects are the building blocks of the proposed decentralised social networking platform architecture.

From a development perspective, each component represents a social network functionality and aspects represent the properties and preferences that are used to decentralise the functionality. The model for the component composition is a major challenge because the use of CAM for social networks has not been attempted before.

The proposed architecture comprehensively integrates the DSNA architectural style into each architectural component. Portability among SNPs by means of decentralisation can be summarised into three steps. (1) Definition of the architectural style, (2) implementation of the architectural style into components and (3) integration of the component composition.

To date component composition approaches have not been used for social networks as a way to develop social network functionality. The concept of middleware has been adapted to achieve the composition feature of the architecture. In the architecture Social Network Support Layer (SNSL) functions as middleware to facilitate component composition. Existing middleware solutions still lack integration of CBSD and AOSD concepts. This limitation is characterised by, a lack of explicit guidelines for composition, a lack of declarative specification and definition model to express component composition and a lack of support for role allocation. This research overcome these limitations.

The application of the architecture is based on the W3C SWAT (Social Web Acid Test) scenario. A Messaging application is developed to evaluate the scenario based on the Design Science Research Methodology. The architectural style is defined in the first stage of design followed by the component-based architecture. The architectural style is defined to guide the architecture and the component composition model. In the second stage, the design and implementation of composition technology (that is SNSL) are developed with architectural style and the rules defined in the first stage. The refined version of the architecture is evaluated in the third stage, according to WC3 SWAT test. The definitive version of the proposed architecture with the benchmarked result can be used to design and build social networking platforms, allowing users to share and collaborate information across the different social networking platforms.

Table of Contents

List of Figures

List Of Tables

Dedication

I would like to dedicate this PhD thesis to my mum. She is the one who bear all the tough time in my absence and supported me with all the possible means. Even when she was extremely sick and weak, she was the one who gave me strength, love, and encouraged me to carry on and conclude my journey of achieving PhD. She is my source of strength and without her unconditional support I would not be here, and this work would not be concluded.

My special thanks to my wife Asma who was always there for me in my tough times. I also would like to thank my sister Farah who looked after all sort of things in my absence and supported me in the time of needs. I am a blessed person and for that, I am thankful to almighty.

Acknowledgement

The completion of this work was only possible due to help and collaboration of certain people. I would like to express my gratitude to all those people. At the beginning of my PhD I spent my time at school information systems and computing which was an immense pleasure to be with world-class faculty and some wonderful people. My time at Brunel Business School was amazing and it was a wonderful privilege. I was very lucky to have Dr Sergio de Cesare as my supervisor. I Also want to thank Dr David for his encouraging feedback during the examination of my thesis. Their passion for quality research inspired me to always try my best. Thank you for believing in me and providing me with all the necessary support.

Now I would like to take this opportunity to express my sincere appreciation and heartiest gratitude to my principle supervisor Dr Sergio de Cesare for his exceptional support, encouragement and guidance during all the stages of my research. It would not be wrong if I say, Dr Sergio, installed a desire for excellence in my mind and taught me the principles of superior quality research and helped me think independently and creatively. He always takes time out to listen to me to hear my thoughts and never discouraged me on my mistakes. His beautiful mind and excellence academic intuition made him an oasis of ideas that inspired me to be a better researcher.

List of Acronyms

AI	Artificial Intelligence
ACL	Access Control Layer
DSN	Decentralised Social Network
DSNA	Decentralised Social Networking Architecture
DSNP	Decentralised Social Networking Platform
DSR	Design Science Research
FOAF	Friend of a Friend
RDF	Resource Description Framework
SN	Social networks
SNS	Social Networking Services
SNSP	Social Network Service Providers
SNP	Social Networking Platform
SWS	Semantic Web Service
SNCS	Social Network Connect Services
XML	Extensible Markup Language
SNSL	Social Network Support Layer
SA	Software Architecture
SAE	Software Architecture Engineering
WWW	World Wide Web
SWT	Semantic Web Technologies
XMPP	Extensible Messaging and Presence Protocol
SQL	Structure Query Language
AOP	Aspect Oriented Programming
AOSD	Aspect Oriented Software Development
CBSD	Component-Based Software Development
AO	Aspect Orientation
CAM	Component Aspect Model
AOADL	Aspect Oriented Architecture Description Language
ADL	Architecture Description Language
OWL	Ontology Web Language
URI	Universal Resource Identifier
URL	Universal Resource Locator
PHP	Personal Home Page

Chapter 1 - Introduction

1. Chapter Introduction

A large part of our everyday activities centres on the handling of information in one way or another. Searching for information, grasping it, sharing innovative ideas and results with other people are some of the key activities performed for work or leisure. With the recent growth of Information Technology (IT), people now have multiple ways in which they communicate and share information. This growth has produced an increase in the use of Social Web applications. For example, to access resources available on the Web people are asked to create personal information records to use the services available. By doing so, they are recreating substantial amounts of their information. Management of such information across multiple social web platforms in not commonplace, and currently primarily carried out in a manual manner. Access and use cross-platform data on different social network platforms in a systematic and architecturally sound manner is a problem not currently resolved in an effective manner. The focus of this research is such a problem, i.e. accessibility, dissemination and portability of personal data across social network platforms in a consistent manner.

1.1. Conceptual foundations

1.1.1. Personal Information Space

The Web was invented with the intention of providing a “shared information space” (Berners-Lee et al. 2001) where humans and machine could communicate. The people who intended to use this system were located around the world, connected through heterogeneous mediums. The challenge was to build such a system that can provide a consistent interface to this information coming from different interconnected platforms (Berners-Lee et al. 2001; Hendler and Berners-Lee, 2009).

The study of information (information science) is an interdisciplinary field that deals with the analysis, collection, classification, manipulation, storage, retrieval and dissemination of information. Borko, (1968) defines information science as a discipline that investigates the properties and behaviour of information and the forces that govern the flow of information and means of processing information to its highest level of accessibility and usability.

Personal information in the context of the social web can be interpreted as information about an individual who uses services provided by an organisation, which stores that individual’s information like date of birth, address details etc. Boardman, (2004) defines ‘personal information' as information owned by a person and is under his direct control so the person can alter information without any restrictions.

Personal information space can be described as a repository of an individual's personal information. It includes all the items (emails, e-documents) used by a person while using the web. According to Jones, users upload their personal details to acquire services from the service providers. The stored information is normally under a person's control but not exclusively (Jones, 2007). For example, when a person sends an email message, before coming into their inbox it goes through a "relay" (Crocker, 2009) which store and transmits message towards its destination. Even if the message is deleted, it is very likely still around somewhere in the system.

Personal information space is an information source, which can be used in several ways. For example, it can be used to customise the way the web is used. It can be used to increase the usability of information. There are some security and privacy concerns that are explained in the upcoming sections. The next section describes the evolution of the web towards the social web and outlines the principles, that playing a significant role in making it collaborative knowledge space.

1.1.2. Collaborative Knowledge Space

Several concepts are used to support the foundation of this research. The concepts of personal information and collaborative space gives some initial awareness about the problem, which is related to management of user information, and data in a way that it can be used or re-used across the social web platforms. The problem investigated by this research strongly relates to information science with the focus on the ways in which information is managed and flows across different social media platforms on the World Wide Web.

The World Wide Web (WWW) was designed as a common information space where people can communicate by sharing information. The Web, which is now used by people so generally that it has become a reflection of the way people do work and socialise (Berners Lee et al. 2001).

The Web 2.0 is a second phase in the evolution of the WWW. It is an umbrella term accompanying various new web technologies (Murugesan, 2007). The term web 2.0 coined by Tim O'Reilly to describe this new generation of websites (O'Reilly, 2005). It encourages users to generate content such as blogs, wikis, and feeds, share their content, upload images and videos. Web 2.0 binds to the web in a more interactive and collaborative manner by promoting social interaction and collective intelligence and presents new opportunities by engaging the users more effectively (Heitmann, 2010).

Web 2.0 is a collection of open source technologies collaboration, interactive and user controlled applications expanding the experiences, knowledge and user power as participants in business and social processes. These applications support informal network of users to facilitate the flow of ideas and knowledge and allowing them to generate, disseminate, share and edit users created content (Constantindes and Fountain, 2008; Murugesan, 2007).

Collaboration and sharing are the most important characteristics of knowledge availability, which requires that all participants (i.e. people and applications) must have common grounds to share and collaborate (Sharman et al. 2007). Therefore, the interoperability is required between different systems, databases and applications to share collaborate and execute various services on the Web. To achieve this vision, the World Wide Web (W3C) developed a new set of technologies for the web called semantic web or Web 3.0 (Heitmann, 2010).

The ubiquitous and seemingly distributed nature of the web has taken the flow of information on the internet to an extreme level. This increase in user created content and services offered by social web networks have raised some important questions in relation to information management, data portability and interoperability between the social networks. The next section discusses the social and semantic web in the context of this research.

1.1.3. Social Web Networks

The phenomenon of the social web is characterised and defined differently in the field of social science and computer science. The social web is composed of a set of social relations between the people linked through the WWW. Social web centres on the definition of social interactions and their contents. The field of computer science provides the foundations, in term of algorithmic means and the design and development of web that foster social interaction, hence called social web (Halpin and Tuffield, 2010).

The term social network is a theoretical construct used in social science to study the relationship between the social interaction of two or more people or organisations. A social network is a combination of social structures made by a set of actors and a complex set of relationship between these actors (Wasserman and Faust, 1994). The combination of computer science concepts and information sciences theoretical constructs forms a social networking service, which is an online platform or website that facilitates building social relationships.

In regard to this research, both “Social Network Platform” and “Social Networking Site” are suitable term as they cover both the technical (development and implementation) and social (people-to-people) context of people interactions on the web. The review of available definitions of the social web or social networks is beyond the scope of this research, but essential in term of laying down the conceptual guidelines.

1.1.4. The Social and Semantic Web

Tim Berners-Lee envisioned the term Semantic Web (SW) as the next stage in the evolution of WWW to enhance the ability of the current web. The semantic web can be thought of as a mechanism for representing, describing and processing information on the web in a way that can be processable by machines.

The SW is intended to reduce human involvement in performing different tasks and to enhance automation, coordination and scheduling of services between different platforms. According to Tim Berners-Lee, the idea of the semantic web is to extend unstructured information with a machine-processable description of the meaning of information and to provide missing background knowledge where needed (Berners-Lee et al. 2001). The need is evident to enhance the ability of the web, to be more people-centric and with advanced filtering and recommendation services (Dasgupta, 2010 and Sfakianakis, 2010) by providing data portability and integration between different websites and networks (Sfakianakis, 2010).

In the semantic enabled social web, content can be easily connected, integrated, navigated and queried. Semantic web technologies can be used to add rigour and descriptive structure to the content of the user contributions in a way that will enable powerful computations and help better manipulation and distribution of data (Gruber, 2007).

Currently, social web applications are more focused on the management of social contents and interactions rather than focusing on the provision of semantically enabled data description. Blogs, search engines and messaging are some prominent features of the social web, can be enhanced using the SW (Sfakianakis, 2010). For example, Valencia-García et al. (2010) used social semantic technologies to constitute a platform, which is capable of automatically managing and suggesting new member of a project team based on their best suitable skills for the development of the software project.

According to the findings of Halpin and Tuffield, (2010), the social web does not suffer from lack of standards as it was a few years ago. Numbers of diverse groups are formed in this area. The data model formats and communication protocols used by the web have been revitalised by their efforts. Lots of work is done on the standards to address the basic issues of identity management and user login information portability. However, not enough is done to solve vital and complex issues such as privacy policy portability and user data portability within and across the network boundary. These issues present scope for further development and research.

1.1.5. Data Portability and Interoperability

Data portability is defined by DataPortability,org (Dataportability, 2015), as the ability of people or applications to reuse their data across different interoperable applications, by allowing the people or application to be able to control their various forms (i.e. identity or media related) of data. Breslin et al. (2009) refer to data portability as a combination of methods that allow people to port their data from one place to another. In the social network, the user cannot access their data and share it across social networking platforms. Data portability in the context of social networks is concerned with, allowing data to be accessible and available to the user and social networking applications within the same or across SNPs. With data portability, different components of the application can be reused within or across the platform.

In contrast to data portability, interoperability is very well defined and standardised, according to ISO 15926, interoperability is the ability of different types of computers, networks, operating systems and applications to work together effectively, without prior communication, in order to exchange information in a useful and meaningful manner (ISO15926, 2016). Kosanke, (2006), reviewed interoperability standards and available research, in order to provide not only a single version of interoperability definition but also how different interoperability standards have been used in engineering, manufacturing and computing research. Another ISO standard, ISO-14258 (ISO-14258:1998, 2014), is used by t organisations, seeking integration between their different independent systems, to define rules and concepts for their enterprise models with the intent to guide the process of interoperation. According to ISO-14258, interoperability may occur between two or more than two different entities that are connected to each other in three ways; integrated (where there is a standard format for all the devices and systems constituents) unified (where there is a common meta level structure across basic models, providing a means for establishing semantic equivalence) and federated (where models must be dynamically accommodated rather than having a predetermined meta-model) (Kosanke, 2006).

Based on the above description, in reality, it is very unlikely that complete interoperability can be made possible following any of the ways mentioned in ISO-14258 because current global information and communication environments do not support global unification, integration and federation of existing systems and make interoperability a difficult task. Similarly, data portability and interoperability between different social networking platform can be hard to achieve as there is no standardised method or architecture available to guide the interoperability process at all the levels of social networking platform. Decentralisation of social networking platform (SNP) is one of the ways to achieve interoperability between SNPs.

1.1.6. What is Decentralisation

The software engineering and software architecture literature has not embraced a formal definition of decentralisation (Khare and Taylor, 2004). Even now it has been described differently in the context of research and mainly considered as a synonym for distribution.

The research community sees decentralisation as a solution to some of the issues with existing social networking platforms. However, decentralisation has its own issues. The area is still underdeveloped, and the process of standardisation is not efficient enough, thus suffering from a lack of available implementation standards. Decentralisation is mainly dependent on open source community standards (explained in chapter2) and the definition of decentralisation and related concepts are based on the opinion adopted by the researcher.

In the Oxford Dictionary, decentralisation is described as the process of transferring authority of decision making to lower level. In the field of computing decentralisation is an allocation of resources to individual clients. In the field of database management, decentralisation is about storing data on clients at multiple locations however the clients are not interconnected by central network or database. Therefore, a decentralised database is best regarded as a collection of independent databases rather than a geographical distribution of single database (Slater et al. 2015).

The earliest related conceptual relevance can be found in McLeod and Heimbigner, (1980) in which, they described decentralisation in the context of databases and in their opinion, decentralisation is a logical combination of components or entities having their own logical and conceptual schema. These components are related but independent and they may or may not be disjoint.

Decentralisation in the context of social web network can be defined as a collection of entities, called peers or nodes that interact with each other without the presence of a trusted central control authority. Each one of them works towards achieving its individual goal (Suryanarayan et al. 2005). Therefore, there is no single point where the decisions are made, and every peer makes decisions towards its own behaviour.

1.2. Research Motivation

The available research in social web decentralisation is mainly done in user privacy, Profile data portability, activity and identity-related issues. There are three main approaches widely used in research to decentralise the social web, distributed web server hosting, federated layer and P2P approaches. The majority opinion goes with the federation of social networking platforms, which is still underdeveloped and has opposition in social network service providers. The general trend in research is, to have portable social data by using semantic web technologies but they do not provide any standard way for social data to be portable. Another popular opinion described in Berners Lee, (2009) is user-centric social data management that is providing personal information space, where the users can manage their information and data based on their own needs, with service providers only providing the interface.

Lack in the implementation of software architectural principles standards and guidelines.

The work done in this thesis addresses new challenges and opportunities for the decentralisation in social networking platforms, that are posed by lack of architectural guidelines, current infrastructure, protocols, standards and service providers restrictions. The proposed solution introduces changes in a way decentralised social network platform should be designed and develop. To serve this goal a comprehensive decentralised architecture for social networking platforms is designed under the guidelines of the proposed architectural style. The overarching goal to achieve by building an application based on the proposed architecture is portability of data at the functional level of different social networking platform.

The solution envisioned in this research attempts to solve the problem of data portability between social networks at the functional level by using a decentralisation approach. The methodology used to build the decentralised architecture, uses similar standards and protocols as used by existing architectures, however, it differs on the principles, whether decentralisation should be done at the central level such as the Federated Social Web, widely explored or at the functional level, which is unexplored. Using the proposed architecture users will be able to decide which functionality they would like to use across their social network platforms i.e. if the user decided to use the message related functions then they will be able to send post to another platform they are registered to.

Lack of data portability and interoperability caused by a centralised form of architecture.

Traditionally social web networks are based on centralised architecture, making the companies providing these services the sole owner of user's data. Due to this reason data stored on these websites is not accessible to another site, and users are not allowed to reuse their own data on other similar sites, thus forming data silos, an isolated island of data (Yeung et al. 2008). Each social application has its own data not knowing of the relevant data available on the other applications and platforms, exposing the lack of interoperability between the applications and services they provide. Similarly, due to these restrictions, ordinary users are unable to have the ownership of their own data and therefore cannot reuse their data and profile information on the other social network platforms (Tandukar and Vassileva, 2012).

These deficiencies in social web architectures affect the user experience and cause problems such as data Portability or interoperability, User Identity and profile reusability, Linkability and privacy of user data (Halpin and Tuffield, 2010). In addition, in recent years' platforms like Buddycloud and Higgins are built with the same goal. Diaspora is a social network centred on the idea of data hosting at different locations connected together in one autonomous network.

These and similar platforms are either insufficient schema agnostic or seemed not to address needs concerning sharing i.e. pertaining to keeping subscriber informed of the changes. Access control was also done in an ad-hoc, non-standard compliant ways and there are some security issues like system data being exposed to external apps (Smith et al, 2012).

Lack of data integration between different SNPs and duplicity of data.

An important requirement for an SNP architecture is to provide seamless integration of data in a distributed setting. In the Nepomuk Semantic Desktop project (Sintek et al. 2009) group collaboration architecture is proposed based on semantic web technologies and peer to peer networks to enable communication between different applications running on different networks. Nepomuk is an ontology-driven and support group collaboration which is an essence of social networking. Although Nepomuk was successfully implemented, it is a desktop application that meant to improve the desktop experience rather than web experience. Desktop applications are becoming increasingly obsolete that is why current research investigate an architecture to facilitate decentralisation in social networking platforms.

In PrPl (Seong et al. 2010) decentralised social networking infrastructure is described, which allows users to share their personal data in a distributed network of peers through butlers. Personal cloud butlers are used as decentralised data storage to index user personal data. While similar to this research, PrPl requires its applications to be developed in a specialised language called SocialLite (Seong et al. 2010) which reduces its adaptability by the developer community. Whereas the proposed solution is not dependent on any specific language or standards and is based on the open web standards.

The solution proposed in this research is inspired by the existing Internet Mail (also known as Email) architecture which appeared to be only standardised architecture available that can be used to enable user communication between different websites using messaging protocols. According to Ballester et al. (2010) each Social network stores user information differently. If the user is interested in using the services offered by others than the site he is registered, with then he may have to register again. As a result, the user may be registered on several social networks, causing data to be scattered, duplicated and disorganised. If it is possible to represent information in a common language or standards like email, then social networks may be able to interoperate.

The envisioned solution provides a mechanism or constructs, rules and guidelines in the form of an architecture to help decentralisation in social networking platforms. A platform that can enable end users to integrate, reconcile and consolidate their different identities from multiple social networking platforms and reduce duplicity of data, as compared to existing centralised social network the proposed solution has a number of advantages that is the most important motivational point of this research.

1.3. Problem Definition

The best known social networking sites such as Facebook, MySpace, Twitter and etc, have limited themselves to relationships between the people on one site, the social web should be extended to the entire web. For example, people can call each other no matter what service provider they are using, similar to people sending a message to each other using email irrespective of their service provider. The social web should allow people to create a network of relationships across the entire web by giving people access to their data and privacy (Halpin and Tuffield, 2010; Hu and Lau, 2013). To solve this, issue a truly open and decentralised architecture for the social web platform is required.

For example, person A is a member of social network SN1 declares person B as his brother. Person C is a member of SN2 and the aunt of person A and B. There no way for her to get in touch with her family members, without joining their social networks. This is because of there being no mechanism of transferability in import or export of personal data across social networking platforms.

1.4. Research Aim and Objectives

The rationale established in the conceptual foundations, problem definition and motivation regarding the problem in current social network platforms abilities to allow complete or partial information portability is not a resolved issue. None of the previous work on web architectures provides enough guidelines on social web architectures, also, there are deficiencies in the software architecture principles implementation to provide concrete guidelines for the design and development of the software architecture for the platform that can help the enablement of social data portability among different social network platforms.

The aforementioned shortcomings in the subject area encourage the proposition made in this research proposal, which is to design and develop an architectural framework for the decentralisation of the social networking platforms following software architectural guidelines.

The main contribution of the proposed architecture is to the concept of the social networks interoperability and software architecture engineering, by using the methodology that is a combination of software architecture and semantic technologies, modern web languages and open-source message transfer protocols to achieve the desired goals.

1.4.1. Aim

The inability of current social networking platforms in providing the data portability and limitations in the existing architectural approaches to provide a satisfactory solution for the data portability by the means of decentralisation in an unsolved problem. Therefore, to solve this problem social networking platforms service providers need a new architecture, which is the aim of this research.

The aim of this research is to investigate the application of software architecture principles and synergies between social and semantic technologies, to design and develop the decentralised architecture that can enable data portability between different social networking platforms.

1.4.2. Objectives

To achieve the aim, it is imperative that the following objectives are met.

O1: Derive basic components of the social web architecture by analysing literature on software architectures and web architectures.

This objective is about finding the underpinning knowledge related to this research, such as the role of software architecture engineering in the design of the social web applications. The main emphasis is on analysing and synthesising the literature on the software and web architecture to understand the key components that are used by the industry and academia to design the web applications and their architectures. Finally, analysing how much the building blocks of the proposed web applications in the literature are influenced by software architecture engineering principles and can be used as the components of social web architecture.

O2: Investigate the literature on synergies between software engineering principles, semantic technologies and social web networks to derive the components considered as necessary to build the decentralised social network platform (DSNP), that can enable data portability between different social networking platforms (SNPs)

This objective is aimed to critically analyse the concerned literature and synthesis of the concepts, approaches and methodologies relevant to data portability issues, in the existing social networking platforms. Moreover, identify the role of decentralisation and semantic technologies towards the data portability in SNPs. The purpose of this objective is to help discover the gaps in the research done so far in concerned area and identify the relevance and disagreement of opinions to present the governing rules for the design of proposed architecture. In regard to the software engineering principle, the integration of CBSD and AOSD is investigated to find how functional independence in the components can be achieved.

O3: Build the design of the component based architecture and architectural style to provide a framework of principles for the design and development of decentralised social networking platform at a functional level.

This objective is aimed at providing component based architecture DSNA and DSNA architectural style to decentralise the SNPs at the functional level. Achieving this objective consists of five steps. In the first step requirements of proposed DSNP are illustrated under the extended SWAT scenario (as explained in chapter2). Requirements are used to characterise SN functions. In the second step, the key foundation principles of the design are explained. These principles are based on CBSD based PACE and AOSD. In the third step, detailed design of DSNA architecture having its foundation in the DSNA style is described. The purpose of DSNA style is to equip the application developer with rules, properties and guidelines to design and develop DSNP. In the fourth and fifth step, the architecture is implemented on simple SN function, components and aspects implementation are also described.

O4: Build a prototype by implementing the DSNA on SWAT scenario and demonstrate the composition of aspects and components in the implementation.

The purpose of this objective is to implement the proposed DSNA on SWAT. The requirement analysis identifies the challenges and needs of the design. Interaction, communication, composition and allocation are identified as four main challenges towards implementation. Design phase explains the design related challenges and build phase handles the implementation related challenges. Dynamic Component and aspect composition are handled at the middleware level. SNSL (Social Network Support Layer) handles the composition of components. Social messaging application prototype is built to evaluate the functioning of the application.

O5: Perform SWAT based evaluation of the DSNA and find the significance in current literature and drawbacks.

The aim of this objective is to define the SWAT evaluation metrics. According to which the application is tested on the basis of interaction and communication. At the end, the improved version of DSNA is presented with the overall findings and challenges.

1.5. Research Methods

The research objectives are concerned with answering the questions raised during the research. Therefore, a research method should be selected based its compatibility to the research objectives. The key motivation behind any research is the desire to build and improve new environments by introducing innovation in building new artefacts and that is a key characteristic of Design Science Research (March and Smith, 1995; Hevner et al., 2004).

To achieve the research, aim and objectives this research follows design science research methodology. DSR is an iterative activity where solution artefacts are designed and developed through various cycles, processes, activities, inputs and outputs. The goal of a DSR is to generate a purposeful artefact that addresses a practical problem, especially, when elements of the problem are not completely understood (Hevner et al., 2004).

DSR provides a suitable approach and comprehensive framework for the analysis of the systems and architectures in question. It comprises of two main activities i.e. construct and evaluate (March and Smith, 1995), to resolve the research problem. DSR supports the design, construction and evaluation of the Decentralised Social Networking Platform (DSNP) through the means of DSNA. The framework of March and Smith, (1995) is selected to support artefact design process. The framework provides foundations for the execution of the research project by articulating the artefacts in four outputs, Constructs, Model, Method and instantiations.

In the current research, DSR is associated with all main activities to the end result of creating and an architectural solution called DSNA. The DSNA has been evaluated to verify its applicability according to the desired objectives. Kuechler and Vaishnavi, (2012), has given a broad outline of the development stages, that has been followed in order to direct the research finding process.

In Kuechler and Vaishnavi, (2012), framework an iterative process of design has been followed to ensure the continuous improvement in designing the artefact. There are five main phases, (1) Awareness of the problem and type of solutions (2) Suggestions for the design (initial Conceptual Design) (3) Design and Development (4) Demonstration and (5) Evaluation. Each phase feeds back the knowledge gained construction and evaluation into the design of the iterations. For the sake of evaluation, Hevner et al. (2004) methods of evaluation have been adopted. The detailed scenario in the area of social network functions has been built to test the utility of the DSNA. Further explanation of the iterations and phases is given in chapter 3.

Artefact Category	**Iteration 1**	**Iteration 2**	**Iteration 3**
Construct	None	None	
Model	DSNA style DSNA component architecture	DSNA Deployment Design DSNA Component composition model	
Method	DSNA framework Aspect component composition	SNSL component and aspect composition	DSNA evaluation mechanism

Instantiation	None	DSNA prototype	DSNA interaction evaluation DSNA communication evaluation

Table 1-1: Classification of DSR Artefact in this research

The artefacts are realised after evolving through the 3-stage development process. The development of DSNP prototype is done in three main phases 'design and development', 'Implementation' and 'evaluation'. Each phase goes through the process of 'build', 'deploy' and evaluate, as described in March and Smith, (1995). The DSR process can be iterative or incremental in nature (Markus et al. 2002). The 3-stage design process can be repeated or incremented during each phase until satisfactory artefact is obtained. In iterative DSR, the 3-stage design process is repeated to improve the quality of the artefact. Whereas in incremental DSR the design artefact is decomposed into granular artefacts each one is developed and evaluated in each increment (Simon, 1996). The DSNA is designed and evolved in the 3-stage process, artefacts are designed, deployed and evaluated using suitable evaluation methods. The research in this thesis is iterative in nature however, some artefacts are decomposed into multiple iterations for attaining the complete functionality.

1.6. Thesis Structure

Chapter 2 presents literature review with the purpose to provide contextual analysis of research done in social web network's capacity to enable data portability and to outline the existing research approaches and methodologies used to deliver the portability of data and information between social networks. Chapter 2 is basically survey based and the first section focuses on software architectures, its elements and styles. The purpose of this section is twofold; (1) examine which software architectural principles and design techniques are used in Web architectures, proposed by the research community and (2), understand if those principles and techniques can contribute towards the design of social web architectures. The second part of this chapter exposes the shortcomings in the current social web network architecture within the domain of data portability between different social web network platforms. The attempt is made to critically analyse the current state of research on social web platforms architectures with the focus on data portability between different platforms and the role semantic technologies have played to achieve it.

Chapter 3, introduced DSR methodology and its application within this research. The stages adopted to implement the DSR are illustrated in Vaishnavi and Kuechler (2004; 2012). There are three main stages, awareness of the problem and type of solution, development and final evaluation. There are three iterations between development and final evaluation that are described as design, deploy and evaluate. For the construction of artefact (Construct, method, model and instantiate), (March and Smith, 1995) guidelines are used. For evaluation, each iteration and artefact follow criteria described in Hevner et al. (2004).

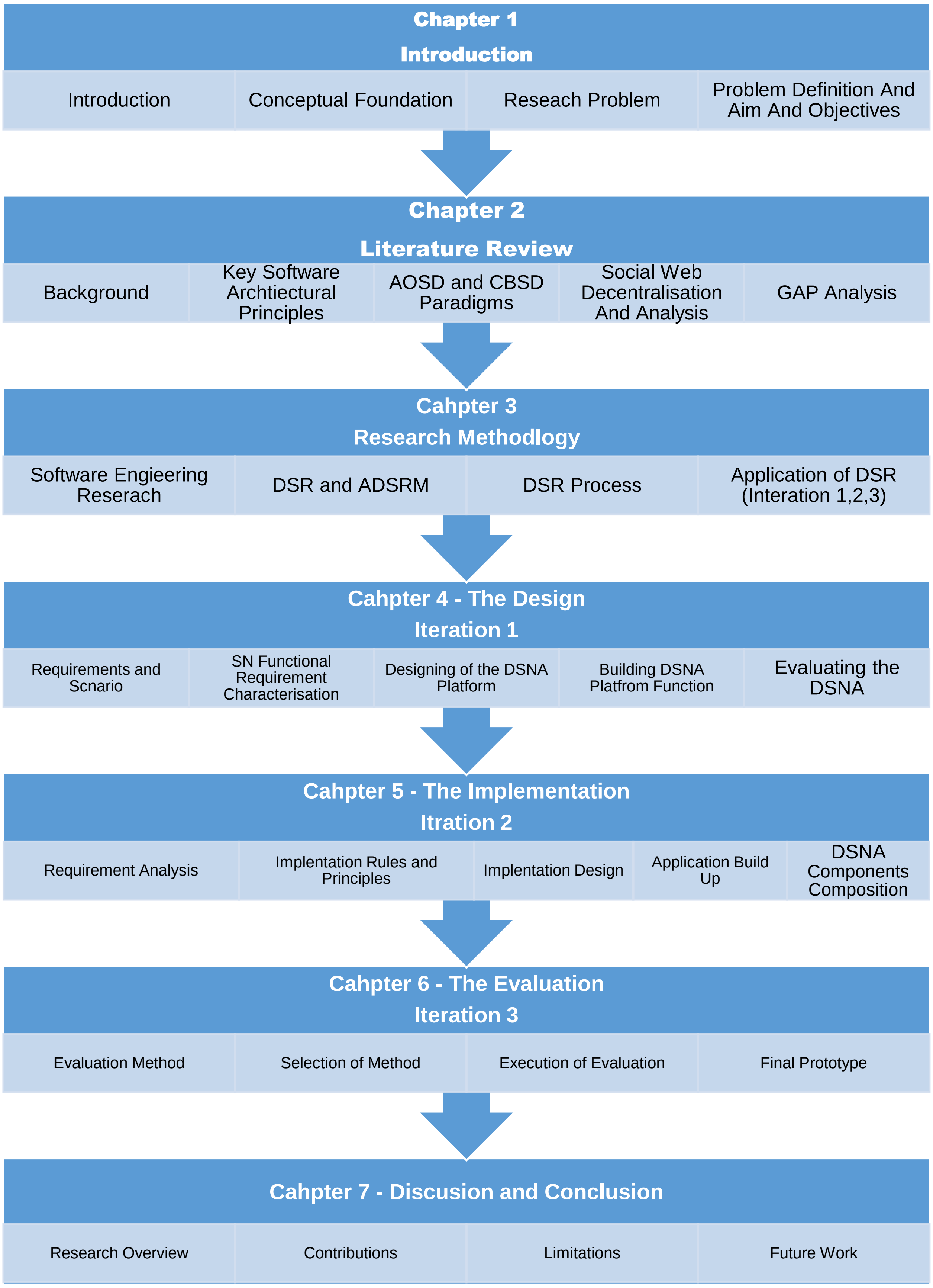


Figure 1-1: Thesis Structure

Chapter 4, introduces the component based conceptual architecture. As the result of the first iteration, the conceptual architecture of DNSA is presented. The research in this chapter provides a detailed description of the architectural components. The description of Architectural components is required to provide grounding structure to build proposed social networks architecture. In the context of this research, the standardised sets of components for the formation of conceptual architectural view are important. In order to achieve its purpose, DSNA must provide a feasible and reliable way for the users of different social platforms to interact and communicate without duplicating their data.

Chapter 5, presents the refined version of DSNA after its successful implementation in the prototype. This chapter describes the prototype implementation of DSNA proposed in chapter 4. The prototype is the result of SWAT Scenario that has been used to check the practicality of the DSNA. The DSNA is tested against most possible Scenario that the DSNA can be implemented. In this perspective, the Implementation of DSNA is done at the functional level of social networking platform, using content sharing functions, such as messaging.

Chapter 6, extends the implementation into the more realistic scenario. In the final iteration, DSNA is implemented in multiple cross-domain social networking scenarios. To test the scenario SWAT v1, which is an initiative social web group at W3, is used. SWATv1 provides test scenario and a set of guidelines for evaluating the decentralised social networking applications.

Chapter 7, presents the summary of the thesis. In this chapter, a brief review of activities performed within the entire chapter is provided. A brief account is provided, about how each objective is realised. The chapter ends with the description of the research limitations and recommendation for future work.

1.7. Chapter Conclusion

The chapter presented the introduction of the research conducted in the remainder of the thesis. Beginning with motivation for undertaking this research and literature reflecting the need, importance and current state of the research. The main problem is defined with the aim and objectives to be completed to fulfil the aim. In the next stage, which research methodology is selected to address the research contributions, is explained. Finally, at the end, the content of each chapter is summarised

Chapter 2 - A Literature Review on the Decentralised Social Web, An Architectural Perspective

2. Chapter Introduction

An understanding of the elements required for the design of any software architecture is an important step. To realise this step, this chapter reviews different software architectural styles and components with the aim of providing guidelines for the design of social web architectures and key elements representing the building blocks of such architectures, are also highlighted. These building blocks are discussed in the context of both, theory and practice.

This chapter addresses objectives 1 and 2 of this research, which is (1) to analyse the best practices on software and web architecture to derive basic components of social web architecture, and (2) to investigate the synergies between the semantic web technologies and social web to determine the components needed to design a required decentralised social web architecture.

The overall goal of this chapter is to provide contextual analysis of research done in social web network's capacity to enable data portability and to outline the existing research approaches and methodologies used to deliver the portability of data and information between social networks. The review also helps, to discover gaps in the research done so far that are still to be addressed and to identify relevant and applicable elements to set governing rules for proposed research. Another important aspect of this chapter is the analysis of the existing tools and technologies have been used by academia to formalise the solution required for data portability between social networking platforms.

2.1. Literature Review Methodology

This literature review is done in four steps, to form the fundamental conceptual study required to build the coherent and systematic outlook for the literature review. In the first step. 70 articles from journals and conferences relevant to the research topic were selected. The details of the articles are maintained in a table having columns; Title of the article, Type (i.e. journal, conference, book, report), Summary, and Relevance to the research. In the second step, selected articles were skimmed, duplication was removed. The skimming process shaped the literature review and in the third step. literature review map is built, which gives the visual presentation to the literature review shown in figure 2-1. In the fourth step, a final literature review is built, from the summaries taken from the skimmed articles. The articles were collected using library database Athens, Google Scholar, universities research group portals and direct from authors (in some specific cases). The literature review map is built using conceptual mapping technique of concept analysis mentioned in Martin and Hanington, (2012). Conceptual mapping is a visual framework that can be used to allow readers to absorb new concepts towards the understanding of new domains to build new meanings. Its main purpose is to connect a large number of ideas and events as they relate to a specific domain. It provides the scaffolding that can help the readers or designers to visualise the structure of various connected concepts and ideas (Martin and Hanington, 2012).

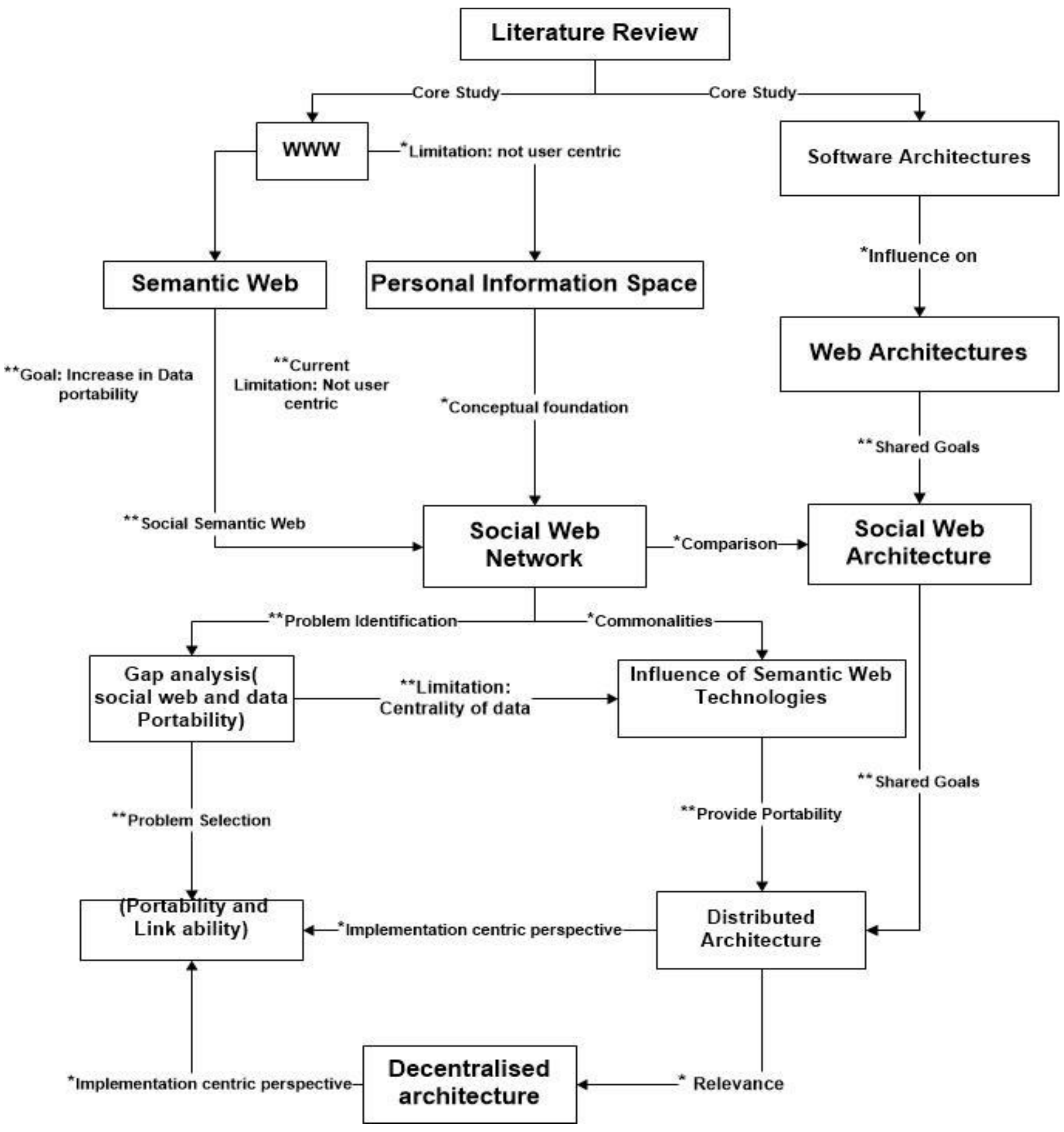

Literature Review
Core Study
Core Study
WWW
*Limitation: not user centric
Software Architectures
Semantic Web
Personal Information Space
*Influence on
Web Architectures
**Goal: Increase in Data portability
**Current Limitation: Not user centric
*Conceptual foundation
**Shared Goals
**Social Semantic Web
Social Web Network
*Comparison
Social Web Architecture
**Problem Identification
*Commonalities
Gap analysis(social web and data Portability)
**Limitation: Centrality of data
Influence of Semantic Web Technologies
**Shared Goals
**Problem Selection
**Provide Portability
(Portability and Link ability)
*Implementation centric perspective
Distributed Architecture
*Implementation centric perspective
Decentralised architecture
* Relevance

Figure 2-1: Literature Review Map

The literature map in the figure 2-1 consists of key domains, related concepts and linking words. The linking forms the meaningful statement. The linking words are distinguished by placing two asterisks **, and one asterisk * at front of the words. Two asterisks ** shows the findings one asterisk * are the linking concepts to form a meaningful statement.

The rest of the chapter is organised as fellow. The first section is a theoretical analysis of fundamental software architectural principles, for example, abstraction and different ways to achieve it. For that reason, architectural style and views and their relevance in Web architectures are discussed, moreover, the current research on the Web and social web architectures and technologies that influence the design of the architecture are also discussed.

2.2. Background

The development of an architecture, for any software, is the main activity during its design. There is a consensus in the literature that software architecture depicts the structure of the software components, with the purpose to provide a functional description of the software structure, components and component interaction. It is also important to adhere to software engineering principles during the architectural design to deliver the details regarding the key elements of the architecture such as component selection, standards, policies, design methods and implementation infrastructure (Gerber et al. 2008). In this research, the term architecture when unqualified is a synonym of software architecture.

2.2.1. Concept of Abstraction and Software Architecture

The core of software architecture is the principle of abstraction, which is about encapsulating the details of a system in order to better identify and sustain its properties (Shaw, 1990). The architecture of complex software might have multiple levels of abstraction to represent the behaviour of the system and architectural components functionalities (Bass et al. 2011).

According to Perry and Wolf (1992), architectural elements provide the abstract view to software architecture. They define “software architecture as a set of architectural elements that have a particular form explained by a set of rationale”. Garlan and Shaw (1994) introduced some of the key principles in software architecture research and described various architectural styles. They described the architecture of a system as a collection of computational components connected through connectors. The role of components is then further clarified in Shaw and Clements (1997), according to them a component is an abstract unit of software instructions that performs some functions during runtime and provides a transformation of data via an interface. For example, programs, objects and processes. This explains the difference between the software architecture, which refers to the abstract structure of software system behaviour and the component as part of software architecture.

The significance of elements in software architecture

Software architecture is defined by the configuration of its elements and their composition according to given requirements to achieve the desired architectural properties (Fielding & Taylor, 2002). The elements are the core of any software architectures. Based on the particular behaviour of the system, there are three basic types of elements, data, processing and connecting elements. Processing elements; process components that perform the transformation of data, data elements are those elements that contain information about the system functionality that is used and can be reused, if called back again during system deployment; the connecting elements are the glue that holds the different elements together (Perry and Wolf, 1992; Shaw and Clements, 1997).

The architecture of a software system often has multiple abstraction levels and each may have its own architecture. For example, in a Web application architecture, a configuration file (Web.config) will be treated as a “data element” (Perry and Wolf, 1992) during start-up as it contains the information that has to be shared with other elements of the architecture. However, during the normal processing of the application, it will not be considered as an element because the shared information is distributed throughout the application and no longer needed until the instance of the application returns to start-up level.

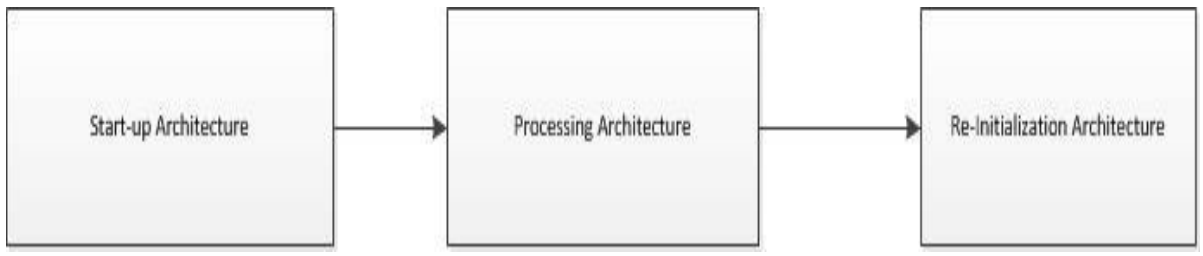


Figure 2-2: Software Architecture Design Stages: Adopted from Fielding and Taylor (2002) and modified.

Therefore, it is important to understand and implement the behavioural properties of the software. To do so the designer may need to design the start-up architecture, processing architecture and re-initialization architecture.

In general, the architecture must be capable of describing not only the operational behaviour of the software such as the communication between the elements but also the transition between different phases (Fielding, 2000). In another opinion, the architecture of an information system or software application is a mixture of structure or structures of a system, which is made of software elements, the externally visible properties of those elements and relationship among them. In the area of software architecture engineering the externally visible properties are those assumptions that elements make about other elements, when different functionalities and behaviours showed by elements to provide services, to solve performance issues or fault handling (Bass et al. 2011).

The definition

As described above, there is no shortage of opinions, when defining the software architecture. The definition used in this research is taken from IEEE std 14712000 (1471, 2000) which was further improved in IEEE P42010/D9, is considered as a standard definition of the software architecture. In IEEE 1471, the software architecture is defined as "the fundamental organisation of a system embodies in its components their relationship to each other and to the environment and the principles guiding its design and evaluation" (IEEE-1471, 2000). This definition incorporates the idea that there is a difference between architecture description and an architecture. An architectural description is a concrete artefact, but an architecture is a concept of a system (Maier et al, 2001). IEEE-P42010/D9, (2011), distinguishes between the architecture description and architecture of a system. According to the standard, architecture description is a work product whereas the architecture is an abstract shape of the system consisting of concepts and properties (IEEE-P42010/D9, 2011).

Certainty, the definitions of software architecture are different, but there are also so many commonalities as well. For example, most of the definitions indicate that the architecture is concerned with elements, structure and behaviour of the system.
Furthermore, the architecture description concerns with views and styles, are influenced by key stakeholders and system environment.

The elements are a basic pillar of any software architecture and made by a set of components, containers and connectors. When these elements are working together, in a well-connected manner then they adopt a form of a view and unique styles (Bass et al. 2011) discussed in the next section.

2.2.2. Architectures styles and views

Perry and Wolf (1992) divided software architecture into architectural styles and views based on the functionalities of the system. Kruchten, (1995) extended this work into the object-oriented environment and proposed four categories of architectures, logical, development, processing and physical architectures. The complex software may have multiple architectural styles to describe its functionalities in depth.

In the literature, there is no hard-dividing line between architectural style and architecture. It is one person's decision, as an architectural style of one person may be the architecture of another. Architecture is an organised arrangement of elements, but an architectural style provides a specific abstraction of the elements regarding their system functionalities (Perry and Wolf, 1992, Shaw and Clements, 1997). Network, distributed, layered, client server and Service Oriented Architectures are examples of specific architectural styles.

The architectural style is also known as an architectural structure. According to Bass et al. (2011), the complexity of a software system can make it difficult for a designer to grasp all the information. Instead, they can restrict their attention to one specific software structure of the system. The development of web systems is done in many phases. Each phase may have its own architectural view. View holds the important information regarding the development of software architecture. The architectural view (or simple View) is a representation of a coherent set of architectural elements as described by key system stakeholders. It consists of elements and relationships between the elements (Brown and McDermid, 2007).

Another perspective on Architectural view is described in IEEE reference standards. According to that architectural view is a description of architecture as specified by its stakeholders, with the purpose to express the architecture of the system of interest in accordance with architectural "Viewpoints" (explained in next section) (IEEE-P42010/D9, 2011).

Software Architecture Development	Explanation
1. Software Architecture and Architecture Styles	Examples: Layered, Network, distributed or Service Oriented Architecture, Composed of Elements Modules, Components and connectors, Allocations or relationship
2. Architectural Views	Divides architecture into different perspectives to reduce complexity during the design process • Conceptual View • Logical View • Physical View From each view, separate architecture emerges with all the information needed to build complete software.
3. Methods	Style and Views (Perry and Wolf, 1992) 4+ 1 Views (UML based) (Kruchen,1995) Attribute Driven Design (Bass et al. 2011) RUP (Clement et al. 2003)

Table 2-1: Software architecture development, based on Kruchten, (1995), Shaw and Clement, (1997), Clement et al. (2003), Bass et al. (2011).

Table 2-1 summarises the main points proposed by Perry and Wolf (1992), Kruchten (1995), Shaw and Clements (1997), and Fielding and Taylor (2002).

Clement et al. (2003) proposed an approach to describe software architectures using a set of Views, ViewTypes and ViewPoints. Bejar et al. (2009) used Clement et al. (2003) approach to design an architectural style for Spatial Data Infrastructure (SDI) as a part of an information system to examine geographical data. One of the main reasons to use this approach was to enhance the ability of designer and system architect to describe the complex functionalities and behaviours of the system. Chen et al. (2008) also included Views and ViewTypes to describe their proposed architecture from a specific perspective.

Advancements in software development technologies and a blend of mobile and web applications have changed the requirements and needs of software architecture development. To meet new challenges IEEE Architecture Planning Group (APG) and Software Engineering Committee, revised software architecture description standard and included View and ViewType and ViewPoint as key components to describe software architecture (IEEE-P42010/D9, 2011).

2.2.3. View, View Types and View Points

A view is a graphical representation of elements and their relationship (Perry and Wolf, 1992). Bass et al. (2011) described the view as a representation of a coherent set of architectural elements as written by and read by system stakeholders. ViewType is a definition of allowed element types and relationship types, which can be used to describe a system from a specific perspective. For example, a view of a web application could be a diagram showing web services as boxes and arrows as the relationship between the services and ViewType could be a constraint saying only web services are represented as boxes and arrows as relationships (Clement et al. 2003).

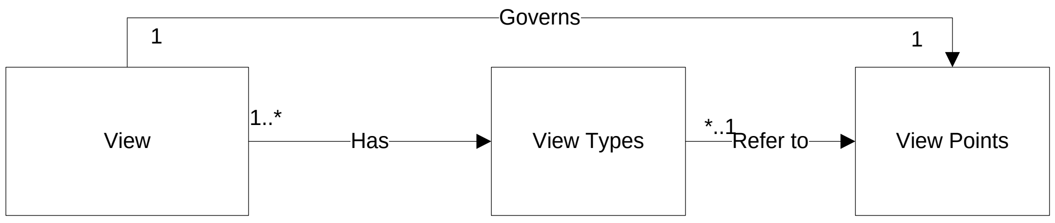


Figure 2-3: Relationships between views, view types and view points

Viewpoint provides a specific perspective of the system by focusing on certain concerns regarding the system. For example, the security viewpoint focuses on the security perspective of the system with relevant elements by suppressing details and providing simplified versions (Bejar et al. 2009). Figure 2-3 shows the relationship between views, view types and view-point. The relationship between the views, view-types, and view-points can vary, depend on the architecture description but the relationship between view and view-point will always be one to one (IEEE-P42010/D9, 2011). As described in above section that architecture design process can have multiple phases.

In the research community, there are different opinions on the number of steps should be involved in the software architecture design. The approach described in Clement et al. (2003), IEEE P42010/D4 standard and Bass et al. (2011) are widely practiced, although the components are expressed in a different manner but main steps to express those components are same. There are three basic types of views, module view, component and connector view and allocation view as shown in figure 2-4.

The module view describes the modular structure of the software. It is a static view and expresses static division of software as a set of units. Component and connector is a dynamic view and describes runtime entities and their runtime behaviour and interactions. The runtime entities include process, thread, object, client server and data store (e.g. concurrency). The allocation view expresses the mapping of software. It describes the different software and non-software structure of the system and their relationships and interactions with each other (Kim, 2006 and Bass et al. 2011).

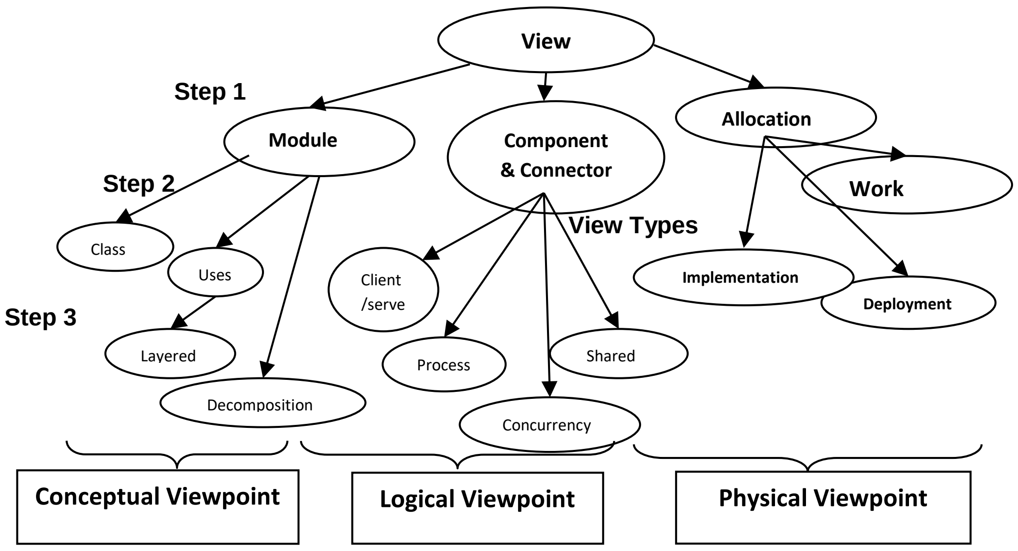


Figure 2-4: Software Architecture Views (Bass et al., 2011)

The comparison between the viewpoint and architectural view or style comes to the almost same conclusion, which is explained in Table 2-1. However, Bass et al. (2011) have given more detailed picture of architecture by describing the conceptual, logical and physical dimension of the software architecture. For example, in figure 2-4 view-types of module, view is a conceptual dimension and represents a conceptual view-point of software architecture, similarly component and connector view represent, and logical viewpoint and allocation view represent the physical viewpoint of software architecture.

The current discussion is not on the historical dimensions of software architecture or its adoption by the industry. However, the goal of the above discussion is to develop some understanding about areas that are relevant to this research. The next section discusses the key software architectural methods, that are used in the design and development of the distributed software applications.

2.3. Software Architecture Engineering (SAE) and Distributed Applications

Traditional methods provided by software engineering are not sufficient enough to cope with the complexity of modern distributed systems (Pinto et al. 2005). The decomposition of the software application into smaller independent and interoperable modules is a major concern of current web development. CBSD (Component based software development) and AOSD (Aspect oriented software development) can provide a solution for the development of the independent and interoperable application. (for details chapter 4).

2.3.1. Component Based Software Development (CBSD)

CBSD is also known as component based software engineering, is a branch of software engineering that focuses on the decomposition of software components into functional or logical components with well-defined interfaces. It comprises of reuse based approach of defining, composing and implementing loosely coupled components (Kwong et al 2010). The main advantage of CBSD is, that various processes and functions of the system can be kept into separate components, so that data inside each component can be semantically related. This phenomenon is known as separation of concerns.

The separation of concerns is a design principle used in CBSD to separate the computer program into different sections addressing a separate concern. With all the benefits, such as clear traceability from requirement to the implementation, the modularisation of concerns in a complex system can cause two problems known in the literature as scattering and tangling. Scattering is when, implementation code spread out on many modules, attached to one concern. The concern influences the implementation of the module. Therefore, the implementation is not modular. The tangling is occurred when the concern is intermixed with other concerns in the code (Kiczales and Mezini, 2005, Pessemir et al. 2008). These issues can be addressed by using AOSD.

2.3.2. Aspect Oriented Software Development (AOSD)

In literature, AOSD is described as a technique that can be used to improve the separation of concerns in a complex distributed software application. Traditional software application development focuses on the decomposition of main functionalities into smaller units. AOSD focuses on the identification, specification and representation of the cross-cutting concerns. AOSD is based on abstraction called aspects. Aspect encapsulates the functionality that may be needed at the several places in a software program (Brichau et al., 2008). Therefore, AOSD is aimed at the automated modularisation of the cross-cutting concerns, the modularisation that goes beyond the generalised procedures.

The approach for adopting the aspect to the architecture can be either symmetrical or asymmetrical. In asymmetrical approach, only the module with the tangled or scattered concerns are modularise using aspect. In symmetrical approach, aspects are represented as components and all the concerns are modularised using aspects (Brichau et al., 2008). The main benefit of this approach is reusability of aspect because components are considered as highly reusable entities. (Chapter 4 describes the adoption approach for the aspects and components)

2.3.3. Integrating CBSD and AOSD

The research in the area of CBSD and AOSD integration is now quite mature as compared to when attempted in Kiczales and Mezini, (2005), Masuhara. and Kiczales, (2003) and Kiczales et al. (2001). The benefits of integrating CBSD and AOSD are discussed in Pinto et al. (2005) and Pessemir et al. (2008). AOSD and CBSD are two different technologies. Integrating AOSD principles into CBSD can help to improve the evolution and maintainability of components by extracting crosscutting concerns from components and putting them into aspects. In addition to that, those crosscutting concerns can be managed separately without affecting the whole functionality of the components. Kiczales and Mezini, (2005)

The functional independence provided by AOSD to the component can be the key to solving the problem of decentralisation. However, AOSD approaches mentioned in Kiczales and Mezini, (2005), Kiczales et al. (2001) and Duclos et al. (2002), prevent the aspects to be reused in a different context. Therefore, a model is required which can integrate the AOSD and CBSD. Such a model (component and aspect model) is proposed in chapter 4, using the fundamental principles described in Brichau et al. (2008), Pinto et al. (2005) and Pessemir et al. (2008). Based on this model each component of DSNA (Decentralised Social Network Architecture) are designed and implemented.

The next section describes the use of SAE (Software Architecture Engineering) principles in designing and implementing distributed applications.

2.4. Distributed Application Architecture

The web is perhaps the largest distributed application. Learning about the key principles and architectural style underlying the current web can be helpful in explaining its technical success and may lead to improvements in other relevant aspects of the web (Fielding and Taylor, 2002). This part of the chapter attempts to explain the architecture of the web as a distributed application, its key components and how these component works together in different architectural styles.

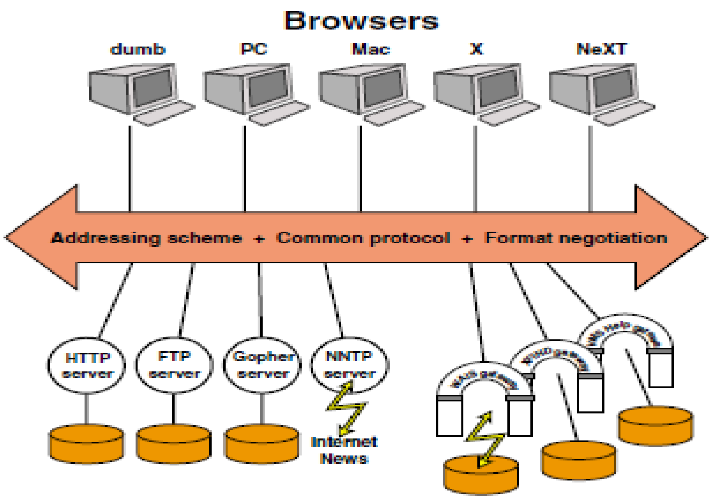


Figure 2-5: Architecture of the web used in CERN Project, taken from Berners Lee et al. (1992)

Figure 2-5 shows the architecture of the system developed by Berners-Lee and team in the CERN project to integrate different source of information. For that purpose, a system was built based on client, server and browser. The main function of that system was to exchange documents located on different systems. A presentable interface (i.e. Browser) was built to make the interaction between various systems possible (Fielding, 2000).
Due to growth in data, soon after this early design, a web community exceeded its limits. The need was to attach and generate pages and data dynamically based on the services needed by the users. On the other hand, the concept of web application also affected the way web architectures were developed. This led to the integration of software engineering and

hypermedia system principles to help develop the architectures for the modern web (Webber et al. 2010).

2.4.1. Web Application Architectures

In the modern web architecture, the elements interact with each other by the means of an interface. The interface encapsulates the detailed view of the system and partitions it in public and private views or sides. There are two partitions in traditional client-server architecture-based web system, one for public view (client side) and another for the private view (server side) (Bass et al. 2011). The web application is a complete piece of software with application and business logic attached to the data and governs by software engineering principles (Langegger et al. 2005).

Perry and Wolf, (1992), Garlan and Shaw, (1994), Kruchten, (1995), Conallen, (1999) proposed concept was originated from the software engineering principles and attempts to cover most of the aspects of the modelling process of web application, however, lacked in providing a framework to integrate the later investigated concepts into the field, for example, web services.

Conallen's basic web architecture did build consensus among the research community on its applicability for building modern web applications, however, has failed to change the stateless nature of the web. For example, the communication between the client and web server is stateless and to manage the session state, either cookies or IIOP (Internet Inter-Orb Protocol) elements must be added and this is accomplished by using different architectural styles (Fielding and Taylor, 2002; Booch, 2001).

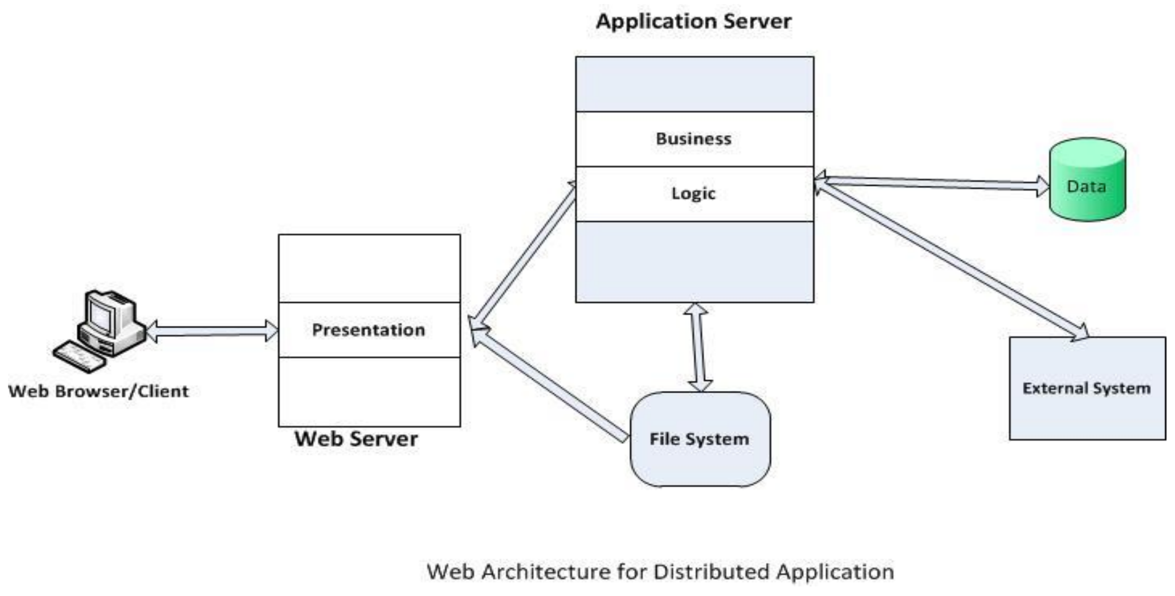


Figure 2-6: The basic Web Architecture; adopted from (Conallen, 1999) and modified according to the need of this research.

The shortcomings of basic architecture were resolved by REST proposed by Fielding, (2000) in his PhD thesis, to overcome the deficiencies of early web architectures. His work resulted in the set of operation with consistent semantics to build an architecture that can support any type of web applications. REST (Representational State Transfer) architecture describes the web as a hypermedia application, which linked resources by exchanging their states. REST is based on client cache, client connector and stateless server.

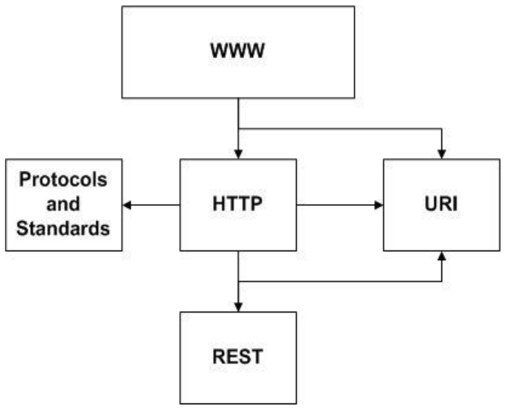


Figure 2-7 Current WWW structure (W3.org)

Table 2-2 summarises important research publications, architectural concepts and their implementations to cover the evolution of the process that software architectures went through towards Web architectures and social web architectures. In the architectural concepts column, software architecture concepts are listed that are discussed in the publications and in implementation column the technologies used to realise those concepts.

Publications	Architectural Concepts	Implementation
Shaw, (1990)	The Concept of abstraction	Software components
Perry and Wolf, (1992)	Architectural Elements, Views and Style	UML-based architectures
Berners Lee et al. (1992)	Network architecture for WWW	Component-based client-server architecture for the web
Garlan and Shaw, (1994)	Architectural styles as computational component of a system	Used Components and connectors to describe the style
Kruchen, (1995)	Architectural styles, views and elements with Object Orientation	UML Models
Shaw and Clement, (1997)	Component and connectors further clarified	Set of a software instruction functions through data and interface. Object and process
Conallen, (1999)	Web Architecture Elements	UML
Fielding and Taylor, (2002)	Redefine and subdivide elements into data, processing, connecting elements	REST based Web Architecture
Clement et al. (2003)	View, View types and view Points	Set of elements, Constraints and styles
Anderson, Graham and Wright, (2000)	Conceptual Architecture	Client/Server architecture
Bejar et al. (2009)	Conceptual architecture	Architecture of web base information system
Zachman, (1998), Noran, (2003)	Logical Architecture Physical Architecture	Data Model Process Model
Zachman, (1998), Noran, (2003), Garland and Anthony, (2003) Rozanski and Woods, (2005)	Physical Architecture	Implemented with the combination of Rules, process, data, functions, structure and structure location
Berners Lee et al. (2006), Berners Lee et al. (2007)	Web Architecture elements and structure of implementation components	HTTP, HTML and URI
Bass et al. (2011)	Implemented Web Architecture	Implemented using Attribute Driven Design, see figure 3.
Meier et al. (2008)	Web Application Architecture element describe according to the need of the modern Web	Multi-Tier, Multi Layers (Data, Logic, Presentation) and REST styles
Yeung et al. (2009) and Hendler and Berners-Lee, (2010) Berners Lee et al. (2006) Berners Lee et al. (2007)	Social web application architectures and necessary improvements needed for the next generation Web	Semantic web technologies

Table 2-2: Evolution of the Web architecture components towards Social web architecture

2.4.2. Social Web Architecture

The social web networks are a mixture of different types of web applications, developed to provide certain functionalities to the users. The success of Social Web Networks is very much due to the use of new software development and communication paradigms such as AJAX (Asynchronous JAVAScript and XML), REST (Representational State Transfer), JSON (JavaScript Object Notation), and web services. In this section, social web architecture is investigated with the aim to present its global version based on software architecture principles. To demonstrate this, an architecture of a prototype application is presented.

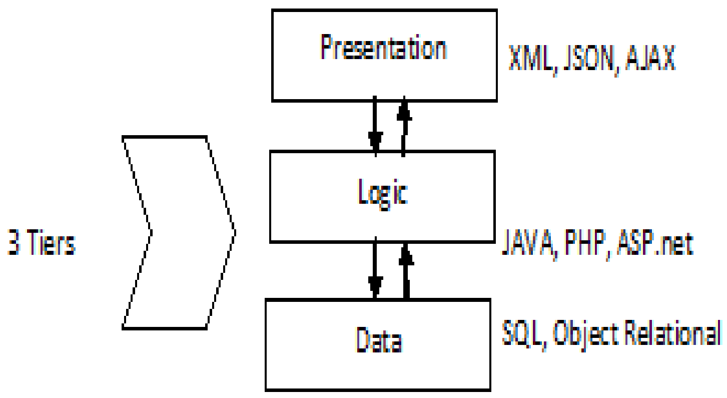


Figure 2-8: Conventional 3 Tier web Application architecture

The Social Web architectures can be designed in various forms-based needs of the application. For Example, multiple tiers-based architecture (Figure 2-8) may enable you to operate multiple environments based on your operational requirements and system resource usage. Different components can be deployed onto the tiers based on matching resources to increase the operational performance (Meier et al. 2008). However, the increase in tiers and component distribution on tiers can reduce the performance, increase the operational cost and complexity. Serious consideration should be taken in choosing the communication paths, protocols and states between the tiers, i.e. Stateless or state-full (Hill, 2009).

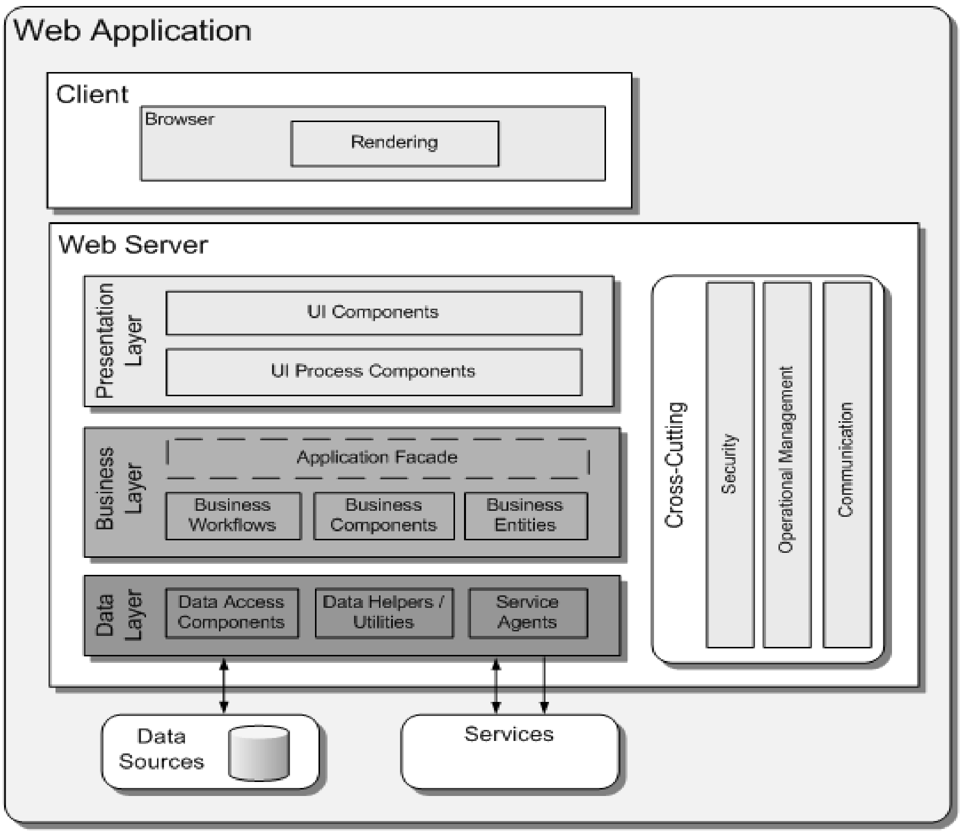


Figure 2-9: Layered Architecture (Generic), from Hill, (2009)

Figure 2-9 is an example of layered architecture. According to (Hill, 2009) layered design can help you to decompose a complex system design into a logical grouping of software components. Layered architecture helps you to differentiate between the different type of tasks that will be performed by different components of the systems, make it easier to create a design and increase the reusability of the components.

Yeung et al. (2008) and Hendler and Berners-Lee, (2010) work on the social web and decentralisation is fundamental and used to define the very basic structure of social web architecture as shown in Figure 2-13. The architecture consists of 3 main components Client, Server and external APIs. Each component is attached to other components, according to a specific rule.

The social web architecture above complies with the software architecture principles proposed in Perry and wolf (1992), Garlan and Shaw, (1994), Kruchen, (1995), Celment et al. (2003) Bass et al. (2011). Based on Bass et al. (2011) the above structure is a structured set of elements bind together in certain rules. The architecture shown in figure 2-10 is a conceptual architecture of the social web and the first step towards much complex architecture proposed discussed later in this research.

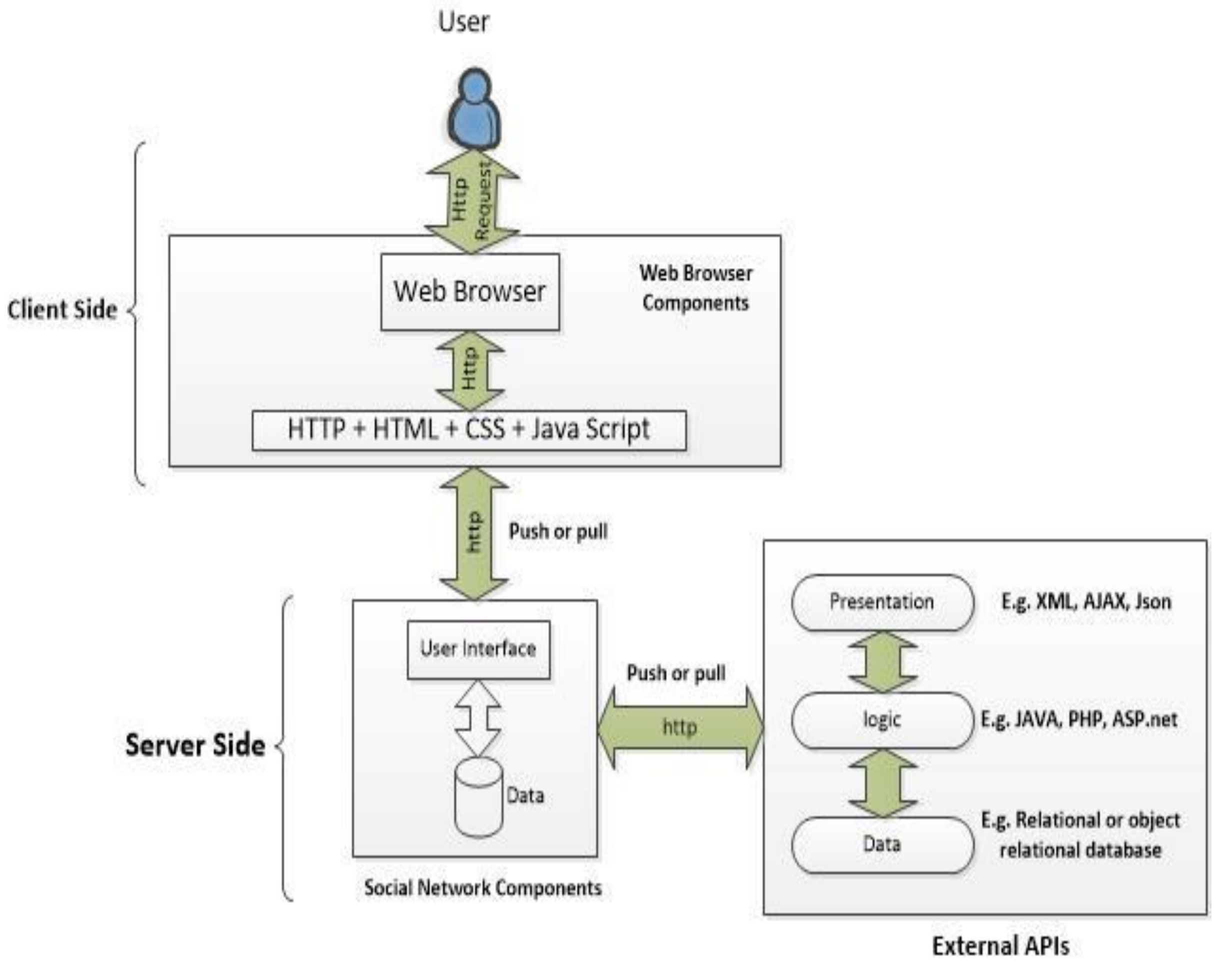


Figure 2-10: An example of Social web architecture with its basic components taken from (Yeung et al. 2008) and modified

The above research can be helpful to form an opinion that, the social web architecture needs standardisation of components in the same way as the architecture of the WWW, because of the changing requirements and new technologies. The work of Laine et al. (2011), Bejar et al. (2009), Brambilla et al. (2006), Langegger et al. (2005), Tiwana and Bush, (2001) on web architectures is either platform specific or done under certain criteria that reduce the chance of the artefact produced in that work, to be used independently.

The work of W3 Technical Architecture Group of WWW proposed an architecture of the web. The purpose was to standardise the essential elements needed to build architecture for modern web applications (W3C, 2004). W3 architecture describes basic needs and concepts but does not provide a solution to integrate modern technologies to produce complex web systems. The recommendations made by the research community to W3 for improvements, opened up new possibilities, leading to the recognition of Semantic Web also known as Web 3.0 or Open Web. Figure 2-11 Demonstrate how the future of the web will look like.

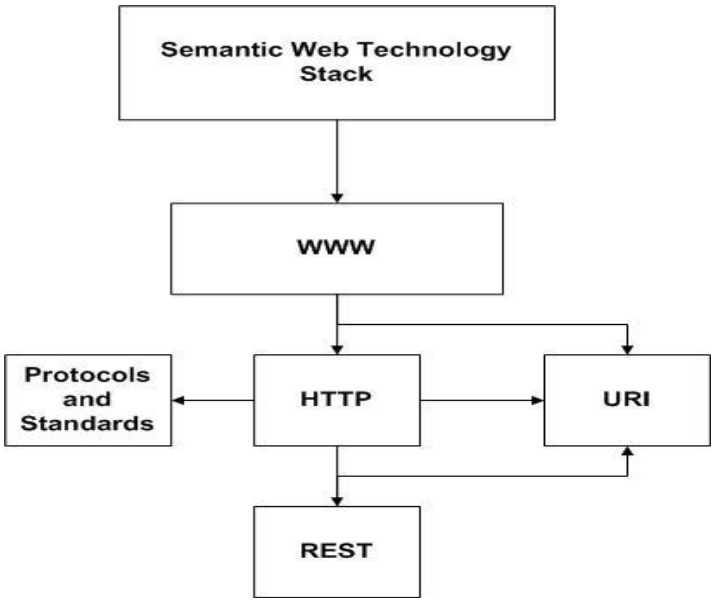


Figure 2-11: WWW and Semantic Web (W3C, 2013)

The understanding of the key functions of the Web architecture's components and their contribution towards the design of social web architecture is key for finding the relevance between software and social web architecture. The findings of this study will help towards the design of the main artefact. The next section extends the research to semantic enhancements of social web architecture.

2.5. The Decentralisation Problem Scenario

In the light of above discussion and to make the vision of this research on social networks more explicit, a scenario has been described to shed light on the problem. The argument is made on the usability of various social network functions across different SNP. The SWAT v1 (W3.org, 2015) is adopted as a standardised scenario. SWATv1 is described in various dimensions of the decentralised social web such as data portability, which is used to test the data portability between DSNPs.

Alice has a profile on SN1 (or server 1) and she manages another profile for photo sharing at SN2, as it provides better multimedia functionalities. Her friends Bob and Tony uses SN3 and want to share their photos with Alice. To do so they must join each other's networks. Their data is stored on a centralised platform and restricted. Because of restrictions posed by the service providers, it is not possible to interpret the data across different networks.

In the decentralised social networking platform, according to SWATv1, Alice should be able to send a message to Tony, and Bob and they should be able to reply back.

In the context of this scenario, there are various opinions in the literature and in open sourced community, to understand and solve this problem. In order to gain insight knowledge, the section explains how various methods and technologies are used to solve the problems in decentralising the SNPs.

In the following sections, the term peer(s) and node(s) are used as a synonym to the user(s) or client(s) and the term social web and social web platform when unqualified is a synonym to the social network and social networking platform.

2.6. Distributed and Decentralised Social Web Architecture

Despite the issues of communication and data portability, another issue decentralised architecture faces is research related that includes the uncertainty and confusion about, whether the decentralised architecture is a distributed architecture or not and if not then how to describe it. In Berners Lee, (2009) view decentralised social networks are the application of semantic web. Tramp et al. (2012), proposed distributed social network architecture based on three basic principles, linked data (for data publishing), service decoupling (enable users to able chose between different services), and protocol minimalism (enable RDF to triple to communicate between different nodes of social network).

The term distributed social networks is frequently but incorrectly used to describe all the decentralised social network (Narayanan et al. 2012).

In literature, social network architectures are described in four categories, federated (still experimental, an ecosystem of interoperable implementations in the client server mode) distributed (peer to peer), decentralised and centralised architecture. The research done in the different areas of the decentralised social network is mainly specific to certain aspects of social network and lacks in the generalisation of requirements that are required to design an architecture. That is why there was a need to illustrate general requirements for the decentralised social network as described in the above scenario.

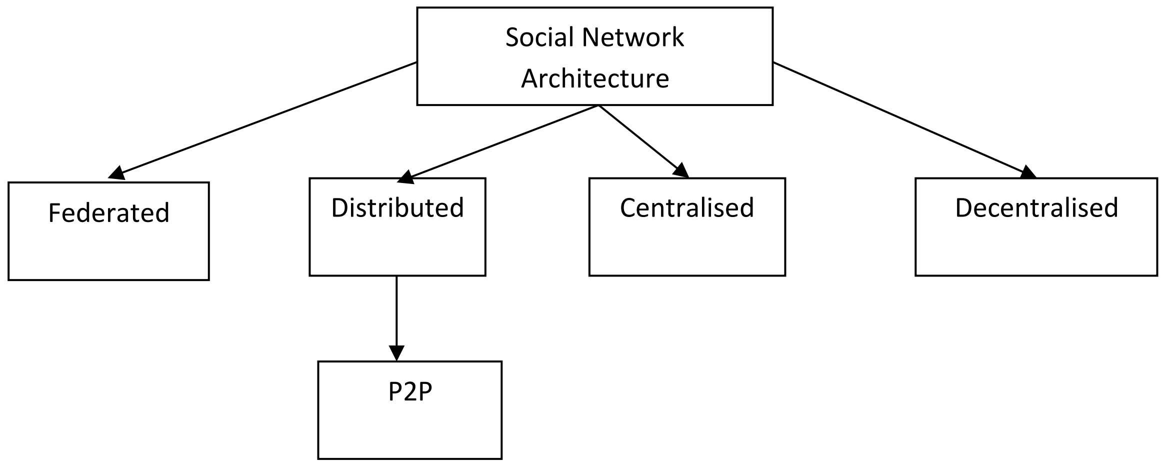


Figure 2-12: Types of Social network architectures

There is a common consensus (Chowdhury et al. 2015), (Famulari and Hecker, 2013) (Buchegger et al. 2009) and (Datta et al. 2010) that current social network structure is highly distributed having central authority (server) but they are not decentralised (Maurer and Labitzke, 2014).

Therefore, this raises an important question that is when to call a social network architecture a decentralised architecture and which rules, standards, and principles a social network architecture must adhere and adapt.

To answer this question, in the next section an attempt is made to differentiate software and system architecture briefly because distributed, decentralised or centralised are considered as architecture structural decision (Bass et al. 2011) which must be made before the actual design of the software or system architecture.

As defined in section 2 of this chapter and in short, software architecture is the fundamental organisation of system components, their relationships, and interaction, with the objective to understands and improve complex application structures (García-Castro et al. 2008). Whereas, system architecture describes the mapping of software architecture components on to the machines (Traz, 1994). The system architecture is divided into four main types, centralised, decentralised, distributed and hybrid, recently federated architecture term is also introduced not in the scope of this chapter.

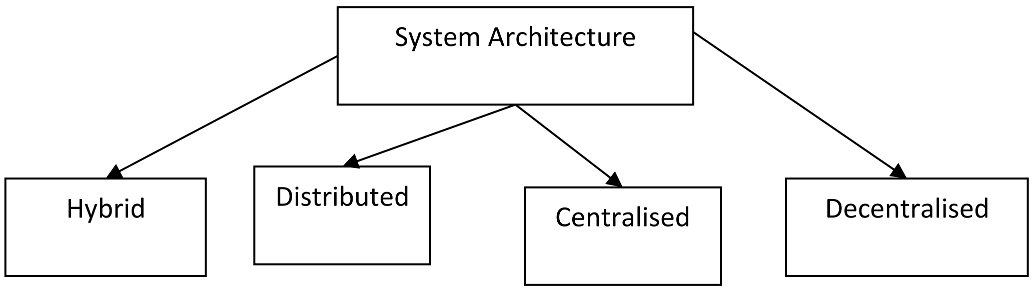


Figure 2-13: Types of system architecture

Centralised architecture, traditionally client server architectures (explained in section 2), in which architecture is divided into two logical division client and server. Figure 2-14 show how different (node or peer) communicated with the server in the centralised structure.

For example, In the given below figure user Alice and Tony are connected to a centralised social network SN1 and in the same way, they are connected to SN2 to communicate with their other friend Bob. In the centralised SN, the data is proprietary and is not shared with another SN, therefore it is not possible for Alice, Tony, and Bob to communicate with each other.

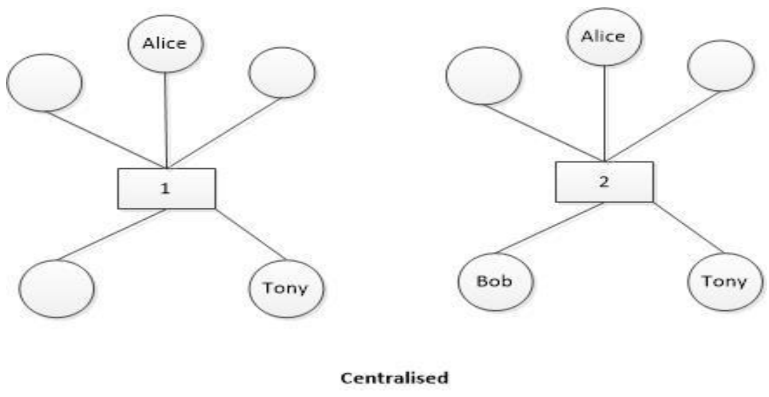


Figure 2-14: Centralised architecture example

2.6.1. Decentralised Network Architecture

To establish the consensus on concepts of the decentralised and distributed architecture, it is important to shed some light on their grounding principles and distinguish between them. In the distributed network system, nodes are located on networked computers communicate and coordinate with each other by passing message in order to achieve common goals. The main purpose distributed network architecture is to describe and define the components their interactions, relationship and deployment (Coulouris et al. 2011).

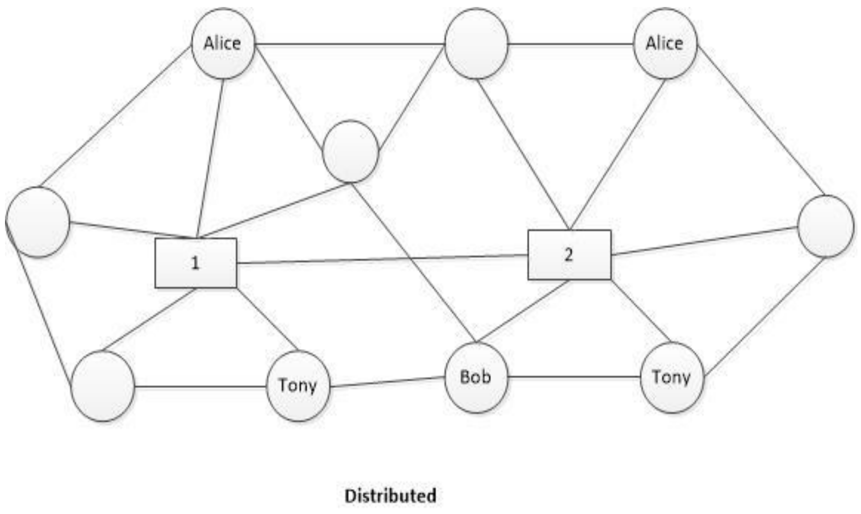


Figure 2-15: Distributed Network Architecture example

In the context of social networking, distributed mean processing of information is shared across the multiple nodes but the decision making may still be centralised. According to Han et al. (2011), a distributed social network does not have a central server but connected peers (nodes) acting as servers. Each peer has its own read or writes data access permissions as authorised by the central server. Peers may be open to communicate and share data with other peers, but the decision is very likely to be made on the central servers and may be connected as well, shown in the above figure 2-15.

For instance, in figure 2-16, SN1 and SN2 are connected through the third-party medium or service (which can be an application). This allows Alice, Tony, and Bob to communicate and share information with each other. At the architecture level, SN1 and SN2 are independent servers but connected nodes or users are made to share information. The next main concern about distributed network architecture is how the user data is stored. In distributed network architecture, data related components are spread physically across multiple locations and are connected to a single logical storage by a communication link.

In Han et al. (2011), they implemented similar concepts and attempted to extend the distributed social networks data access related functions. They proposed a flexible distributed storage to allow users to organise their personal contents at the place of their choices like cloud storage or personal device. Similar personal data storage projects include the LockerProject (Miller and Smith, 2010) and Owncloud (owncloud,org), BuddyCloud (buddycloud.com), and all of them provide some degree of easy to create personal cloud storage.

Indeed, the distributed networking-based platforms have their own benefits, related to data access and management, mentioned in (Coulouris et al. 2011). But the social platforms designed and developed so far are either insufficiently schema agnostic to be an application platform (as in Diaspora) or seemed not to address the need concerning sharing, that is pertaining to keeping the subscribers notified of changes. Similarly, access control functionalities provided in these platforms are also done in ad-hoc nonstandard complaint ways (Smith et al. 2012).

McLeod and Heimbigner, (1980) differentiated between distributed and decentralised database systems. They described decentralisation in the context of databases and according to them, decentralisation is a logical combination of components having their own logical and conceptual schema. These components are related but independent and they may or may not be disjoint. The figure 2-16 shows, how different peers (nodes) are connected to each other in a decentralised network environment. There is no server to server contact, each node can have connection to multiple servers.

According to the scenario, Alice is connected to SN1 and wish to communicate with her friends, irrespective of which SN they belong to. In the example decentralised SN shown in figure 2-19, Alice can communicate with any SN by fulfilling their requirements. The given depiction of DSN is only to show how DSN should work according to the above-mentioned scenario.

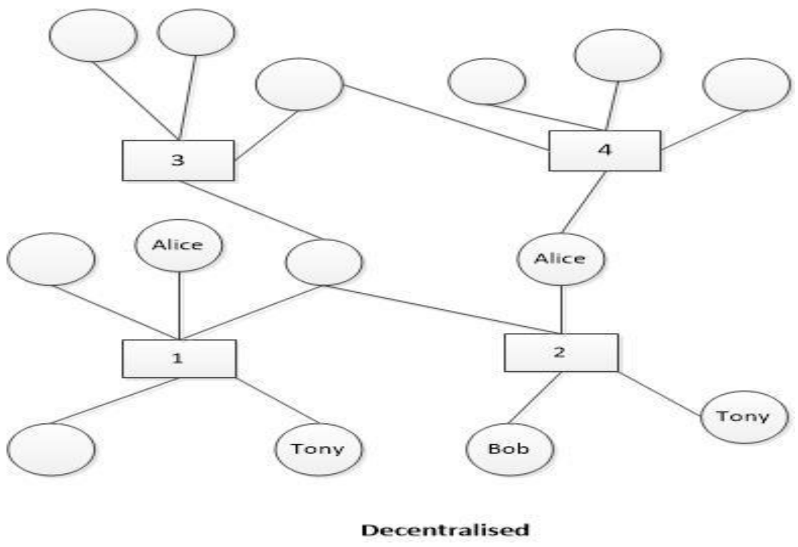


Figure 2-16: Decentralised network architecture

Currently, decentralised architectures are described in the context of author's perspective in the selected research and involved principles. In the current context, the opinion adopted on the conceptual description of decentralised network architecture is based on, how decentralisation is explained in computer networks and decentralised database related studies as earlier mentioned.

To sum up, a decentralised architecture or decentralised network architecture is a collection of entities called peers or nodes that interact with each other without the presence of a trusted central control authority. Each peer work towards achieving its individual goal (Suryanarayan et al. 2005). Therefore, there is no single point where the decisions are made. Every peer makes decisions for or towards their own behaviour. The next section describes the decentralised network architecture in the context of social networking.

2.6.2. Decentralised Social Network Architecture

The research in the decentralised social networks is mainly in the areas of security, privacy, and trust related issues. The research on these issues has described the decentralisation within these contexts. For example, Suryanarayana et al. (2005) explained decentralised architecture in context to trust enablement, between different peers in decentralised applications. Similarly, Seong et al. (2010) presented a decentralised social networking architecture which is security and privacy centric. The fundamental function of the decentralised architecture is to provide an open environment to different applications to interact with each other.

In open decentralised architecture, there is no authority preventing the addition of peers. Therefore, each decentralised peer is responsible for the task of determining the validity of information received from other peers. This local autonomous determination is the defining principle of open decentralised architectures (Suryanarayana et al. 2005, Khare and Taylor 2004).

Problems with the decentralised peer to peer applications are discussed in (Suryanarayana and Taylor, 2004) and (Suryanarayana et al. 2005), in which they emphasized on the placement of central authority that can coordinate the behaviour of different peers within the application. But the purpose of the decentralisation prevents the existence such central peer or authority in the architecture. Therefore, to solve this issue they introduced trust layer in the decentralised architecture. The purpose of this layer is to add trust component on each side of peers with the aim to handle all the information related queries including seeking, sharing and storing information. Hence in their view, it is essential that decentralised architecture-based applications must enable efficient storage and a reliable search mechanism for them.

The research literature is quite vague on decentralised social network architectures and limited to few architectural perspectives such as peer to peer. To define the DSNA and describe its structure, the effort was to differentiate between decentralisation and distributed architectures. Therefore, the above presented research shows the evidence that structurally, distributed (peer to peer) and decentralised architecture are different.

Berners Lee, (2009) view on the social network decentralisation which is mainly adopted by the research community, in which he described decentralisation as an implementation of semantic web and open standards into the social networks. However, in his paper he has illustrated components (mainly mixture of open and semantic web standards) of the architecture needed to decentralise the social networks but unable to give generalise description of decentralised social network architecture which is made up of the component he illustrated. Similar lacks are shown in (Chowdhury et al. 2015) and (Seong et al. 2010), as they described existing research efforts such as (Famulari and Hecker, 2013), (Buchegger et al. 2009), and (Datta et al. 2010) and in their opinion, the decentralisation offered by these systems is based on peer to peer architecture, therefore decentralised architecture and peer to peer architecture is equivalent.

To describe decentralised architecture in the context of social networks, McLeod and Heimbigner, (1980) provides conceptual ground that is, a decentralised architecture is a logical combination of components having their own logical and conceptual schema. These components are related but independent. Suryanarayana et al. (2005) defined decentralised network architecture in a context when two different application need to communicate in a decentralised way. These different applications are mentioned as entities that are also called peers, nodes or users. In this regard, decentralised network architecture is a collection of these peers that interact with each other without the presence of central authority and each peer is responsible for its own behaviours. Based on these aspects, a decentralised architecture is divided into two layers of abstraction, external architecture and internal architecture. The external facilitates the interaction between peers by describing the topological arrangement of peers and basic network infrastructure. The internal layer is responsible for describing the behaviour of the peers towards achieving their goals.

From the above discussion, it is plausible to conclude that the definition of decentralised social network architecture depends on how the components associated to, the role of the peers, nodes or users, the interactions between peers, nodes or users and the rules to relate those components are defined. As mentioned in chapter 1 section 1.1, a social web network is a combination of social structures made by set of actors and complex set of relationship between these actors (Wasserman and Faust, 1994). With all that is said, a decentralised social network architecture must be able enable description of components that are associated to the nodes role and interactions.

Within the context of this research, a decentralised social network architecture (DSNA) can be define as an architecture that embodies internal (i.e. the components responsible for internal functions of the social networking platform) and external (i.e. the components responsible for the external functions of the social networking platform such as communication) components, composed within the constraints and rules illustrated in architectural (DSNA) style (See DSNA Style in chapter 4 for details).

2.7. Drawbacks in the Existing Social networking Platforms

Since the focus of this research is on the functional design of the SNPs. Therefore, the drawbacks of the existing social networking platforms functionalities should derive the need of decentralisation to their structure.

2.7.1. Breach of trust

Unwanted information disclosure caused by the functionalities that allow third party applications to use user personal information. These applications are facilitated by the SNSPs to enhance the user experience and given full control over the user information. This exchange of information is explained in term of service document. In reality, only few users understand the meaning of this data exchange. In the architecturally decentralised SNP users have more options in term of which information should they share with other and where it will be stored, thus avoiding the breach of turst problem.

2.7.2. SNP Business Model

Another major drawback of existing SNPs is their advertisement centric approach. Due to that reason they lock user's data and exploits their personal information for targeted ads and other marketing purposes. Because of the network controls, it is not hard to envision a situation where information of very large part of the population can end up in the hand of an individual or a group. Considering the obvious privacy concerns this will ultimately harm the end user.

The business model of the social network service providers (SNSPs) is centred around the ads and users, and not much changes are made to improve the access to information for users. Recently, there was data breach in the Facebook services. A large amount of Facebook user's personal information, which meant to be used for data mining was used for other purposes without the consent of the users (Kayes and Lamnitchi, 2017).

The drawbacks in the design and model of the existing SNPs needs system of principles and guidelines to solve the issues related to the user security and privacy, data, and its usage. The drawbacks of the existing SNPs are very well documented in (Seong et al. 2010) (Paul et al. 2012)(Zhong and Sastry, 2017) (Bahri et al. 2018) and propose various solutions, among them decentralisation of the SNPs is commonly discussed. However, their debate is mainly restricted around the concepts of distributed data storages, peer to peer structures and dealing with security and privacy of the user data.

Halpin and Tuffield, (2010), sum up the drawbacks in two main categories, walled garden, and Centralisation. These two main categories are used to further identify the problems in the SNPs, also how the need of decentralisation reflects on solving the main issues.

2.7.3. Walled Garden

The open and distributed nature of the web as universal space of information and knowledge collaboration have always been a key to its success. Until recently, the phenomenon of the social web has created a problem known as Walled Garden. The problem is caused by restrictions posed on the access and manipulation of user personal data via proprietary interfaces, so creating a "wall" around connections and personal data (Yeung et al. 2008), as illustrated in figure 2-17.

The best known social networking sites such as Facebook, MySpace, Twitter and etc, have restricted themselves to relationships between the people on one site, the social web should be extended to the entire web. For example, people can call each other no matter what service provider they are using, same as people can send a message to each other using email irrespective of their service provider. The social web should allow people to create a network of relationships across the entire web by giving people access to their data and privacy (Halpin and Tuffield, 2010; Hu and Lau, 2013). This issue can be solved using a truly open and decentralised architecture for the social web platform is required

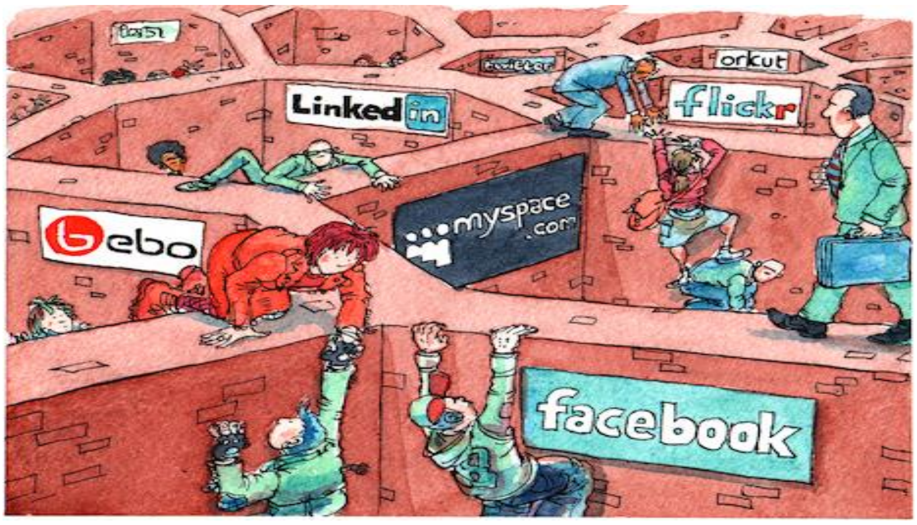


Figure 2-17: Depiction of Walled Garden by Yeung et al. (2008)

2.7.4. Centralisation

Traditionally social web networks are based on centralised architecture, making the companies providing these services sole owner of user's data. Due to this reason data stored on these websites is not accessible to another site, and users are not allowed to reuse their own data on other similar sites, thus forming data silos, an isolated island of data (Yeung et al. 2008). Each social application has its own data not knowing of the relevant data available on the other applications and platforms, exposing the lack of interoperability between the applications and services they provide. Similarly, due to these restrictions, ordinary users are unable to have the ownership of their own data and therefore cannot reuse their data and profile information on the other social network platforms (Tandukar and Vassileva, 2012).

The above-mentioned issues in social web architectures affect the user experience and cause the following problems, shown in the table 2-3, such as data Portability or interoperability, User Identity and profile reusability, Linkability and privacy of user data (Halpin and Tuffield, 2010).

Problems	Description
Data Portability	The user cannot access their data and share it as they like (they have limited options). Information available on the social websites can be accessible to other applications but current limitations and restrictions hinder the usability share their information across the different social web.
User Identity and Profile Usability	When user goes to a new site, they have to recreate the profile and all profile information and entice the friend again
Link-ability	There is no way of being notified for users if they are mentioned in any other social web, they are not a member of. There are restrictions that do not allow the users to create and distribute links between different social web platforms.
User security and privacy	Because existing social web platform are centralised that is why user access to their data is limited, which raise security and privacy concerns. For example, the user cannot control the way their information is viewed using various social web applications.

Table 2-3: Main problems in current social web platforms (Halpin and Tuffield, 2010)

2.8. Decentralisation in the Social Network Platforms

Developers are already using semantic web technologies to enhance the ability of the social websites to link, create and reuse content. According to Berners Lee, (2007), online communities can serve as a rich data source for semantic web technologies, and this linked up data can enhance the view of individual or community across social web platforms.

There are many proposal and implementation in the field of social network data portability at industrial, developer and academic research levels. They all address some issues associated with current social web platforms, e.g. identity management, content management etc. However, few of them managed to jump start and only a few of them grown to millions. There are different aspects of portability problem between the social web platforms. For example, according to Ballester et al. (2010) solving security and privacy problem can help to achieve total or partial data portability.

Ballester et al. (2010), presented semantically enabled security architecture using decentralised approach. The proposed semantic interoperability and access control layer (SIAC) is intended to make applications independent from data, privacy policies and empower users to take control of their own personal information. This approach attempts to give users an ability to define their own access control rules. Their suggested architecture will be able to collect information distributed all over the social networks, using Semantically-Interlinked Online Communities (SIOC), (Breslin et al. 2005 and SIOC-Project, 2015) to aggregate information and FOAF (Foaf-Project.org, 2013) vocabulary to describe user information, allowing users to have one global version of their information. In their research, much of the focus is on the security of user's data, not on the organisation of knowledge gathered from different resources, such as blogs, message boards, online discussion and mail posting.

The technologies mentioned in Ballester et al. (2010) research can provide services for social networks independently like FOAF provide profile portability by semantically describing user information using RDF graphs and SIOC to aggregate user information from distributed resources. The combination of security rules and access control layers embedded with semantic technologies are also suggested by Sloni, and Sharma, (2011) and Seong et al. (2010).

However, the work of Seong et al. (2010) is unique in a way as they used Personal Cloud Butlers (PCB) as decentralised data storage to index user personal data. Personal cloud butler is a program that manages user personal could of information on his or her behalf (Song et al. 2010). Berners-Lee, (2009), presented socially aware cloud storage for user personal data, which is conceptually similar to cloud butlers, but the elements of implementation are different.

According to Breslin and Decker, (2007), Sintek et al. (2009) and Cena et al. (2013) there is not a single consistent, standardised approach or method available to allow two different social web networks to interoperate and enable complete system-wide data exchange.

For example, Internet Mail (also known as Email) architecture is appeared to be only standardised architecture available that can be used to enable user communication between different websites using messaging protocols. As compared to the Internet Mail, there is no seamless way of communication between two different social web networks (Sintek et al. 2009).

Each Social network stores user information differently. If the user is interested in using the services offered by other than the site he is registered, then he may have to register again. As a result, user may be registered on several social networks, causing data to be scattered, duplicated and disorganised (Ballester et al. 2010).

If it is possible to represent information in a common language or standards like email, then social networks may be able to interoperate. From this perspective Email or SMS can provide guiding principles for the development of an architecture for data portability (Hu and Lau, 2013). Given below table provide four parameters to transfer message within social networks. The link can be static or dynamically created on the initiation of interaction between the users. The message or the communication is either unidirectional (one-sided) or bidirectional (two-sided). Accessibility is, how the message will be displayed or viewed. Verification is about the authentication of the message and sender. In table 2-4, the parameters that can be attached to the social message have been listed.

Parameters	Values
Links	Static, Dynamic
Direction	Unidirectional or Bidirectional
Accessibility	Read, Write
Verification	Authorisation, Authentication

Table 2-4: Parameters for social network messaging service (Hu and Lau, 2013)

2.8.1. List of standards and Protocols

The decentralisation of the social web is standardised at identity, profile, privacy and activity levels. The main assumption is that all these frameworks, protocols and standards should be working together seamlessly in an architecture of the federated social network.

Federated social network aimed at creating an ecosystem of standards-based interoperable implementations of social networks. For example, Diaspora is a hybrid social network that means a combination of both distributed and federated, OStatus, being coordinated by W3C, uses existing protocol for microblogging, rather developing them from scratch, which is a positive side (Narayanan et al. 2012).

The use of so many protocols and standards to solve social network problems is one of the flaws decentralised social network has, which not only complicate the development but also affects the friendliness of user interface. Table 2-5 below, highlights the list of protocols and various standards that are the part of decentralisation initiative in social network platforms to achieve portability.

Identity	Profile	Privacy	Activity
OAuth (Server Side) **Purpose:** Token-based Authentication protocol **Status** (Stable)	**XRD or YADIS (**XML and RDS**)** **Purpose:** Portable Contact Information **Status** (Ongoing Work)	**P3P (Platform for Privacy Preferences)** **Purpose:** Expressing Privacy via machine-readable languages **Status:** Some features Implemented in IE and FireFox later discontinued. (Too complex and unstable)	**XMPP (Extensible Messaging and Presence Protocol) or Jabber** **Purpose:** Initially developed as messaging services now can be used for passing XML message or data between machines **Status:** Stable, widely used by Google GTalk and open source messaging projects
OpenID (Server Side) **Purpose:** Centralised Authentication **Status: Stable**	**VCard or VCARD 3** **Purpose:** Portable contacts (Mail Programs) **Status** (Stable)	**POWDER (Protocol for Web Description Resources)** **Purpose:** Privacy Description Describing group URIs and linking them to XML and RDF. **Status:** (Discontinued, failed to describe single URI)	**Pubsubhubbub (Publisher subscriber hub)** **Purpose: (**Provide push request architecture over Pull based HTTP web architecture**)** used with ATOM to provide feeds and status updates **Status:** Stable with ongoing work deployed in STATUSNET and DIASPORA projects
WebID or FOAF+SSL (Client Side) **Purpose:** Decentralised authentication **Status** (Stable)	**FOAF (XML, RDF, URI)** **Purpose:** Describe social network user information (support Decentralise applications) **Status** (Stable)	**AIR (AMORD in RDF)** **Purpose: Policy description language** **Status (**Stable but with only RDF data**)** No practical implementation	**ActivityStreams** **Purpose:** List the activities performed by the user. With Atom serialisation, the goal is to make functions like Status Updates cross platform. **Status:** Stable with ongoing work (JSON Serialisation) widely deployed in Facebook, Google and the BBC
InfoCard (Client Side) **Purpose:** Identity Authentication and storage **Status:** (Ongoing Work)	**PortableContacts** (VCAD4XML, XML and RDF Support) (VCard 3.0 extension, OAuth integration) **Purpose:** Profile Provider, Rich in attributes **Status** (Stable)	**XACML (eXtensible Access Control Markup Language)** **Purpose:** Express access control rules in machine-readable format **Status: (**Stable with ongoing research on extending it to control privacy on social web**)**	**OStatus:** **Purpose:** Manage user status updates in the open social web. It works in conjunction with other activity standards as mentioned above. **Status: (**Stable with ongoing research**) HTTP based meta-architecture to provide Activity based functionalities in social web applications**
XAuth Server Side **Purpose:** Identity Connect Protocol **Status:** (Ongoing Work)	**OpenSocial** JavaScript APIs **Purpose:** Get access to Profile data using Open Authentication Protocols **Status** (Stable)	**RIF (Rule Interchange Format)** **Purpose:** Exchange rules languages between different rule engine **Status (**stable but not practically mature, limited to research projects**)**	
		ORDL (Open Digital Rights Language) **Purpose:** Express policy in machine-readable format (XML, RDF) **ORDL 2,** additional functions of access control and permission control and privacy control. **Status:** (Stable, Used in OneSocialWeb Project for policy, privacy and control description) Ongoing work on its binding with XMPP	

Table 2-5: List of open standards used for social web applications decentralisation

2.8.2. Decentralised Social Networking Projects

In recent years, a lot of work undertaken towards making decentralised social networking real. The projects listed below meet W3 FSW standards, are implemented in various forms and have major followings. Decentralise Social Networking Project are divided into three main approaches, web server hosting, federated server approach and distributed server or nodes approach (P2P).

Projects	Status	Protocols	Features	Privacy Supports	Approach
StatusNet	Work in progress	OStatus, OpenID, FOAF	Microblogging	None	Web server hosting
OwnCloud	Work in progress	WebDAV, Open Collaboration Services, Distributed Repositories	Photos, Media sharing, RSS	Yes	Web server hosting
GNU social	Work in progress	Status, StatusNet, Mysql and PHP	Microblogging	None	Web server hosting
OneSocialWeb	Beta version, work in progress	XMPP	Microblogging, Profile	Yes	Federated
Higgin's Project	Inactive, work in progress	RDF, OWL, Personal Data Store	Social Networking	Yes	Federated
Diaspora	Stable with work in progress started in 2014	OAuth, OpenID, Ruby	Social Networking	Yes	Federated + distributed nodes
Disco Project	Under Development	OpenID, OAuth	Microblogging	None	Web server hosting
SMOB	Work in progress	FOAF, SOIC	Microblogging	None	Web server hosting
Google Wave	Work in progress	XMPP	Messaging, Microblogging	Yes	Federated
Friendlka	Work in progress	Atom, Pubsubhubbub, Salmon, ActivityStream, OpenID	Social Networking	Yes	Federated
Elgg	Work in progress	FOAF, Pubsubhubbub, REST API, RDF, ActivityStream	Profile, Microblogging, Stream	Yes	Federated

Table 2-6: Decentralised Social Networking Projects

The table 2-6 highlights some important features and projects related to social network architecture. The mentioned solutions to solve SNPs aforementioned problems are a complex, mixture of many standards, protocols and lacks in too many successful implementations. In the literature, there are various viewpoints available regarding the complexity of decentralised SNP.

For example, according to Hu and Lau, (2013), building a decentralised platform from scratch is unwise, instead, they proposed a network of all social network a Meta Social Network. To achieve this, cross-platform middleware is proposed to unify interfaces and data structure at the services level.

2.9. Social Network Platform Decentralisation Initiatives

The social network connects service (SNCS), use architecturally decentralised form of platform to provide identity authentication services to the user. Major social networking sites are providing third party websites connection services such as Facebook Platform, Google Friends Connect and MySpaceID using various kinds of open standards. These services allow third-party sites to build applications or (APIs) to extend social network access to their users without building their own social network.

For example, a third-party website can utilise the authentication services provided by social networking website to draw the attention of the users using that social networking site and the users can avoid creating more login and profile; instead, users can draw some bits of basic information of their social network profile to the third-party website.

Ko et al. (2010) and Tapiador et al. (2012), analysed SNCS services and done a comparison between different connect services. Google has a decentralised platform for social connect services, which can provide users with more customisation options regarding the handling of their data and profile, whereas Facebook and Myspace use their own platform and proprietary interface for data handling. In their opinion, a decentralised platform such as Google Friends Connect can enhance the user’s experience, but it increases the administration cost of data handling as compared to Facebook, which uses centralised platform.

The decentralisation of the social web is standardised at identity, profile, privacy and activity levels. The main assumption is that all these frameworks, protocols and standards should be working together seamlessly in an architecture of the federated social network.

Federated social network aimed at creating an ecosystem of standards-based interoperable implementations of social networks. For example, Diaspora is a hybrid social network that means a combination of both distributed and federated, OStatus, being coordinated by W3C, uses existing protocol for microblogging, rather developing them from scratch, which is a positive side (Narayanan et al. 2012).

The use of so many protocols and standards to solve social network problems is one of the flaws decentralised social network has, which not only complicate the development but also affects the friendliness of user interface. Table 2-6 above, highlights the list of protocols and various standards that are the part of decentralisation initiative in social network platforms to achieve portability.

2.9.1. Distributed Networking Initiatives

The decentralisation of the social web demands standardised means of exposure of social data to structured data web. Basically, Implementation of decentralised architecture is an application of semantic web and it relies on semantic web technologies and protocols (Berners Lee, 2009).

The groundwork to build a decentralised social network is available to certain level. There are certain projects developed that grown to millions, such as Diaspora, (Diasporafoundation, 2013) is an open source project based on distributed social networking, instead of having a centralised server they used PUBSUBHUB mechanism to allow the user to host their profile to a POD while networking. POD is like an independent space on an independent server. It allows publishing of your profile feeds through “ActivityStream” (Snell et al. 2011) and import or export your profile data. However, it does not have the notion how to deal with profile data scattered on different PODs. For example, someone steals your data and you do not know from where he got your data as POD send your data to servers you do not know without your consent. In the case of The Facebook or Myspace, you know the operator.

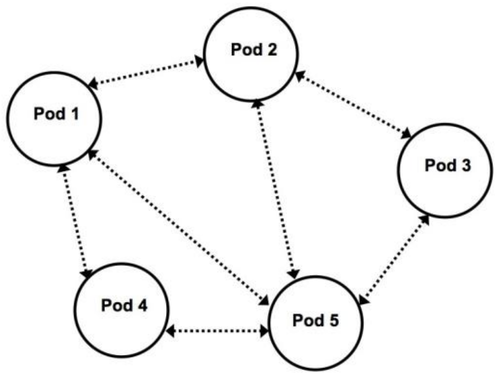


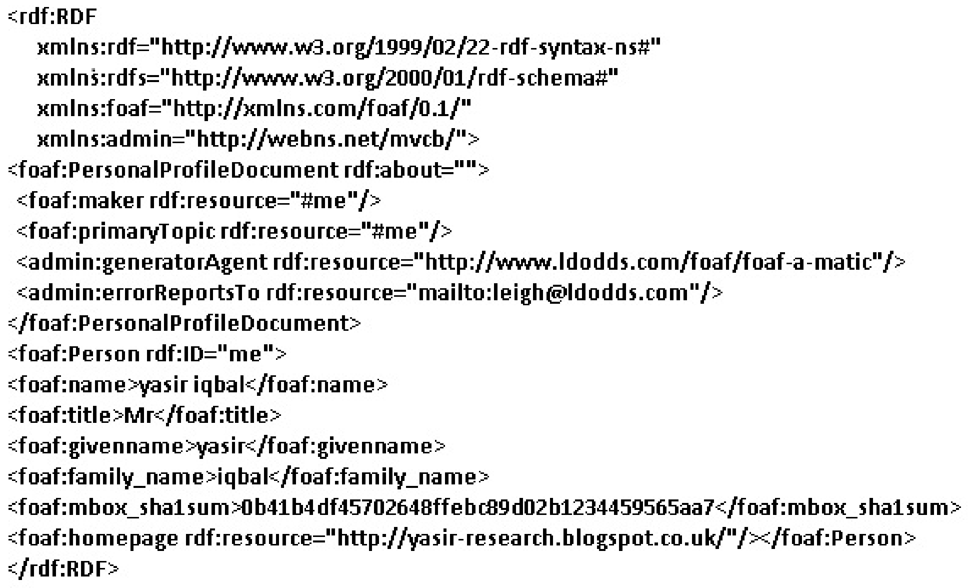
Figure 2-18: Distributed Host-based Decentralisation (an example) adopted and modified (Tandukar and Vassileva, (2012)

The rest of this section discusses details regarding the relevant efforts that are underway in open source community as well as novel platforms and architectures.

2.9.2. Open Sourced Initiatives

FOAF is a machine-readable ontology describes person's activities, their relations to other people and objects. The FOAF project, which defines the FOAF vocabulary, is considered as one of the first open standards for a social semantic application that constitutes of RDF technology with social web concerns (Brickley and Millers, 2007). Foaf-O-Matic is a first application that creates FOAF profile, enabling users to describe themselves using FOAF (Foafproject, 2013) properties and generating RDF based FOAF profile.

The main drawback of this application is the fact that it does not support the editing of the profile. Thus, to modify the profile one has to recreate from scratch or edit the RDF file. Bojars et al. (2008), presented the improved version of FOAF by adding more detailed about describing social network using SIOC ontologies. The following code snippet is an example of FOAF profile.



```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:admin="http://webns.net/mvcb/">
<foaf:PersonalProfileDocument rdf:about="">
 <foaf:maker rdf:resource="#me"/>
 <foaf:primaryTopic rdf:resource="#me"/>
 <admin:generatorAgent rdf:resource="http://www.ldodds.com/foaf/foaf-a-matic"/>
 <admin:errorReportsTo rdf:resource="mailto:leigh@ldodds.com"/>
</foaf:PersonalProfileDocument>
<foaf:Person rdf:ID="me">
<foaf:name>yasir iqbal</foaf:name>
<foaf:title>Mr</foaf:title>
<foaf:givenname>yasir</foaf:givenname>
<foaf:family_name>iqbal</foaf:family_name>
<foaf:mbox_sha1sum>0b41b4df45702648ffebc89d02b1234459565aa7</foaf:mbox_sha1sum>
<foaf:homepage rdf:resource="http://yasir-research.blogspot.co.uk/"/></foaf:Person>
</rdf:RDF>
```

Figure 2-19: FOAF Profile Example

Bojars et al. (2008) does provide the solution to reusability but does not provide any solution for how different profile should share the same URI to identify the same person.

In figure 2-20, a FOAF-based decentralised social network system architecture is illustrated. The proposed system allows users to manage their information on a trusted server (as shown in the figure, there are 2 trusted servers A and B) relying on some access controls policies to enable social network applications to use their information from FOAF for social network activity. A user manages data by themselves and central access point manages for social network applications, are some key points of their architecture (Yeung et al. 2008).

On the weak side, trusted server information repository can affect the reliability of social network services. Whenever there is a change in application user has to update the information repository in order to enable access to new services. According to Bortoli, Palpanas and Bouquet, (2011) opinion user should have the ownership of their data but the social network service providers should keep attending the privacy issues to provide better services to the users.

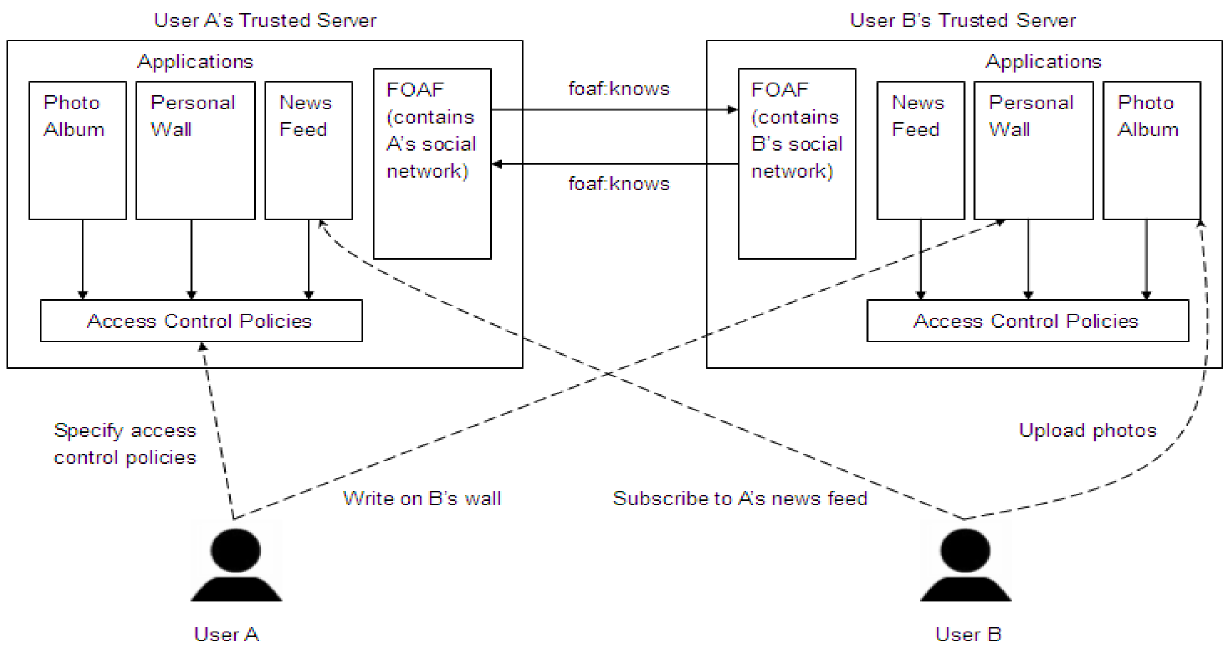


Figure 2-20: A framework of decentralised online social networking (Yeung et al. 2008)

2.10. Existing Versions of Decentralised Social Networking Architecture

Decentralised social network platforms are the application of semantic web, which not just about putting data on the web but also about making links so that people or machine can explore the web of data (Berners Lee, 2009). Decentralised architectures are distributed structures with trusted network of servers to provide a safe haven (Tandukar and Vassileva, 2012). These notions are the basic principles of social network decentralisation to achieve data portability.

Seong et al. (2010) presented an architecture for a decentralised social networks platform that has relevance to this research. Their proposed platform allows the user to retain control over their data by using distributed, decentralised storage, handled by butlers (explained in 2-4). Open APIs are used to access distributed data, which has integration with the access control APIs to avoid personal information disclosure. OpenID and certificate authority are used foster trusted communication and query propagation across the distributed personal data. The data is stored in RDF triple and ontologies are used to describe the system and user resources.

According to Yeung et al. (2008), a platform that allows users to share and communicate social data with other users. A prototype data browser "Tabulator" (Berners Lee, 2008) and linked data (RDF) editor is developed as an interface to provide a mechanism for user interaction. These types of platforms encourage users to store their data on the web in open standard formats such as OpenID, RDF and it should be accessible through URI. Therefore, users can use any social application that supports these open standards to access their profile and its data.

In the figure 2-22 platform, users are hosting their social data in an independent storage. Another perspective on decentralised platforms is the use of P2P architecture. In P2P, architecture data is stored at the peers and the availability of social data depends on the online behaviour of peers. However, this approach lacks in standards and stable solutions for user social data propagation and dissemination between the peers.

Cutillo et al, (2011), presented Safebook, a platform based on 3-tier architecture for online social networking platform, having focus on security and privacy. The first-tier handles data, storage, user relationship, content, and communication privacy. The middle tier is P2P overlay provides the application services, for example, look up services. The top tier consists of internet, transportation, and communication services. Social networking tier is core of the Safebook architecture. The users are connected in circles called logical rings. The innermost ring handles the relationship and trust between the friends. The outermost handles the requests for accessing the data and pass them to the inner rings. The safebook is an interesting concept but still to be implemented in any SNP.

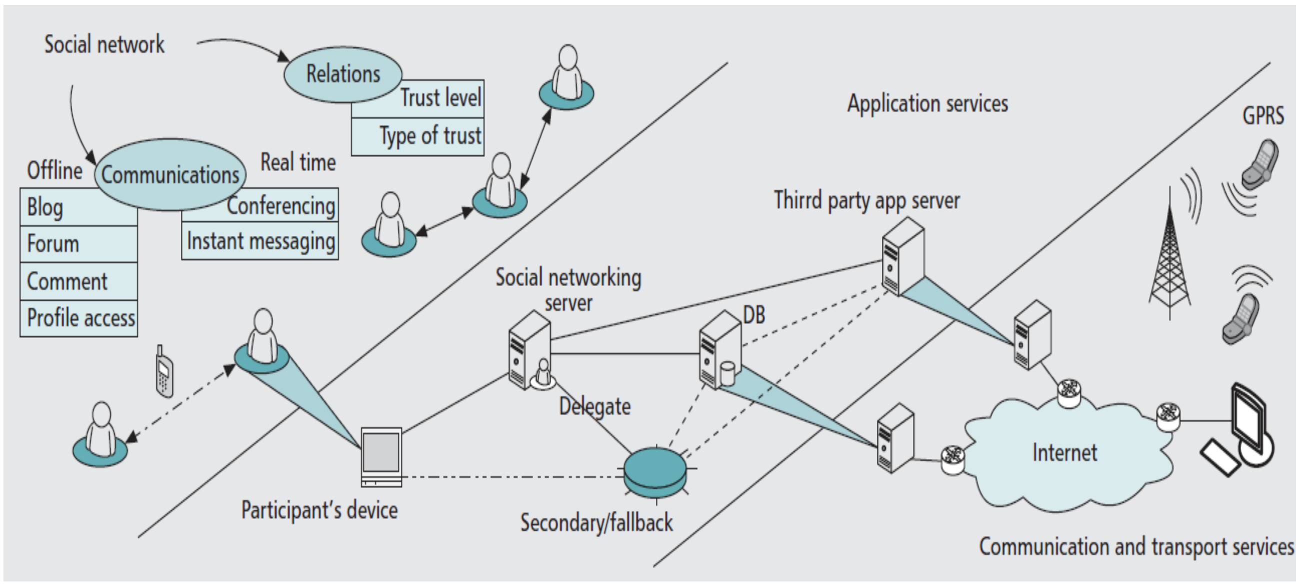


Figure 2-21: Safebook architecture (Cutillo et al. 2009)

According to Tandukar and Vassileva, (2012), if the user stores their data on the web in a standard format like RDF which can be accessed through URI attached to an independent interface, will give the user more accessibility of their data. The architecture they sketched down, gives the users the access, to host their data to their trusted hosts. A machine can host data of more than one user, but its accessibility is control through applications and by the users. As shown in figure 2-22, the system is a multi-agent system where agents are distributed on different machines. Each agent is a web application having its own database to store social data and accessible through URI. One machine can have more than one agent and can connect to each other in their respective social graph (Tandukar and Vassileva, 2012).

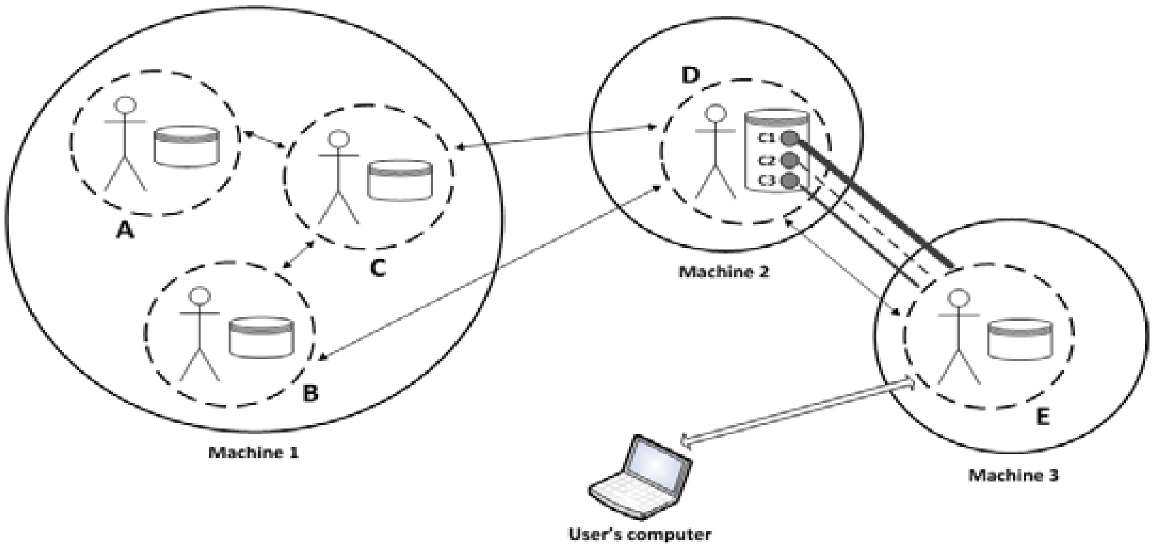


Figure 2-22: P2P based Decentralisation (Tandukar and Vassileva, 2012)

Berners Lee, (2009) and Bortoli et al. (2011) proposed decentralised social network architectures. At the centre of it are the same open standards such OpenID, WebID, distributed independent storage, but Berners Lee, (2009) separated the user access data and social data. There are many open web data initiatives, and some have been standardised shown in below table 2-7. Federated social network is one of the long awaiting outcome of these standards. A global social network based on decentralised architecture and open standards, where all centralised social networks can combine to give user data portability and data ownership, that is one global version.

Indeed, there are some commonalities in above-mentioned research, which leads to open standards such as OpenID, WebID, and distributed repository for semantic data that can be queried using SPARQL. Based on the comprehensive analysis, main decentralised architectural components are listed as follow.

Architectural Components	**Details**
WebID	Originally Known as FOAF+SSL is a single sign on system which binds the user to its URI in the web. OpenID anther single sign on system that can also be used.
URI	URIs are used as names for users, groups and documents on the web.
RDF Ontology	Such as FOAF can be used to allow URI of people to be looked up and return from the group of people.
Access Control Ontology	A simple ontology of terms that allows the access control
WebDav	Gives tools for creating, updating and rewriting data files.
SPARQL	The query language is used to making changes to the data repository.

Table 2-7: Architectural Components of decentralised social network platform (Berners Lee, 2009)

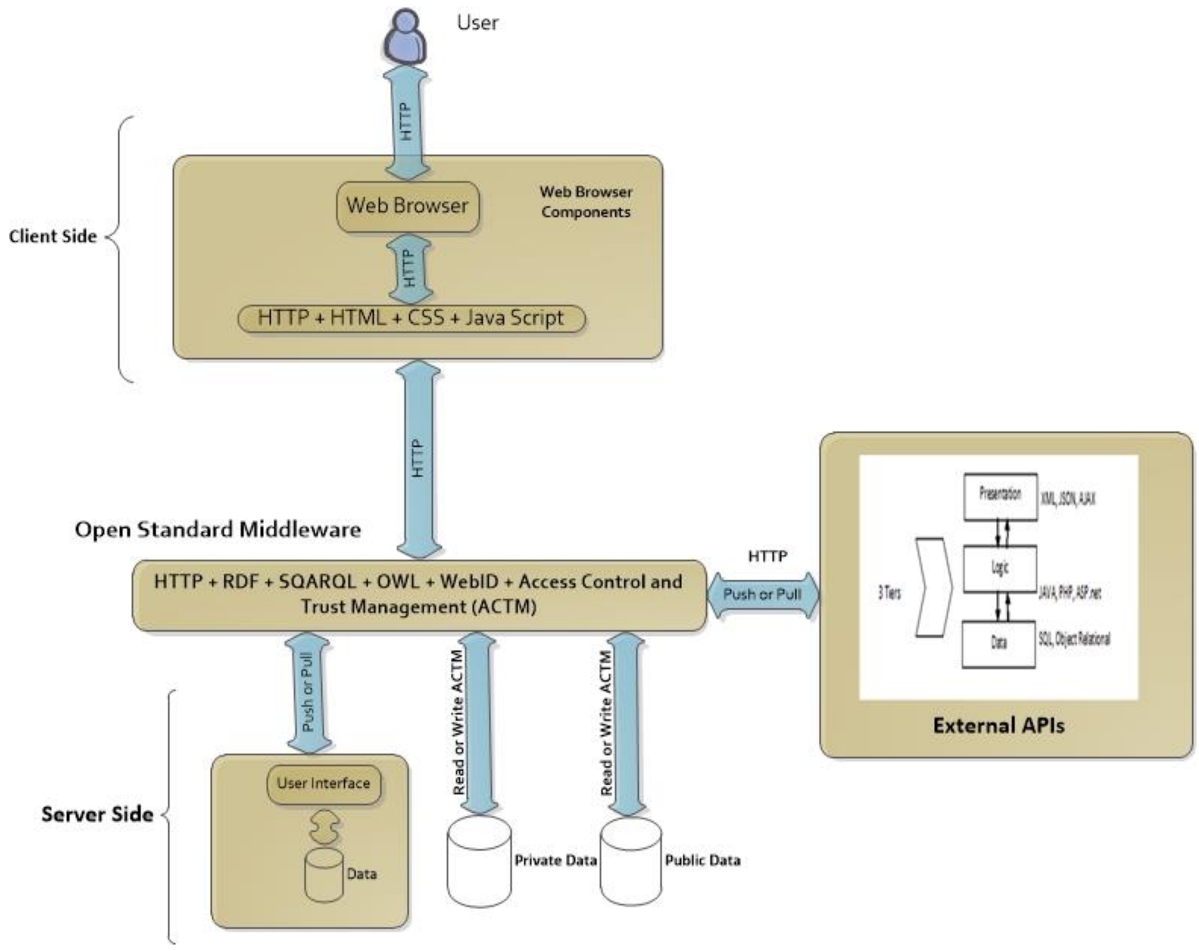


Figure 2-23: Basic Decentralised social network architecture based on existing research

The Decentralised architecture shown in figure 2-23 is a result of Berners Lee, (2008), Bortoli, et al. (2011) and Halpin and Tuffield, (2010), (See Table 2-6) work and above described standards and protocols. The general trend to develop a decentralised social platform is by adding a federation layer, which is a middleware (Open Standard Middleware) structure composed of open standards on top of the social network platform, with the aim to provide a structure of technologies to enable portability between social networks.

Instead of taking a global approach, using standards and protocol that are underdeveloped, current research attempts to undertake bottom-up approach that is from functional level to federation level. For example, enabling portability between the social network's functions, which can be extended based on user needs. The notion of using functional approach is based on results of the research taken in the field of social and computer sciences.

2.11. Drawbacks of Decentralisation in the Social Network

With all the advantages, decentralised architecture for social networking has some underappreciated drawback. Not all the drawbacks apply to the architecture in question, nor is any of the drawbacks may have a tendency to decisively affect the implementation of the architecture. But they may help explain why decentralisation of social networking faces the steep road ahead and why the implemented decentralisation may not provide the estimated benefits.
There are many types of computations that are hard to implement without a unified view of data. Fraud detection, spam, search collaborative filtering and analytics are some example. Network unreliability, lower data consistency and availability are some that can be mentioned here. Other than that, data duplication is another important challenge (Narayanan et al. 2012).

Decentralising the existing functionality of the SNP requires finding ways for distributed storage of data, update sharing, protocols for search and security, mechanism to find friends, openness for third party applications and meeting user demands for resource availability, are some of the known challenges (Datta et a. 2010). Meeting these challenges towards the decentralisation leads to certain drawbacks.

2.11.1. User Acceptance

The lack of user acceptance and adaptability of the decentralised social networks is the most known drawback. Centralised SNPs have larger established user base and more accessible infrastructure, which enhances their ability to attract more users and generate more revenue.

Convincing these traditional SNP users to migrate their data to decentralised platform can be difficult, because there is not a single well established decentralised platform. The study of the user behaviour to understand the usability of SNP functions, is done by using Social Network Analysis (SNA) techniques. SNA techniques are useful to identify the usability, performance and effectiveness of SNP functions but not the behavioural and psychological factors that led to the acceptance of SNP functions.

For decentralised or centralised social network site, attempts to understand user's behaviour to adopt these online technologies, have not yet achieved much success (Rad et al., 2014). According to, Pai and Arnott, (2013) and Vannoy and Palvia, (2010), the most current information system research on technology adoption has focused upon the technology adoption in organisations, mainly utilising Technology Acceptance Model (TAM) (Davis, 1989). It has been suggested that a new perspective on technology adoption is needed, to fully capture the nature of the technology acceptance in social networks, where the technology is embraced rather than just accepted by the user and where the actions made by technologies are seen as a behaviour, embedded in society (Vannoy and Palvia, 2010). Task Technology Fit(TTF) (Goodhue and Thompson, 1995), argues that individual will adopt a technology based on the fit between the technology characteristics and task requirements.

According to Rad et al., (2014), the research on technology adoption is the most mature stream of IS research but missing the social factor. Collaboration is the key component of the social networking sites. In the context of collaboration Brown et al. (2010), measured the technologies performance based on their progression. Rad et al. (2014) integrated UTAUT (Unified Theory of Acceptance and Use of Technology) model (Venkatesh et al. 2003) and TTF, to study the Social Research Networking Sites (SRNS). In another attempt, to overcome the missing social factor in technology Vannoy and Palvia, (2010) proposed Social Information Model (SIM). The SIM model posits to inform the current knowledge by the development of a social influence construct applicable to technology adoption where social influence results at the confluence of four phenomenon, social computing action, social computing consensus, social computing cooperation and social computing authority (Vannoy and Palvia, 2010).

As social networking become prevalent, new ways are needed to examine the human behaviour toward such technologies. The complexities of decentralised social networking may not fit into existing methods. The solid research in the area of decentralised social networking, in the context of user behaviour is not available. The research community have more focus on the importance of the networking structure and factors such as performance and storage, in which users have keen interest.

2.11.2. Performance

Another crucial factor that hinder the acceptance of decentralised social networking site is their performance. To determine decentralised social network performance, one has to assume that they need to become as useful as their counterpart centralised platform. Existing P2P approaches causes message transfer and profile update delays, because P2P replaces the database queries with node messaging. Cachet (Nilizadeh et al. 2012) introduced data caching strategy to improve the performance, by maintaining the encrypted channels to friends. This strategy can be successful in smaller network but in the larger network data caching itself cause maintenance issue. Comparable to the centralised SN, a single authority responsible for updates and data management, optimises the caching and reduce the cause that leads to performance related issues.

2.11.3. Usability

Since the selling point of the DSN (Decentralised Social Networks) is security and privacy therefore, knowledge about the cryptography may be necessary to use the DSN. For example, in certain networks user may need to know how to exchange public keys. In the above-mentioned approaches the user may need to install the client software which also may require administrator privileges, that user may need to install on the local machines. Using the decentralised social network should be simpler and any web connected device should be able to use it, without the obstacles of learning about key exchanges and software installations, to achieve better usability comparable to centralised platforms.

2.11.4. Functionality

The DSNP approaches are based on academic ideas rather being more practical, which can be used by users. This is one of the reason why existing DSNPs are backward in their functions. The most commonly use functionalities such as recommender systems, search functions and third-party applications, influence the users. In P2P based DSNP peers are connected to each other in the form of ring like structure. The absence of social graph, which index the users based on social links, reduces the ability of the sophisticated search mechanisms, because in the graph-based structure one can search user friends (neighbours) and extended friends. However, one can also argue that such search and recommendation capabilities affects the user privacy. Hence challenge the core concept of decentralised social networking, according to which such functionalities should be available in the privacy preserving manners.

2.11.5. Data Storage

One of the main architectural elements of decentralised architecture in a social network is decentralised autonomous storage mechanism, however, an architecture without a single point data storage may have many disadvantages. One of the major concerns of the user is about their content data and how it will be stored in the decentralised platform. Will it be stored exclusively at the node run by user or will it be encrypted and stored at random node. The selection of data storage type characterises how the DSNP will be designed. Since the data may be stored at many places, based on node location. This may increase risk of data duplication, data unreliability and data unavailability

2.12. Literature Review Finding and Research Direction

In this chapter, the literature is reviewed on the social networking platforms (SNPs), the application of semantic technologies and software architecture principles, to enable data portability through the mean of decentralisation. The main emphasis was on the technologies and software architectural designs principles that can be helpful in implementing such architectural structure on top of SNPs to enable decentralisation. Not many attempts are made such as Bortoli et al. (2011), in which they emphasised on the description of social network user functions rather than creating and following federated social network (a network of the networks).

The assumption of using different frameworks and protocol together to produce harmonisation of standards in order to enable wide variety of improvements across the social networking platforms and applications is only possible with the combined effort of interoperable architectures, instead of single monolithic architectures.

The purpose of the above mentioned (section 2-10) frameworks (such as semantic web, hybrid distributed or P2P) in most of the social network decentralisation research seems to make web resources machine understandable, shareable and reusable among different applications. This phenomenon already used by many websites that interoperate user generated content and semantic annotations. The use of semantic technologies to add extra semantics to the user generated content has provided ways to represents reuse and share information across the web platforms (Cena et al. 2012). On the other hand, SNS providers have independent control of the data, creating the value of this data coming from the different application is one of the main roles of decentralisation.

Existing SNP research, in the domains of decentralisation and data portability, is mainly done in user privacy, Profile data portability, activity and identity-related issues. There are three main approaches widely used in research to decentralise the social web, distributed web server hosting, federated layer and p2p approaches. The majority opinion goes with the federation of social networking platform, which is still underdeveloped and has opposition in SNS providers.

The general trend among the SNS providers to achieve complete or partial data portability between SNPs, is by using the semantic web technologies however there is no standard mechanism available. The standards and protocols mentioned in tables 6 and 7 are used to solve the problem, but there are not easy to use. Another popular opinion described in Berners Lee, (2009) is user-centric social data management that is providing personal information space (section 1.2) where the user can manage their information and data based on their own needs, service providers can only provide the interface.

The appraoches mentioend in the above literature are specific towards solving one part of the problem related to decentralisaiton. Indeed, the standards and protocols mentioned in the above sections 2.8.1 have the potential to produce a viable solution but building a decentralised platform from scratch is a difficult and cumbersome task that is why widely unwelcomed among the Service Providers. Therefore, SNS providers are keen in adopting technologies like SN integrators and social connect services. The complexity of design requirement, the difficulty of distributed data storage management, cost of network availability, are some of the known facts behind the failure of existing DSNP approaches (P2P, hybrid and federated).

The focus of the existing research on the social network decentralisation is on developing tool and technologies to use distributed networking approaches to build tools that may allow portability of data. The alternative would be to work towards to set standard design and development principles using software engineering standards and to come up with the technical architectural framework for design and development of the DSNP. The framework may illustrate the need of semantic technologies and other standards based on the design requirements.

This research attempt to solve the problem of data portability between social networks at functional level by using decentralisation approach. The methodology which is used to build decentralised architecture, uses similar standards and protocols as used by existing architectures, however, differentiate on the principles, whether decentralisation should be done at the central level such as the Federated Social Web, widely explored or at the functional level, which is unexplored.

The functional level approach gives user control on their social network functionality. Using proposed architecture users will be able to decide which functionality they would like to use across their social network platform that means if the user decided to use the message related functions then they will be able to send scrap or post on another platform they are registered to.

The functional level approach is based on CBSD and AOSD. Each function of the SNP is designed as a component, under the guidelines of CBSD. According to which, a component is equal to a functionality and each functionality has a certain behaviour. The behaviour of the functionality is controlled by an aspect. In the context of current research, an aspect contains the attributes that are required to decentralise the functionality. Based on this opinion, if social network functions are described semantically following the decentralisation protocols and

Key Publications	Concepts	Analysis
Suryanarayana and Taylor, (2004) Suryanarayana et al. (2004) Suryanarayana et al. (2005) Suryanarayana et al. (2006)	Trust management in peers of the P2P decentralised application	An event based architectural style is presented to show how various kind of technologies can work together in different decentralised platforms towards the management of trust among the connected peers. The concept of separate architectural style for decentralised application is adopted and mapped on DSNP with the help of component and aspect-based design analogies.
Fuentes et al. (2003)	Model Driven Application (MDA), AOSD CBSD	They presented MDA based joint model of CBSD and AOSD and lay the foundation of CAM. CAM is a key element of the proposed decentralised architecture. CAM is mainly used for the design of distributed applications. The use of CAM to develop decentralised software application is never been done before.
Pinto et al (2003) Pinto et al. (2005) Pinto et al (2011)	CBSD, AOSD, CAM Aspect Oriented architectural description language (AOADL)	Further to their previous research they solidify the concept of CAM and how the combined form of AOSD and CBSD can be used to design and develop complex distributed application. The CAM model is crucial to the proposed decentralised architectural style. The key attributes of the proposed architectural style are component and aspect.
Pessemier et al. (2008)	Component and aspect integration. Component, aspect binding, that is component to component and component to aspect.	They described the general model for components and aspects integration. They also argue that in certain conditions aspect can be used as a component. The similar concept is adopted to describe the behaviour of the functionality of the DSNP.

Table 2-8: List of key publications

standards then this idea can negate the need of building a completely new decentralised social network. With this innovation, existing social network will be able to interoperate at the functional level.

Since key concepts, tools and technologies have been described. The question that should be answered in the further research is, can the integration of CBSD and AOSD help achieving the data portability by the mean of decentralisation. As mentioned in section 2-3, the integration of CBSD and AOSD has improved the performance of complex distributed systems. The functional independence, reusability, and adaptability of components provided by integrating the CBSD and AOSD can be ground-breaking in decentralising the social network at the functional level and should be investigated further.

2.13. Chapter Conclusion

In the context of current research, the study of software architecture for intended social web architecture has evolved around the observation of the software design principles and that designer or software architect follow when they take actions while working on the application. The web architectures and their related concepts as mentioned above are useful in certain specific environments and they all can do a good job to some extent but they all have limitations.

The web application produced by the early Web architectures were stateless, static, and asymmetric in nature. The work of Berners-Lee et al. (1992) and Fielding, (2000) is considered as fundamental, there were some issues, which later on solved by JAVA-Scrip, AJAX and Web Service. Perry and wolf, (1992), Garlan and Shaw, (1994), Kruchen, (1995), Conallen, (1999), Celment et al. (2003) and Bass et al. (2011), works provides the fundamental software engineering concepts and are used by researcher in most of the Web architecture related studies. UML approaches used by Kruchen, (1995), Conallen (1999), Ceri et al. (2000) and Booch, (2001) to developed web application provide an alternative for web development.

The decentralisation of the social web to achieve data portability is complex. The research community attempted to develop various tools technologies using P2P, hybrid and federated style of architectures to achieve decentralisation in social web. But so far there are no successful implementations. This research proposed CBSD and AOSD based architectural guidelines in the form of technical architecture to set the standards for the design and development of the decentralised social networking applications. The section 2.6 differentiate between the distributed and decentralise architectures and argue on why P2P or hybrid architecture-based applications are not decentralised. Building on the key concepts related to decentralised network architectures and define the decentralised social network architecture.

In the section 2.7, the drawbacks of existing social networking platform as discussed and why do people need decentralised social networks. The breach of trust, business model and centralisation are the highlights of the existing SNP drawbacks.

The critical analysis of the existing decentralised social networking tool, technologies, protocols and standards, gives the insight knowledge necessary to understand the user needs with respect to decentralised social networking platforms. The analysis of the existing DSNP shows, user acceptance, performance, usability, functionality and data storage are the main issues hindering the adoption of the DSNPs. Lastly, the literature review provides the grounding for the main artefact by describing the fundamental needs of the decentralised social web architecture. The basic form of decentralised social web architecture based on existing research and shows best combination of tools and technologies proposed in well cited academic research. In doing so this chapter addressed the objective 1 and 2 of the research and also provided the knowledge needed to address rest of the research objectives.

Chapter 3 - Research Methodology

3. Introduction

This chapter outlines the research activities seeking to improve the theoretical and methodological approaches available to study decentralisation in social networks. A series of approaches are investigated to develop an approach to analyse the different aspects of social networks to help enable decentralisation in social networking platform. Therefore, to achieve this goal research methodology is presented to shape the research process and guide the investigation. The suitable research method that is adopted by this research is design science research (March and Smith, 1995; Hevner et al. 2004; Peffers et al. 2008). DSR is an iterative activity where solution artefacts are designed and developed through various cycles, processes, activities, inputs and outputs. The goal of a DSR is to generate a purposeful artefact that addresses a practical problem, especially, when elements of the problem are not completely understood (Hevner et al. 2004).

3.1. The Need of Research Methodology

The architecture proposed in this research follows both existing and previous work on decentralised environments and theories. The proposed artefact is guided by a research methodology for the delivery of desired goals. This section describes the need for suitable research methodology and its application to the proposed research. Peffers et al. (2008), describes research methodology as a system of principle, practice and procedures applied to a specific branch of knowledge. An effective research methodology can enable the researcher to conduct research successfully by fulfilling requirements of a particular task and activities.

This thesis addresses an area that has been identified as significant but lacks in researched based implementations. The addressing of interoperability challenges in social networks is one aspect, the other challenge this research face is related to the need of right research methodology to implement this research finding. The proposed framework uses software engineering and information sciences principles as its foundation. The required methodology can be based on multi-pragmatic and mixed method approach. For example, Gacenga et al. (2012) proposed a research approach which is based on behavioural science and design science paradigm.

In the development of the software for any system, tasks and activities are performed under the guidance of software engineering methodologies. Software development methodologies such as Agile is widely used for quick systematic deliverance of software product.

The main purpose of the research methodology or software development methodology is to provide systematic, plan-driven guidelines, that are to be processed through valid information and rational decision making. In the perspective of this research, the most important aspect of both paradigms will be the support for software architecture through-out the development cycle.

In general, to support the cycle of software engineering, the artefacts are designed and developed using various software development processes and methodologies such as RUP (Rational Unified Process) and Agile methodology. The design and development process of the software can also be based on mixed approach as few authors (Gacenga et al. 2012 and Conboy et al. 2015) have proposed in their research. ADSRM (Conboy et al. 2015), is a similar attempt in which DSR best practices and Agile methods more pragmatic approaches are aligned to compose new components by amending the existing DSR best practices.

The Information Systems research is mainly influenced by referring to prior published ideas. For example, the literature relevant to the Design Science Research is discussed reflecting its evolution with the key methodological guidance referring to Hevner et al. (2004) work, which draws extensively on March and Smith, (1995) and similarly Gregor and Jones, (2007) work which is based on Walls, Widmyer and El Sawy, (1992). Similar approach is taken to compare DSR methods with agile methods in the next sections, taking the analysis further to selection of research methodology.

3.2. The selection of Research methodology

The systematic study of design, the development and evaluation processes with the aim to establish a sound solution for the research problem based on theory, requires a methodology. The methodology adopted in this research to solve the portability problem between the SNPs, comes from the comprehensive comparison between the explanatory research, software engineering design paradigms (such as Agile) and design science research paradigms (such as DSR).

The paradigms are composed of, assumptions about knowledge, how to acquire it and about the physical and social worlds (Hirschheim and Klein, (1989) and Gregg et al. (2001). There are three questions that need to be addressed in order to define the paradigm. What is the nature of the reality (ontology)? What is the nature of the knowledge (epistemology)? What approach is the best approach to understand and obtain the desired knowledge (methodology)? Positivist or Post-positivist and the Interpretive or Constructivist are two main paradigms of interests for IS researchers (Gregg et al. 2001).

There is an evidence based on previous research, such as (Mertens, (1998) and Schwandt, (1994) and recent research such as (Gregg et al. (2001) and Gacenga et al. (2012), Positivist or Post-positivist and the Interpretive or Constructivist paradigms provides the good basis for the majority of the IS research, but they do not fully address the requirements software engineering-based research projects.

The software processes and methodologies used for the application of technological success to the IS systems is mainly based on the understanding of the organisational units. The Positivist or Post-positivist and the Interpretive or Constructivist paradigms are used in IS research to understand impact of technological success, but not the creation of unique product associated with the development of the software system, which is the case in this research, a design and development approach is created and implemented.

In another example, building a hypothesis using explanatory research, to find the explanation behind the decline of the people interest in a specific type social network, is possible. Also, the hypothesis results can be helpful to obtain certain process to improve the social network ratings, but these features can not be used to obtain the components required for the technical framework design of the social network.

To alleviate limitations of the fundamental IS research paradigms and to describe the software engineering practices in IS research Nunamaker et al. (1991) presented multi-methodological approach, March and Smith, (1995) presented design science and Gregg et al, (2001), presented Socio-technologist / Develop-mentalist approach. The main purpose is to build the connection between the software contributions and scientific knowledge building and provide the suitable method to describe the whole process.

3.2.1. Design Science Research (DSR) and Agile Methodologies

At this point, it is important to distinguish between the design and development research and software product development. One could develop and launch a successful software application or product but does not meet criteria for design and development research. In general, research involves, addressing an acknowledge problem based on existing literature and making an original contribution to the body of knowledge (Ellis and Levy, 2008).

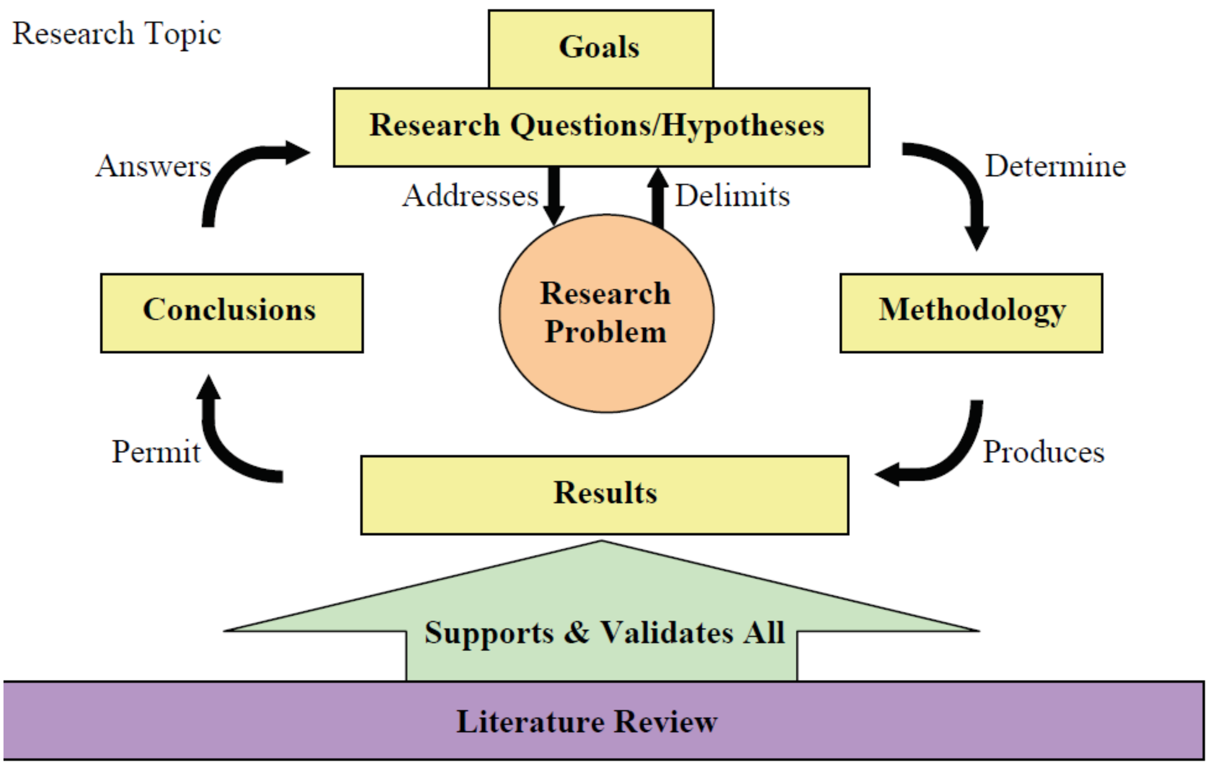


Figure 3-1: Basic Conceptual Map of the problem based research (Ellis and Levy, 2008).

The conceptual map shown in the figure 3-1 is a testbed on which theoretical foundation of the research can be built on. In the figure 3-1, there is a two-way relationship between the research problem, the goals and the research questions. A research goal is the main intended objective to solve the problem. To find the answers to those research questions, the cycle continues between activities known as determine, produce, permit and answer. By attaining the answer to the research questions, that means the research problem is solved and contribution is made (Creswell, 2005). Furthermore, in solving the research problem the methodology directly impacts the driving of the research. Since methodology is a step to find the answers to the research questions, therefore the grounding for the needs and requirements for the methodology need to be known before selection.

3.2.2. Requirements for the Methodology Selection

Software designing is a theoretical and empirical study of software creation and modification including its methodology. In DSR, the design is a research method, which depends on its key elements, theory and design process (Hevner et al. 2004). Still, there is an ongoing work on design process in term of understanding and implementation. In the cu rrent research, the aimed methodology should support problem-solving using architectural design.

The main requirements that should be fulfilled by the research methodology in current research perspective are,

- The methodology should support and focus on the problem-driven approach,
- Theorising of the problem so the solution can be extended in case there are many solutions,
- Product-centric, in case there is an end-product of the research, practicality support is required so that solution can be more practical based on the theory presented and lastly,
- The methodology should support the identification of clear and original contribution to the knowledge.

3.2.3. DSR As Problem Driven Approach

Simon established the foundation of the Design Science by emphasising on the uniqueness of the sciences of the artificial. The science of the artificial focuses on the artefact that serves a human purpose (Simon, 1996). The key motivation behind DSR is the desire to build and improve the new environment by introducing new and innovative artefacts and processes for building those artefacts. Good design science research often starts by identifying opportunities and problems in actual application environment (Hevner, 2007). DSR also compliments from behavioural science research and natural science (Hevner et al. 2004).

The expected outcomes of design research are discussed in detail in March and Smith, (1995), Gregor and Jones, (2007)) and Gregory and Muntermann, (2014) with their different perspectives. In Information Technology, there are two kinds of scientific research interests, descriptive and prescriptive research (March and Smith, 1995) that can be used to explain DSR outputs (Gregory and Muntermann, 2014).

3.2.3.1. Descriptive vs Prescriptive

In information systems, DSR is described in two perspectives for the understanding of technological and social environments (design science and behavioural science) and their relationship within the IS discipline. The behavioural science perspective is concerned with the theory development, justification and evaluation. It primarily uses the natural science research, considering IT artefacts as extant objects to be studied. The Design Science perspective is more concerned with building and evaluating the artefact that addresses important human and organisation problems (March and Smith, 1995; Hevner et al. 2004).

The descriptive research aims at understanding the nature of IT systems and prescriptive research aims at improving them. This division of interests has caused confusion among the researcher over what constitutes legitimate scientific research method. However, regardless of the dichotomy of interests, both descriptive and prescriptive research relates back to natural and design science. According to March and Smith, (1995) natural science is descriptive and explanatory in intent, whereas Design Science offers prescriptions and creates artefacts that represent those prescriptions, hence more relevant to existing research problem.

In a multi-disciplined paradigm, such as problem-based research and specially in the software system development cycle, the primary purpose is to add the body of knowledge about the creation and evaluation of software design. Also, document the activities during development and implementation to enhance the understanding of the issues related to the research problem. In the DSR framework proposal of Hevner et al. (2004), DSR is an iterative activity where problem’s solution is designed and developed through various cycles, processes, activities, inputs and outputs. The goal of a DSR is to generate a purposeful artefact that addresses a practical problem, especially, when elements of the problem are not completely understood (Hevner et al. 2004).

In Hevner et al. (2007) and Hevner et al. (2004), they focused on three design cycles for the development of IS research outputs (artefacts and theories). The relevance cycle (bridges the contextual environment of research project with the design science activities), the rigor cycle (connects the design science activities with the knowledge base of scientific foundation, experience and expertise that informs the research project) and the design cycle (iterates between the core activities of building and evaluating the design artefacts and processes of the research).

The selection of the research approach is often dependent on the domain the research is conducted in (Gregor and Jones, 2007). In the current research, the domain is software architecture engineering. The next section discusses software engineering methodology (SEM) or software development methodology (SDM) and to what extent SEM could be or cloud not be adopted as a research methodology to solve problem-driven research.

3.2.4. Software Engineering Research

The software development methodology (SDM) has been omitted from most of the classifications of the research methods. Mainly due to the assumption that system development does not lie within the research domain (Burstein and Gregor,1999). The legitimacy of the system development methodology (SDM) as valid research activity was first debated by Nunamkaer et al. (1991). They compared IS research methods such as design science and system development methods and proposed multi-methodological research framework to guide IS research activities. The approach consists of four strategies, that are observation, theory building, system development and experimentation as shown in figure 3-2.

The software development approach as a research method can be used to bridge the gap between the technical and social side of the IS research (Burstein and Gregor,1999). There is numerous recent research attempts to extend the framework of IS research and software development components integrated, to form a research cycle, that can present complete, comprehensive and dynamic research process. This will allow multiple perspectives and methods to be considered in various stages of the research process (Bai et al. 2013).

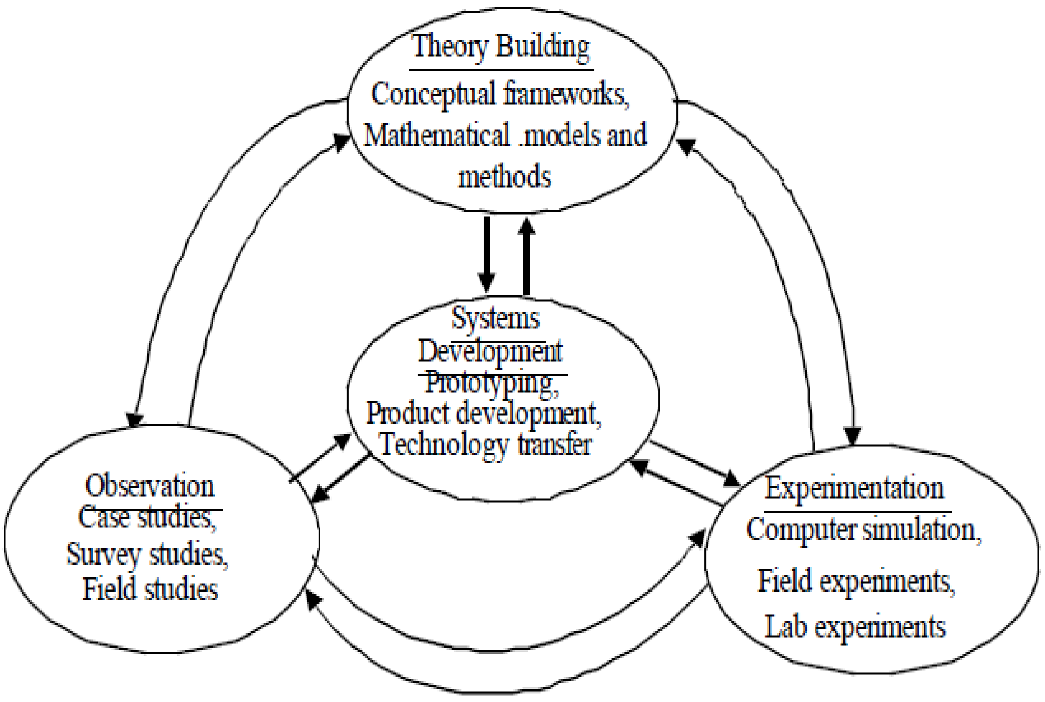

Theory Building
Conceptual frameworks, Mathematical .models and methods
Systems Development
Prototyping, Product development, Technology transfer
Observation
Case studies, Survey studies, Field studies
Experimentation
Computer simulation, Field experiments, Lab experiments

Figure 3-2: A Multi-methodological approach to IS Research taken from Nunamaker et al.(1991)

Morrison and George, (1995), described the objectives of software engineering research, are to investigate all the aspects of the software development process including, software formulation, description, implementation and evaluation (Morrison and George,1995).

The software development methodologies (Gregg et al. 2001), such as Agile which is regarded as highly effective software development methodology in many studies, (Cao et al. 2009) (Vidgen et al. 2012) is used when the rapid transformation of system design to the prototype is required. It starts with implementing confirmed and well-understood requirements and continuously refines and add more functionality to the developed system based on user feedback (Bai et al. 2013).

In general, the agile software development is characterised in, incremental (refers to small software releases with rapid development), cooperative (refers to close customer and developer interaction), adaptive (refers to the ability to make and react to the last moment changes) and straightforward (refers to easy to learn and easy to document development process) (Abrahamsson et al. 2003). For further detailed discussion on the characterisations of agile methods, readers are referred to (Cao et al. 2009, Vidgen et al. 2012 and Fowler and Highsmith, 2001).

3.2.5. Agile and ADSRM

Since social networks are continuously changing and building new ways to improve their services. To achieve this, they stay in a continuous development process. Maintaining the changing need of the DSNP rapid software development approaches such as Agile, can be useful but the most incumbent part of this process is the knowledge attained and how this knowledge can be useful in solving the pursued research problem.

Therefore, the main question that arises is from the above discussion is, whether software engineering methods can be considered as a research process? The answer is quite vague as far as the academic literature is concern. However, the integration of software engineering process of software development to problem-driven research can enhance the understanding of the software operational environment. Conoby et al. (2015), Vidgen et al. (2012) and Jalali and Wohlin, (2012), proposed research methodologies that are composed of multiple methods such as the integrated approach of Agile and DSR.

Methodologies	When to use and Limitations	Type of research questions and examples
Exploratory Study	Should be used when there is high level of uncertainty, problem is not understood and very little existing research on the subject. Can cause indecisiveness when concluding the research.	What is the case or key success factors? For example, what are the key critical success factor of the decentralised social networks
Explanatory study	Identify the links between the factor and variables relate to the problem. Used for case control study. Limited in the ability to provide deep contextual data	Based on the explanatory nature of the research question. Explains why a phenomenon is happening. For example, why the crime rate high, or examining various kind of social trends.
Design Research	Solutions oriented, problem driven, can be used when new knowledge based on artefact is formed. Prototype may not be similar to real world. Mechanism complexity may exceed to the level where it become difficult to manage.	Based on the descriptive nature research question. For example, an application is required for to find social trend based on already available data.
Software Engineering Research and (ADSRM)	Solution oriented, problem driven, can be used to investigate the software development, including formulation, description, implementation and evaluation. SDM limitation such as lack in identification of knowledge contribution.	Based on descriptive in nature research question. For example, social network is getting used for message sharing and next version is needed to add multimedia sharing functionality. It can used for the development of the next version

Table 3-1: Short comparison of the key methodologies

Agile methods consist of the set of practices for software development aiming to overcome the limitations of plan-based methods by changing system requirements, with the focus on intensive collaboration between the customer and developers. As Agile methods rely heavily on frequent communications and requirement gathering, therefore there are challenges associated with this combination to make it work effectively (Boehm and Turner, 2005). The main challenges can be related to communication, personnel, trust and knowledge management (Jalali and Wohlin, 2012).

Conboy et al. (2015), extended DSR using the best practices of Agile. The aim is to enhance the ability of DSR to balance the procedural rigour with the need to consider empirically driven problem or solution and improve the handling of unanticipated problems. The model of extended DSR is referred as ADSRM as shown in the figure 3-3.

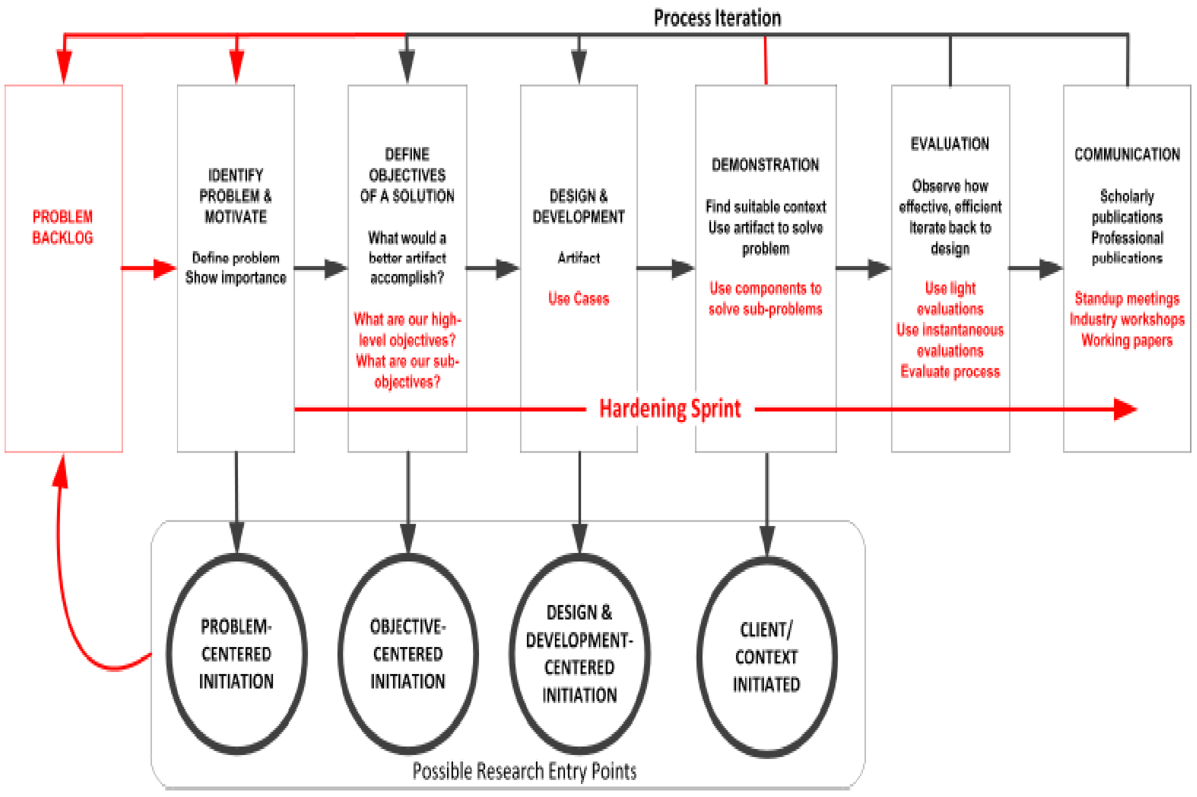

Process Iteration
PROBLEM BACKLOG
IDENTIFY PROBLEM & MOTIVATE
Define problem
Show importance
DEFINE OBJECTIVES OF A SOLUTION
What would a better artifact accomplish?
What are our high-level objectives?
What are our sub-objectives?
DESIGN & DEVELOPMENT
Artifact
Use Cases
DEMONSTRATION
Find suitable context
Use artifact to solve problem
Use components to solve sub-problems
EVALUATION
Observe how effective, efficient
Iterate back to design
Use light evaluations
Use instantaneous evaluations
Evaluate process
COMMUNICATION
Scholarly publications
Professional publications
Standup meetings
Industry workshops
Working papers
Hardening Sprint
PROBLEM-CENTERED INITIATION
OBJECTIVE-CENTERED INITIATION
DESIGN & DEVELOPMENT-CENTERED INITIATION
CLIENT/ CONTEXT INITIATED
Possible Research Entry Points

Figure 3-3: Agile DSRM research model

ADSRM is a fundamental epistemological shift for DSR which encourages the use of the Agile approach to problem identification. The integration of Agile elements to DSR allows for greater rigour and knowledge accumulation, in how it is conducted analysed and reported. Moreover, it can create a more detailed understanding of the design for researchers. The main contribution of ADSRM is the introduction of two additional components to the DSRM model. Problem backlog and hardening spring.

In practice, customers are not aware of the capabilities of the until the first version of the product is released and the related concepts and issues made tangible. To better capture this scenario the ADSRM introduces problem backlog. This component represents the broader problem space from which individual problem can be identified and motivated. The feedback from the later stages of development can be added to problem backlog, representing the ability of that stage to provide insights into the problem.

The concept of “hardening sprint” (Conboy et el. 2015) is applied as an additional design component. This component consists of three main mechanisms, (1), Freeze the problem (2), Freeze the process and (3), Add to the process. They are used to enhance the change with agility into the rapidly changing design environment, a level of rigour is added where improvisation is allowed (Conboy et el. 2015), under the principle stated in the “Agile Manifesto” (Fowler and Highsmith, 2001).

3.2.5.1. Why DSR?

Based on the requirements mentioned in the section 3.2.2, and overall above discussions, they are some commonalities and differences between DSR and software development methods. DSR practice is entirely different than SDM but there is relevance on how the objectives are achieved and communicated throughout the design process. Like identifying a problem that is useful and sustainable in an organisation and communicated with the relevant stakeholders during the cycle of development.

Hevner et al. (2004), addresses the difference between the routine system design and DSR by defining the design as an application of knowledge to solve previously unsolved problems. DSR main position is more design oriented driven by the use of existing theory to solve the problems and validate them on the basis of experiences. In contrast, the software or system development methodologies such as agile, literature do not explicitly demand that system design should be based on a theory. In the academic literature, multi-methodological approaches are proposed to fix these lacks. Already discussed in the above sections.

A good example of some of the lacks can be, Peffers et al. (2007) have described four initiation contexts for a DSR project namely (1) problem centred initiation, (2) objective centred solution, (3) design and development centred initiation and (4) client context initiation, are considered as important. Yet, these contexts could be based on observations or non-DSR based approach, analysing and explaining important considerations that would help the designer (Gleasurea, 2015).

The DSR literature contains many unsolved questions, for example differentiating between definitions of practical problem and knowledge problem. Which is more suitable to define the whether the problem is DSR problem or not. In general, DSR is considered and mentioned as a problem-driven framework to solve an ununderstood problem. Important tasks in the problem-driven investigation are describing the problematic phenomena, formulating and testing the hypothesis about their causes and priorities for problems to be solved (Wieringa, 2009).

On the contrary side, the researcher may adopt other investigative approaches like goal-driven, solution-driven and impact-driven approaches or the mix-theory approach to support the foundation and desired results. For example, Adipat et al. (2011) study adopts the DSR approach to address the prescriptive problem and incorporate both cognitive fit and information foraging theory to understand the user searching and browsing behaviour on the mobile web applications. The prescription element of their research helped them to understand and explain the need for cognitive fit theory support for mobile web applications.

The proposed research has substantial technology-based artefact content. The multi-mythological approaches are suitable in case the proposed artefacts are getting used to improve an existing structure or platform which is not the case. The approach proposed in Conboy et al. (2015) (ADSRM) is likely to be more suitable for complex projects. Their approach is still underdeveloped and is more appropriate to be used to solve the problems that are well established. For example, next version of the already developed application. To support this, they added problem backlog component explained in the above (section 3.2.5) discussion.

The expansion of DSR using agile practices can be seen as an alternative to solve the problem where the problem space and solution space both are evolving. In contrast to that, the standard DSR focuses on the problem and evolution of the problem. In the current research perspective and to keep the concentration on the process of design and development, this research adopts the standard DSR approach proposed by Hevner et al. (2004) and Kuechler and Vaishnavi (2012), as they satisfy the very basic requirements illustrated in section 3.2.2 to formalise the research methodology.

The important characteristics that distinguish the DSR from SDM is the clear identification of contribution to the archival knowledge base of foundation and methodologies, (Hevner et al. 2004), systematic documentation of a discussion of design choices made, option considered and alternative (Ellis and Levy, 2008) and use of rigorous, accepted research methods (Hevner et al. 2004).

3.3. DSR Process

Drawing on the above discussion, the key element that separates the design science research from routine system development is the creation of the design. Design means “the art or action of conceiving of and producing a plan of something before it made” (Oxford Dictionary, 2017). Thus, design deals in planning and creating a new artefact. If the knowledge required to create an artefact that already exists then that design is routine design, else it is innovative. The innovative design called for new knowledge to fill the gaps in the current knowledge. Problems in routine design can lead to DSR. Thus, DSR is used to find out the missing knowledge in the new area of design.

The focus of this section is to describe the selected research methodology. The methodology is described in the hierarchy of model, guidelines, and process.

3.3.1. The Model

The final objective of a DSR process is to provide a mental model for the characteristic of research outputs. A mental model is a small-scale model of reality, constructed from perception and imagination to form a logical understanding of the structure to form formal rules and theories (Peffers et al. 2008).

Similarly, DSR process model should provide some guidance about what to expect from DS research. The model shown in figure 3-4 is a design process model taken from Kuechler and Vaishnavi (2012), which shows different phases of design research process and activities carried out within various phases.

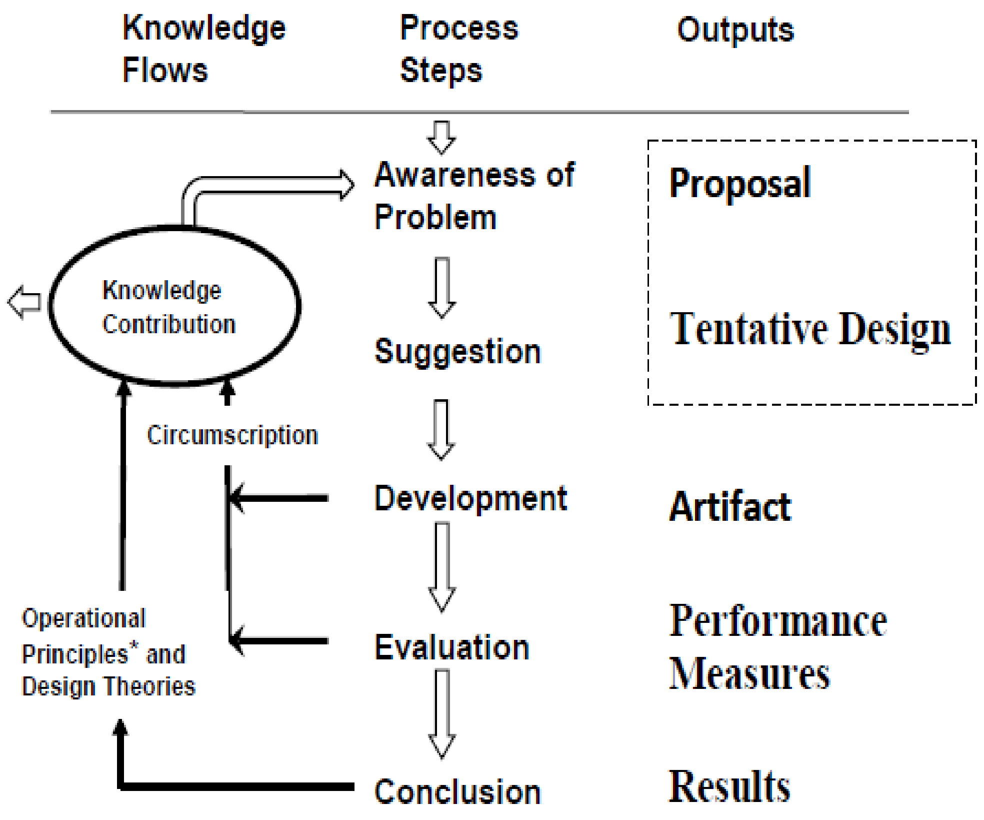


Figure 3-4: Design Science Research Process Model taken from Kuechler and Vaishnavi (2012).

With reference to above figure, typical design science research proceeds as follows,

Awareness of problem: This is the first phase of DSR process for the investigation of any research problem. According to Kuechler and Vaishnavi, (2012), the awareness of an interesting problem may come from studying multiple sources including new developments in the industry. Investigating related discipline may also provide the opportunity for the application of new findings. The output of this phase is a proposal for new research.

Suggestion: The second phase follows right after the proposal and is connected with tentative design. A tentative design which is the output of the suggestion phase is a prototype based on the initial design. Basically, this phase is a creative process in which new functionalities are envisioned based on a novel configuration of new or existing elements. This phase is not very well understood in the design science research as human creativity is poorly understood in cognitive science (Kuechler and Vaishnavi, 2012). However, the main purpose of this phase is to gain insight knowledge into the problem domain to form initial design and increase human curiosity to solve the problem.

Development: This phase is about the development and implementation of the tentative design. The process for implementation is different depending on the artefact to be created.

Evaluation: The evaluation of the artefact is done in this phase based on the criteria made explicit in the proposal. In this phase, all glitches hindering the expectation, both quantitative and qualitative must be carefully noted and explained (Kuechler and Vaishnavi, 2012). Hypotheses are made to explain the behaviour or the artefact. If the results are unsatisfactory then design process goes back to the initial phase. Otherwise, the cycle moves to conclusion phase.

Conclusion: This phase could be the end of the research efforts. The final effort is the satisfactory behaviour of the artefact from the evaluation. If the artefact behaviour deviates from the desired results then revised hypothesis results are judged as "good enough" (Simon, 1996). During the hypothesis revision cycle, the knowledge gained is processed by the designer and guidelines are built for the practitioners as part of the "communication" (Hevner et al. 2004), explaining how to use the artefact. The leftward arrow (figure 25) coming out conclusion indicates the knowledge contribution (Kuechler and Vaishnavi, 2012).

Other Models:

There are a number of excellent process models, guidelines and descriptions of design science research process such as Peffers et al. (2008), Hevner et al. (2004), Purao, (2002), March and Smith, (1995) and Nunamaker et al. (1991). The above-described model of Kuechler and Vaishnavi, (2012) is similar to these models but its focus is more on the generation of the knowledge.

3.3.2. Research Guidelines

In order to assist researchers Hevner et al. (2004) prepared seven guidelines to help understand the need for effective Design Science research and implemented them using their information system research framework. The fundamental principle from which these seven guidelines are derived is knowledge and understanding about the problem are acquired in the building and application of the artefact. Hevner et al. (2004) guidelines are described in Table 3-2.

Guidelines	Descriptions
Design as an Artefact	Design science research must produce a viable artefact in the form of a construct, a model, a method, or an instantiation
Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems
Design Evaluation	The utility, quality, and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods
Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artefact, design foundations, and/or design methodologies.
Research Rigor	Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact
Design as a Search Process	The search for an effective artefact requires utilising available means to reach desired ends while satisfying laws in the problem environment
Communication of Research	Design science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 3-2: 7 Design Science Research Guidelines (Hevner et al. 2004)

March and Smith, (1995) and Hevner et al. (2004), guidelines for DSR influenced Peffers et al. (2008) Design Science Research Methodology (DSRM) process model to enable researchers to conduct their research by following the commonly understood framework.

3.3.3. Research Process explained

The DSRM presented by Peffers et al. (2008) incorporates principles, practices and procedures required to carry out design research to meet objectives. As shown in figure 3-5, the DS process consists of six steps following each other.

The process is structured in sequential order but there is no exception that researchers would always process in sequential order starting from any step until the demonstration step, depending on purpose and objectives of the research (Peffers et al. 2008).

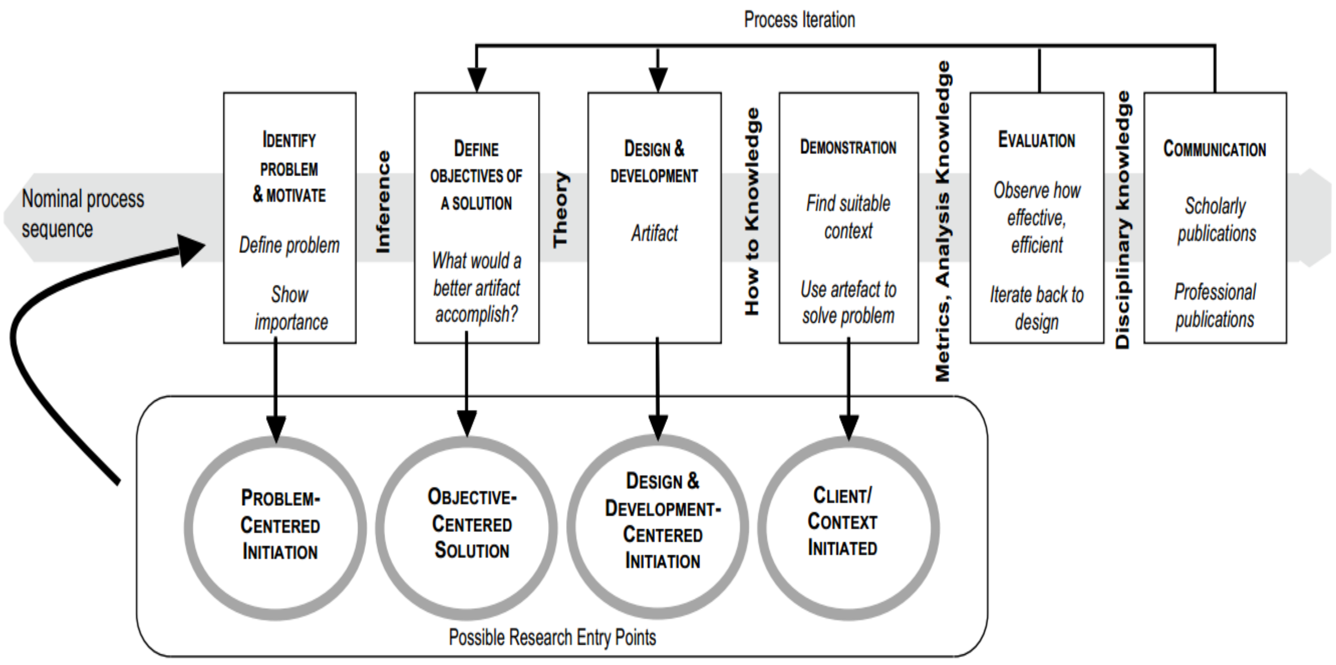

Process Iteration
Nominal process sequence
IDENTIFY PROBLEM & MOTIVATE
Define problem
Show importance
Inference
DEFINE OBJECTIVES OF A SOLUTION
What would a better artifact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to Knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, Analysis Knowledge
EVALUATION
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
COMMUNICATION
Scholarly publications
Professional publications
PROBLEM-CENTERED INITIATION
OBJECTIVE-CENTERED SOLUTION
DESIGN & DEVELOPMENT-CENTERED INITIATION
CLIENT/ CONTEXT INITIATED
Possible Research Entry Points

Figure 3-5 Design Science Research Methodology Process Model (Peffers et al. 2008)

This model as compared to the model shown in figure 3-4, breaks the awareness of the problem phase into two phases, identify the problem and motivate and define objective of a solution, merges the suggestion and development phase into a single phase, design and development breaks the evaluation into two phases demonstration and evaluation and conclusion is renamed as communication (Kuechler and Vaishnavi, 2012).
The outcome of the DS research depends on how theory and design are tested, and process model should provide some guidance about what to expect from DS research outputs (Peffers et al. 2008). March and Smith, (1995), gave the essentials of the DS research outputs. Hevner et al. (2004) further elaborated the essential elements of the DS outputs. The next section explained the role and importance of the DSR outputs.

3.4. DSR Outputs

The output of DSR can be an artefact or DS theory or both. March and Smith, (1995) purposed four general outputs for the Design Science research and each output referred as an artefact.

For example, constructs (specific data modelling formalisms), models (a set of interrelated data modelling formalisms), methods (data modelling language) and instantiation (the realisation of model method and construct in an environment) can be the artefacts produced during implementation of the design research (March and Smith, 1995).

3.4.1. Artefacts

The artefact can be a prototype, an architecture or set of guidelines for the improvements. In the case of current research, the architectural framework of the decentralised social network platform can be considered as an artefact. In the research, literature artefact is described in various contexts, but the most common description of artefact can be found in Simon, (1996). According to Lee, (2010) for the science of the artificial, the first and the foremost requirement of knowledge building is its efficiency and effectiveness for bringing into existence an artefact to solve the given problem.

Outputs	**Descriptions**
Constructs	The conceptual vocabulary of a domain
Model	A set of propositions or a statement expressing relationships between constructs
Methods	A set of steps used to perform a task
Instantiation	The operationalization of constructs, model and methods
Better theories	Artefact construction as analogues to experimental natural science together with reflection and abstraction

Table 3-3: Outputs of Design Science Research

Simon conceptualised an artefact as a man-made product that “can be thought of as a meeting point” which is an interface between various environments (Simon, 1996). According to the Simon, (1996) the designer main concern should be “how things ought to be” and focus on prescription and finding ways in which that “adaptation of means to environments is brought about” until a satisfying solution is found. Table 11 summarises the way concept of output is described in the DS research. March and Smith, (1995) divided DS output into four types table 3-3.

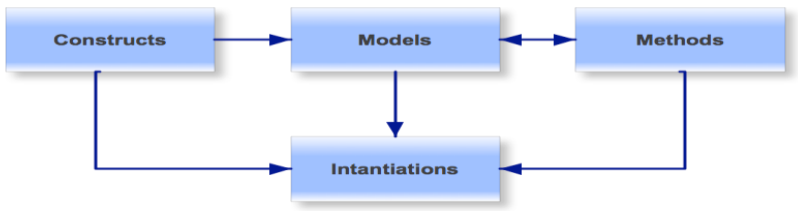


Figure 3-6: Relationship between DSR outputs (March and Smith, 1995).

The evaluation of an artefact is basically, demonstration of the artefact’s ability to solve the planned problem. Having explained the DSRM and its outputs, the next section describes the DS research evaluation.

3.5. DSR Evaluation

There is little guidance available in DSR literature about the adoption and choice of strategies and methods for evaluation (Venable et al. 2012). It is necessary to demonstrate that the developed artefact coincides with functionalities and requirements established during the design and development phase. The artefact must be evaluated to check its validity in the context of the problem described. The researcher must ensure that the prototype produced some viable results in addressing the problem (Ellis and Levy, 2010).

According to March and Smith, (1995) the evaluation of the artefact is a process of finding how well the artefact perform, that is the rigorous demonstration of the utility of the artefact.

3.5.1. Purposes of Evaluation in DSR

Venable et al. (2012) outlined 5 purposes for evaluation from DSR literature.
(1) Evaluate an instantiation
(2) evaluated the formalised knowledge,
(3) evaluate a designed artefact by comparing it with the formalised knowledge to understand whether it achieves a similar purpose,
(4) evaluate designed artefact with the purpose to know the consequences of evaluation,
(5) evaluate the designed artefact to find weakness and areas of improvement for an artefact under development.

According to Hevner et al. (2004), evaluation of an artefact is established by the requirements set by the business environment. Therefore, evaluation is an integration of the artefact within the technical infrastructure of the business environment. In notable DS literature (March and The form of artefact also affects the criteria of requirements in which the artefact will be evaluated. As shown above (figure 3-4 and 3-5) DS research is an iterative and incremental activity, the evaluation phase provides feedback to the design and development phases to improve the requirements and quality of artefact. This cycle can continue until the artefact satisfies all the requirement and constraints meant to be solved (Hevner et al. 2004; Peffers et al. 2008).

3.5.2. Selection of Evaluation Methods

There are different methods discussed in the literature for DSR evaluation, such as Hevner et al. (2004), Peffers et al. (2008 and Venable et al. (2012) have identified some methods for DS evaluation. This research adopts scenario-based evaluation style from the descriptive evaluation methods described in Hevner et al, (2004). The description of evaluation methods is given in table 3-4.

Method	Description
Observational	**Case Study**: Study artefact in depth in a business environment. **Field Study:** Monitor use of artefact in multiple projects.
Analytical	**Static Analysis:** Examine structure of artefact for static qualities (e.g., complexity) **Architecture Analysis:** Study fit of artefact into technical IS architecture **Optimisation:** Demonstrate inherent optimal properties of artefact or provide optimality bounds on artefact behaviour **Dynamic Analysis:** Study artefact in use for dynamic qualities (e.g., performance)
Experimental	**Controlled Experiment:** Study artefact in controlled environment for qualities (e.g., usability) **Simulation**: Execute artefact with artificial data
Testing	**Functional** (Black Box) Testing: Execute artefact interfaces to discover failures and identify defects **Structural** (White Box) Testing: Perform coverage testing of some metric (e.g., execution paths) in the artefact implementation
Descriptive	**Informed Argument:** Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artefact's utility **Scenarios:** Construct detailed scenarios around the artefact to demonstrate its utility

Table 3-4: Design Evaluation Methods (Hevner et al. 2004)

Furthermore, the selection of evaluation methods must be matched appropriately with the designed artefact and the selected evaluation criteria. In Prat et al. (2013), Cleven et al. (2009) and Hevner et al. (2004) goals, environment and system structure is mentioned as key dimensions of evaluation criterion. For example, goals can be subdivided into efficacy, validity and generality. The criterion of evaluation also depends on the artefact development and objectives of the research. The evaluation can lead to the final conclusion or further modifications or both as mentioned in Peffers et al. (2008) and Kuechler and Vaishnavi (2012).

In order to realise the final version of DSNA, this research adopts the scenario-based evaluation method. According to which detailed scenarios are built to verify the practicality of the proposed final artefact. The goal criteria of evaluation are on the basis of which requirements of evaluation and scenarios are illustrated. For example, the interaction between the users of the different social network can be evaluated on the basis of efficiency or efficacy. The evaluation is explained in chapter 6. The next section describes the application of the DSR to the current research.

3.6. The Practical Application of DSR

This section describes the application of DSR methodology in the proposed research and outlines the development of phases as conducted by following the general DSR methodology of Kuechler and Vaishnavi, (2012) under the guidelines of Hevner et al. (2004) and Peffers et al. (2008). The research process is divided into five main phases;

(1). Awareness of the problem and type of solutions
(2). Suggestions (or Conceptual Design based on literature)
(3). Design and Development
(4). Demonstration or implementation and
(5). Evaluation.

The first phase forms the understanding of the problem and the knowledge required to enable the development of the DSNA. During this phase, guidelines based on academic literature and industry sources are analysed to form the understanding of social web architectures and role software architectures for the development of decentralised architecture. To form the understanding regarding the portability problem in social web networks, various available solutions in academia are analysed to construct a basic conceptual framework of components needed to design and develop general purpose decentralised social network architecture. The purification of knowledge from the awareness of the problem phase leads to the initiation of design and development phase. The whole development process concludes with an evaluation of developed artefacts implementation and further improvement to the final solution.

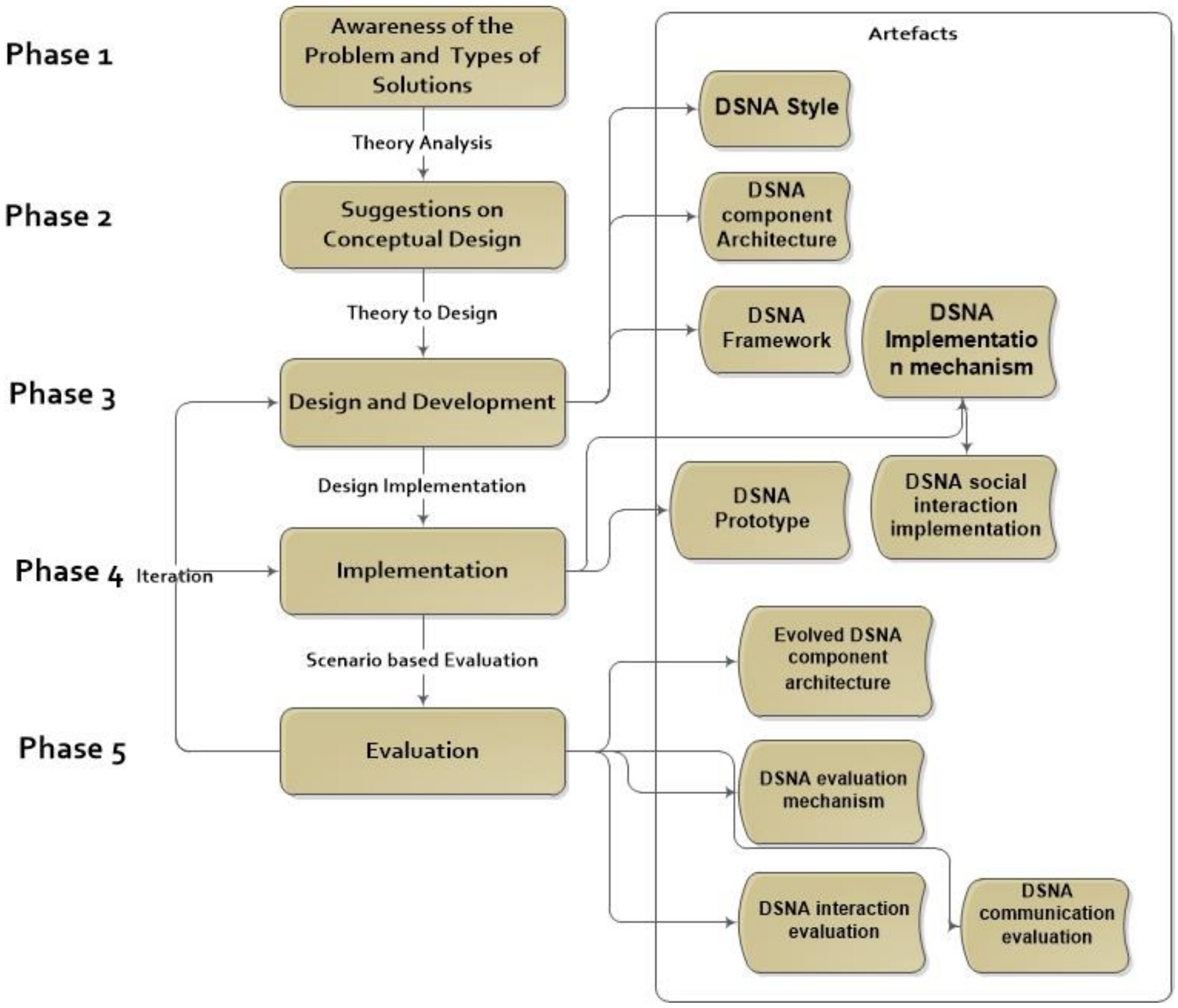

Phase 1
Awareness of the Problem and Types of Solutions
Theory Analysis
Phase 2
Suggestions on Conceptual Design
Theory to Design
Phase 3
Design and Development
Design Implementation
Phase 4
Iteration
Implementation
Scenario based Evaluation
Phase 5
Evaluation
Artefacts
DSNA Style
DSNA component Architecture
DSNA Framework
DSNA Implementatio n mechanism
DSNA Prototype
DSNA social interaction implementation
Evolved DSNA component architecture
DSNA evaluation mechanism
DSNA interaction evaluation
DSNA communication evaluation

Figure 3-7: Outline of the research main phases

The development of artefact has been done in three phases and is called development stage. Each phase in the development stage is completed under the guidelines of Mark and Smith, (1995) 3 stage iteration strategy i.e. design, deploy and evaluate.

The first version of general social web architecture based on theory and academia is produced in phase 1. The requirements and components of the architecture are illustrated in phase 2 that gives the conceptual architecture for social web platform based on software architecture and decentralisation principles. The cycle of iteration continues between 3, 4 and 5 until an effective solution is found. The iteration stops when either the process is interrupted or required criterions are met and effective solution is found.

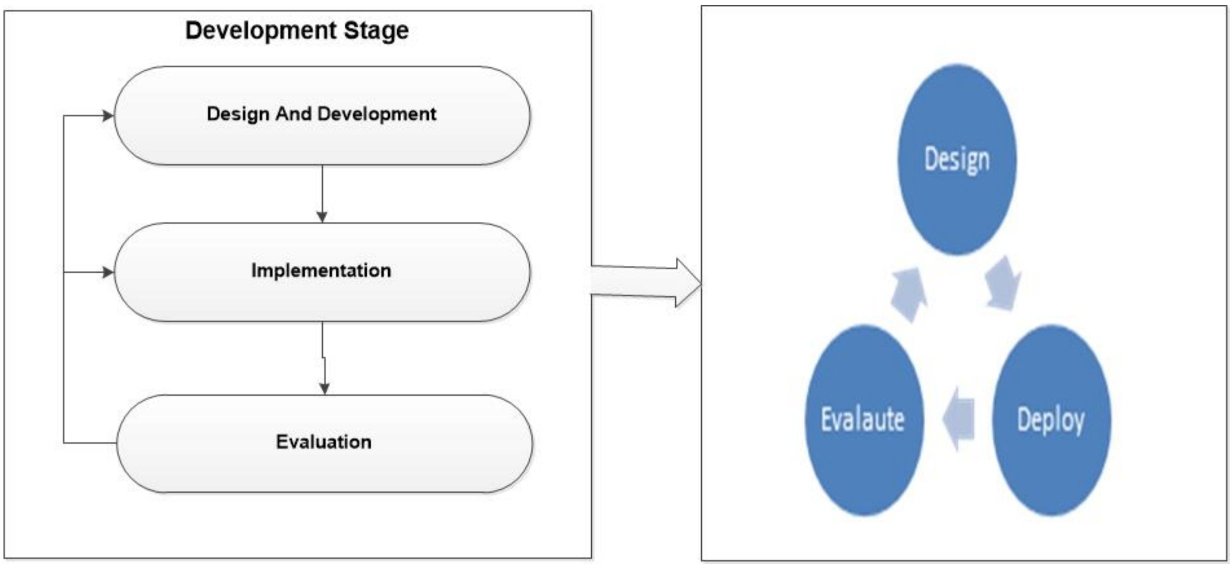


Figure 3-8: Structure of the development stage and iterations taken from (March and Smith, 1995) and modified

The outcome of the "Awareness of the Problem and Type of Solution" phase suggests an initial conceptual framework of components for the decentralised semantic architecture for the social web, based on already available solutions in academia of similar problems. The knowledge from chapter 2 is used in chapter 4 to list the requirements for artefact design. In chapter 5 more logical underpinning of the proposed architecture is formulated based on the requirements. The artefact which is in the current case is decentralised semantic architecture is evaluated in chapter 6 on the scenario-based implementation of the proposed artefact. The evaluation continues in iteration until the set objectives are achieved.

3.6.1. Awareness of the Problem and type of solutions (Phase 1)

This phase investigates the lacks in the application of the software architecture guidelines in the formation and standardisation of social web architecture. The necessary groundwork for this research is done by deriving initial requirements based on the research objectives. The primary activity in this phase is the formalisation of social web architecture components and requirements needed to build initial conceptual architecture.

In this phase, the differences between the various areas in software architecture discipline are investigated. That also includes the study of software architecture basic principles, design elements (components, styles, views) and their adoption to visualise the components required for general social web architecture.
There are three main activities performed, that are deemed helpful in addressing the research problem.

- Determine the requirements for architectural design
- Identifying Web Architecture reusable components
- Generalisation of the solutions

3.6.2. Suggestions on Conceptual Design

The work done in this phase is grounded in three main areas, software architecture engineering, web application architectures and the semantic web technologies. The issues identified in the first part of the literature review are investigated in the context of data portability between social web Network platforms. The attempt made in this phase to critically analyse three main domains of this research

As illustrated in figure 3-9, with the aim to review the existing research, approaches, methodologies and tools that can be used to enable data portability between social network platforms.

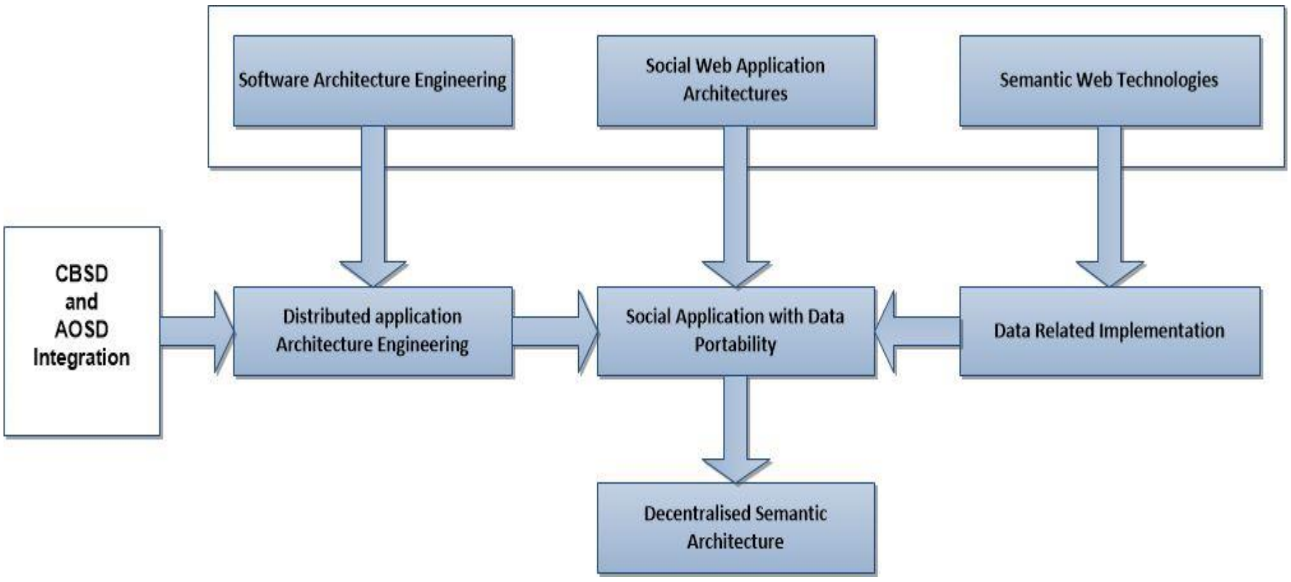

Software Architecture Engineering
Social Web Application Architectures
Semantic Web Technologies
CBSD and AOSD Integration
Distributed application Architecture Engineering
Social Application with Data Portability
Data Related Implementation
Decentralised Semantic Architecture

Figure 3-9: Structure of Concepts and their relationships with each other in phase 2

The literature reviews also helped to discover the gaps in the existing research and helped to identify the relevant components, rules and principles for the proposed artefact. The construct of the decentralised semantic architecture is based on 3 domains as mentioned in figure 3-9. Software architecture engineering provides design and development rules and principles, starting from requirement gathering to the identification of components, services, dependencies and constraints between the components. The study of Social web application architectures provides the essential knowledge, approaches and tools required to enhance the ability of current social network platforms.

3.6.3. Design and Development (Phase 3) Iteration 1

As part of the development stage which consists of three iterations, this first iteration presents, how the integration of CBSD based PACE principles and AOSD based CAM gives the framework of rules and components that are required to build DSNP. The DSNP which is decentralised at a functional level. The component-based conceptual architecture is derived from the literature explained in earlier phases. The derived architecture is based on component-based software development (CBSD) architecture C2 style and Aspect-oriented based software development (AOSD) CAM style. The PACE architecture which is an extension of C2 style provides fundamental principles required to define and describe selected DSNA components. On top of it, CAM is used to provide a component composition and relationship rules.
To demonstrate the functionality of the DSNA in this iteration very basic messaging application is built. The key principles of the DSNA style and architecture are followed during the design, deploy and evaluation phase of this stage.

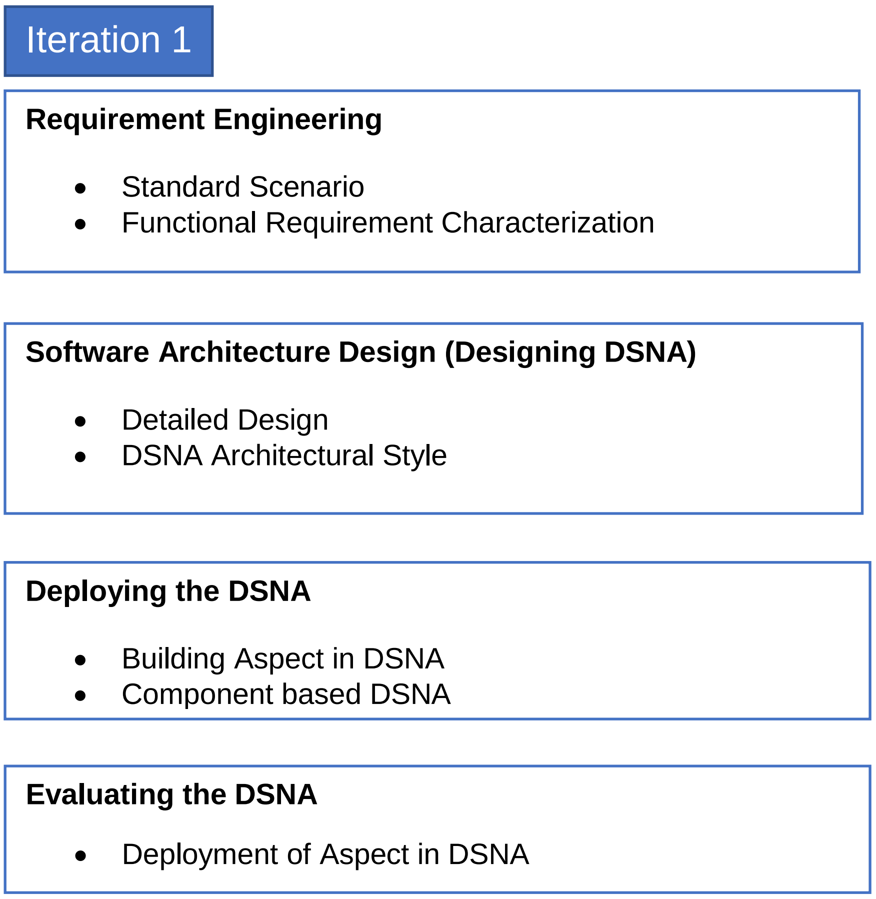


Figure 3-10: Iteration 1 structure

3.6.3.1. Requirement Engineering

The purpose of this step is to illustrate the requirement for the DSNA based application. The requirements are based on the W3 SWATv1 scenario which is explained in chapter 2. The same scenario is extended to include the needs of the decentralised SN application.

3.6.3.2. Software Architecture Design (Designing DSNA)

In this step of the iteration, the foundational design principles required to build the DSNP are explained. The attempt is made to provide the justification regarding the need for the prescribed rules and guidelines. The rules and guidelines include how the communication between the different component will take place. The component communication guidelines are based on C2 architecture. In the advanced stage of this part explains the CBSD and AOSD role in supporting the architecture, which can help enable the decentralisation in SNPs. The core part of this whole procedure is the integration of the CBSD based PACE style and AOSD based CAM. As result of this integration, an architectural style (DSNA style) is obtained. The DSNA style provides specific instructions for the design and development of DSNPs. The role component and aspect are defined moreover, how the component and aspect are described in the architecture.

3.6.3.3. Deploying the DSNA

This step of the iteration is aimed at providing the description of the component required to design and build the DSNP based on the DSNA. During this iteration, the focus is on the composition of components and aspects. The implementation of aspect is demonstrated in the simplest form of functionality. A DSNA conceptual view is explained, including the key elements of the architecture. At this stage, how components and aspects are distinguished, their representation in the overall platform is explained.

3.6.3.4. Evaluating the DSNA

At this final step, of iteration one DSNA is evaluated against the simple messaging functionality. According to the research design process, each of the above steps has contributed to the evolution of DSNA. Each step produced an improved version of the DSNA.

The initial version of artefact obtained from this phase is a result of design and deployment phase of DSR. In this stage of iteration, the main pillars of DSNA, that are Component, Aspect and Role are demonstrated in the messaging application. Tools and technologies used for the demonstration of all the important components are explained. System sequence diagrams are used to demonstrate the behaviour of the application. The end results from the demonstration are used to build better, evolved version of DSNA in the next stage of the development phase.

3.6.4. Implementation (Iteration 2)

This stage is aimed at introducing the improvements required under the guidance of the iteration one and by following the DSR process. As per iterative design guidelines, improvements are made in the artefact. In this version of DSNP application, DSNA is implemented in a complex application than the one in iteration 1.

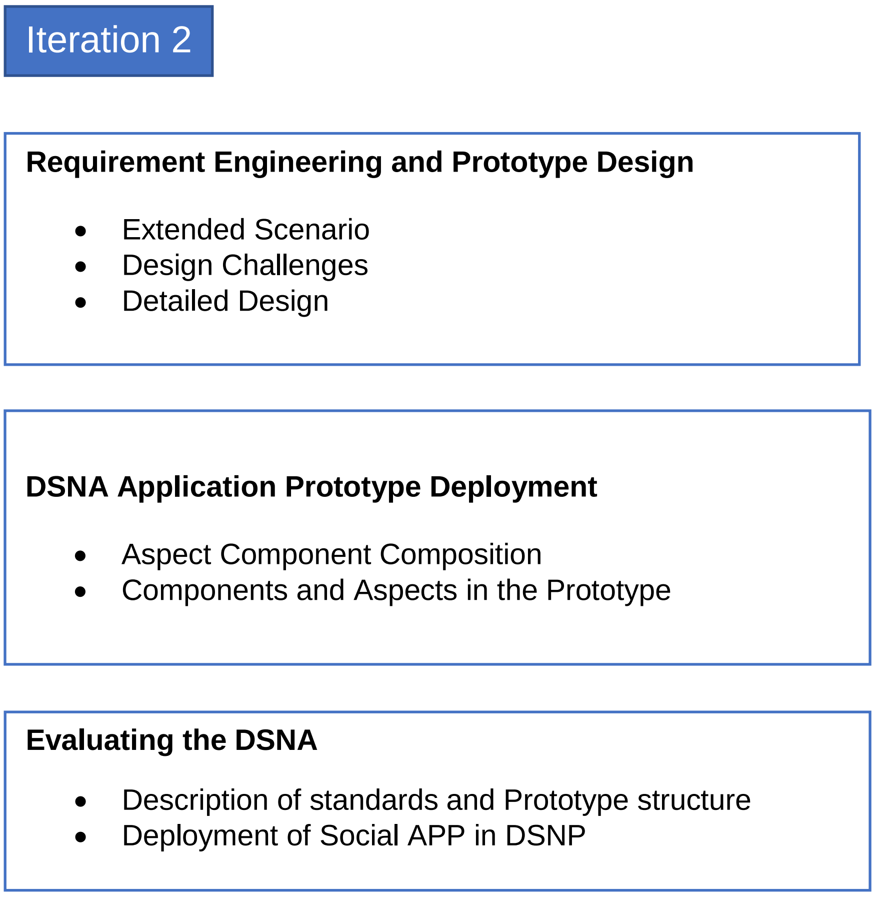

Iteration 2
Requirement Engineering and Prototype Design
Extended Scenario
Design Challenges
Detailed Design
DSNA Application Prototype Deployment
Aspect Component Composition
Components and Aspects in the Prototype
Evaluating the DSNA
Description of standards and Prototype structure
Deployment of Social APP in DSNP

Figure 3-11: Iteration 2 Structure

3.6.4.1. Requirement Engineering and Prototype Design

Based on the scenario already explained in chapter 2 and extended in chapter 4, in this step of the iteration the requirements are illustrated to describe DSNP application design. W3 SWATv1 scenario is improvised to accommodate the needs of the design. The challenges involved in the implementation of the prototype are described in four categories, interaction, communication, composition and allocation. Design related challenges are handled in designing and deployment related are handled in the deployment of the DSNP

The detailed design of the DSNP application describes the core aspects of the component and Aspect-Composition design. Detailed design not only describe the component composition of the DSNA, but also explain how SNSL can be used as middleware to handle the composition of the components.

3.6.4.2. DSNP Prototype Deployment

The aim of this step is the deployment of DSNP application. Which is built under the requirements of the design scenario. DSNP social messaging functionality is built under the guidance of DNSA. All related protocols and standards are explained to provide the knowledge required to procure the decentralisation in SNP's.

During this iteration, the problem related to the DSNA component interpretation in the form DSNP, is solved. Which is done by defining the components based on their roles. The allocation of roles is done in three levels, functional, distributional and co-ordinational. To achieve the full benefits of the DNSA, dynamic component composition is crucial, and the allocation of role describes how it can be achieved at the component specification level. Another part of the component composition is an Adapter design pattern. Adapter act as a bridge between the DSNP and other SNPs. Adapter work in conjunction with role allocation and interpret the data related to functionalities and make it portable to DSNP.

3.6.4.3. Evaluating the DSNA

The evaluation of DSNA is done by demonstrating the social messaging app. The purpose of the evaluation is to discover the effectiveness of the artefact in the proposed design. Also, explaining the importance of the contribution made by the solution.

As part of the evolving research process, the artefact is evolved to solve more complex problem of component composition. There are two important part of the DSNP prototype function, one is SNSL, second is an Adapter. To implement the DSNP protype, the design part adds Adapter as a new addition to the evolution process of the DSNA. The purpose of an Adapter is to guide, interpret and connect DSNA components through the Social Network Support Layer (SNSL). By doing so DSNP accomplish user profile reusability and data probability. To demonstrate DSNP social app is built and behaviour of the application is shown in the sequence diagram.

3.6.5. Evaluation (Iteration 3)

The aim of final stage of the three-phased iteration process is to measure the effectiveness of the solution, to have the final version of DSNA. The SA evaluation can be performed for number of reasons, but the common goal for most of the evaluation is to evaluate the potential of design and to facilitate the achievements of the quality attributed of the architecture. Iteration 3 is aimed at fulfilling the goal of component composition by implementing the stage 2 of the SNSL.

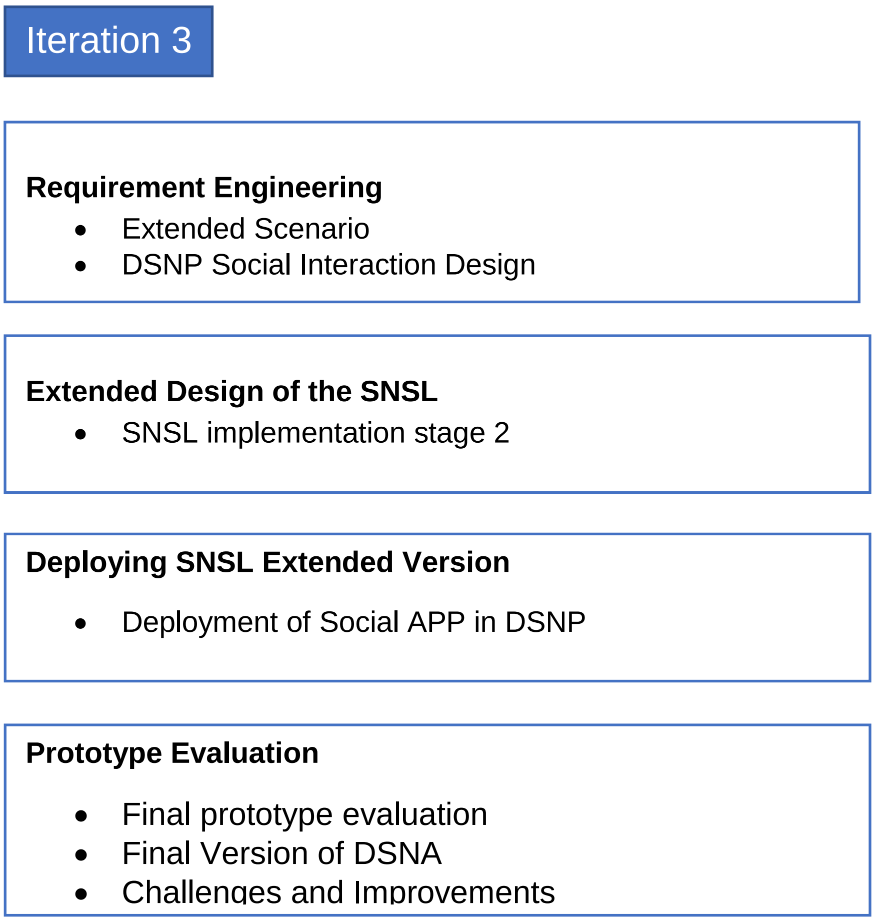

Iteration 3
Requirement Engineering
Extended Scenario
DSNP Social Interaction Design
Extended Design of the SNSL
SNSL implementation stage 2
Deploying SNSL Extended Version
Deployment of Social APP in DSNP
Prototype Evaluation
Final prototype evaluation
Final Version of DSNA
Challenges and Improvements

Figure 3-12: Iteration 3 Structure

3.6.5.1. Extended Design of the SNSL

This step of the evaluation is aimed at providing the extended needs that are required to building the definitive version of the DSNA. The lesson learned in the previous iterations are also used in the new design for the evaluation of the DSNP application.

To meet the prescribed requirements, an extended scenario of SWATv1 is used to evaluate the finalised version of the DNSA based prototype. A specific criterion is set to test the social interaction between the different SNPs in the DSNP environment.

The iterative process of development stage is completed, when the design requirements are met, and final version of the artefact is accomplished. DNSA architecture's better version is produced during the protype implementation. To complete the development cycle, DNSA's SNSL stage 2 is implemented so that final version of the DSNA can be evaluated.

3.6.5.2. Deploying DSNP Extended Version

Lessons learned from the implementation of the SNSL stage one, are used to improve the next version of DNSA. Some issues appeared during the implementation of DSNP, that are regarding the data consistency and persistence. To resolve these issues SNSL second stage propose necessary components. These components are part of DSNA's SNSL and work in conjunction with an Adapter.

3.6.5.3. Prototype Evaluation

After the successive iterative phases, this stage produces the refined version of the DSNA based on the lesson learned from the previous iterations. The challenges posed to the success of the implementation of the DSNA are described. The importance of the inclusion of the various components is explained. For example, why the dynamic composition of the component is required to accomplish a key task that is crucial for the implementation of the DSNA. At last stage of evaluation, the prototype behaviour is evaluated based on the SWAT scenario and prototype interaction and communication with other SNPs is assessed.

3.7. Chapter Conclusion

The chapter presents the detail of the research activities performed in this thesis. The centre of these activities is research methodology. The research methodology adopted for this research is design science research methodology. The methodology consists of the construction and evaluation of the artefacts that resolve a significant and recognised problem (March and Smith, 1995). The design science research is used to get the reliable and practical outcome from the implementation of the DSNA. In addition to that DSR methodology is an iterative activity where solution artefacts are designed and developed through various cycles, processes, activities, inputs and outputs. The goal of a DSR is to generate a purposeful artefact that addresses a practical problem, specially, when elements of the problem are not completely understood (Hevner et al. 2004) thus increasing the validity and reliability of the artefact.

The principal guidelines of DSR dictate that the initial artefact must be refined in the form of constructs, model, methods and instantiation to propose an effective solution to the problem. For instance, to identify the issues with current social networking platforms, a literature review was done and used to produce a solution that evolved and improved in three iterations, in the design and development phase. Now the research methodology in place, in the next chapter component based DSNA is produced as a result of the first iteration.

Chapter 4 - Iteration 1

Chapter 2 Decentralised Design of the Social Networking Applications

4. Chapter Introduction

The main contribution of this iteration is to present the design of the architecture and architectural components to support the realisation of decentralisation in different SNPs. This chapter also provides a detailed description of the architectural components. The description of architectural components is required to provide a structure in which to ground the proposed social networking architecture. In the context of this research, it is important to have standardised set of components for the formation of conceptual architecture.

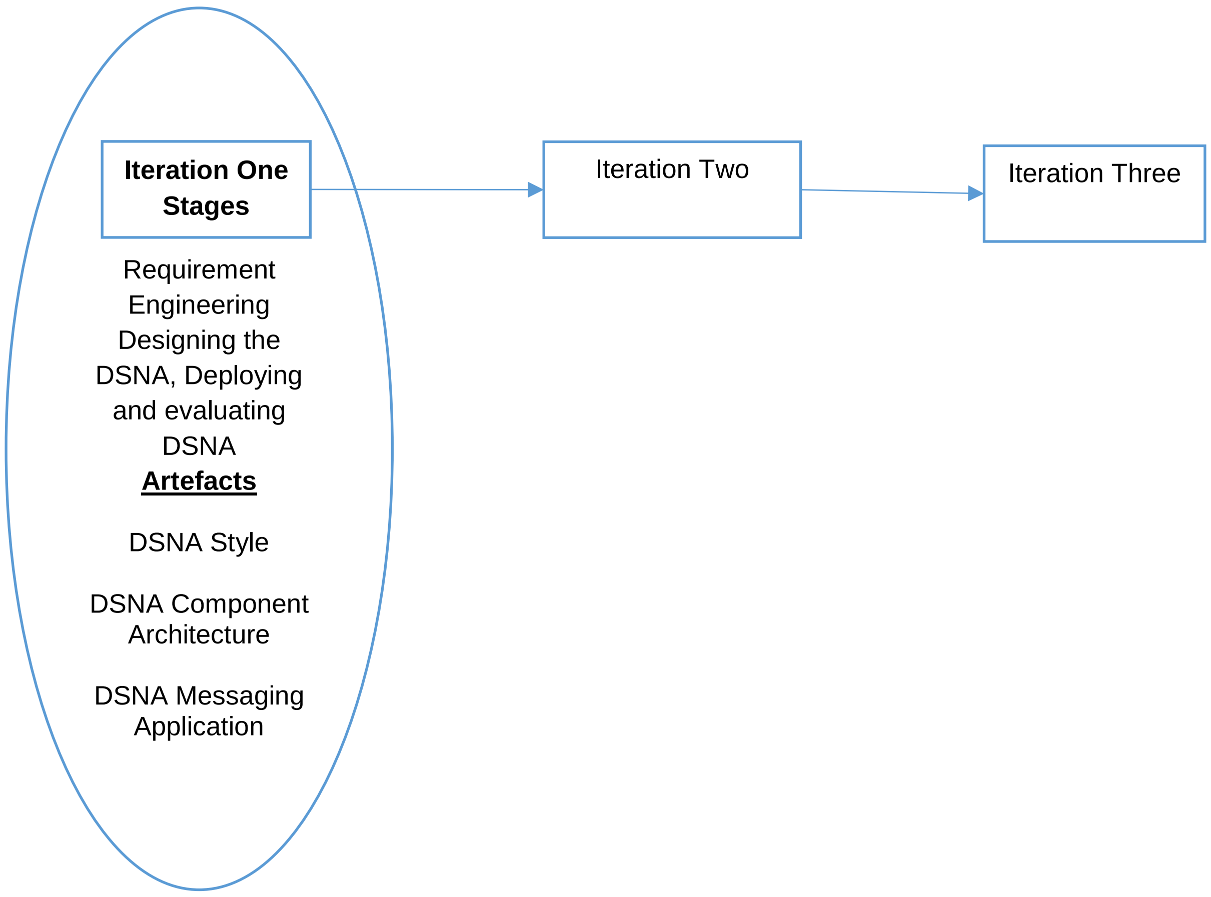

Iteration One Stages
Requirement Engineering Designing the DSNA, Deploying and evaluating DSNA
Artefacts
DSNA Style
DSNA Component Architecture
DSNA Messaging Application
Iteration Two
Iteration Three

Figure 4-1: Iteration one structure

This iteration focuses on objective 3, which is about Designing a component-based architecture (DSNA) to achieve decentralisation between social networking platforms by decentralising the social networking functions. Another part of the objective is to describe the suitability of available architectural styles for DSNA, define and describe the DSNA architectural style and the main components required for the conceptual architecture of the proposed decentralised social networking platform (DSNP).

The architecture is realised by combining the component-based software development (CBSD) architecture C2 style and Aspect oriented software development (AOSD). PACE which is an extension of the C2 style grounding principles is used to define and describe the foundation of DSNA and its components. The Component Aspect Model (CAM) is used to define the DSNA architectural style and its main elements. The style explains how every component in the architecture should be designed and develop.

Iteration one is mainly design focused. The design provides a foundation in term of components and rules. Iteration one produces three artefacts, evolving through three phases of development stage (design, build and evaluate). DSNA style, The DSNA style and its architectural component provides the blueprint for integrating the required technologies into the social network platforms and DSNA messaging application is built to demonstrate the architecture. The extended version of architecture is Instantiated in chapter 5 where the prototype architecture is implemented.

4.1. DSNA Requirements

Drawing on the problem scenario described in chapter 2, this part of chapter 4 extends the problem scenario so that explicit and detailed requirements for the proposed DSNA can be defined. This is done by using a standard test scenario proposed by SWAT v1 (Social Web Acid Test) (W3.org, 2015). SWAT v1 is an extension of SWAT v0. SWATv1 is described separately in various dimensions of the decentralised social web such as data portability, messaging, content deletion etc. According to SWAT, each test should be used to test the level of decentralisation at the functional level between different platforms.

The Scenario

The scenario explains the nature of functionality required from the proposed platform.

- **Users**
  Alice, Bob and Tony
- **Social Networks**

SN1, SN2, SN3

- **SN Functions**

  Data Portability

  Messaging

- **Description**

  - Alice has profiles on SN1 and SN3. She also has another profile for pictures sharing on SN2, as it provides better multimedia functionalities.
  - Her friends Bob and Tony uses SN3 and want to share their pictures with Alice.
  - After Alice joins SN2 Bob and Tony are still her friends

- **Goal**

  In the decentralised social networking platform, according to SWATv1, Alice should be able to use messaging functions to send a message to Tony, and Bob and they should be able to reply back. Data portability between the different SNPs should allow such shared functionalities.

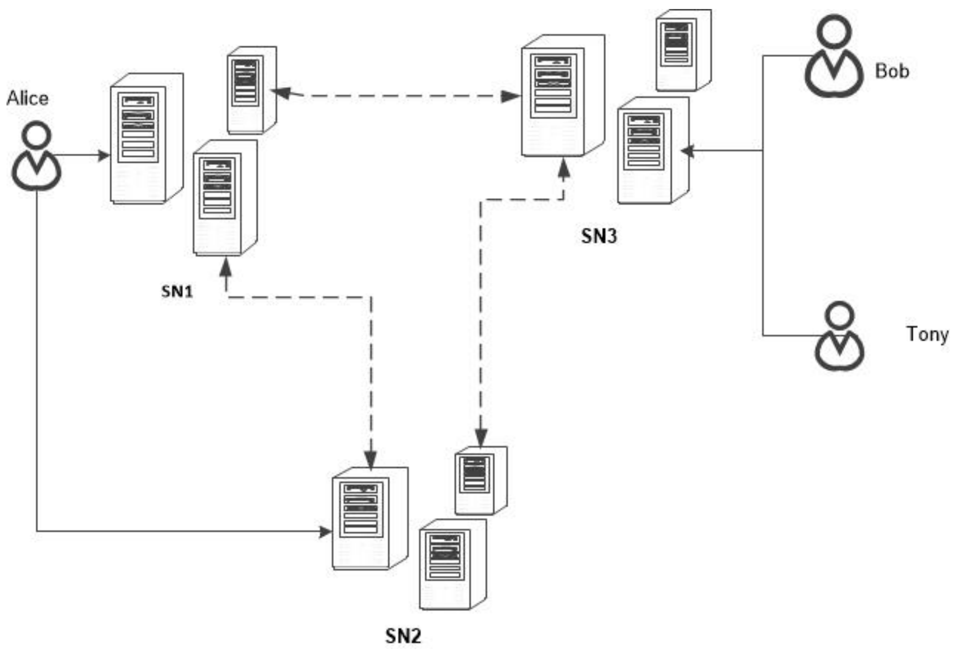


Figure 4-2: Problem Scenario based on SWATv1

In the figure 4-2, dotted lines show how the data portability should enhance the connectivity between the different SNPs according to SWATv1.

Based on the analysis in Chapter 2, the basic requirements for the proposed architectural framework are divided into four main categories that the decentralised social network architecture is required to fulfil.

1. Security and Privacy
2. User Link-ability
3. Data portability

4. Profile reusability

4.1.1. Security and Privacy Requirements

R1. The proposed architecture should provide components to achieve security and privacy by providing more control to the user of their data and privacy.

4.1.2. User Link-ability Requirements

R2. The proposed architecture must provide components to resolve or convert user profile preference to standardised identification, which can be understood by rest of the platform. This is required to improve user link-ability in the DSNP and other SNPs the user belongs to.

4.1.3. Data Portability Requirements

R3. The proposed architecture (DSNA) should provide components to enable data availability across other SNPs. Which means the user should be able to access their data from the proposed platform (DSNP) and other SNPs they are connected to.

R4. DSNP should provide data access and aggregation service to the user profile, accessing from multiple SNPs. A user profile registered with one social network should be able to gain access to other SNPs.

4.1.4. Profile Reusability Requirements

R5. The DSNP should provide components to enable the user to gain access, even if their profile-ID is unknown to the platform user intended to gain access. For example, the third party connect services can provide profile neutrality feature to the user. Third party connect service is explained in chapter 2.

4.2. Social Network Functional Requirement Characterisation

Since the proposed architecture must provide support to the general types of social network functions, therefore, this section analyses the social network functional requirements and the role decentralisation in the different areas of SNPs with more recent examples. Furthermore, analyse how social network function may work in a decentralised environment. The main purpose is to define the proposed platform functional level requirements. This is achieved by characterising basic functions of SNPs and comparing them with existing DSN environment.

The diverse nature of available research on social networks makes a characterisation of SNPs functionalities a cumbersome process. Richer and Koch, (2008) and Kietzmann et al. (2011) work on SNPs functionalities is found relevant to this research, as they provide insight and clear description of functionalities in term of their implementation and usability. Richer and Koch, (2008) characterized social network functions in six basic functionalities.
They are

1. Identity Management,
2. Expert Finding,
3. Context Awareness,
4. Contact Management,
5. Network Management and (6) Exchange as shown in the figure 4-3.

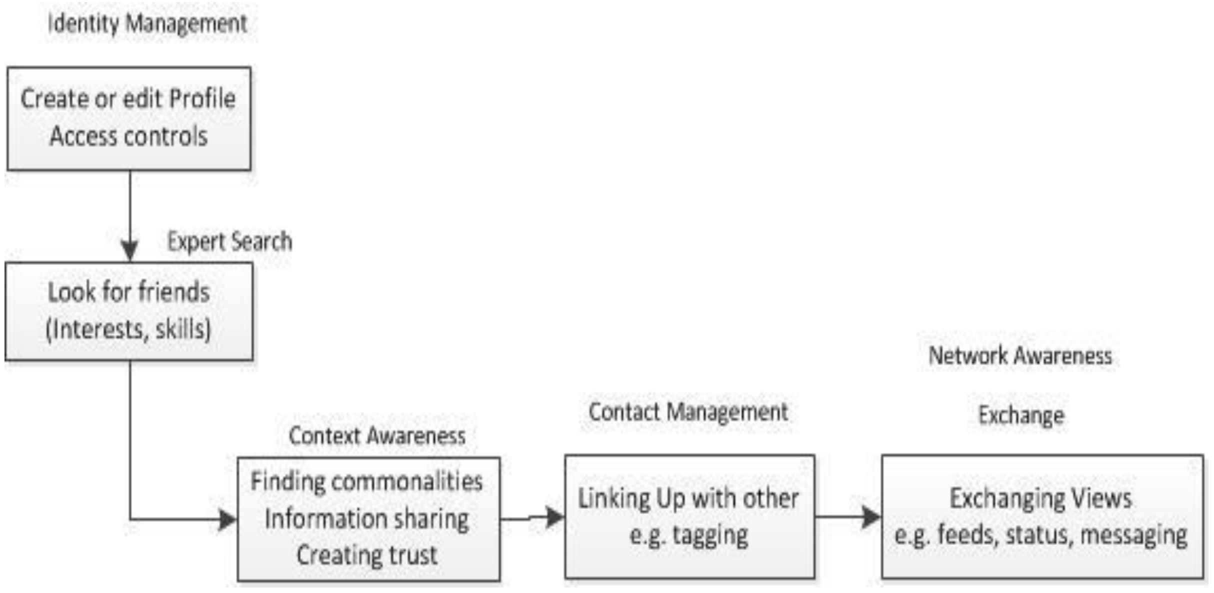


Figure 4-3: Structure of Social Network Functionalities implementation perspective, adopted from Richer and Koch, (2008) and modified

4.2.1. Identity Management

Identity management means the management of identity information availability, that is how the information is stored, setting the access rights i.e. who is allowed to see what (Richer and Koch, 2008). User profiles and group memberships are the most enablers of this functionality. In addition, Kietzmann et al. (2011) suggested identity block (as shown in figure 4-3) that identity management is also responsible to control, to what extent users want to reveal themselves on the social media. For example, to what extent they want to disclose their information such as age, gender, location etc.

One of the many requirements of decentralised social network environments are confidentiality and integrity of user profile related data that are stored in distributed and untrusted storage nodes. The user should be able to have complete control over the permissions to content they create (Nilizadeh et al. 2012).

4.2.2. Contact Management

According to Richer and Koch, (2008), Contact management is a combination of all functionalities that enable maintenance of the personal network. Linking up with other people using tags and adding access restriction to the profile contents are the example of contact management. Kietzmann et al. (2011), described contact management as to the extent user can communicate with other users in each social network environment.

The purpose of contact management in a decentralised environment is related to providing the ability to users that allow them to control their visibility in a social network environment. For example, management of, how the conversation between certain contacts will be displayed and shared in a social network environment.

4.2.3. Content Management

Content exchange combines all possibilities to exchange information directly (messages) or indirectly (photos or messages via wall) (Richer and Koch, 2008). Kietzmann et al. (2011) explained content sharing in a context to the social network as a mode to exchange content between the users. Therefore, management of exchange, sharing and distribution of content from a central interface is content management. The function of content management is to provide a mechanism to store and organise content related data.

In the decentralised social network environment, mainly users are responsible for managing their content. That means user choose where their content will be stored. Up till now research on social networks provides various ways as proposed by Nilizadeh et al. (2012) and Aiello and Ruffo, (2012) to achieve this functionality however strong security and privacy control must be enforced to ensure the safety of user data.

4.2.4. Context Awareness

A social interactive environment such as Facebook, LinkedIn and etc, were not capable of acquiring the information based on common intelligence (Irfan et al. 2013). The term context refers to the relevant information that can be used to categorise the situation of an entity. An entity means a person, place or object considered relevant to the interaction between users and an application, including users and applications themselves (Abowd et al. 1999).

In current social network environments, context awareness provides an appropriate platform for the integration of information that can be collected from tagging of a picture or joining the same group. That context related information referred to user profile giving a basic understanding of the user’s behaviour.

In the decentralised social network, context awareness is subject to requirements. As decentralised architectures are more user centric therefore it’s easier to implement context aware functionalities, (Google hangout is the best example, in which user can feed data that is later used to make context aware decisions and recommendations) therefore context awareness can be used to make recommendations and decisions based on people personal experience or the experiences of other associated people.

4.2.5. Social Network Awareness or Network Awareness

Communication technologies are not enough to promote communication and information sharing. It is important to be aware of other sources of information in a network, to communicate and collaborate. Therefore, social environments must provide means to communicate social cues and context information (Cadima et al 2010).

Cross et al. (2001) and Cadima et al (2010), accentuates that, the accessibility of information in social networks is directly connected to social network awareness, which in their perspective is the awareness of social relationships within the community, the awareness of “who knows whom” and “who knows what”. Social network awareness can be helpful to map access relations at as network level to understand who can reach whom.

In decentralised social network environment as peers, nodes or users are mainly independent that can lead to deeper mutual awareness, more expressive communication, and coordination of ideas between the peers, nodes or users. These functionalities are implemented differently than centralised platforms. All centralised SNP provides sharing status update feature, however, in decentralised SNP messages are distributed in an efficient way with more privacy features.

By summing up here, to achieve decentralisation at the functional level of SNPs, an architecture is required provide principles, guidelines, standards and protocol support to achieve this goal.

4.3. Designing the DSNA Platform Functions

In iteration one, the first version of the DSNA is designed. The aim is to utilise principles of C2 based PACE and CAM to ground a style on which the DSNA is built. The architecture is then deployed and evaluated using a messaging functionality of the proposed platform (DSNP).

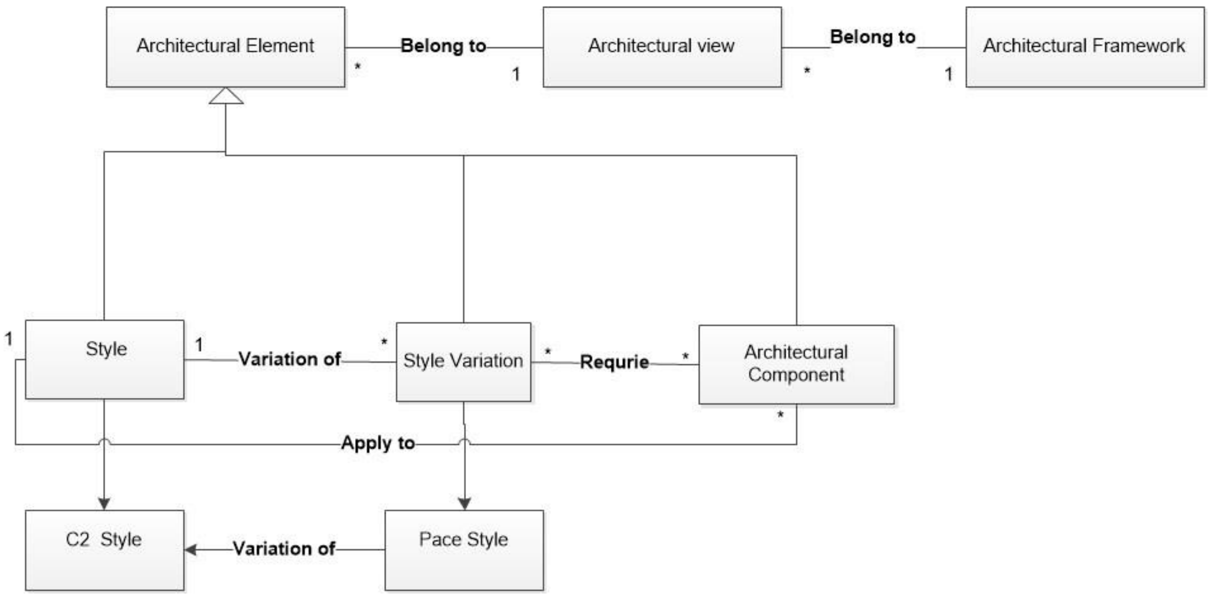


Figure 4-4: Conceptual map of design phase of development stage

Figure 4-4 provides an overview of how the different concepts are associated with each other in the DSNA.

In pursuing the design goal of the research, iteration one attempts to fulfil the design requirement R1 and R2 of by developing a messaging application under the guidelines of the proposed architecture.

4.3.1. Foundations of the DSNA Style

<u>**Component Communication Rules**</u>

The search for a suitable architectural style for the decentralised architecture of software application begins by recognising that nodes (peers or users) are autonomous as they can choose when and how to respond to the information they receive (Suryanarayana et al. 2005). The interactions between nodes and components are divided into two types.

1. Internal interactions

2. External interactions

Since there will be Internal (in context of this research, the communication with the browser and other nodes within the network) and external (which is the communication of components with external SNPs), interactions of architectural components, an architectural style capable of supporting dynamism between coupled components is required (Dooren et al. 2013).

The interactions happen between nodes in the decentralised architecture either synchronously or asynchronously. In the context of message communication, synchronous interactions are suitable for scenarios in which a sender must need a response back and wait until the response and asynchronous interactions are suitable for a scenario in which responsiveness is important and the sender is not sure about the availability of the target.

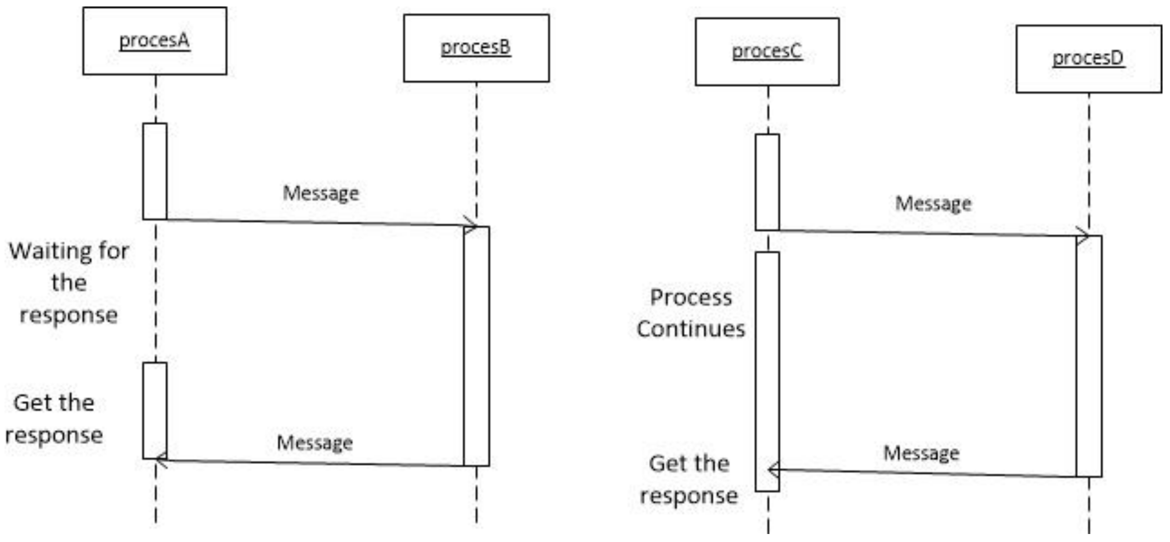


Figure 4-5: Synchronous and Asynchronous Interactions

For example, in the figure 4-5, process A communicates with process B synchronously, that means A send a message to B and waits for B to reply. Process A does not do anything until it gets a reply from process B. In contrast to that when communication is done asynchronously process C continue with another task while waiting for the reply from process D.

One of the drawbacks of asynchronous communication is network connection uncertainty that means it does not guarantee network connectivity or target availability. In Morandi et al. (2013), they suggested using store and delivery mechanism to avoid losing the message. When choosing store during the design process consider the use of local caches should be considered to store messages for later delivery in case of network or system failure.

The proposed architectural style is formally summarised as a network of concurrent component hooked together by message communication (Taylor et al. 1996). Event based architectural styles have been successful in addressing constraints of asynchronicity, dynamism and loose coupling. The C2 (Component and Connector) architectural style fits within these constraints and also provides support to facilitate rapid development (Suryanarayana et al. 2005).

4.3.2. C2 Architectural Style

The building blocks of C2 architectures are components (computational element) and connectors (interconnection elements). This segregation of two architectural concerns that are, computation and communication enable the construction of flexible, extensible and scalable system. The style places no restriction on the implementation languages or granularity of components and connectors, allowing the style to use multiple interoperability technologies for its connectors (Natarajan and Rosenblum 1998).

Drawing from the above section where asynchronous communication is described. C2 is an asynchronous event based architectural style which promotes reusability, dynamism and flexibility through limited visibility (i.e. component independence). In this style, components are arranged in a layered fashion, and a component is completely unaware of the components below. This independence of the layers shows a clear potential for the fostering substitutability and reusability of components across the architecture (Natarajan and Rosenblum 1998; Suryanarayana et al. 2005). Complexity is the main issue with this architectural style, caused primarily by the prevalence of asynchronous behaviour which must be managed by making the interaction between the components more consistent.

Regarding the structure of C2 style, components and connectors have a defined top and bottom that allow them to be arranged in layers. Because of this arrangement the components communicate by passing messages or notifications, travels down in the architecture and requests travels up (Taylor et al. 1996).

<u>Component Interaction Rules</u>

Components can interact with each other following the rules described below and proposed by (Taylor et al. 1996; Natarajan and Rosenblum 1998).

1. The top component can connect to the single connector at the bottom of the layer.
2. The bottom component can connect to the single connector at the top.
3. There is no limit on the number of components or connectors that may be connected to the single connector.

4. When two or more connectors are connected to each other, they must be connected bottom to the top of the other.
5. Components can only communicate through connectors.

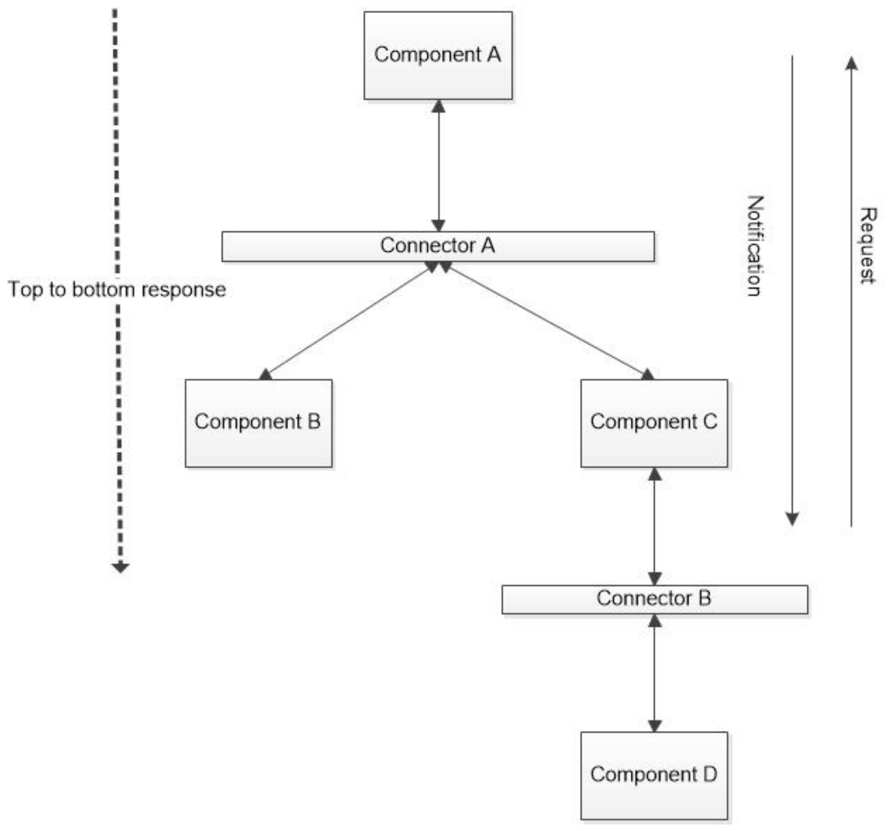


Figure 4-6: Architectural style's components interactions rules

The figure 4-6 shows the relationship between the components and connectors in a top to the bottom approach of C2 style.

Conceptual architecture based on C2 style can be extended and instantiated in a number of different ways. Many potential issues such as interaction constraints and performance are discussed in (Suryanarayana et al. 2006) and (Pinto et al. 2005). In the context of this research, an extension of the component style is required which can be used to conceive the design of DSNA style.

Comparing C2 and PACE

PACE (Suryanarayana et al. 2006) stands for Practical Architectural Approach for Composing Egocentric Trust. The PACE is an improved variation of the C2 style. PACE imposes additional constraints on the structural behaviour of C2 components in the context of peers to address trust management issues in the decentralised application architecture.

PACE is a trust-centric architectural style that addresses the concerns of trust management in decentralised applications. PACE provides explicit guidance on the incorporation of trust mechanisms. The adoption of PACE to the proposed research is based on the approach, that provides the mechanism for integrating communication, data, trust models within an internal architecture (explain in next section) to support the properties that allows trust and data privacy related challenges to be addressed in the decentralised social networking platforms.

Furthermore, the PACE architectural style is selected because it can facilitate the incorporation of trust model into the architecture of the decentralised network. The PACE is about the guidelines and about the components that should be included in the peers, as well as their arrangement and their interactions (Suryanarayana et al. 2006; 2005). Peer corresponds to a system and peers represent a network of systems or nodes connected in a decentralised manner. (Chapter 2 section 2.6 describes the terms nodes and peers)

PACE gives detailed guidelines and implementation strategy of the existing distributed networks to incorporate trust by using the decentralisation in peers. In contrast, social networks are also made of large distributed networks and peers connected to each other by the mean central authority. Therefore, an attempt is made to utilise principles of C2 PACE architecture to achieve seamless decentralised behaviour among the users or peers connected to one or more social networks.

4.3.3. Fundamental Principles of C2 PACE Style

4.3.3.1. Digital Identities

Identity is defined as a set of attributes related to an entity and digital identity is an information on an entity used by the computer system to represent that entity. An entity can be a person, concept, thing, or group (ISO/IEC 24760-1, 2011). Identity management is crucial to the success of the decentralised social networking platform.
The concepts of identities both physical and digital are necessary to facilitate meaningful relationships. The purpose of digital identities is to identify peers in the system through their digital identities allowing the possibility that a single user may pose as multiple peers by using multiple electronic identities. Each digital identity carrying trust information is separately determined and maintained irrespective of the physical identities it represents (Suryanarayana et al. 2006).

There are few constraints that arise because one to one mapping between digital and physical identities may not be possible as one person may have multiple digital identities. Therefore, it is not possible to attach a digital identity to one physical individual. Instead, critical criteria of trust relationships in decentralised applications should be the actions performed by digital identities not by physical identities. Therefore, PACE considers trust relationships only between digital identities (Suryanarayana et al. 2006). For example, in the context of trust management in decentralised SNP, an anonymous user may be present and resisting the acceptance of digital identification. In that case it is not possible to attach a digital identity to one physical individual.

4.3.3.2. Separation of Internal and External Data

This is how the information related to a peer's interactions is stored in a proposed decentralised application. The distinction is made between the internal and external interaction of the peers.

The separation of internal and external data helps to resolve conflicts between externally reported information and internal perceptions. Therefore, PACE makes clear distinctions between the internal beliefs of a peer and the beliefs communicated externally to it by the other peers in the system. Such a distinction is required as there may be a chance that the information received from other peers may be faulty. Therefore, PACE explicitly divides data storage between internal and external information repositories (Suryanarayana et al. 2006).

4.3.3.3. Explicit Trust

In a decentralised application, trust must be visible to other components in the architecture to make accurate decisions. For example, trust related information should be available to the components to make decisions internally with the architecture. This will enhance the collaboration among the peers and provide them the knowledge to make decision related to their privacy.

Each peer needs information to make decisions without the influence of controlling authority. Active collaboration between the peers may provide enough knowledge to make their local decisions. The trust related information can be processed only when it is not localised to one component but distributed across the entire architecture. Each component in the peer is responsible for making local decisions and take the advantage of the trust perceived from other components (Suryanarayana et al. 2006).

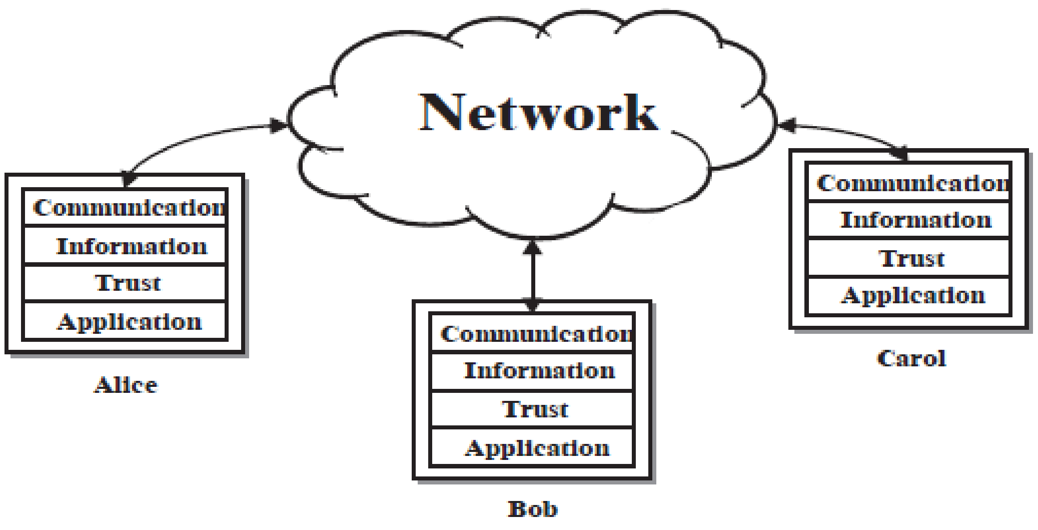


Figure 4-7: External Architecture of PACE (Suryanarayana et al. 2006)

4.3.3.4. Implicit Trust

In a decentralised platform, the purpose of the implicit trust is to handle the internal communication of architectural components.

The components in the internal architecture (Shown in figure 4-8) of PACE are linked via an implicit trust. The only difference is the communication layer (explained in next section) because it is not responsible for validating the messages from other peers. Any notification sent by the communication layer cannot be trusted. (Suryanarayana et al. 2006).

As shown in the figure 4-8, the communication layer handles the external communication of the peers and situated at the top of the architecture. It issues communication requests to other layers and it originates from the components of the layers below the communication layer. Since the request is for the internal communication of the components and it is considered as implicitly trusted. Because of this, the components of the architecture treat communication request differently. An internal request is generated for the communication of the components within the architecture and an external request is generated to communicate externally with other peers (Suryanarayana. and Taylor, 2004; Suryanarayana et al. 2004).

For example, the information layer (explained in the next section) only allows requests to query, update or delete stored information and prevents the notification from external peers received through the communication layer to do the same (Suryanarayana et al. 2006).

4.3.4. Component based PACE Architectural Style

Drawing on the above principles, PACE divides the decentralised architecture into four layers, communication, information, trust and application layers. Each layer and architectural component must adhere to the fundamental principles (explained in section 4.3.4) during the design and development of the platform.

In this section, PACE architectural style is introduced with its specific topological and component constraints. This architectural style is used in the designed and development of the proposed platform. The fundamental principles i.e. identification of identities, separation of data and separation of trust are adopted with the style of architecture to design and develop the component architecture of the DSNA. The figure 4-8, below illustrate the sample architecture constructed in PACE style.

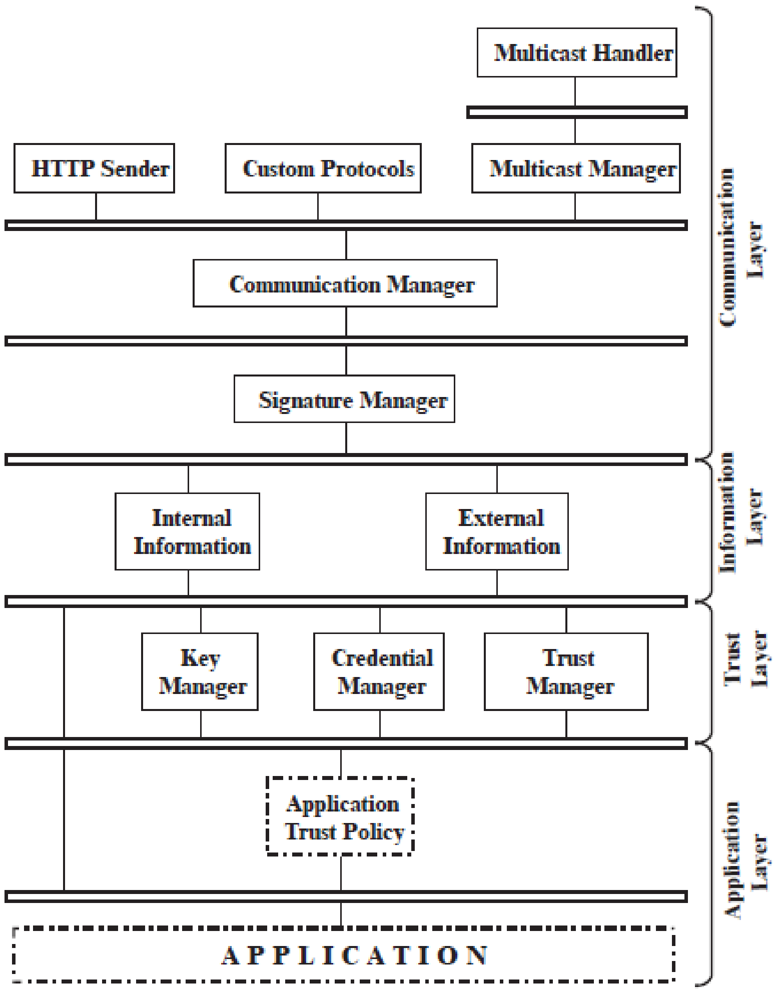


Figure 4-8: Sample internal architecture designed in PACE style (Generic), (Suryanarayana et al. 2006)

The components shown in the above figure are generic and can be replaced based on the requirements of the system.

4.3.4.1. Communication Layer

The purpose of this layer is to handle the communication between the peers in the system. This layer has three main functions,

- Provide abstraction to underlying connection protocols
- Provide a mechanism for multiple connections
- Identity management

To achieve maximum flexibility, the type of data is used by underlying protocols is isolated to protocol handler component. Each protocol handler is managed by communication manager. Underneath the communication manager, there is a signature manager that verifies the communication messages inside the architecture (Suryanarayana et al. 2006).

As shown in the above figure, there are three main components of communication layer.

- Communication Manager
- Protocol Handler
- Signature Manager

The protocol handler enables multiple network communication which is responsible for translating internal events into the format understood by the associated external protocol and vice versa. The communication manager responsible for the dynamic creation of protocol handlers and the signature manager is responsible for signing and verification of requests (Suryanarayana et al. 2006; 2005).

4.3.4.2. Information Layer

The purpose of information layer is to store data and separate the internal information of the peers from external peers. The information layer consists of two components.

- Internal information component
- External information component

Internal information stores request messages that originate from internal components and external information stores the request messages from external peers. The data related to internal information is persistent, to allow the peers to keep the record of their actions. In contrast, the data in external information components need not to be persistent.

4.3.4.3. Trust Layer

The trust layer is a combination of components that enable trust management and policies at the local peer level. To achieve this, the layer is divided into three components;

- Key Manager
- Trust Manager
- Credential Manager

The key manager is responsible for generating and storing the public-private key or a unique key pair for the message authentication in the internal information components. The purpose of a credential manager is to manage the credential of the local peers and that is done by storing peer identity at locally situated cached in information layer or internal information components. Finally, the trust manager is responsible for assessing and computing trust between the peer's based prescribed models and algorithm decided in the requirement design of the trust manager.

4.3.4.4. Application Layer

The application layer consists of application specific components, that means the component is dependent on the specific need for the application. Therefore, these components should be decided during the requirement design process by the developer. In PACE application layer includes application trust rules and application components. Trust rule encapsulate the rules that are assigned to the semantic meaning of the messages and the application component may include the components that may represent the behaviour of the peer, which may include a user interface.

4.3.5. Component based Architecture and Separation of Concerns

The component based architecture such as PACE relies on achieving an accurate functional decomposition of a system into independent components. The goal is the reduction of cost, development time and efforts while improving the flexibility and maintainability of the final application (Pinto et al. 2005).

The advantages of loosely coupled application are well accepted but at some point, when many different applications are interacting with each other in a decentralised environment and producing new objects, then at some point there may be chances of having duplicate functionalities caused by duplicate code. This problem may arise because of low cohesion between the components in the PACE architecture, hence reducing modularity.

The monolithic description of components provided in PACE lacks in the level of modularity (grouping of the logically related element of the application) required to achieve appropriate “separation of concerns” (Dijkstra, 1982) across the different architectural views and roles. This lack of modularity may reduce the ability of PACE managing the multiple variations of applications and functionalities developed and deployed in decentralised social networking platforms.

Therefore, this research adopts the concept, separation of concerns (which is used in the development of highly distributed application under the principles of aspect orientation) (Pinto et al., 2005; Pessemier et al. 2008), to achieve a higher level of modularity, cross-functional integrity and reusability between decentralised applications.

In computer sciences, separation of concerns is a design activity that is used to divide the program into separate distinct sections and each section addresses a separate concern. A concern is a set of information that affects the code. A concern can be some general detail or as specific as some name of the class (Laplante, 2007).

The separation of concern is the core design activity in both AOSD and CBSD. The problem solved in this research is software design related and from that perspective, various concerns are identified and implemented. A concern of an application is related to the functionalities the application provides. For example, a calculator application needs to provide mathematical operators and a user interface to interact with operators. The implementation of operators and user interface are two separate concerns.

During the implementation, the concerns are scattered over many modules. When the concerns crosscut, each other it becomes problematic and creates the code tangling problem. This breaks the key principle of the separation of concerns, according to which each module should not contain more than one concern. This issue hampers the code reusability and effects the modularisation of concerns (Sommerville, 2006).

The aspect orientation is used to tackle the crosscutting concern problem using an aspect (see chapter 2 for definition). In the light of, Kiczales et al. (2001), aspect is used to modularise the SN functions, for the composition or decomposition of aspects is implicit to the mechanism placed for the implementation of the aspect. The next section explains the design mechanism which will instantiate the proposed style of the DSNA.

4.3.6. Design Mechanism of DSNA Style

The core concern of any system are those functional concerns that are related to the system's primary purpose. For example, social network functionalities are the core of an SN and concerns are related to the primary purpose. A general mechanism in the form of concern should be put in place to guide the SN functions in term of their implementation. This mechanism is guided by an architectural style which is mentioned in this research as a DSNA style.

At the core of DSNA style is a unified approach, which is a combination of component-based style such as PACE and Aspect Oriented Software development (AOSD) concept of separating the concerns (Pessemier et al. 2008). A detailed description of AOSD technologies can be found in (Pinto et al 2011; Fuentes et al. 2003).

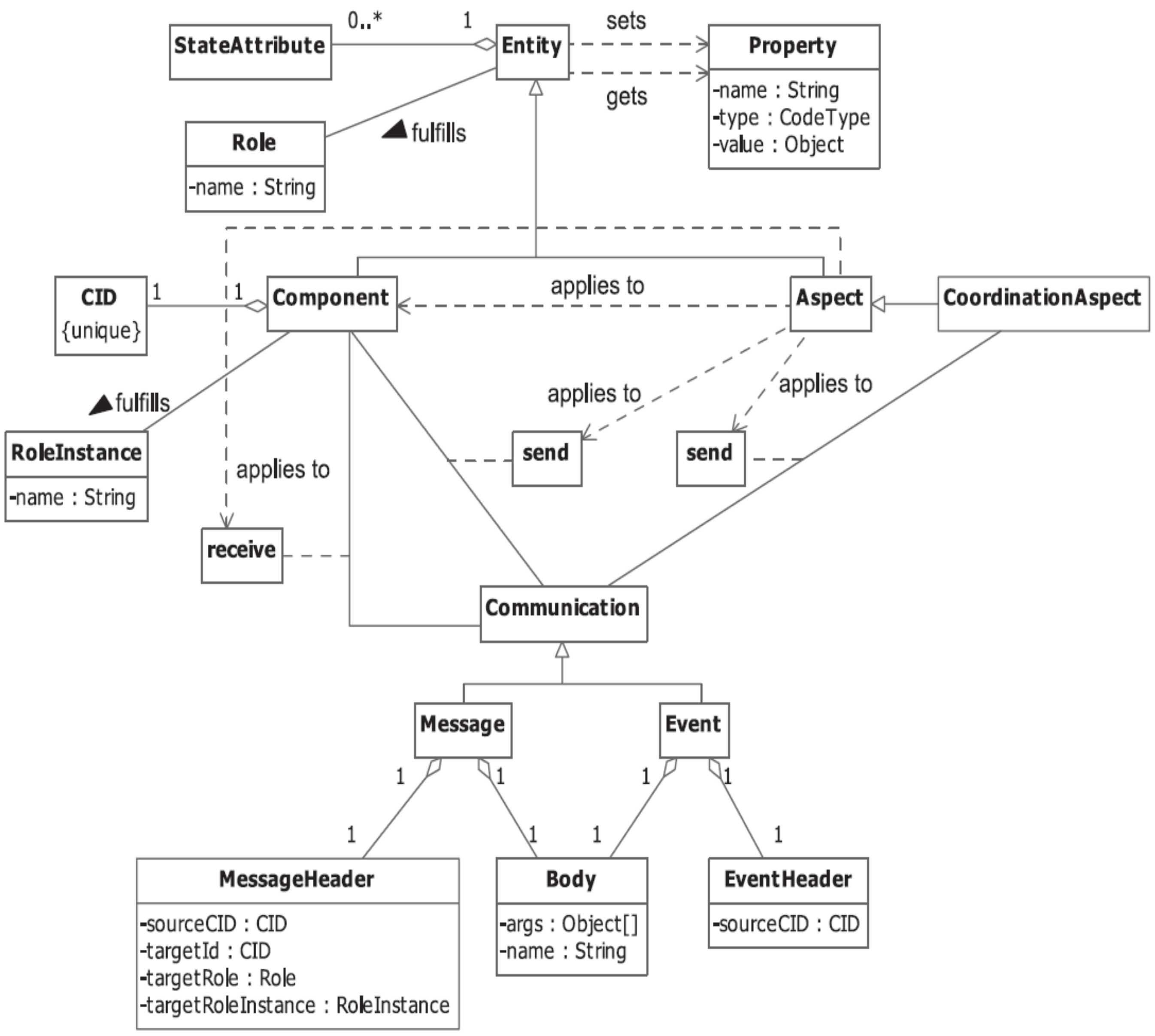


Figure 4-9: CAM Meta Model, (Fuentes et al. 2003; Pinto et al. 2005)

To derive architecture style for DSNA, both aspect based, and component-based techniques are combined to obtain their mutual advantages. In the current case, CAM (Component

Aspect Model) is used. The CAM provides foundation rules for the component composition of DSNA. The CAM defines the basic entities and structure of the system from the architectural point of view. In the case of DSNA components and aspects are the basic building blocks.

The figure 4-9 is a UML diagram with the basic entities of CAM and the relationships that can be established among them. The above diagram is known as UML profile for CAM.

UML Profile: According to, Alhir, (2002) UML profile purpose is to provide an extended mechanism for customising UML models for specific domains or platforms. Stereotypes tags and constraints are used to define profile elements such as classes, activities, entities, attributes and operations. A collective combination of the elements customised to represent a particular domain can be called as UML profile of that domain (Alhir, 2002). The figure 4-9 is a UML profile of CAM with stereotypes and constraints, to design application using CAM.

There are two main entities of CAM, components and aspects. Both components and aspects are required to have STATEATTRIBUTE that represent their current state that is public or private. ROLE and PROPERTY are assigned to the components and aspects to distinguish them from the final implementation of the application (Pinto et al. 2005). The next section describes them in further detail in the context of design.

4.3.7. Design of Social Messaging Application Using CAM

As part of the iteration one, the next step towards the instantiation of DSNA, social network functions such as messaging, or scraping is mapped in CAM. The figure 4-10 shows the design of SN messaging application using the CAM. The core behaviour of the application is modelled, as per rules described in Pinto et al. (2005) and Fuentes et al. (2003) the application of separation of concerns allows the separation of crosscutting functional requirements such as authentication or message filtering. This method makes easier to reuse that social network function components that may have or may not have their properties changed. Pinto et al. (2005), have mentioned that the aspectual properties of the entities that can also be reused in another context.

For example, authentication aspect of the component is applied when the user wants to join the social network to send a message, that means the user must enter some relevant identification information as required by the application. A local instance for messaging function component is created only when the user is authenticated and registered in the network.

Another example could be if the requirements are changed from simple messaging to messaging with chat functionality. In this case, persistent chat aspect can be added or called, which stores the states of the chat component in the data storage. The figure 4-10 describes the relationships between the component and aspects in an implementation of CAM in social network messaging functionality.

4.3.8. Components and Aspects of the Social Messaging in DSNA

In this section, the role of components and aspect in DSNA is described. Moreover, this section also explains the associated entities of the components and the aspects, their relationship principles and how the effective cohesion between all the entities is formed to form an effective DSNA style.

The main entities of the CAM are components and aspect as shown in figure 4-10. In principle, there are no restrictions on the granularity of these entities because of the distributed nature of the application and the way they are composed. The components are disturbed they interact with each other by exchanging messages and aspect are attached to the components to impose some recommendation regarding the level encapsulation. Both components and aspects are considered as course-grained encapsulated entities and act as a unit of composition with contractually specified interfaces and explicit dependencies (Pessemier et al 2008; Fuentes et al. 2003).

4.3.8.1. State Attributes

In the CAM based architecture or models, aspects are treated as a special kind of component having some shared common features with components. The components may have a set StateAttributes to represent their public statements, for example, the information that should be made persistent to restore the state of component or aspect. This information can be used to implement some properties (Pinto et al. 2005).

4.3.8.2. Roles

In their implementations of CAM pinto et al (2005), Fuentes et al. (2003) and Pessemier et al. (2008) have used a new class of ROLE (see Role class in figure 4-9 and 4-10), in order to detach component and aspect interfaces at the final implementation of the application. A unique role name is assigned to identify both component and aspect classes. A specific functionality is encapsulated inside the role that can be executed by the component when it’s called. According to, Pinto et al. (2005) role names are architectural names that are used for component and aspect composition and interaction allowing loosely coupled communication among them.

To demonstrate CAM implementation, simple social network messaging functionality is selected. In the figure 4-10 component, aspect and their relationships with other entities are shown in the context of DSNA messaging functionality.

In the functionality, components with the Role Name ‘ChatRole’ ‘MessageRole’ and three aspects ‘Authentication’, ‘Persistence’ and ‘Filter’ are added. It is quite possible that in a decentralised or distributed application’s several components have the same role, for example, a user using more than one functionality such as having a conversation with more than one person at the same time. To handle such variations, RoleInstance is introduced, which is created on the initiation of any interaction between the users and allocated to the component (Basically it is an instance of the component created by the component).

In the light of the scenario explained in section 4.1.1, a message interaction between user BOB and ALICE will create a MessageRole (Name=Message) and new RoleInstanceMessage (Message_UserNames) and similarly if the BOB and ALICE turns the messaging to chat, then a new RoleInstance, RoleInstanceChat (Chat_UserName) and Role name ChatRole (Name=Chat) are created to differentiate between different chat or message Roles.

4.3.8.3. Component

Differentiating between the component and component instance will be another subject and out of the scope of this research. As there are various definitions of the components are available as such in Shaw and Garlan, (1996) and Szyperski, (2002) (see chapter 2).

In the purposed DSNP the components are made of a set of classes assembled and executed as a single functionality to be deployed by the social networking platform. The implemented form a component in the DSNP is the SNP functionality with specific aspects, properties and roles when initiated by the user as a request.

4.3.8.4. Property

One of the main goals of CAM is to keep aspect unaware of other aspects information and that is applied at the same time to the components as well (Kiczales et al. 2001). This hinders the composition of aspects and components. Fuentes et al. (2003) solved this problem by the adding the adding an extra class PROPERTY. Property is identified by a unique name, type and value. Therefore, in CAM aspects, directly or indirectly resolve their dependencies by sharing properties. The main purpose of the property is to define truly independent component having any kind of data dependency as a shared property (Pinto et al. 2005).

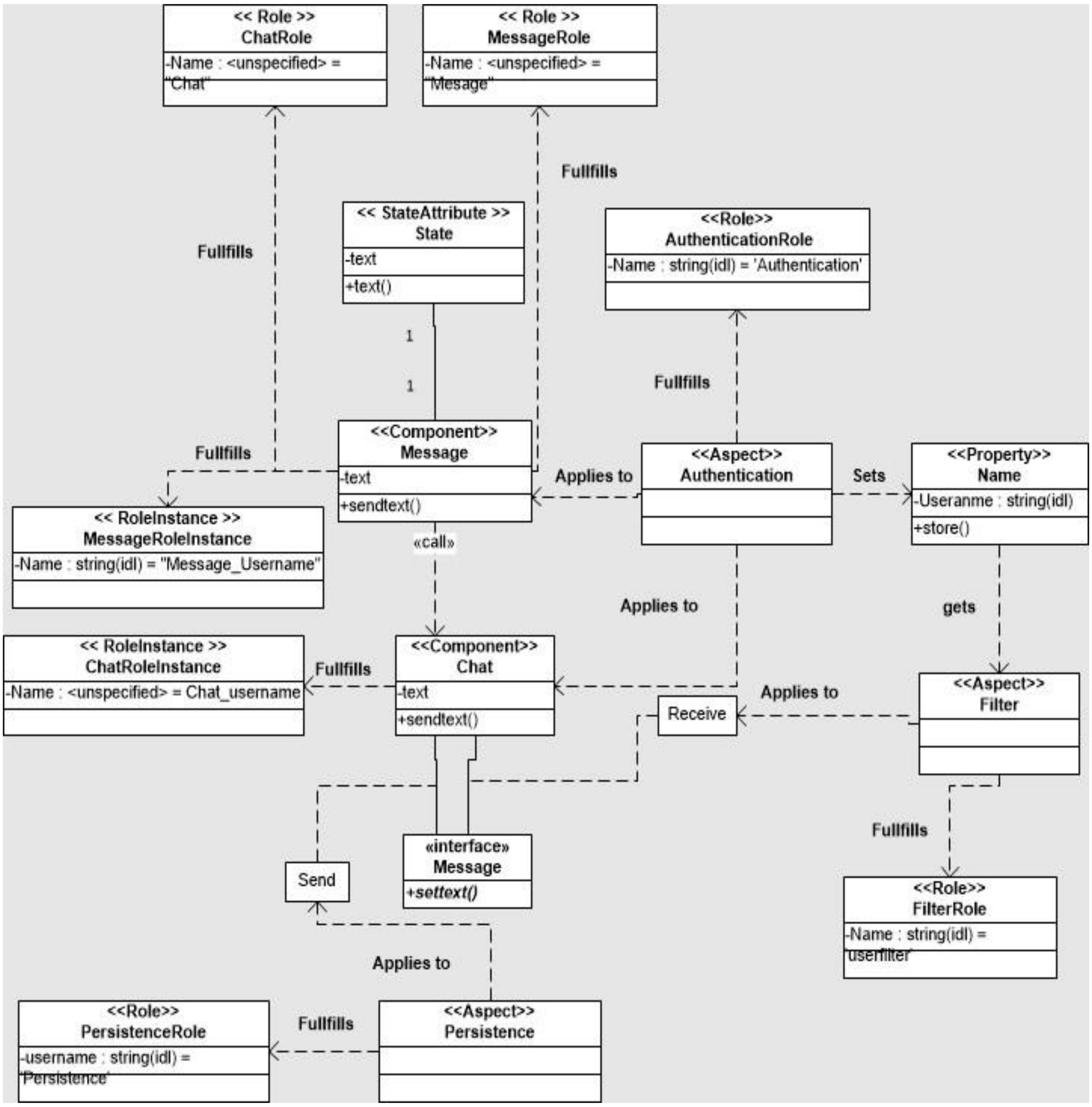


Figure 4-10: CAM Model of Message Functionality of the DSNP

The figure 4-10 shows an example of Property called Name. The figure shows that aspect with the role name AUTHENTICATION and FILTER shares the property USERNAME. The purpose of this property is to authenticate the user once the user information is authenticated, authentication aspect set the value of the property to username and stores it. In case, the user does not want to see the message of the specific user this is where FILTER aspect gets the value, that how the message to be displayed or not to be displayed.

4.3.8.5. Relationships

CAM follows standard practices of C2 or CBSD i.e. component based architectural style for the communication between all the entities of the CAM. In CAM component interact with each other by exchanging messages and events. According to, Rathfelder et al. (2014) messages are sent to communicate with specific target entity and events are also a form of a message that is asynchronously transferred between the components to trigger a certain behaviour.

The value of CAM

- The use of properties in CAM will allow the description of data dependencies and more independent and reusable entities during the design phase.

- The information generated in the description by aspects may have been generated by components as well. This provides a standardised information sharing mechanism between the components and aspects. For example, a PROPERTY USERNAME is created by an AUTHENTICATION aspect based on consultation from MESSAGE component, similarly, it can also be consulted by the CHAT component.

To implement and deploy such architecture in a realistic setting, one must take this into to account that, nowadays the infrastructure of SNP's is continuously evolving. New functionalities (related to publishing, authentication or profiling) are added to deal with new trends. Various functionalities from multiple platforms can make the problem of cross-cutting concern very complex and AOSD, CAM provides a quality solution.

4.3.9. Architectural Style of DSNA

The purpose of the architectural style is to provide a specific abstraction of the elements regarding the system functionalities. According to Fielding, (2000) a style provides set of architectural constraints to restrict the roles and features of the elements and relationships between the elements.
A unified approach is used to combine the component-based style such as PACE and Aspect Oriented Software development (AOSD), to obtain mutual advantages to conceive a style of an architecture that can solve the complex issues related decentralisation problem between

social networking platforms. This section presents DSNA style main entities based on the detailed description mentioned in section 4.3.8.

Fielding, (2000) definition can be adapted to define DSNA style, according to which DSNA style is an architectural style that provides architectural constraints to restrict the roles and feature of the element and their relationship with other elements. The main elements of the DSNA style are;

- Component
- Aspect
- Role
- Property

In the context of this research, the component is a set of classes assembled and executed as a single functionality that to be deployed by social networking platform. In the context of the scenario, BOB wants to send a message to TONY. But first BOB will login to a web application which will initiate, for instance, a login component. Login component has various aspects such as CreateCookie, Authentication etc. initiating login also initiate aspects its roles and properties. When BOB presses login button Authentication aspect is used to check whether BOB is register with the website or not. For this purpose, Property USERNAME is called which check the user validity and set the value in response the user either enter the website or more information is asked.

For the relationship between the components, as mentioned in above section CBSD rules are applied to DSNA. In the case of aspects as they are applied to the component, therefore 'applies to' is considered for DSNA style as for of relationship between the components and aspects.

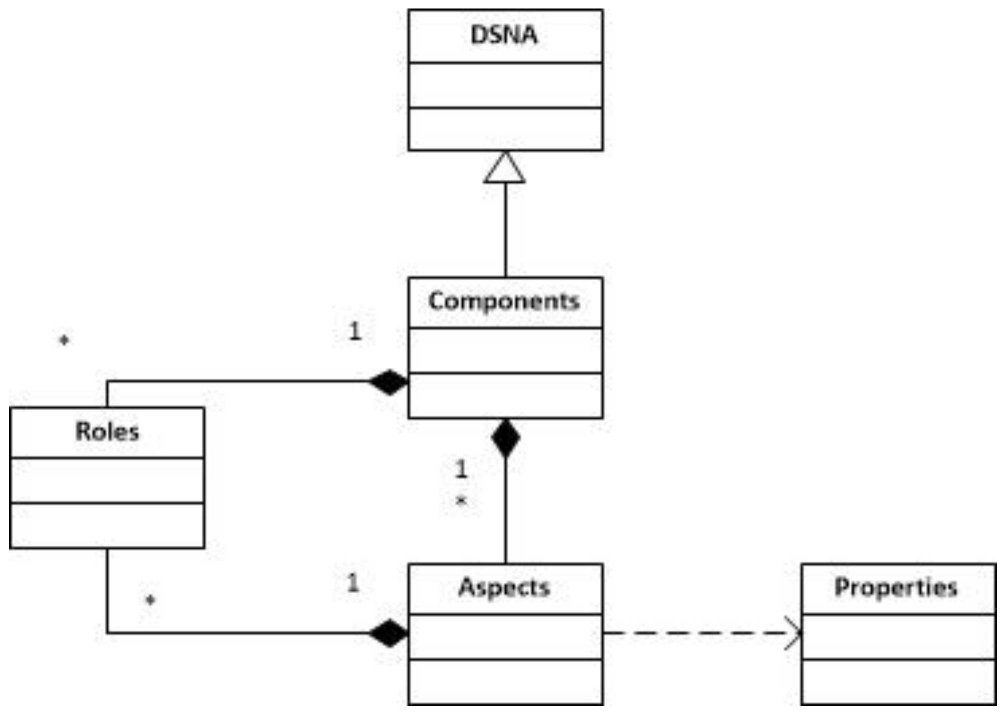


Figure 4-11: Architectural view of DSNA style

Aspects are applied whenever the components are created, and this is same with the role.

Figure 4-11 basically shows the higher-level relationships between the elements of DSNA style using UML diagram. It shows that DNSA (here DNSA mean the application based on DSNA) is a superclass of the components. Each component is associated to roles and aspects. A component may have many roles and many aspects. The aspect and property are associated to with each other in 'dependency' relationship. The outcome from the property can directly or indirectly affect the component.

4.4. Deploying DSNA Platform Functions

As part of iteration one, the transformation of principles and methods proposed in the design of DSNA style is done in the building and deploying of DNSA platform. In this stage, an attempt is made to describe how the DSNA style can be used to help (the developer) decentralisation of social networking platform functionalities. The build process of components and aspects of SN application are described to show the functional view of the DSNA.

In the build stage of the iteration one, DSNA is applied to achieve the very basic function of social network, such as chat messaging function. The most important part of DSNA deployment is the composition of aspects and components to demonstrate their working in the DSNA based platform.

The approach taken in this thesis consists on focusing the SN decentralisation at functional level. This is achieved by separating the behaviour of SN functions at component level, by using non-limited set of aspect component, that are then further used to describe the behaviour of the component interface. With this approach, the SNP functions can be addressed separately and thus enhancing the level of decentralisation among the SNPs.

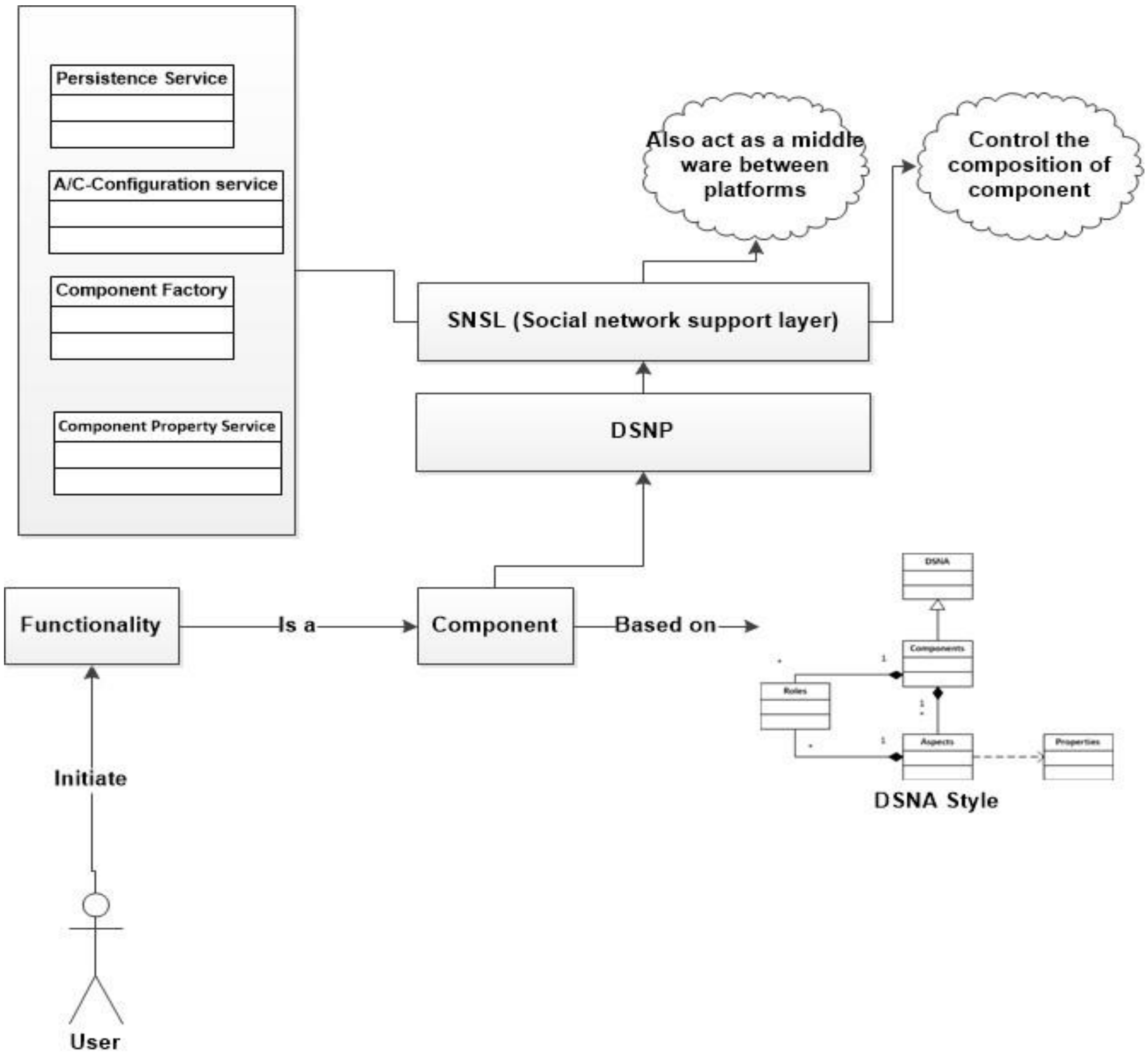

Persistence Service
A/C-Configuration service
Component Factory
Component Property Service
Also act as a middle ware between platforms
Control the composition of component
SNSL (Social network support layer)
DSNP
Functionality
Is a
Component
Based on
DSNA
Components
Roles
Aspects
Properties
DSNA Style
Initiate
User

Figure 4-12: Key building blocks of component composition in DSNA based application

One of the core feature of DSNA is the dynamic composition of component and aspect at runtime level. Which is achieved by following the DSNA style and the design mechanism described in section 4.3. In CAM aspects are represented as components, to increase the reusability of aspects. Figure 4-12 describes the place and work of the component in DSNA. The figure which is a simple depiction of larger architecture shows the relationship between the component, platform and SN functions. The figure also shows the key building blocks of component building in the DSNA based application. SNSL and composition of the components are discussed in more details in chapter 5.

4.4.1. Deploying Aspects in DSNA

The core development concept of any application developed using modern languages is the breaking down of the problem into separate objects and each object grouping together data and behaviours into a single entity. The aspects in DSNA uses the same design, with addition of concerns. DSNP is composed of component and each component is deployed based on the need of the user. Each component represents the functionality used by user of the SNP and it can be either in use or needed to be deployed. Aspect allow user to decide which functionality of the SNP they want to use, and concern make this possible by solving that functionality concern. As explained in CAM ` model a concern is an additional class associated to each aspect and it represent the application requirement which may have been described in the requirement or arises during the application building process.

The CAM approach uses four main functions, that are part of pointcut-advice model of AOSD (Mouheb et al. 2015). Therefore, the implementation of aspect components is done using these four main functions.

Pointcut is an expression in an aspect component that designates set of jointpoints. Basically, pointcut is a statement included in an aspect that defines joinpoints. Pointcut also exposes data from the execution context to the joinpoint.

JoinPoint is a point or an event in the aspect component or any point in the code where aspect is called or executed. Therefore, joinpoint can be method invocation or calls, exceptions, constructor or a catch block. This is the point where joinpoint provide service to the object or class (Mouheb et al. 2015, Sommerville, 2007).

Advice or advice code is the implementation of the concern in an aspect. A concern implements the behaviour into the aspect component. A behaviour can be injected or called anywhere in the code dynamically. For example, according to the current requirement, that can be used to create a behaviour of such as adding SN functions based on user request. Like class or method, the behaviour of an aspect can be split into many types of advice codes. During the DSNA design model as the flow reaches any joinpoint, it is bound to trace the advice to implement the behaviour.

Weaving is about putting the jointponts in the place where it needs to be executed. The inclusion of an advice at the joinpoints specified in the pointcut is the main responsibility of the weaver. The waving function from CAM is used in the DSNA to perform the final execution of the application with the desired aspects included at the specified component.

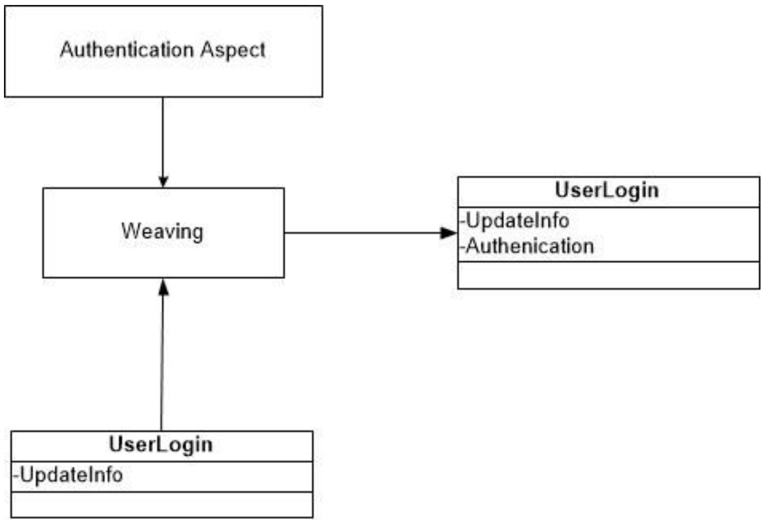


Figure 4-13: Example of Authentication Aspect in context to Weaving

SN Authentication Function Aspect

The user authentication and authorisation functions are the most used in any system, they may have to be included in several different places based on the requirements. In the CAM, based system an aspect can represent any change or any concern that requires additional functionality. For example, updating user password or user security related malfunctioning which may trigger various events like forget password, policy change and may need to call multiple methods at multiple places in the system to fix the problem. In the case of CAM, aspect can be called as soon as required based on rules as defined in the aspect, therefore the need of recreating the methods and calling them at various level may not be required when using aspect. This enhances the distributed behaviour of the system at the semantic level and enhances the reusability of code.

The notion of decentralising the SNPs at the functional level by changing the very semantics of the way they work can be achieved by an aspect representing a function and each function representing a component. The figure 4-13 shows how an aspect includes a specification of where the raised concern need to be woven into the component at the code level.

The notations that are used in the authentication aspect example follows AspectJ (Kiczales. 2000) style but modified and simplified so that it can be understandable to anyone. AspectJ is one of the earliest AO extension written in JAVA. With few new constructs, it provides the support for AOSD.

```
Aspect LoginVerify {
Before: call (public void Authenticate* (…))        // start of pointcut, pointcut is a
collection of jointpoints . Before execution of any method start with the authenticate ,
pointcut should be executed. The Advice carry out the execution.
{
//From here you can add jointpoints based on advice that should be woven at runtime.
//Joinpoint with advice
Int loginAttempts =0;
Declare String UserPassword
//compare user password with entered password
UserPassword= Password.get (loginAttempts)
//set the rules which is a part of advice and start of the point
while (loginAttempts < 3 and userPassword != thisUser.password ) {
//Rule: User is allowed 3 attempts
loginAttempts = loginAttempt + 1
UserPassword = Password.get (loginAttempt)
}
//This is a usual code
If (UserPassword != thisUser.password () )
//Then redirect to forgotten password or log out.
System.logout (thisUser)
}
}
```

Figure 4-14: LoginVerify Aspect Example

The Weaving can be used to add additional functionality to the code. As shown in the figure 4-14, if there are multiple request to create a functionality then instead recreating the methods or calling a method at several places, an Aspect can be used with the specification of where the functionality needs to be woven (Created, In CAM terms). Following this procedure SN functionality can be created dynamically. Chapter 5 discusses the implementation of whole CAM based architecture in further details.

4.4.2. DSNP's Component based Conceptual Architecture Explained

As explained in chapter 2, components are the most basic unit of architecture composition that specify interfaces and set of requirements. Component based software engineering, which is used to define, implement and compose loosely coupled independent components into a form of a system (Somervile, 2007). Component based engineering depends on independent components specified by their interfaces and component standards that facilitate the integration of component into middleware that provide software support for the components integration and deployment (García-Castro et al. 2008).

To finalise the component based conceptual architecture of DSNP this section provides basic description of architectural components. The architecture is divided into 4 layers.

1. Application Layer
2. Social Network Support Layer
3. Data Access Layer
4. Communication Layer

The architecture is based on DSNA style, which has foundations in PACE (component-based style) and CAM. The combination of PACE and CAM provides rules and structure that is required to build a decentralised social network platform. The components in DSNA communicates each other asynchronously based on descriptions mentioned in section 4-3 In the next section DSNA components are explained. The relationship between the components is govern by the rules discussed in above section 4-3.

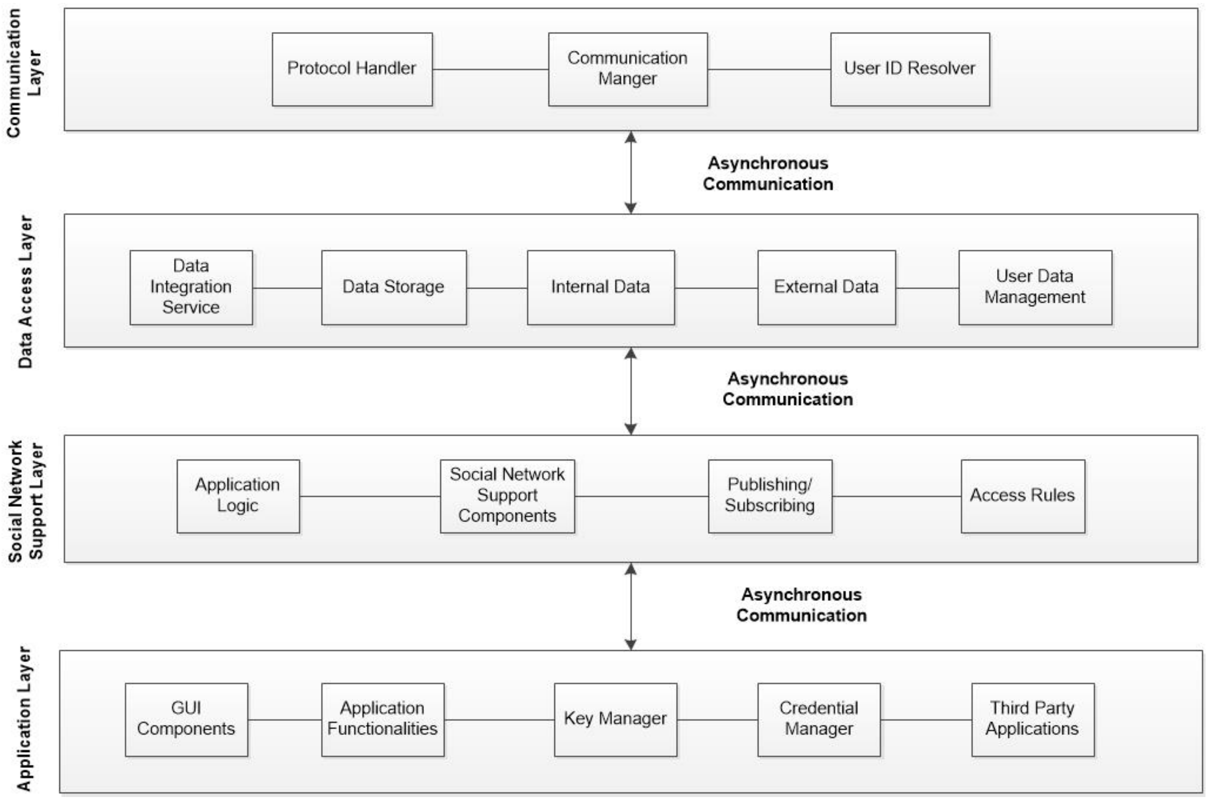


Figure 4-15:Component based conceptual DSNA

4.4.3. Application Layer of DSNA

The application layer consists of application specific components, that means the component are dependent on the specific need of the application. Therefore, these components must be decided during the requirement design process by the developer. DSNA includes the component that are considered important to build an application layer for DSNP. The application layer of DSNA consists of the following components;

- GUI Components
- Application Functionalities
- Key Manager
- Credential Manager
- Third Party Application

GUI Components are the combination of components that makes the user interface and described differently for different applications. Their main purpose is to provide a human accessible interface for navigating the application enabled with different technologies (such as semantic technologies).

It provides social network user to perform basic user interface (UI) tasks such as messaging or posting. Dadize and Rowe, (2011) and Seong et al, (2010) have given the overview of the approaches used for enabling visual interface to decentralised social network applications based on semantic technologies. The most discussed are Python, Java, PHP and JavaScript as they have rich set of libraries to support the semantic enabled applications

The implementation of the interface can be standard dynamic navigation based on data or metadata. The presentation of the interface should be in a standard social application format, for example, all most social applications allow users to post content.

- The **application functionalities** are the functional component of application layer that are initiated by the user. This gives user control of which functionalities he/she would like to decentralise and share with another user on DSNP.

- The main function of **key manager** in DSNP is authentication. It is responsible to provide authenticated communication between the components, by generating and storing the public private key or unique key pair for the message authentication in the internal components.

- In DSNP, the purpose of **credential manager** is to manage the credential of the users and that is done by storing their identity at locally situated cached in data access layer.

- In the DSNP, the **third-party application** component handles the association of the DSNP with external applications. For instance, how the external applications will interact with DSNP and at what level they have access to DSNP functions.

4.4.4. Social Network Support Layer (SNSL)

The purpose of social network support layer components is to implement the rules and protocols to determine how the data will be used across the social network. It comprises of 4 main components;

- Application Logic Component
- Social Network Support Component
- Publishing / Subscribing
- Access Rules

The role of **Application logic** provides interface between data and user interface components.

The application logic is not always same in the social networks. For example, the format of application logic in the Facebook is different than Myspace. In the Facebook, a common mechanism to submit a request to use external API (see chapter 2) services include PHP, AJAX, HTML and XML. The main difference would be the structure of the mechanism as the same request call in JavaScript may have different implementation structure than the one in PHP (Nathan et al. 2015).

The purpose of **social network support components** is to provide rules and protocols, on how social data will published and accessed over the social network.

Another important feature of SNSL is to provide middleware functionality for DSNP applications. Similar concept DOAP is proposed by pinto et al. (2003) as an architectural language to help AO based distributed applications for runtime composition of aspects. Some of feature of DOAP were found suitable and adapted to realise the concept of SNSL.

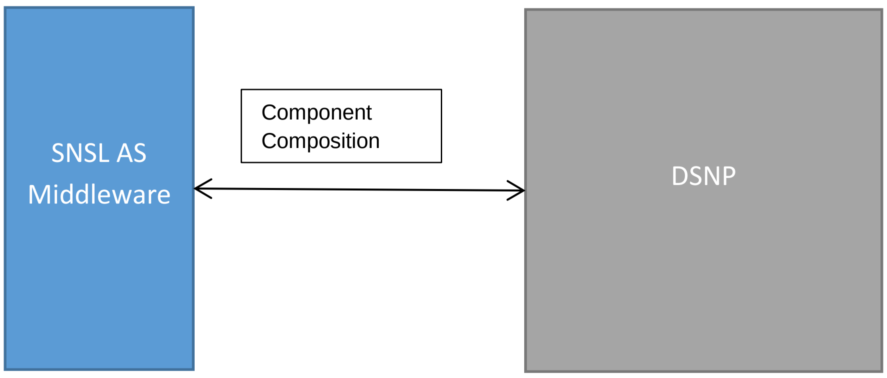


Figure 4-16: SNSL as Middleware

SNSL act as global configuration entity that performs the dynamic composition of component and aspects. Chapter 5 describes the functions and implementation of SNSL in details.

4.4.5. Data Access Layer

Data access layer is also known as graph access layer or data layer. The main purpose of this component is to provide interface to application logic to access the data sources. Moreover, this component translate data from native data model of the programming language to local level (in which the data will be stored, for example for graph based data RDFStore can be used as storage, but to store data it may need to be converted to data model of RDF. According to Tramp et al. (2012), data access layer provides resources for the description and representation of the data. In addition to this data access layer provides an abstraction on to top of storage and data integration services. It consists of the following components;

- Data Integration Service
- Data Storage
- Internal Data
- External Data
- User Data Management

Data integration service is used in the decentralised application to aggregate data of different forms coming from multiple sources. Mainly its purpose is to provide means to solve semantic and structural issues caused by heterogeneous form of data resulted from data access. By addressing this issue, it provides homogenous view of data for all the applications. After the homogenisation, the data is stored into the database.

In the decentralised application data integration service or integration of data is mainly handled at external servers. The implementation of data integration may not be very important in decentralised applications. Conceptually data is distributed in the decentralised applications and it depends on the implementation strategy of the applications.

Data Storage is the most important component of the decentralised applications. The purpose of this component is to provide persistence storage to homogenised data. According to survey done by Heitmann, (2014), RDFStore is the most used storage in the decentralised applications for graph based data and MySQL is used for relational data. The data storage is accessed through data integration service. Bizer and Schultz, (2009) have given an overview of the features and the performance of RDFStores as part of their experiment.

Data storage implementation strategy in decentralised applications is mainly based on open sources standards and protocol. RDFStore is a possible standard for storing RDF based data. Another option is use relational database to store RDF data. SQL and SPARQL can be used as query languages to explore the data.

The separation of **internal and external data** is one of the key grounding principles of DSNA. The aim is to resolve the conflicts between the data that is generated locally or internally (that means from the network where user is registered) and externally. The internal data is originated from internal components of the social network and stored internally, and external data stores the data generated for or by communicating the external users (that user of another social network).

Therefore, DSNA explicitly divides data storage between internal and external data repositories. The internal and external data is persistent in order to allow users to keep the track of their actions.

Finally, the purpose of user data management component is to provide tools to users for their data access. (Explained in detail in chapter 5).

4.4.6. Communication Layer

The purpose of this layer is to handle the communication between the users of different social network. This layer has three main components,

- Communication Manager
- Protocol Handler
- UserID Resolver

Communication layer has two main functions,

- Provide abstraction to underlying connection protocols
- Provide a mechanism for multiple connections

In order to achieve maximum flexibility, the type of data is used by underlying protocols is isolated to protocol handler component. Each protocol handler is managed by communication manager. The purpose of communication manager is to dynamically create protocol handler on the initiation communication request by the users, it also verify the authenticity of the request by verifying the identity of the user.

The use of protocol handler in DSNA is to enable multiple network communication by translating the internal events into the format understood by the associated external protocol and vice versa.

The conceptual architecture for decentralised social network and its components are explored and instantiated in detail in chapter 5.

4.5. Evaluating the DSNA

In this stage of iteration one, the architectural style DSNA is implemented using the social messaging application. The effectiveness of artefact is assessed based on the results achieved in this evaluation. The purpose is to demonstrate the successful application of the DSNA. To demonstrate the functioning DSNA, in this stage of the iteration, key elements of the architecture are developed, including Component, Aspect, Role, Properties. The main aim to achieve from this iteration is an application that can prove that the requirements R1 and R2 of decentralisation scenario are met.

4.5.1. Importance of DSNA Evaluation

Evaluation of the DSNA can provide the detail about, how useful can the DSNA be towards the decentralisation of SNP. The application of the DSNA can be helpful in making the reusability and adaptability of the SN functions to different SNPs. The assumption, if the DSNs are designed using DNSA and its services are accepted by the SNPs service providers then users should be able to choose the functions they wish to decentralise, hence creating DSNP which is customisable based on user needs. On principles, this eventually can achieve all the requirements set in SWAT scenario in section 4.1.

The decentralisation of the SN functions to build a DSNP which can allow customisation of SN functions is not in the current scope and is considered in the future work because of the huge implications. That is why evaluation of DSNA is important to prove, to what extent it can successfully achieve the desired goals. With the more success, the scope can be extended to the complex assessments.

4.5.2. Application Skeleton

The application skeleton is based on simple messaging requirements that are extended from already described scenario in section 4.1. The USER 1 BOB login to SN1, which is on server S1 and send message to ALICE on SN2 which is server S2.

This whole interaction between BOB and ALICE is designed using DSNA. The activities performed by both users are explained in the table 4-1.

User	Activity	Function	DSNA element
Alice	Security related	Login	Component.py, Aspect.py, Role.py
Bob	Security related	Login	Component.py, Aspect.py, Role.py
Alice	User Link-ability	Messaging	Component.py, Aspect.py, Role.py
Bob	User Link-ability	Messaging	Component.py, Aspect.py, Role.py

Table 4-1: Application function outlook

Figure 4-17, shows the messaging application basic design in the context of evaluation. Three main classes are created to handle the functionality requested by the users.

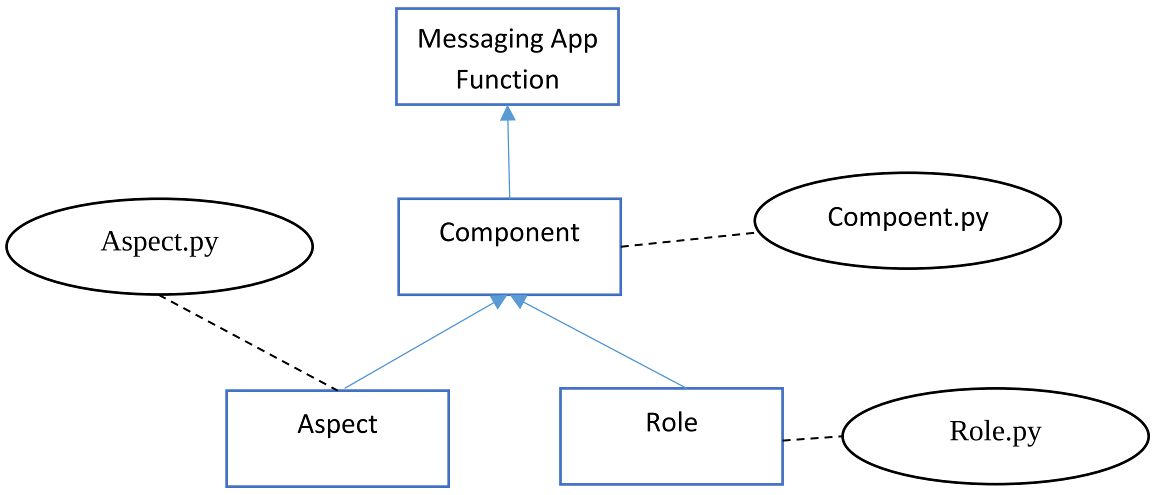


Figure 4-17: Application Code Skeleton

The table 4-2, define the roles key the elements of the DSNA style. As shown in the figure 4-17, each element is represented by a Class. Aspect.py, Component.py and Role.py are basically designed as libraries so that they can be included anywhere in code during the development stage of DSNA application. Detail is given in the next section.

DSNA components	Life Cycle	Definition
Component.py	Dynamic	Component.py represents the component Class and how it forms the functionality in the assumed DSNP. For detail, reference see Appendix 1
Aspect.py	Dynamic	Aspect.py represents all the properties for the aspect and all the functions. See Appendix 1a for reference
Role.py	Dynamic	Role.py represents all information related to the role of the functionality. See Appendix 1b for reference.

Table 4-2: Code related description of DSNA in simple application

4.5.3. Tools and Application Behaviour

The purpose of this section is to describe the tools that are used to develop the DSNA based application. In addition, application behaviour is also explained in the form of system sequence diagram.

4.5.3.1. Tools and technologies

Tools and technologies are selected on the basis of their best suitability for the paradigms (CBSD and AOSD) DSNA depends on and social network platforms. Considering the importance of AOSD in this research Java based ApsectJ (Kiczales, 2000) is deemed suitable. The reason is stability and range or support available, however, AspectJ seriously lacks in supporting SN related tools and technologies. There are various extensions of AspectJ in various popular languages that support new SN platforms.

Python is selected to develop the application, because of its open sourced and dynamic nature and support for component-based platform. There are many strategies proposed in python platform to support AO development. For example, Spring Python (Turnquist, 2010) and the list is long.

4.5.3.2. Component.py

Following the AO concepts, the Component.py has been created as main class that links all the feature of the component to aspects and reset of the DSNA. The Compoment.py represents component of the DSNA and functionality that need to be produced based on the user request. A DSNA component made of set of classes assembled and executed, to create new functionalities. Complete reference to the Component.py available at Appendix 1.

4.5.3.3. Aspect.py

Aspect.py is based on an open sourced project, AspectLib (python-aspectlib. 2016) is modified for the need of this research. The Aspect.py is general purpose class that can be initiated by simply using an import function of the python.
Property: The role of property is very important in Aspect generation. As it changes the behaviour of the functionality based on the Advice. For example, BOB would like to add some extra feature to message functionality he is using. Aspect will be able to change the request and new component will be initiated and new properties to the functions will be added.

4.5.3.4. Role.py

The purpose of the ROLE class is very important as it decides which aspects are going to be added into a specific functionality. A unique role name is assigned to identify both component and aspect classes. A specific functionality is encapsulated inside the role that can be executed by the component when it's called.

In the light of the scenario explained above, a message interaction between user BOB and ALICE will create a MessageRole (Name=Message) and new RoleInstanceMessage (Message_UserNames) and similarly if the BOB and ALICE turns the messaging to chat, then a new RoleInstance, RoleInstanceChat (Chat_UserName) and Role name ChatRole (Name=Chat) are created to differentiate between different chat or message Roles.

4.5.3.5. Application Behaviour

This section describes the behaviour of the DSNA based application at the system level. For example, in the context of the scenario, Bob initiates a functionality (It is assumed that Bob is already logged in to the DSNA based platform and have the access to the SN1 by the mean of his ID).
The request is received at the Component.py, which send the request to ComponentProducer. The CompumentProducer act as Aspect activator. Here, IdentifyAspect() is invoked. ComponentProducer want to know what features of the SN functionality can be decentralised. Similarly, IndentifyRole() is invoked to add any additional role the functionality need, to be fully functional. ComponentUpdate receive the ComponentUpdateAdvice() from Aspect with all the parameter that are required by the component that need to be updated. These parameters

transform into new component. UpdateProducer Sends the component with the UpdateReceived(). Lastly, Bob get the decentralised functionality.

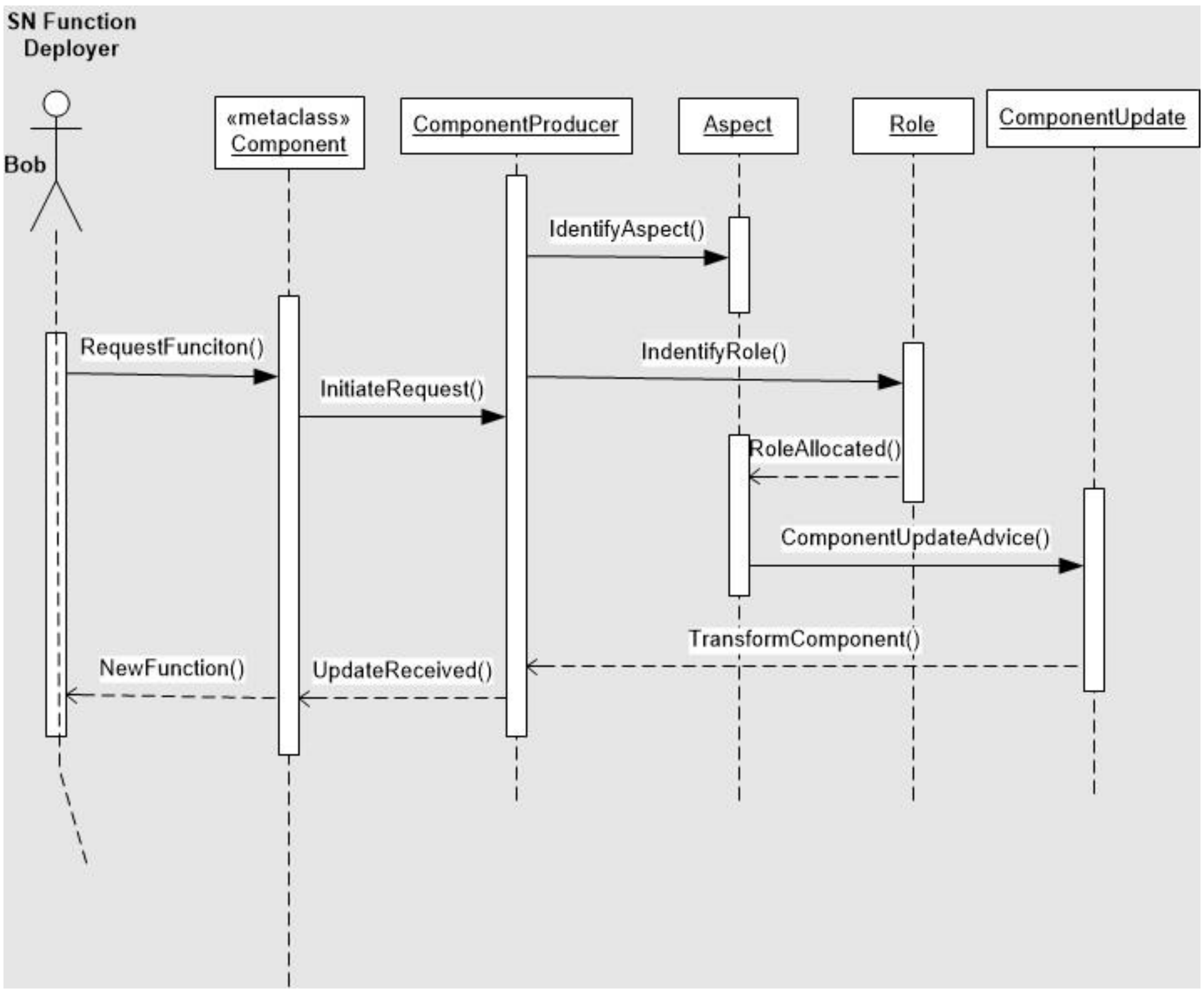


Figure 4-18: System level behaviour of DSNA based example application

4.5.4. Demonstration of the Messaging Application

To evaluate the DSNA, in light of the scenario, Bob (SN1, S1) should be able to send message to Alice (SN2, S2) to prove the R1 and R2 requirements of the decentralisation scenario. To demonstrate how DNSA based application achieve functional decentralisation. Sequence diagram is built to depict the communication pattern between the User and DSNA components.

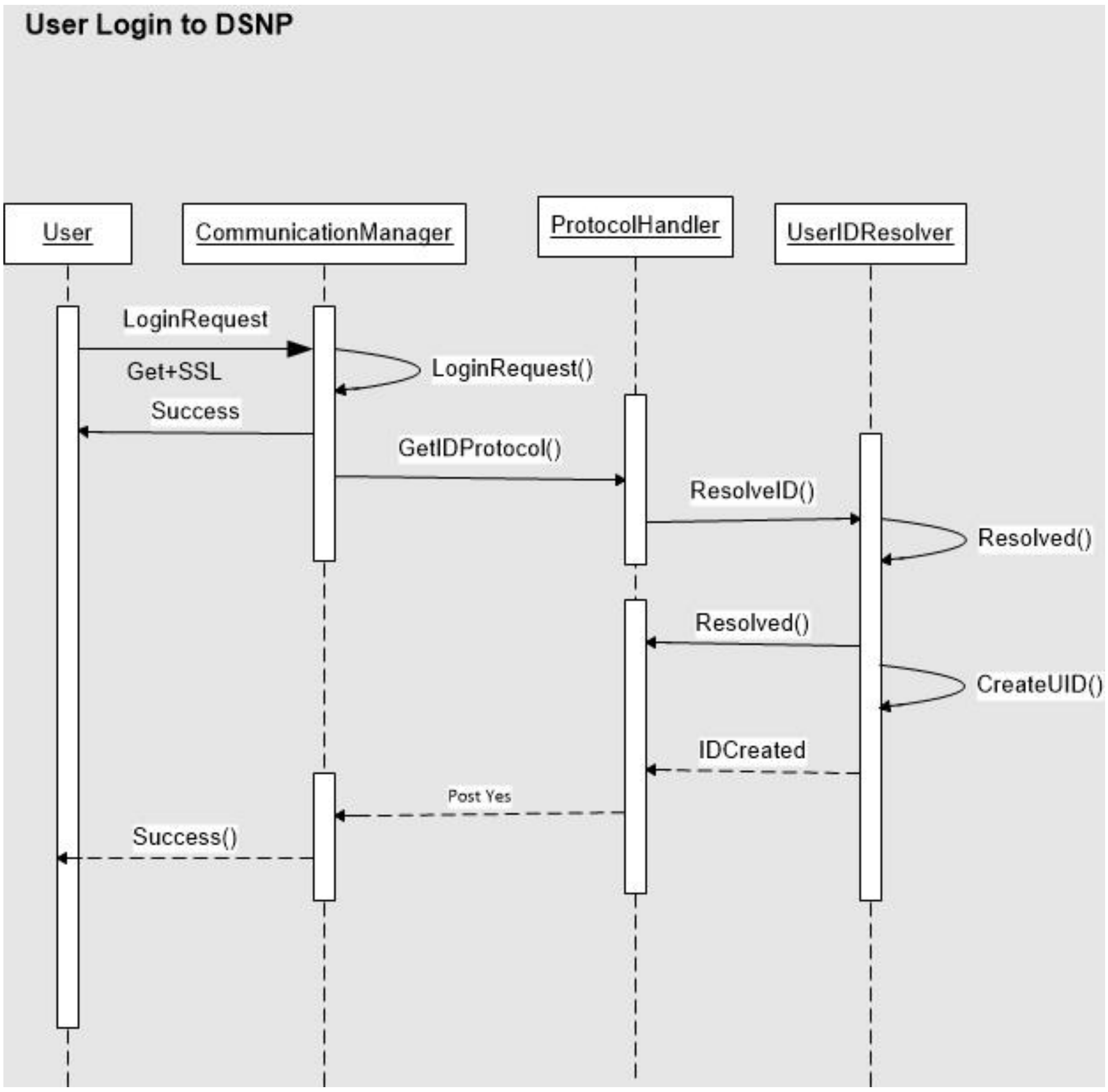


Figure 4-19: DSNA based Application Login

To enable the feature of DSNA based application there are certain precondition to make sure the design criteria of the DSNA is met, to ensure the stability.

Preconditions:

Bob is a member of SN1 and a member of DSNP (DSNA based platform)

Alice is member of SN2

Bob and Alice must have access to their accounts on SN1 and SN2 respectively.

Activities Performed by Bob

The first step to use the DSNA based platform is to login to the platform. To do various components are come into use to verify user. When Bob login to DSNP he can either make a new ID, use the same ID as of SN1 or he can use universal WebID, which can be generated by the Protocol Handler IDResolver function. Once login conditions are met the Bob can choose which functionality he wishes to use to connect to Alice.

As soon he initiates the functionality, the above-mentioned cycle initialises to decentralise his chosen functionality. In the context of the scenario, which is messaging.

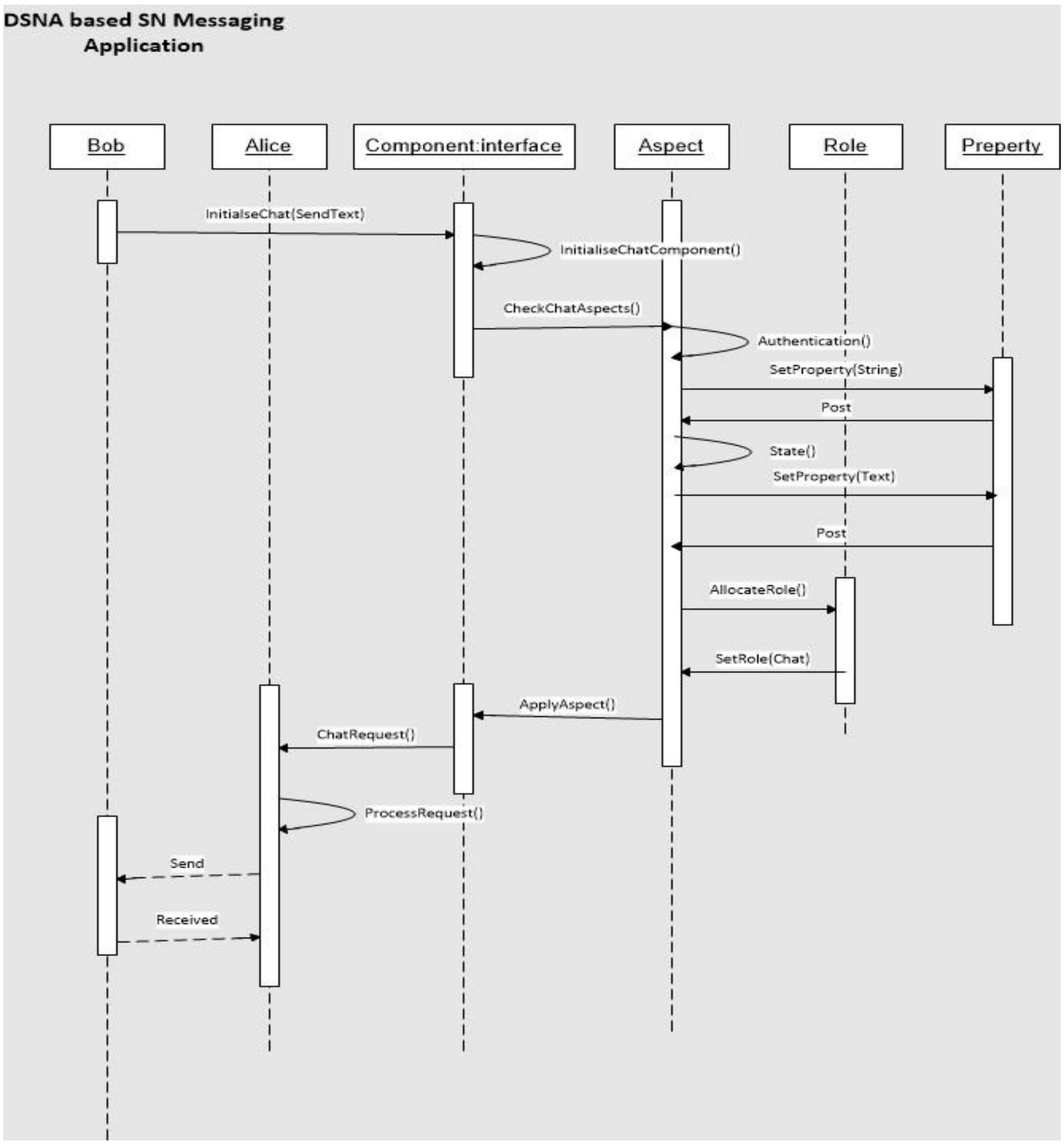


Figure 4-20: DSNA based SN Messaging Application

User	DSNA Components	Functions	Explanation
Bob -> InitialiseChat (SendText)	Component (Interface)	InitialiseChatComponent() CheckChatApsects()	InitialiseChatComponent() Creates the messaging function and CheckChatApsects() check the feature and content required by Bob request of message
	Aspect	Authentication () SetProperty(String) State () SetProperty(Text) AllocateRole() ApplyAspect()	Aspect component assess the request from the main Component and set properties and Role. AllocateRole () set the Aspect according the context of the request. ApplyAspect(), apply the changes to the component
	Role	SetRole(Chat)	In the context of the request, function role is set to chat
	Property	Set additional properties	Any additional properties are added by Property.
Alice -> ProcessRequest ()	Component (Interface)	ChatRequest ()	The new functionality is shared with Alice in the context of Bob request. Message is received at the Alice end. The component stay active Until the connection is terminated by Bob or Alice.

Table 4-3: DSNA based SN Messaging Application Implementation

DSNA is successfully implemented on the messaging functionality and thus decentralising functionality between SN1 and SN2, in the context of scenario requirement R1 and R2.

The login functionality is used to assess the security and privacy (R1), and messaging functionality is used to assess user link-ability (R2). Figure 4-19 and Table 4-3 shows how assessment is done. Appendix 1C includes the screenshots of the application. Lesson learned and challenges are explained in the next section.

4.6. Discussion

Leaning the CAM and DSNA is important to describe and clarify, how decentralisation is examined in this research and how it can be achieved at the functional level. The functional approach is selected because in the context of SNP functions and activities are considered as the lowest denominator. Hence this method can be helpful in introducing the decentralisation at the semantic level. Here semantic level refers to the data and code levels of the platform. Understanding, the fundamental guidelines and all three steps of the development stage reveals learning involved and challenges resolved to reflect the success of the proposed research. The integration of CAM and PACE remains the main challenge, and the most learning is reflected in this area. The distributed nature of CAM and PACE was fundamental towards their adoption, moreover their relevance to the social networks and distributed database principles that are the core of SNPs as well.

In the current iteration, all key components of the DSNA were created according the design principles, however the process was time-consuming. Some of the components of the DNSA involve the use of some static coding techniques for the sake of demonstration. The analysis of DSNA based application development exposes the need of dynamism for creating components. This aspect can enhance the ability of DSNP easiness and reusability.

The application developed to demonstrate the feasibility of DSNA is dependent on the tools and support for AOSD. The process of selecting the suitable language and platform was crucial for the success of the DSNA deployment. The support for dynamic composition of the components is available to some extent but the work is still ongoing, and the case is same with the composition of the aspect. These two remains the key challenges for the success of the DSNA in achieving the goal of DSNP. In the next iterations, the subject is explored further, and attempt is made to achieve the dynamic composition in DSNA based social platform and thus to achieve goal of the complete or partial portability between the SNPs.

4.7. Chapter Conclusion

In this chapter, conceptual architecture of the decentralised social network application is derived from unified approach of using component-based architecture style such as PACE and AOSD CAM (component aspect model). The proposed architecture describes the high-level structure of components and their functionality and how they can be used for developing decentralised social network platform.

In order to provide solid foundation to the perceived conceptual architecture, component-based architecture C2 and PACE were explored to provide guiding principles to DSNA. In the next stage, AOSD CAM approach is used to provide rules, constraints and relationships guidelines for the composition of DSNA architectural style. Based on DSNA style component-based architecture of decentralised application was conceived to provided set of components glued together by component, aspect, role and property elements and relationship rules. In the last stage, the key principle of architecture is evaluated by implementing them to simplest chat application.

Chapter 5 - Iteration 2

DSNP Prototype Implementation

5. Chapter Introduction

The architecture presented in chapter four is aimed to provide the guidelines and rules for component relationship and composition. A unified approach to combine the component-based style such as PACE and AOSD based CAM style have been used to obtain the mutual advantages for the design of DSNA style. The architecture is based on the DSNA style which is grounded in PACE (Component-based style) and CAM. Combining PACE and CAM provide rules and structure that are required to build a decentralised social network platform. This chapter describes the prototype implementation of DSNA proposed in chapter four. The prototype is the result of generic scenario, that have been used to check the practicality of the DSNA.

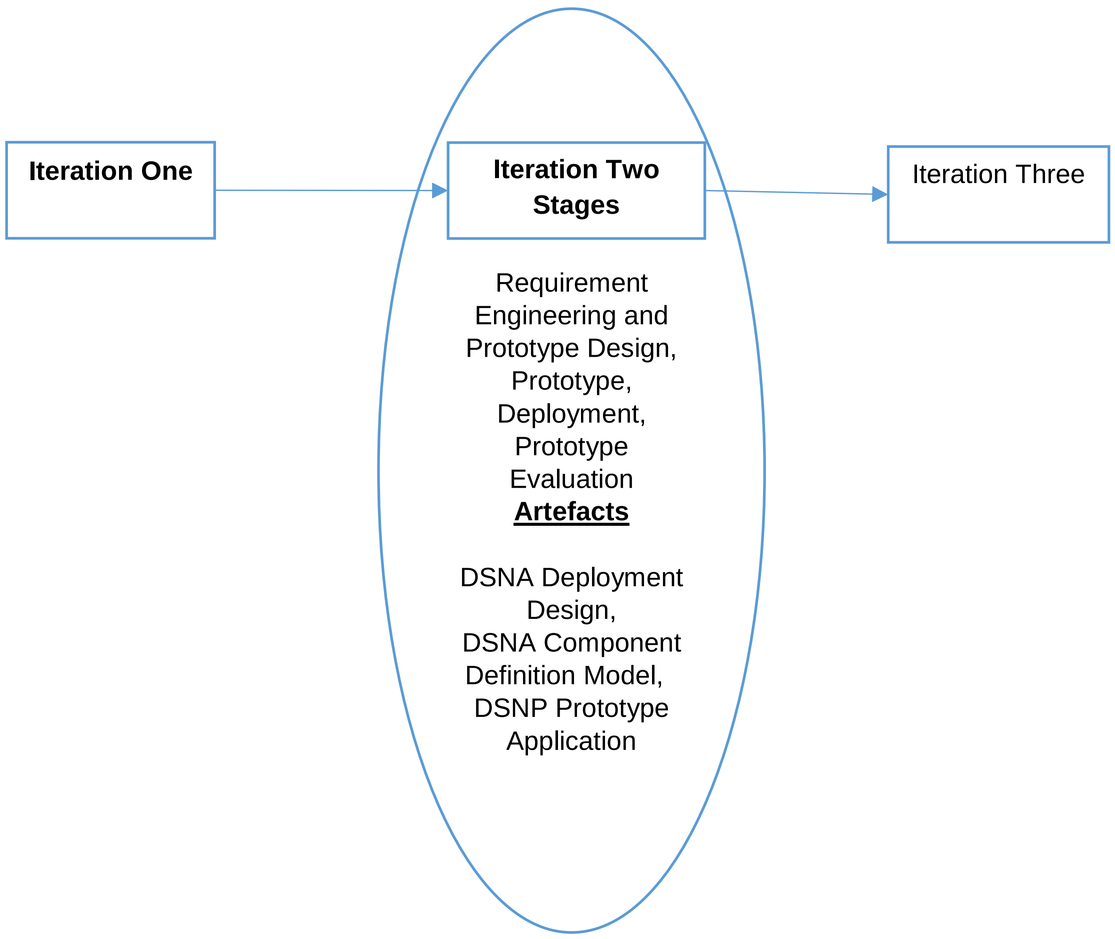

Iteration One
Iteration Two Stages
Iteration Three
Requirement Engineering and Prototype Design, Prototype, Deployment, Prototype Evaluation
Artefacts
DSNA Deployment Design, DSNA Component Definition Model, DSNP Prototype Application

Figure 5-1: Iteration Two Structure

To facilitate the possibility and practicality of pursuing DSNA Style based approach for social network decentralisation, in iteration two an attempt is made to focus more on social network functions. The iteration one focused more on the architecture design and definition of all the possible elements of the architecture. The lessons learned from iteration one, are applied in iteration two, for more evolved version of DSNA.

In iteration one, the proposed architecture and DSNA style is demonstrated in the messaging function. The goal was to describe fundamental components of DSNA and to demonstrate their implementation in the form of simple SN functionality. The extended version of the DSNA in iteration two, attempts towards the core part of implementation, which is the composition of components and aspects. Demonstration of how DSNA component composition enhances the portability between SNPs through the mean of DSNP is crucial towards achieving the main goal of the iteration two.

Iteration two aims to provide more refined version of DSNA by implementing DSNA on social networking platforms. Iteration two achieve objective 4 by producing the prescribed implementation framework. The dynamic component and aspect composition are central to the refined architecture and is handled at the middleware level. The description of SNSL (Social Network Support Layer) and the handling of component composition by the SNSL is a key feature of the evolved version of DSNA. At the final stage, the social messaging application prototype is built to evaluate the functioning of application.

5.1. Prototype Design

In the previous chapter, this research explored the possibility of using existing approaches to design the Decentralised Social Networking Architecture (DSNA). The successful combination of component-based architecture PACE and AOSD based CAM have produced suitable set of rules and components needed to build the DSNP (Decentralised Social Networking Platform). The result of this unified effort was DSNA style and architectural framework based on DSNA style. In the next stage, DSNA style component-based architecture of decentralised application was conceived to provide a set of components glued together by component, aspect, role and property elements and relationship rules.

The outcome of chapter four is used to refine the architecture. A prototype is developed to implement the evolved architecture to present the refined version. Since social networks are distributed in nature, therefore a possible implementation of SWAT scenario can be used in the distributed enterprise.

An important implication of the proposed architecture can be related todistributed enterprise. Large organisations build and maintain multipurpose systems to manage their various types of large amount of data that is used by various type of workers. The steady shift of organisational landscape from centralised to distributed has given the organisation opportunities to take the combine benefits of mass collaboration and scalability by using decentralised networking. One of the challenges that remains and most discussed in building such a collaborative platform is, how to ensure the consistent availability of the content and on different peers situated on different domains (Skaf et al. 2008)

5.1.1. Design Challenges

The shift from monolithic platforms to distributed platforms is suitable for a decentralised approach that can take the distributed nature of user profile with their preferences from multiple domains into consideration. Thus, the proposed social platform prototype base on DSNA need to provide support for an open environment where users from different social platforms can interoperate across their respective platforms, by addressing the four main challenges;

5.1.1.1. Interaction

Aggregation, integration and resolution of the user profile data produced by the interaction between SNPs for the desired platform

A thorough consideration is taken in the design of DSNA to address these challenges. For example, to address the first challenge, according to Tams et al. (2011) data aggregation, integration and resolution of different social platforms requires efficient data synchronisation tools, to make relevant retrieval of content. DSNA provides 'USER ID RESOLVER' 'PROTOCOL HANDLER' and 'COMMUNICATION MANAGER' components for content or event exchange in a decentralised environment (described in chapter 4, section 4.4.).

5.1.1.2. Communication

Providing consistent form of communication to the users to make their interactions feasible and possible.

The 'Pull' approach is the most common approach used in the client/server environments. The communication model is made of request from an active client and response from the passive server. 'Polling' mechanism is related to pull approach that relies on clients, continuously sampling the server status through repetitive requests. Polling has its issues, such as scalability and reliability as far as the interactions between client and server are concerns. To overcome these issues 'Long Polling' was introduced, which supports asynchronous delivery of events with better performance and saleability. Long Polling is based on request/response model in which the server keep the request open until the response is generated or set timeout limit is reached. As far as the Push approach is concern, it uses passive client that is actively kept informed on subscribing to the server, about any occurrence of event (Griffin & Flanagan, 2010).

DSNA uses asynchronous model for the communication between the components and events. For that purpose, Publish/Subscribe (PubSub) component has been introduced in the architecture (see chapter 4 section 4.8) to handle the interaction. PubSub is an interaction paradigm that uses push model. It uses agents to subscribe to a specific event such as profile updates and receive asynchronous notifications from the publisher whenever the repetition of event occurs. The benefits of PubSub over Pull approach lie in the optimisation of the number of requests and synchronisation between the publisher and subscriber (Eugster et al. 2003).

For the proposed DSNA prototype, publish/subscribe interaction model is adopted, as it supports better decoupling between the distributed parts of the platform, which is important for successful implementation of decentralised application.

5.1.1.3. Composition of Components

Providing a mechanism to convert user interaction into a form of the component(s).

The composition of distributed components itself is a complex task and suffers complex interactions between the other components within the architecture and middleware. From the application developer perspective, the complexity consequences into intense focus on the programming APIs and middleware components. The component repositories are considered as solution to the complexity problem and to some extent, tackle the problem by composing and configuring the components. But at a certain level, the issue tackling the complex interaction arises again. This problem is considered common within complex distributed system as mentioned in (Piessens. 2009, Surajbali et al. 2014).

In the DSNA, SNSL (Social Network Support Layer) is functioning as middleware to handle only the component communication and interaction composition. In DNSA, to solve the complexity issue dynamic composition of the component has been introduced.

The concept of using middleware for dynamic composing and reconfiguring the component is promising but still underdeveloped. The main purpose of using such to technology is to solve two problems with the decentralisation of social network, encapsulating the independent functionalities into aspects and then using the weaving function to transform that into a system behaviour.

The DSNA uses already available software technologies for the composition purposes as some mentioned in section 4.5.3. The core part SNSL is component composition and the key feature of the prototype deployment phase.

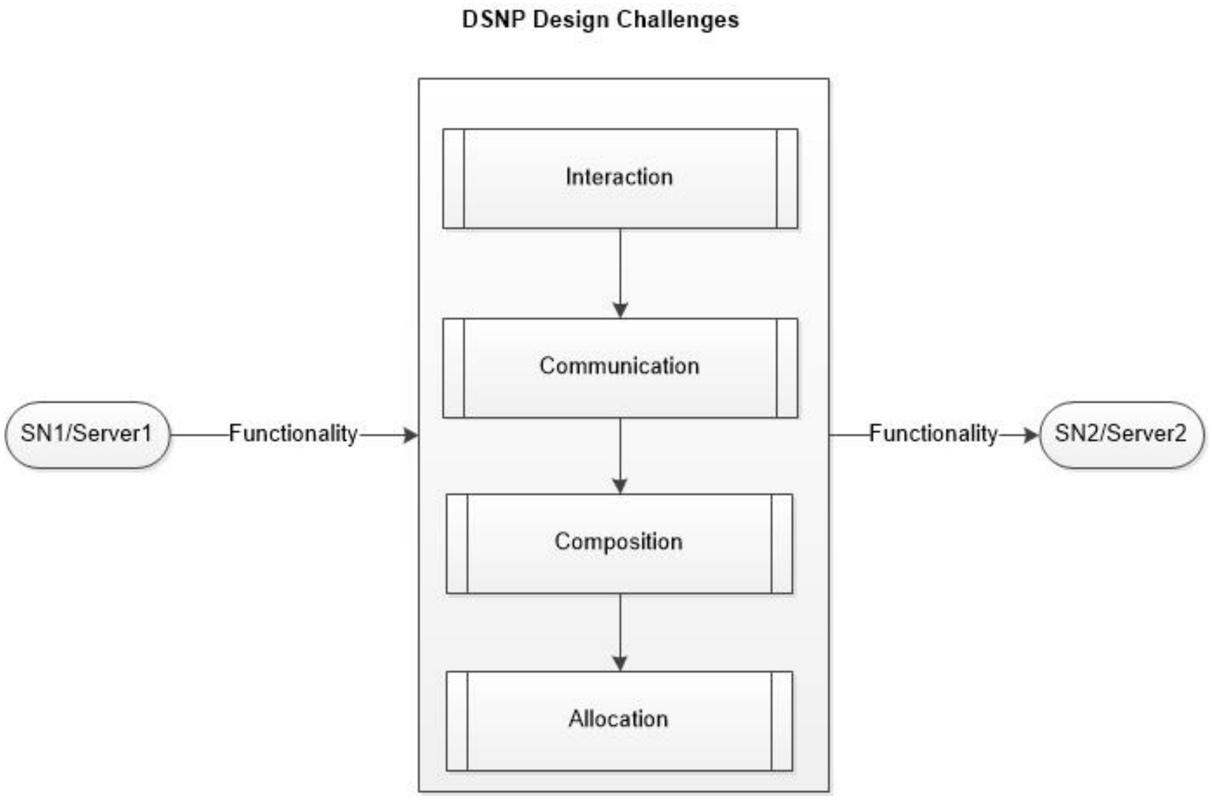


Figure 5-2: Prototype Design Challenges

5.1.1.4. Allocation

Allocation of roles, relationships and aspects to the components towards the creation of decentralised functions.

The allocation is a part of the component composition. Composition allows the decentralised social network to add or remove functionality as initiated by users. Allocation guide the composition, where to be sent, executed based on the rules and guidelines described in DSNA style.

To overcome these of challenges the proposed platform should fulfil the certain requirements described in the next section 5.1.

5.1.2. Analysis of the Requirements

How the above challenges are dealt with, is demonstrated in the social data sharing application. The prototype is based on SWAT scenario (section 4.1) and is required to fulfil requirements R3 and R4 of Data Portability and R5 of Profile Reusability. To demonstrate specific functionality, the SWAT scenario is modified to a use case.

5.1.2.1. Prototype Use Case

In order to introduce the prototype in the perspective of the social networking functions, this section presents a use case as an extension of the SWAT scenario. The use case presents requirements for the architecture implementation.

The use case is implemented on the supposed social networking environment. As mentioned in the scenario, in section 4.1 but in the use case the users of the platform are now connected to DSNP and attempting to share their functionality through the mean of DSNP. What is meant by functionality and its relationship to the component and aspect is explained in section 4.4.

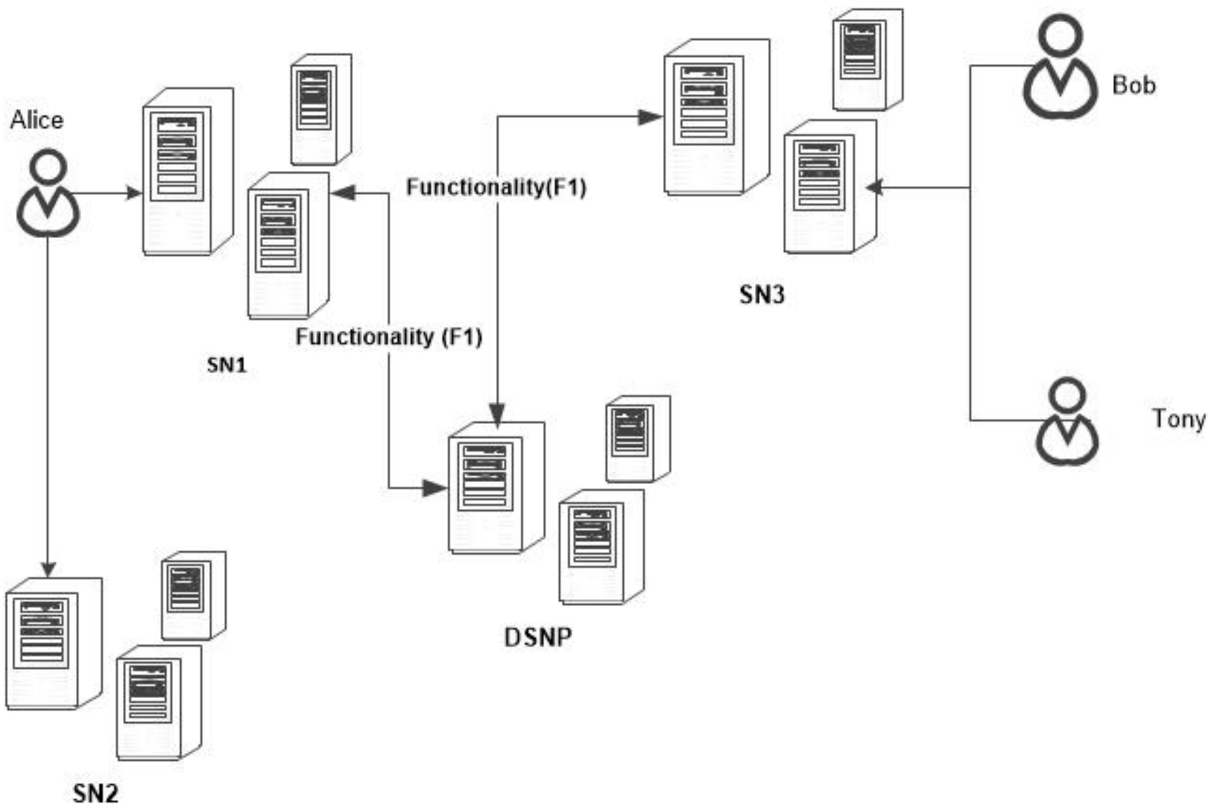


Figure 5-3: Extended SWAT scenario-based use case

5.1.2.2. Common Setting of the Prototype Design

The requirements and above-mentioned challenges are associated to the main goal of the prototype design. In the context of design, the purpose of the common setting is to illustrate the composition of a component in the design process of the prototype. Common setting describes how DSNA style must be implemented during the design of each component. For example, in the figure 5-4 the Communication Manager component design is shown that consists of a style and component application structure.

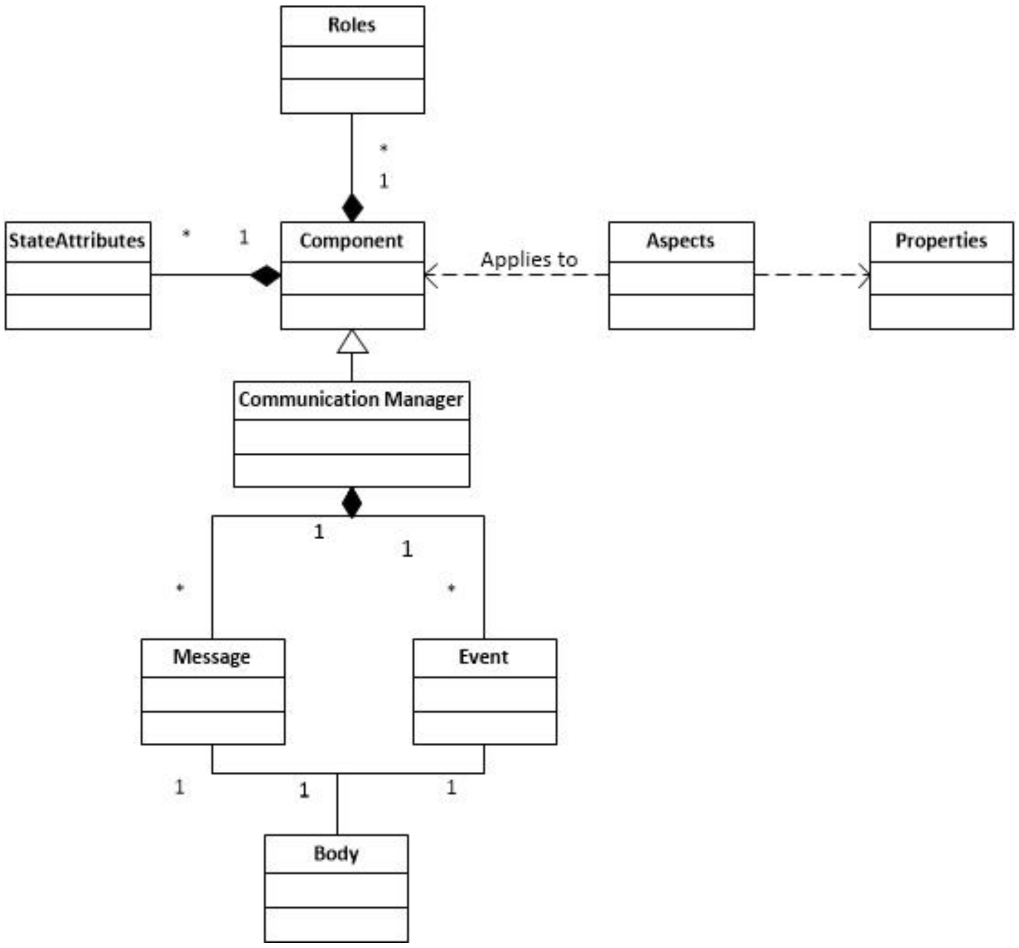


Figure 5-4: Common setting of the DSNA components for prototype DSNP

In the context of use case, Alice makes a content sharing request. For instance, let's say the process starts from the Communication Manager' which is a 'Component' and one of the 'Aspects' of Communication Manager is to 'Authenticate' Tony. 'Aspect' now has a Role 'Authentication' that it needs to 'Fulfil', as a consequence, the 'Property' is assigned to the 'Aspect', that contain values required by the user to validate the identity. 'StateAttribute' may or may not be assigned to the component which is mainly depend on the requirements of the component. Figure 5-4 shows the implementation perspective of each component mentioned in DSNA.

5.1.2.3. Separation of Data and Design Process

One of the most important aspect of DSNA is the separation of internal and external data, with the aim to avoid the conflicts and duplicity in data. In this context, users are directly linked to their data. Therefore, user are separate based on their data, there are two types of users who can access proposed prototype, internal user and external user. The internal users are the registered users of the DSNP and their data is stored in internal data storage. The external users are the users that are not the registered users of the DSNP but interested in using DSNP services and their data is stored in external data storage. An external user can be a friend of an internal user on another SNP than DSNP.

For example, a User (Internal User) registered with the prototype application (DSNP) and Friends (External Users) registered with other social networking platforms (SN2, which is FACEBOOK or SN1 GOOGLE CIRCLE). To make the clear distinction between the users and their description in the proposed DSNP figure 5-5 shows that a user can be internal or external or friends and the purpose of this distinction is to keep external data which belongs to external user or friends registered to (SN1 or SN2) separate.

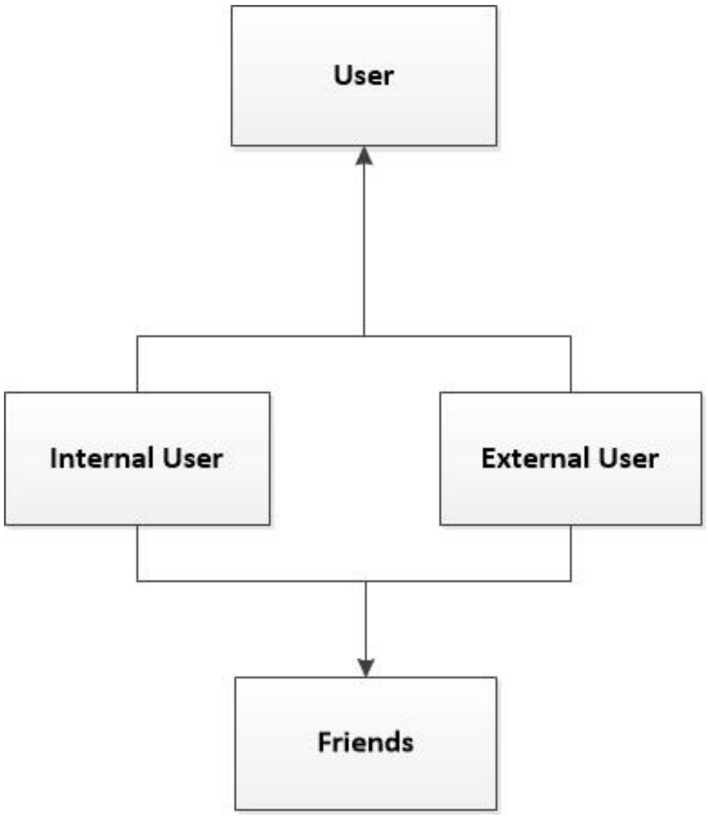


Figure 5-5: Definition of User in the Proposed DSNP

To further describe the use of the components in the DSNA based prototype Figure 5-6, describes the design process of content sharing functionality to demonstrate the design of the content sharing application in the DSNP.

The following are the preconditions according the use case;

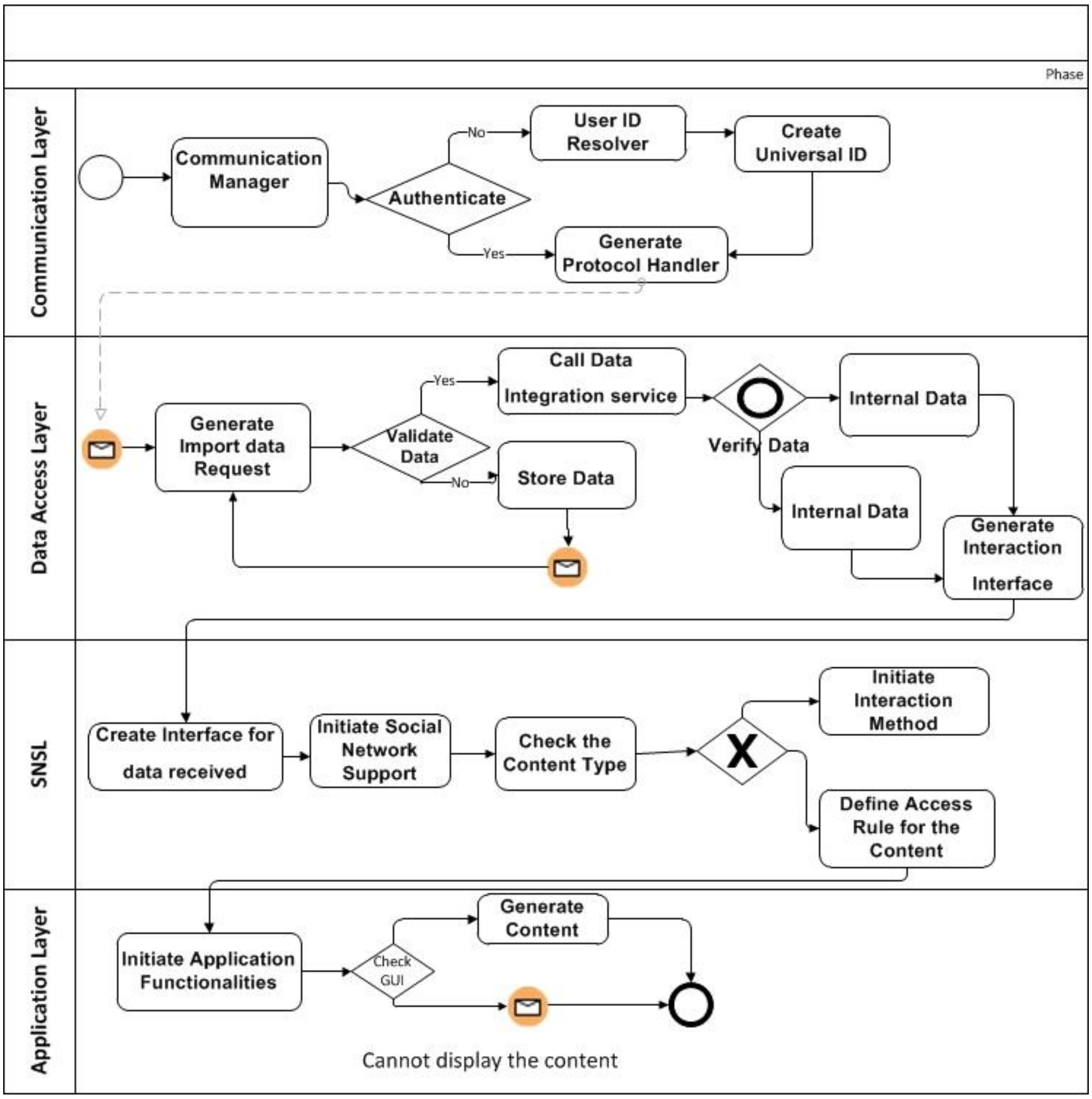
- Alice has a DSNP account
- Bob is a user of different social network
- Alice has a content in electronic form to share.

In the current example, the external user initiates the request. The communication layer handles the interaction initiated by the external user and the application layer handles the interaction initiated by the internal user (the user of DSNP). In the given process, only external interaction is investigated.

Alice is registered with SN1 and wants to share her content with her friends on other social networking sites. To achieve this, she can either install/enable DSNP API to her profile in the Facebook (out of the scope of this research) or she can join DSNP and start sharing content with other sites. Alice initiates content sharing request.

In the DSNP communication layer ‘Communication Manager’ analyse what kind of protocol is needed to handle the request and authenticate the user identification, i.e. Alice has a valid account or not. If the account is valid then the communication manager creates the ‘Protocol Handler’, which dynamically accesses and creates the protocols needed to communicate with the network that the user belongs to. If the user is not authenticated by the ‘Communication Manager’ then the request is sent to ‘User ID Resolver’ that analyse the request and create universal user ID (like Web ID or Open ID) when authentication is completed, ‘Protocol Handler’ is generated. This is how communication layer provides mechanism for multiple connections to DSNP.

In the next step, the request is sent to ‘Data Access Layer’. Frist, it generates the ‘Import Data Request’. At this point preferences are generated on basis of data, to be imported from the other SNP (for example Profile data or content data). After the validation of data, the ‘Data Integration Service’ is called.



Phase
Communication Layer
Communication Manager
Authenticate
No
Yes
User ID Resolver
Create Universal ID
Generate Protocol Handler
Data Access Layer
Generate Import data Request
Validate Data
Yes
No
Call Data Integration service
Store Data
Verify Data
Internal Data
Internal Data
Generate Interaction Interface
SNSL
Create Interface for data received
Initiate Social Network Support
Check the Content Type
Initiate Interaction Method
Define Access Rule for the Content
Application Layer
Initiate Application Functionalities
Check GUI
Generate Content
Cannot display the content

Figure 5-6: Design Process of simple form of content sharing in DSNP

Data integration service is used in DSNP to aggregate data of different forms coming from single or multiple sources. Mainly its purpose is to provide a means to solve semantic and structural issues caused by heterogeneous form of data resulting from data import. Integration service decides how the data will be categorised, for example as internal data or external data.

If the data is not valid to be processed, then any information regarding that data is stored and a message is sent to the user to generate valid import requests. After the data is homogenised, data store component generates the request for the data that is to be used for the interaction with the interface. At this point the social network support layer, Application Logic component creates the necessary support, which include rules, web standards and protocols, required to generate user interface and data interaction possible.

In the next step ‘Social Network Support Components’ initiate the support required to generate the content desired by user. After knowing the content type, method for interaction is initiated (Pub/Sub component) and how the content will be accessed (access rules are set).

In the last stage, when content type is being check in the social network support layer at the same time ‘Application Functionalities’ component is initiated which generate the script that is required to display the content. At this point, the content is tested whether DSNP can display the content or not. If yes, then content is displayed to its location otherwise message is sent to the user that the content cannot be displayed, and process of content sharing ends with this message.

The purpose of the explained application design process is to depict, how the components are deployed in the DSNA based application.

Considering research design process, under the guideline of iteration 2, the next stage further investigates the design in the context of deployment.

5.2. Prototype Deployment

As part of iteration 2, this stage follows the research process as defined and the concept of DSNA progresses further towards the implementation. In the context of application developer, the design stage described the process of designing each component of the DSNA. The design scenario is used to facilitate the design process. In light of design challenges and requirements, the deployment stage attempts to solidify the DSNA by demonstrating the process of deployment in DSNA style-based development.

During deployment, the main problem which is solved is regarding the connectivity of the DSNA components in the DSNP with another SNP. The focus is on the composition of the component and the elements that connect them together and with another SNP. For example, when Alice attempts to decentralise her social networking functionality using DNSP, how components making this possible are deployed and the requirements needed to connect the other SNPs. In the context of data portability requirements (R3 and R4), which is regarding the user data, this facet of the DSNA deployment is very important.

5.2.1. Deployment Levels of Component and Aspects Definition in the DSNA Prototype

Allocation of the role to components and aspects and differentiation based on roles in the application is an issue that must be handled during the implementation of any CAM based architecture. Role allocation in DSNA is done by defining the components and aspects at higher level of abstraction to show the architectural pattern.

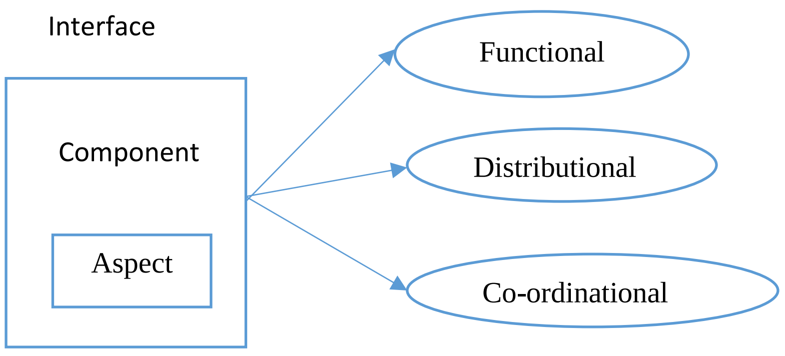


Figure 5-7: Definition levels of Component and Aspects

The figure 5-7 depicts a higher-level view of how DSNA components communicate with other components. The interface is a part of the DSNA that is used to interpret the services that are required to be invoked or published by the components and aspects. Defining them (component and aspect) is an integral part of deployment during the DSNA based application development.

To configure the components and aspects on the CAM based model and define them as shown in the figure 5-7, AOADL (Aspect Oriented Architectural Description Language) (CAOSD, 2017) (Pérez et al.2006) rules are used. AOADL is used to define component and aspect at the three level, functional, distributional and co-ordinational.

In Pérez et al. (2006) model, they divided an aspect into three levels as per requirement of their research. In contrast, the DSNA defines the component and aspect in three levels as described in the figure 5-7. Defining the component and aspect at the functional level is based on the core objective of this research.

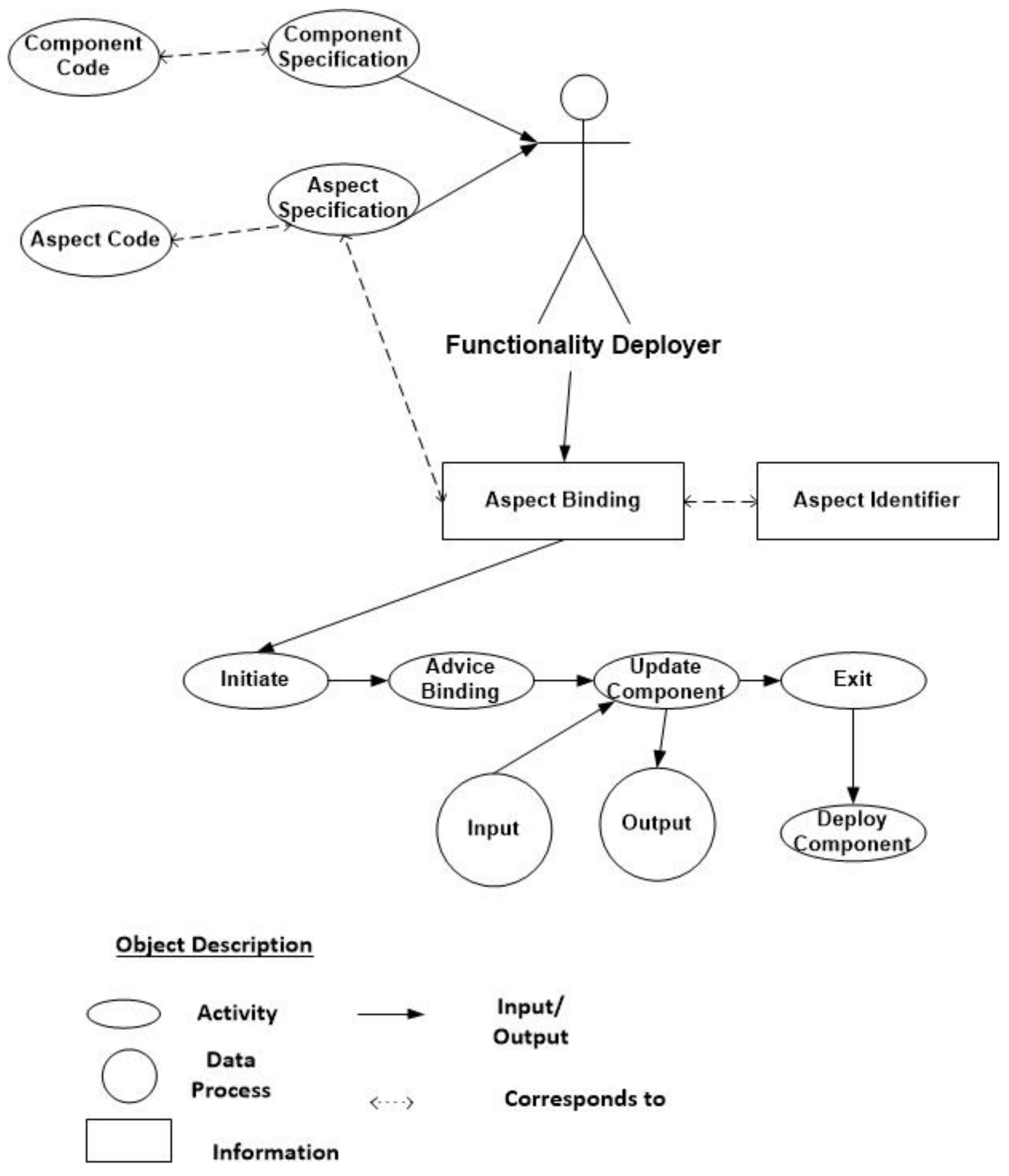


Figure 5-8: Deployment Flow of the Prototype

The main benefit of defining DSNA components at these levels, is enhancement of the component ability to define and structure the behaviour of the specific concern (Concern refers to the changes/attributes/properties required to change the functionality based on the SN user request).

The figure 5-8 gives an overview of the deployment model by showing the sequence of steps that are performed to deploy, aspect, which update the component. The "Functionality

Deployer" can be a user or SNSL. The SNSL middleware uses API from application logic to deploy the aspects and components. Based on the specification or preferences that are attached either by the users / SN or generated by the SNSL, which initiates the "Aspect Binding". The component to aspect interaction is managed by aspect binding.

The "Aspect Identifier" is used to uniquely identify the aspect based on semantics, the capability, functionality and feature that interest the users of the functionality. The binding initiates the process of weaving, that updates the component based on advice, and the "Input Data" which is an old data and "Output Data" which is a new updated data, changes are made to components. The waving ends at the "Exist" advice and new updated component is executed.

The process shown in figure 5-8, summarises the aspect and component deployment process which described in the next section. By defining the aspects and components at these levels helps them to support the concurrent adaptations, which enhances the chance of the aspect to be executed at the right point.

5.2.1.1. Functional

The functional definition of the component and aspect refer to the functional properties and behaviour of the functionality, generated by the DSNP for the user. These properties ascribed to the component interface and the semantics of the interface are defined by the functional definition of the aspect. A functional component and aspects are defined using AOADL Eclipse based plugin, which provides the architectural knowledge regarding the semantics of the content sharing interface.

Appendix 2a contains the snippets of the execution of content sharing functionality using AOADL under guidance of DSNA.

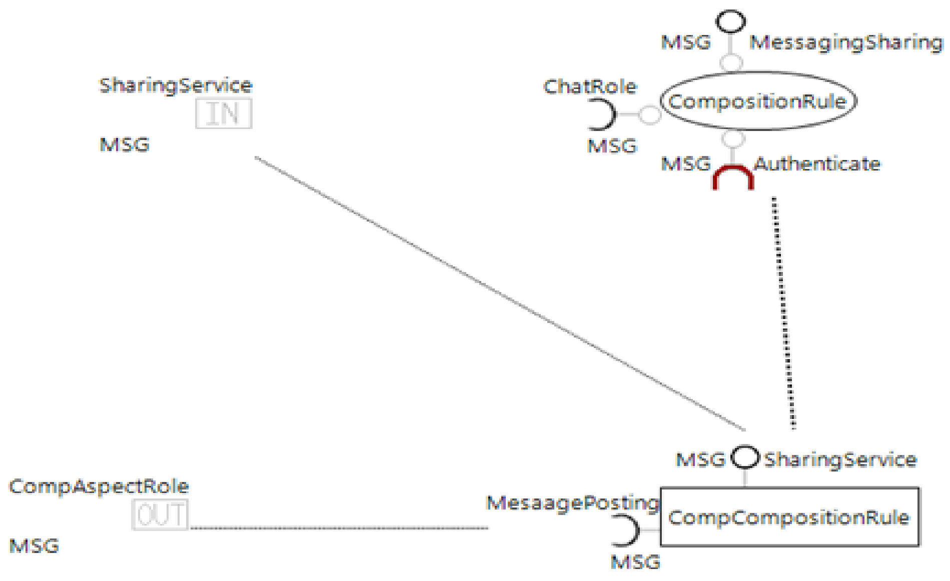


Figure 5-9: DSNA Messaging component sharing design in AOADL

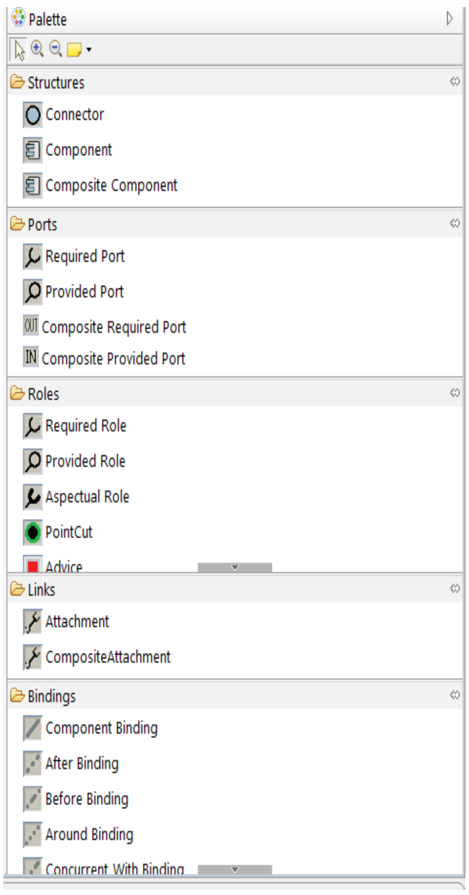
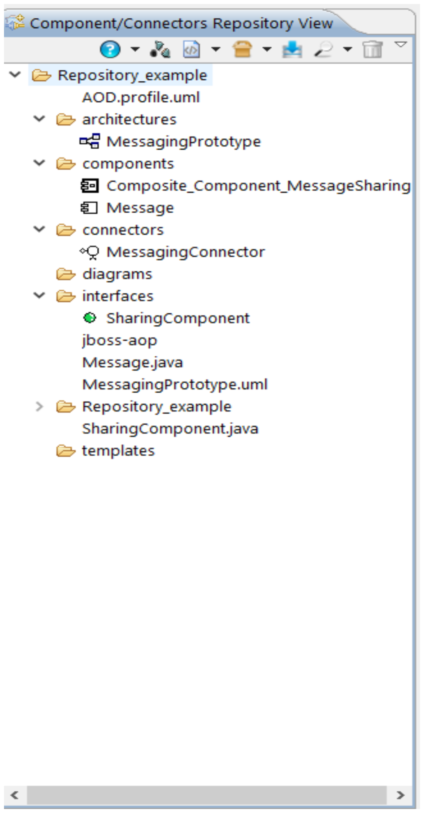


Figure 5-10: AOADL Legends description



```
Component/Connectors Repository View
Repository_example
    AOD.profile.uml
    architectures
        MessagingPrototype
    components
        Composite_Component_MessageSharing
        Message
    connectors
        MessagingConnector
    diagrams
    interfaces
        SharingComponent
    jboss-aop
    Message.java
    MessagingPrototype.uml
    Repository_example
    SharingComponent.java
    templates
```

Figure 5-11: DSNA messaging component and connector

The comparison between the figure 5-8 and 5-9 shows how the DSNA components can be deployed to achieve a certain social network functionality. Figures 5-9,10,11 demonstrate the use of CAM based AOADL to design and deploy the DSNA based messaging functionality. This method also helps to assess the application of DSNA to create a certain decentralised functionality.

For example, figure 5-12 is a snippet from the content sharing functionality deployment. The figure demonstrates the connector component “Messaging Connector”. The connector components are used in the DSNA to link the Aspect and components. It enables communication between the “Message” component and Aspect having Role “ChatRole” by the mean of the “SharingComponent” interface. After the allocation of the Role, there are two stages, “Initiate” and “Exit” for aspect to be woven into new functionality. “Aspectual Binding” is used to bind the attributes specified for the aspect and Input and Output data stores the changes. For instance, Messaging functionality is required to be decentralised to start content sharing between SNPs. When the process Initiated, the aspect binding call for Role Authenticate, which collect information as specified in the aspect attributed and also communicate with other components if required.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<connector xmlns="http://caosd.lcc.uma.es/AO-ADL/AO-ADLSchema"
name="MessagingConnector" type="Connector">
      <provided_role roleName="MessagingSharing"
role_specification="//interface[@name='SharingComponent']"
type="MSG" minOccurs="1" maxOccurs="1"/>
      <required_role roleName="ChatRole"
role_specification="//interface[@name='SharingComponent']"
type="MSG" minOccurs="1" maxOccurs="1"/>
      <componentBindings>
      <binding name="MessageBinding">

      <source>//provided_role[@name='MessagingSharing']</source>
            <target>//required_role[@name='ChatRole']</target>
      </binding>
</componentBindings>
<description>Triggers messaging functionality</description>
<aspectual_role roleName="Authenticate"
role_specification="//interface[@name='SharingComponent']"
type="MSG" minOccurs="1" maxOccurs="1"/>
      <aspectualBindings>
            <aspectual_binding name="StartMessageSharing">
                  <pointcut_specification>

      <pointcut>(//provided_role[@name='MessagingSharing']) and
(//operation[@name='Initiate'])</pointcut>
                  </pointcut_specification>
                  <binding operator="after" order="first">
                        <aspectual_component
aspectual_role_name="Authenticate">
                              <advice label="Initiate">
                                    <attachment>
                                          <argument_binding
target="UserID [String]"/>
                                          <argument_binding
target="ContentType [Array]"/>
                                          <argument_binding
target="DestinationID [String]"/>
                                          <argument_binding
target="Token [String]"/>
                                          <argument_binding
target="String [returnType]"/>
                                    </attachment>
                              </advice>
                        </aspectual_component>
                  </binding>
            </aspectual_binding>
            <aspectual_binding name="EndMessageSharing">
                  <pointcut_specification>

      <pointcut>(//provided_role[@name='MessagingSharing']) and
(//operation[@name='Exit'])</pointcut>
                  </pointcut_specification>
                  <binding operator="after" order="last">
                        <aspectual_component
aspectual_role_name="Authenticate">
                              <advice label="Exit">
```

Figure 5-12: AOADL notations for the Message Sharing

5.2.1.2. Distributional

The distributional definition is applied in a same way as functional. However, when aspects are required to connect other aspects and components, this level specifies the location of the components or instances. For example, attribute URL can be added in the DSNA component and aspect, which can be used if the component or aspects are distributed at different location. In light of DSNA scenario distributional level is not applicable but it is used to keep the track of components and aspects, in the case of specific invocation of their instance is required.

5.2.1.3. Co-ordinational

The co-ordinational is most important deification level in the DSNA deployment because its purpose is to do synchronisation of the data between the architectural components. That includes components, aspects and their relevant connectors.

In the current scenario, in which the content sharing has to be achieved between different SNPs through the mean of DNSP. The Messaging functionality is used as it is associated with all SNPs functionalities. For Data synchronisation, in DSNA “PersistenceService” instance is adopted from the Pinto et al. (2005) model. This service is deployed with all the components and handled by SNSL. It stores components and aspect data states and ensure the consistency when components and aspects are deployed. For reference, check the Appendix 2a for UML deployment model and XML deployment Model.

For the basics on the components and connector see the chapter 2 section 2.1 and 4 section 4.2.

The main advantage of using the AOADL for DNSA prototype deployment is its ability to equip application developer to manage component and aspect architecture. AOADL enhance the accuracy and help to test and deploy application logical and physical interface. Appendix 2a provides the snippets of DSNA prototype deployment.

This section has given the detail account of DSNA based application deployment. To assess the practically and applicability of the DNSA prototype the next section evaluates the prototype by implementing the scenario explained in section 5.2 of the requirement design.

5.3. Prototype Evaluation

In the next step of the proposed architecture evolution, the prototype is developed. As a part of evolving research process, in the iteration 2, the artefact is evolved to solve more complex challenges towards the implementation of DSNA. In chapter 4, the DSNA is assessed by developing simple messaging functionality. The implementation prototype concerns more about solving the core issue by addressing the requirements stated in 4.1 and in above requirement design. The figure 5-2 depict a very high-level design.

In the deployment stage of the iteration, as a part of the messaging functionality, a foundation is prepared by demonstrating the connection between the component and aspect. Furthermore, how aspect deploy the concerns(specification) and change the component to new component to have new functionality.

In this stage, the deployment is converted into an executable application to demonstrate data portability between different social networking platforms. The application of the DSNA is same as done in iteration 1 however the extended scenario-based use case required some changes. These changes are reflected in the prototype skeleton in the next section.

The objective achieved in this stage is the stage 1 of component composition (in the form of adapter) and aspect and thus DSNA prototype artefact is produced.

5.3.1. Prototype Skeleton

The purpose of the prototype is to demonstrate the how a messaging (which is content sharing) functionality can be imported to DSNP and then by the mean of DNSA and SNSL is used for another social networking. One of the features of SNSL is “Adapter”, introduced which facilitate the portability of the functionality.

As per SWAT scenario extended use case, Alice wants to send message to Bob. She is already using DNSP and she wants to use message sharing functionality.

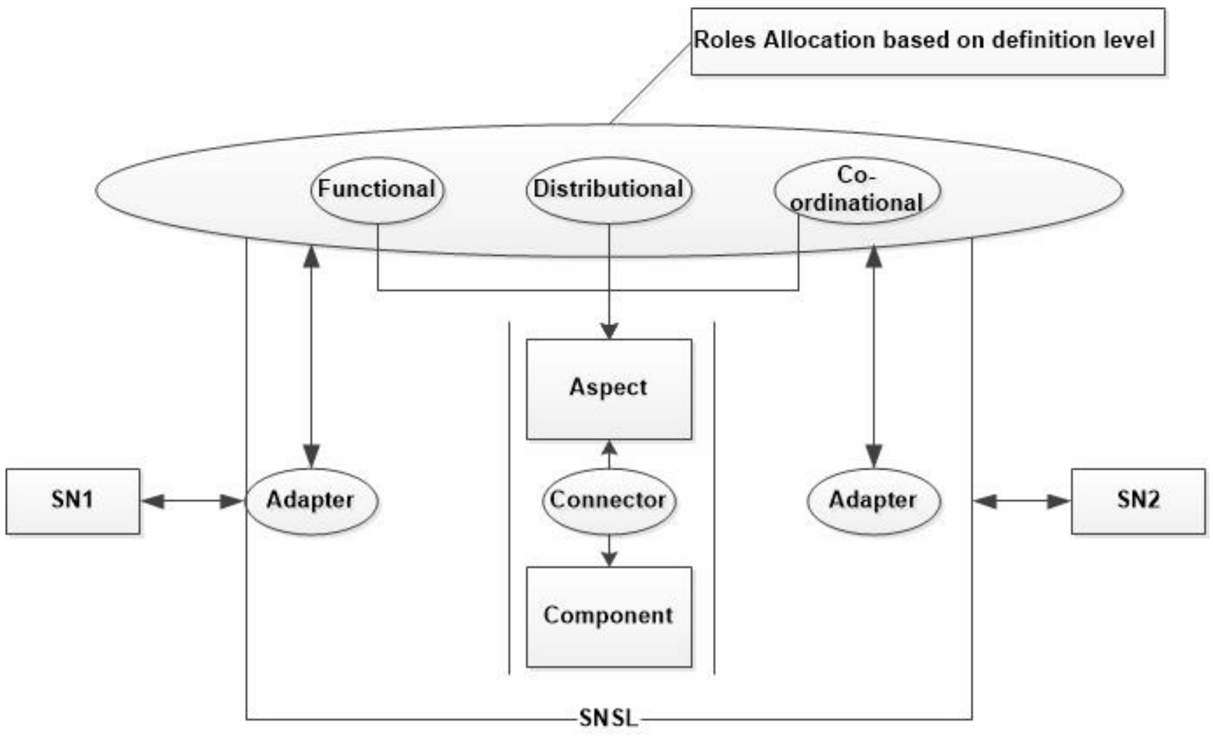


Figure 5-13: Stage 1, structure of SNSL middleware role in DNSP

Figure 5-13, is a depiction of how various components of DSNA have been evolved with more practical knowledge towards the implementation. The description in figure 5-13, demonstrates one of the roles of SNSL as middleware. Furthermore, in stage1 SNSL components are arranged according to the requirements of the prototype scenario.

The implementation steps are the same as proposed in the chapter 4 section 4.5, with the addition of SNSL adaptor component.

User	**Activity**	**Function**	**DSNA Element**	**SNSL Element**
Alice (DSNP)	Profile Reusability	Login	Component.py, Aspect.py, Role.py	Adaptor.py, connector.py
Alice (DSNP)	Data Portability	Messaging	Component.py, Aspect.py, Role.py	Adaptor.py, connector.py
Bob	NA	NA	NA	Adapter.py, connector.py

Table 5-1: Prototype Function outlook

Table 5-1, define the key roles of the elements of the DSNA style. As shown in the figure 5-14, each element is represented as Class. Aspect.py, component.py, Role.py, Adapter.py and connector.py are basically designed as libraries so that they can be included anywhere in code during the development stage of the application. Detail is given in section 4.5.2. Adaptor.py is used only when external connections/actors are involved. The figure 5-11 described an extended skeleton of application presented in section 4-5. The figure shows, that connector.py will be used to handle the communication between aspects and components. Adapter.py act as a wrapper and interprets the external communication (functions) and make them compatible to the new environment.

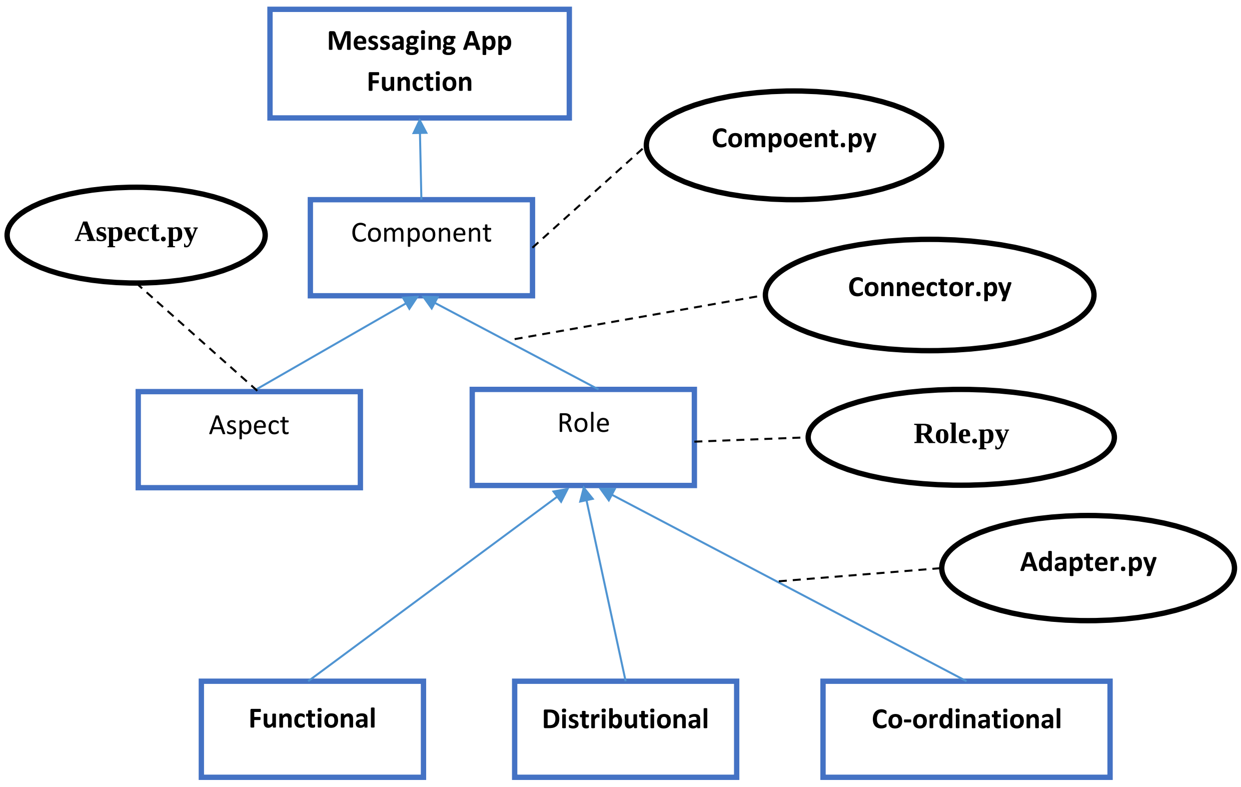


Figure 5-14: Prototype Application Code Skeleton

The figure 5-14 depict the whole function of the Adapter in SNSL. The information received from the adapter is used to define the component and aspect role definition and allocation. This information is used in the composition of new aspect and component.

5.3.2. SNSL Implementation (Stage 1)

As the research progressed, the need to adhere to the standard pattern and vocabulary for the deployment of the DSNP increases. For that reason, DSNA style's rules for the component and aspect design are proposed. For the graphical representation, AOADL element of CAM are used to set standard pattern of actives needed to deploy DSNA based application. AOADL provides standard design element to define the DSNA components based on UML design pattern. Following the same notion, the concept of adapter is adapted from software design pattern (Larman, 2012). The concept of Adapter is fundamental for the deployment DSNP prototype and is central to the process of decentralisation and data portability.

As mentioned in the introduction of the chapter, this stage of design and development phase attempts component composition issue as the next level in the evolution of DSNA architecture. There are 2 stages of the component composition in the DSNA. Figure 5-14 describes the stage 1. The main purpose of stage 1 is to show the SNSL and adapter working, in the form of DSNP prototype, which is a social messaging application.

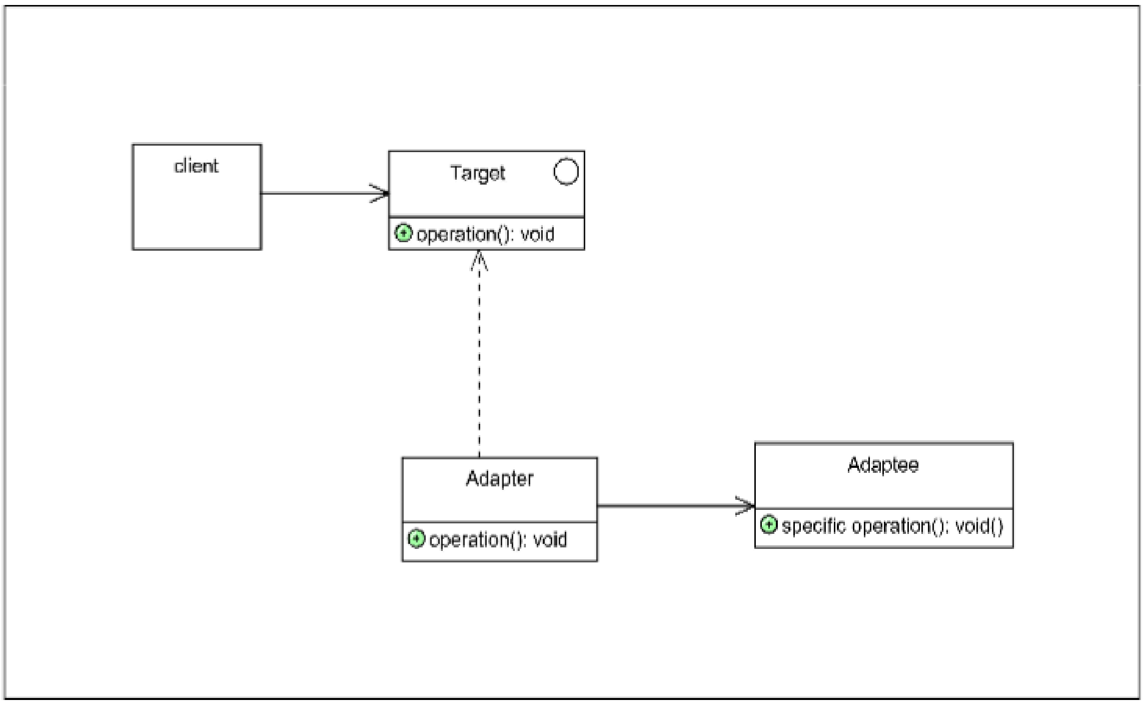


Figure 5-15: Example of Simple Adapter adapted from (Larman, 2012)

In the above figure 5-15, the client class depends on a "Target Interface", cannot reuse the Adaptee class directly because its interface does not initiate the "Target Interface". Instead

the client class work through an “Adapter” class to implements the Target interface as a form of Adaptee.
In contrast, the role of an adapter in the SNSL is communication related and it includes internal and external components (Internal refers to the communication of components in the DSNP and External refers to other SNPs, communicating with DSNP section 5.1.2.3 for details). The important function of the adapter is the conversion of incompatible interfaces (or classes in software design term) to the one requested by the user. The adapters are placed in the SNSL to interpret and gather information received from external or internal users (or actors/clients) regarding the interface. The received information is then used for the procurement of new interface. In the design pattern, simplest Adapter consists of three main classes, Client, Adapter and Adaptee, as shown in the figure 5-15.

The above figure 5-15, which is a static class structure, Client refers to the target interface. Target defines an interface that the Client class requires. Adapter implements the Target interface as required, by calling SpecficOpertaion() on Adaptee object. Adaptee defines the new specification in terms of the class that gets adapted (W3sdesign, 2016). The features of adapter can be accomplished and interpreted in number of ways. In the DSNP prototype implementation, the adapter design pattern is composed by three methods.

- getConnection
- getAction
- settarget

The SNSL manages the adapter and methods are deployed as per specifications. SNSL which also act as middleware is in command of the information gathered by the adapter. The information is retrieved from the external or internal users. This information is used in the definition of the role’s allocation to the components.

The example of the information retrieved by the adapter for SNSL can be access controlled or privacy related or it can be simple request by user for data access.

Each method in the adapter class contains two key parameters.

- ConnectionParameter:
- UrlParameters:

getConnection(), getAction(), setTarget()

SNSL can deploy many adapters based on the ongoing interaction between the users and DSNP. The deployment of the adapters also depends on the already available information in the database about the initiated interaction. For example, as per requirements, Alice initiate the request to interact with Bob on SN3. Bob is not a member of DSNP. To resolve this request SNSL initiates the adapter and getConnection() method is used based on the already available information or prescribed information by the user having specific parameters to adhere the request. Open and Close objects are used to control the getConnection() states until the specified objective of the connection are met.

getAction(), depends on getConnection() and on the information received from the external SNP (SN3). This method identify Role based on the information received and update the database. ConnectionParameters, UrlParameter and specified information by Alice is used to identify the right Role for the allocation.

In the last step of the adapter functionality setTarget() is deployed. The information from other two methods is inherited to setTarget() method. Based on that information Role is allocated to the aspect and the specified information is used to update component.

The entire process is run at the program level. Thus, modifying component at the semantic level and updating the functionality at the DSNP based on the user need.

The implementation of adapter to facilitate the composition of component is a multifaceted activity and can enhance the size of demonstration. To keep the evaluation within the iterative cycle, component composition is divided into two stages. Second stage is a part of next phase of design and development of DSNA based application.

To present the evolved version of DSNA and demonstrate the above finding social messaging application is built. The next section describes the tool and technologies used in the form of DSNP technology stack and application behaviour.

5.3.3. Application skeleton

This part of the evaluation describes the application skeleton of messaging prototype, developed to show applicability of the DSNA. In the prototype complex requirements are explored to develop a new version of messaging application based on scenario explained in section 5.1. In this version of the prototype all three users ALICE, BOB and TONY are connected to three different social networks (SN1, SN2, SN3). ALICE is connected to SN1 and SN2, BOB and TONY are connected to SN3. They share the functionality F1 by using the DNSA based platform DNSP.

This whole interaction between ALICE, BOB and TONY is designed using DSNA. The activities performed by all the users are explained in the table 5-2.

User	Activity	Functionality	DSNA element
Stage 1			
Alice	Security privacy related, Profile reusability	Identity Management, Contact Management,	Component.py, Aspect.py, Role.py Adapter.py
Bob	Security privacy related, Profile reusability	Identity Management, Contact Management	Component.py, Aspect.py, Role.py, Adapter.py
Tony	Security privacy related, Profile reusability	Identity Management, Contact Management	Component.py, Aspect.py, Role.py, Adapter.py
Stage 2			
Alice	Security privacy related, Data portability, Profile reusability	Message Exchange	Component.py, Aspect.py, Role.py, Adapter.py
Bob	Security privacy related, Data portability, Profile reusability	Message Exchange	Component.py, Aspect.py, Role.py, Adapter.py
Tony	Security privacy related, Data portability, Profile reusability	Message Exchange	Component.py, Aspect.py, Role.py, Adapter.py

Table 5-2: Application functionality outlook

Figure 5-14 shows the messaging application design in the context of current evaluation. The extended version of the artefact contains the Adapter component of the SNSL. Adapter.py represent the component code structure which is explained in the next section. The process in which the key elements of the code are explained is same as described in section 4-5. According to which each class is designed as a library so that they can be included anywhere in code during the development stage of DSNA application. Detail is given in the next section.

DSNA components	Life Cycle	Definition
Adapter.py	Dynamic	Adapter.py represent the components of the SNSL. The Adapter.py demonstrates the implementation of design pattern into the SNSL towards achieving dynamicity in DSNA component composition. Its main role is to interpret the information coming from the other SNPs to procure the SN functionality in DSNP.

Table 5-3: Code related description of Adapter

5.3.4. Technology Stack

Key tools and technologies used for the implementation are already explained in the chapter 4. Technology stack explains how the different technologies are used to develop the proposed DSNP prototype.

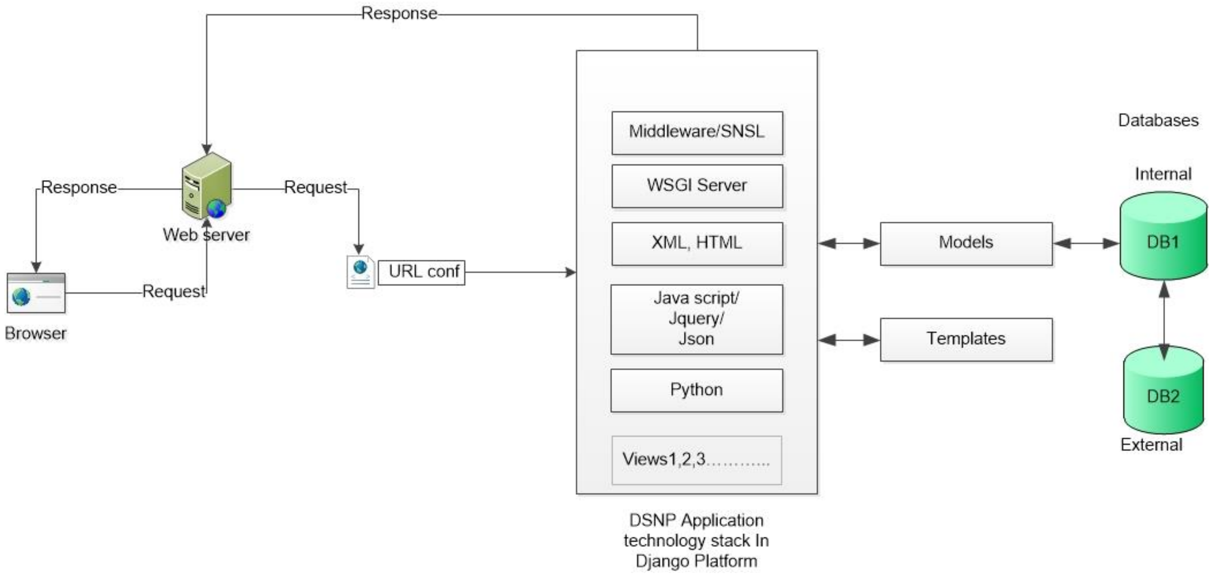


Figure 5-16: DSNP application technology stack in Django platform

The figure 5-16 describes a very simplified process of web request from browser to Django based DSNP application. There are few steps that are important to understand to know how the request work.

1. In the first step browser send request to the web server.
2. Web server hand over the request to WSGI and SNSL.
3. Unlike web server, WSGI can interpret and run python applications. The request populates a python directory having necessary description files and environment variables details.

4. UR configuration is contained in the urls.py of the DSNP application. The file contains the description of all the web URL attached on each view.
5. The selected view talk to the database, renders HTML/XML or any other formatted response using templates and if unable to render response raise an exception.

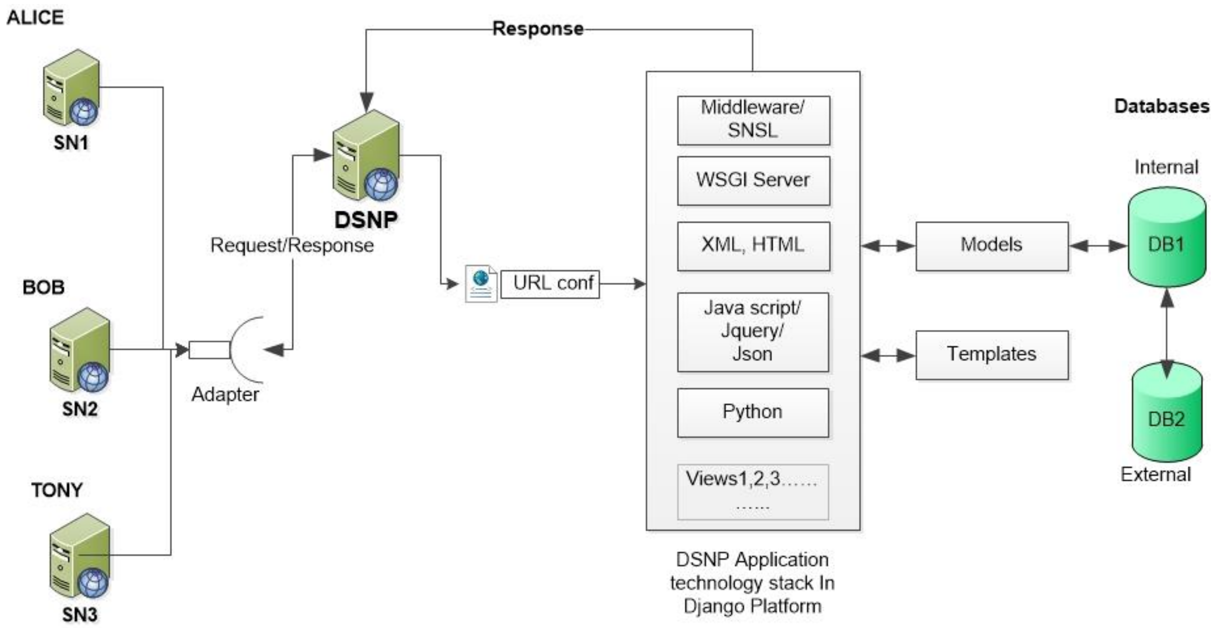


Figure 5-17: DSNP structure in the line of DNSA scenario

The figure 5-17 shows the simplified high-level view of the DSNP prototype in line with the implementation scenario explained in section 5-1. The figure also explains the place of the adapter in the application. To see how the application function, next section describes the behaviour of the application

5.3.5. Application Behaviour

The prototype is developed around the standard guidelines and protocols for user access as depicted in figure 5-17. Following the same format of procedures as mentioned in section 4-5, this section describes the behaviour the new extended version of DSNA based prototype. To enhance system flexibility, the tools and technologies are carefully selected to ensure the platform independence.

The proposed Adapter pattern plays key role in building and importing functionality, working together with communication layer and SNSL. Adapter is central to solve the composition issue related to component towards procuring the SN functionalities. The prototype fulfils the prescribed requirements of data portability and profile reusability (R3, R4 and R5) at the first stage of SNSL implementation. To explain the behaviour of prototype in the first step and the overview of the Adapter algorithm is given, in the second step system sequence diagram is built, which gives the implementation perspective of the prototype.

Variables Declaration	**Description**
Fn **Ds** **Co**	Represent the behavioural allocation to the aspect. Fn is functional, Ds is distributional, and Co is co-ordinational allocation.
Cp **Up** **Str**	Cp refers to the connection parameters and Up refers to URL parameters. Str is a string.
Target	Target variable is to store information related to the functionality targeted deployment.
Action **Protocol**	Action variable is to store the information related actions (database related) needed to deploy the functionality. Protocol contains the information about the protocols required to secure the connection.
Check	Check is used for method call
Uname	Username
Pwd	Password

Table 5-4: Algorithm Terminologies

```
Declare class functionality deployer
Class FunctionalityDeployer: //user or source of the functionality
Declare variables
        Fn, Ds, Co, Cp, Up, Str, Action, Target, Protocol
        Define methods
        adapter methods with relevant parameters
        Def getCon(Cp, Up, Str):
        Def getAction(Cp, Up, Str):
        Def setTarget(Cp, Up, Str):
        Behavioural Allocation
        Fn -> Sn. Functional behaviour (self):
        Ds -> Sn. Specify location (self):
        Co -> Sn. Data sync (self):
Declare Adaptee
Class TargetAllocation(FunctinalityDeployer):
Method call to check the connection to SN and get the required data
        getCon.open (Cp,Up, Str):
open object to open the connection to external SN by the mean to communication layer.
```

```
If
        Check -> CommunicationManager (Uname, Pwd):
        Print (“connection successful”)
        getCon.close
        Else if
if connection is successful allow the communication otherwise, resolve the ID and protocols
required to secure the communication.
        Check -> UserID Reslover(Uname, Pwd):
        Print (“connection successful”)
        getCon.close
        Else if
        Check -> ProtocolHandler (protocol):
        Print (“connection successful”)
        getCon.close
else
getCon.close
return exception
After securing the source, assess the behaviour of the functionality by selecting action
Class TargetInterface:

        getAction (Cp, Up, Str):
Based on user preferences get action will secure the necessary needs of the SN function.
Method RoleAllocation is called to assess the role
Check - > roleAllocation (Fn, Ds, Co):
         functionalBehaviour(Fn):
         specifyLocation(Ds):
         dataSync(Co):
//Adapter class is where the process of allocation and adaptation is finalised.
Class Adapter(TargetInterface):

        Call method settarget() to set the functional behaviour to targeted user and component
        After checking the basic requirements of the targeted component
        Initialise the target allocation.
        setTarget (Cp, Up, Str):
Client class according to adapter pattern
Class Functionality:
        Do all the checks performed in TargetAllocation
        Check connections, use getcon() to secure the connection to the source
        Call getAction()and setTarget()
        Check the behaviour and set the target according to the parameters. Do this for all types
        of allocations
        FunctionalityDeployer(Fn) = TargetAllocation ()
        Adapter= Adapter(TargetInterface)
        Func = Functionality (Adapter)
        Return
        End
```

Figure 5-18: Adapter Algorithm

The deployment is performed by the human actors in our case (Alice, Bob, Tony) referred as functionality deployer (see figure 5-8). These actors are trusted within the boundaries of the platform or system they are attached to. In the current case, all three users are connected to their respective SNPs. The functionality is shared and deployed at the DSNP. Alice would like to share F1 which is assumed as related to the profile reusability and data portability requirements. The deployment is triggered by an adapter at the SNSL and Aspects are deployed, un-deployed or replaced based on behavioural needs of the functionality at the run time level. Aspect Binding function (see figure 5.8) which contains pointcuts, in Aspect.py updates the component and prepare for the deployment within DSNP or for the targeted SNPs.

To further evaluate the prototype under the prescribed requirements R3 R4 and R5 a sequence diagram is built to show the complete behaviour of the implemented application. There are certain preconditions to ensure the requirement design criteria is met for the sake of stability.

Precondition:

- Alice is a member of SN1 and SN2
- Bob is a member of SN3
- Tony is a member of SN3
- They must have access to their accounts.

The procedure of securing access to the DSNP is done in four steps. Normally, access to any platform start with the login. Alice, Bob and Tony are declared as one single entity the “User”. The first step, Alice initiates the request to use DSNP as Alice is not a registered user therefore Alice either creates new user account with DSNP, as allowed security and privacy policy of the platform or access the DSNP with existing SNP ID. In the second step “Communication Manager” which receives the request looks for the relevant methods of access and security policy. Protocol Handler call method GetIDProtocl() to know the protocols required and UserID Resolver uses ResolveID() method to give access to the User.

Since Alice wants to use her existing social network ID and profile in the DSNP therefore SNSL middleware queries access and control service by the mean of an Adapter. In the third step Adapter calls a set of methods to allocate the behaviour desired to achieve the functionality in the DSNP. The diagram shows Adapter communicating with SNPs and getting the required data related to the user’s login details, and during the process it create and update user data.

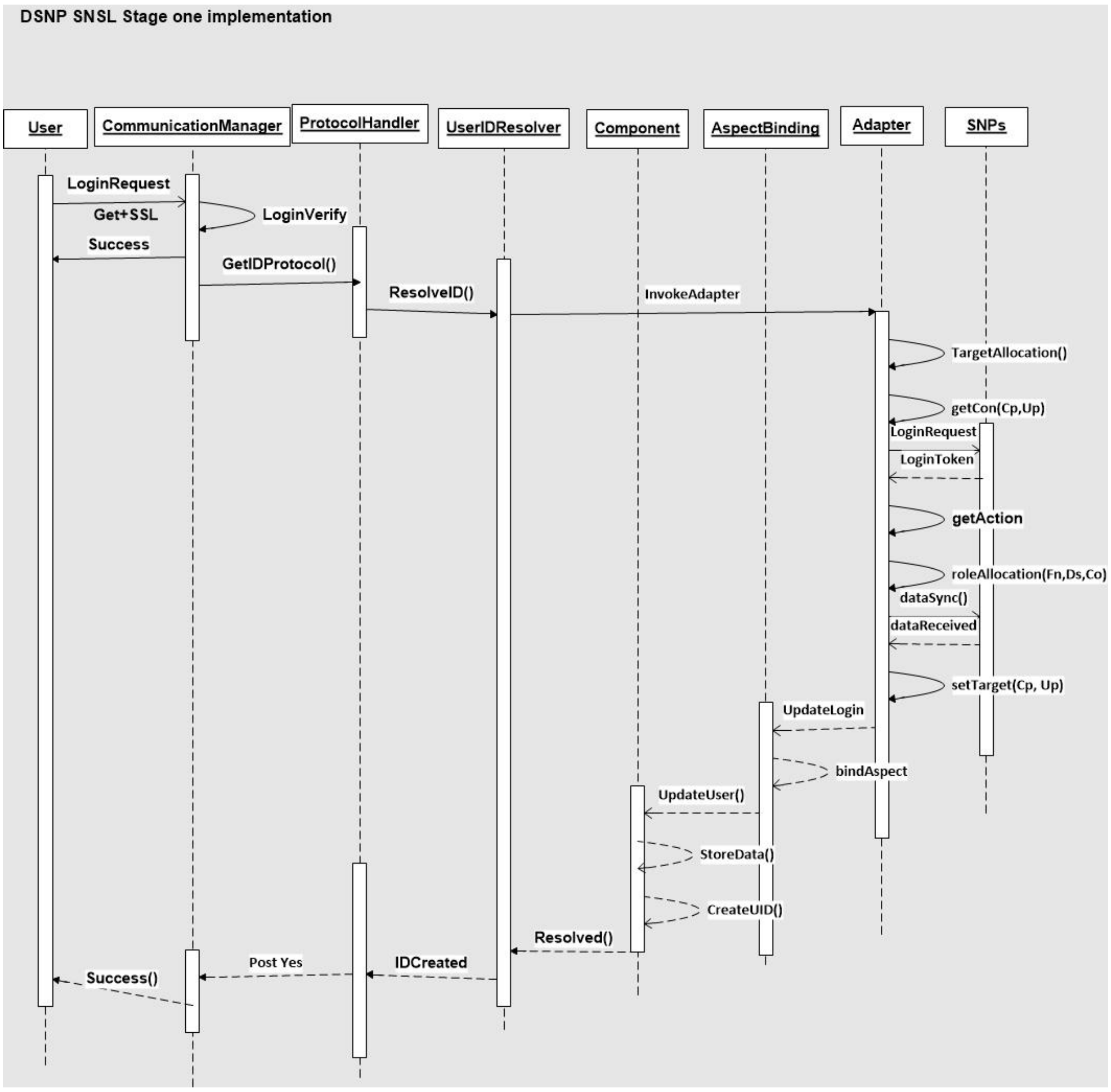


Figure 5-19: Sequence Diagram of SNSL Stage 1 implementation

In the fourth step Aspectualbinding (section 5-2) method of the Aspect is used, updating the component interface to provide new functionality in the DSNP as requested by the user. In current case allowing the user to use their existing profile and credentials to access and share data on the DSNP.

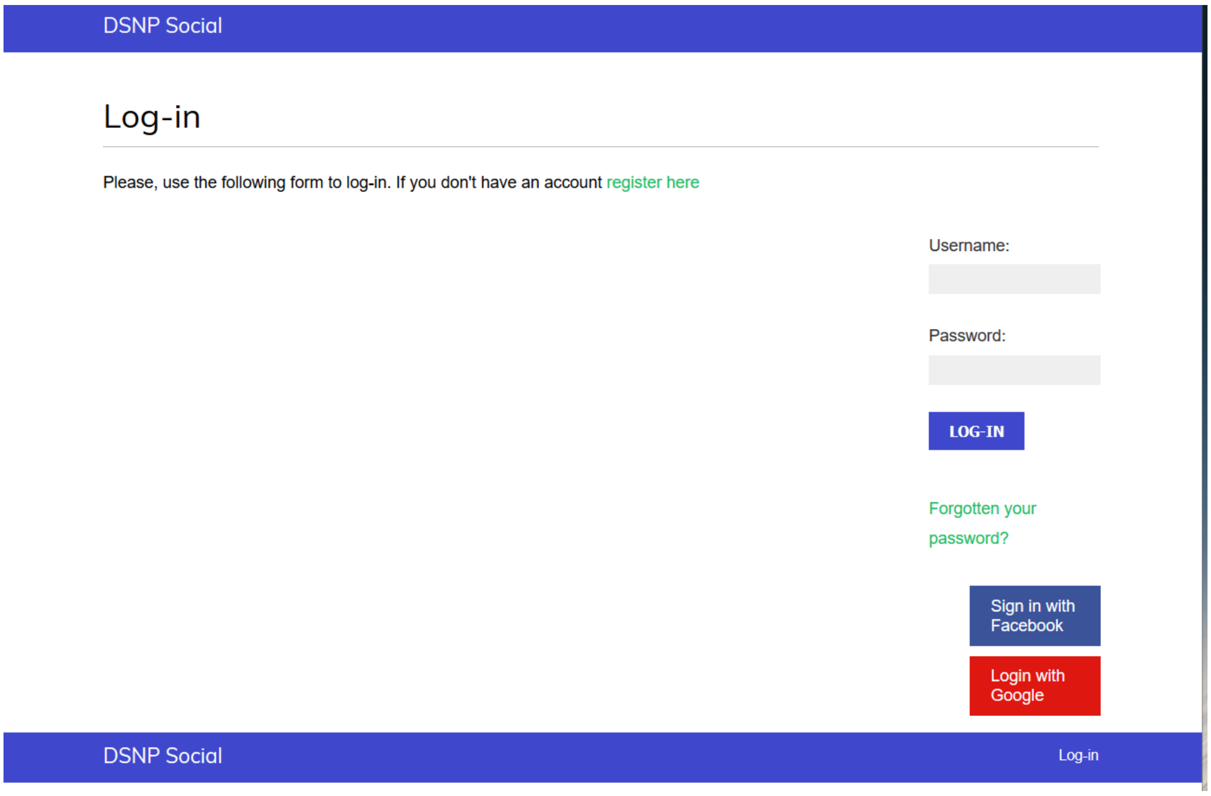


Figure 5-20: DSNP Main Page

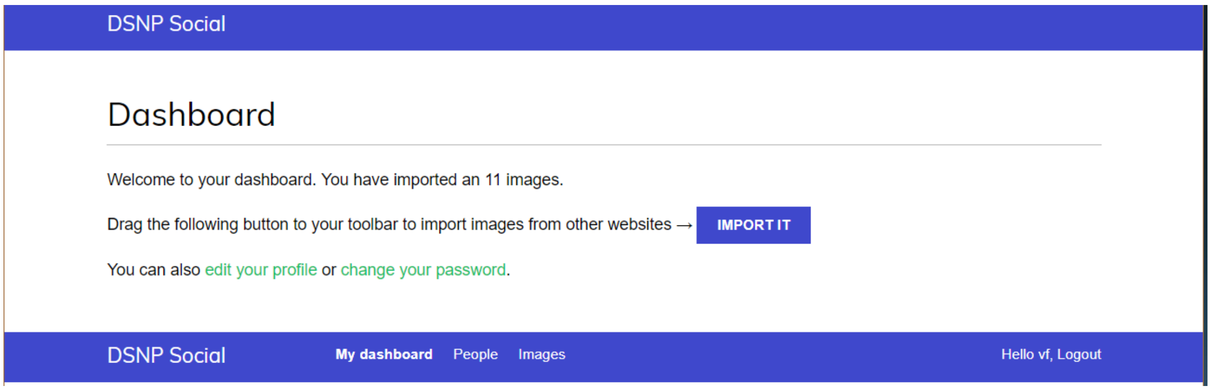

DSNP Social
Dashboard
Welcome to your dashboard. You have imported an 11 images.
Drag the following button to your toolbar to import images from other websites →
IMPORT IT
You can also edit your profile or change your password.
DSNP Social
My dashboard
People
Images
Hello vf, Logout

Figure 5-21: DSNP Dashboard

The artefacts produced in this iteration gives the detail knowledge about how the DSNA components are composed into a functionality. To achieve the set objectives, very specific requirements are designed under the guidelines of the SWAT scenario and certain challenges are specified. The implementation of the prototype demonstrates that proposed concept of DNSA is feasible. Deployment stage of the iteration, key process towards achieving the component composition in the DSNA are described as role allocation. The purpose is to define the specification of component so that the composition process has the required information when interpreting the information coming from other platforms. Adjusting the adapter into the process of role allocation is very complex and the most tedious task of the implementation. Still there are complexities and 100% result are not achieved, for example the process of adaptation become more complex when binding with the Aspect. For example, importing complex functionalities such as Friend List and historic data of the posts are still an ongoing work. Although, there is a partial success in achieving objectives of DSNA and setbacks are due to not enough help related to tools and technologies. Every component is designed from scratch, the changing access requirements of SNPs makes it more difficult to achieve the desired goals.

5.4. Chapter Conclusion

The evolved version of the DSNA is presented in this chapter. The rigorous approach is taken to design, deploy and evaluate the architecture. The architecture of the prototype shows the necessary structural requirements needed to implement the decentralised social networking environment. To implement the evolved version of DNSA extended version of SWAT scenario is used to encompass more complex design requirement.

In the pursuit of the answers regarding the implementation of DSNP, first, the background knowledge on the DSNP related to architectural components and main challenges that needed to be addressed, are explained.

The focus of this chapter is to address the interaction, integration, interaction and allocation related challenges in the design and development of the decentralised social application. In doing so the goal is to achieve data portability requirements. Component and aspect composition are the key to the success to the DSNA. SNSL which also act as middleware is central to the composition problem, as SNSL not only controls the interaction of the DSNA components to other SNPs but also plays significant role in the composition of the components by the mean of adapter. The adapter act as bridge between the external information and the components. At the final stage, the DSNP messaging prototype is build based on design requirement mentioned in the scenario, to demonstrate the functions of the DSNA with the addition of adapter.

Chapter 6 Iteration 3 – Final Evaluation of DSNA

6. Chapter Introduction

This chapter presents the research carried out in order to further investigate the practicality of DSNA by testing it in different domains. In addition to this, new components are introduced based on lessons learned in iteration 2. The prototype is based on the additional architectural feature to solve the problem of data consistency and persistence. The evaluation is done on the criteria mentioned in chapter 4, by using method known as Social Web Acid Test (SWAT v1) introduced by the W3C federated social web group. This Test provides guidelines and numerous use cases that can be used to validate the practicality of the decentralised social web. Due to the complex nature of the experiment, the prototype is evaluated on the basis of interaction and communication use cases. These use cases are the extensions of the main scenario.

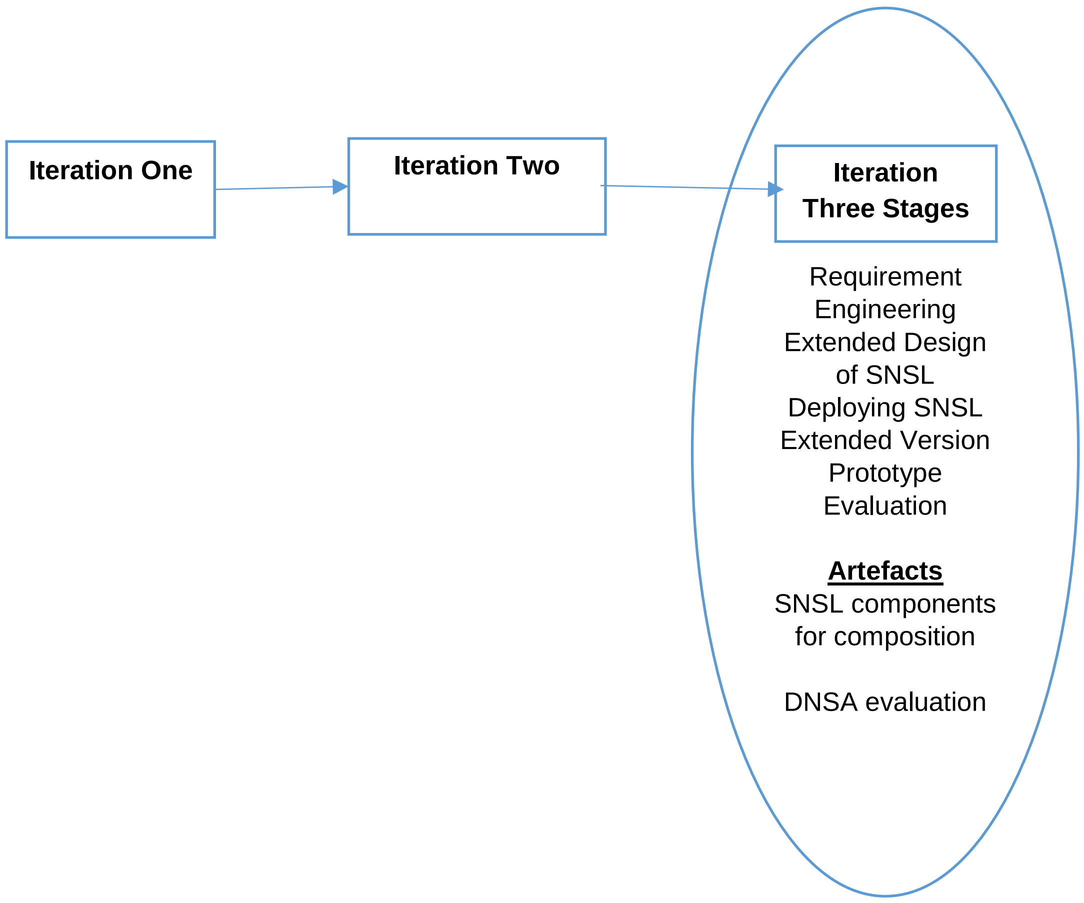


Figure 6-1: Iteration 3

6.1. Extended Design of SNSL

In the previous chapter, a prototype of the DNSA is designed and implemented. The key part of the implementation was to achieve objective 4 of the research by implementing the component composition method for DSNA. As described in chapter 4 SNSL acts as middleware and handles component composition. SNSL stage 1 implementation uses the adapter design pattern to support the data portability and reusability features of DNSA. This step of the development stage assesses the previous inconsistencies and lesson learned to contribute to the final version of the design. The focus of the evaluation design is to overcome the design challenges mentioned in chapter 5.

6.1.1. Analysis of Requirements

To produce the extended version of DNSA, this iteration uses the same requirements described in the previous chapter. There are also no changes to the common setting of DSNA components (See section 5.1).

6.2. Component Composition and SNSL

This section describes the additions that are necessary to support component composition in DNSA.

As already explained in chapters 4 and 5, the main characteristic of the SNSL is that components and aspects are first order entities that are dynamically composed at the runtime. The figure describes all the elements of SNSL that are crucial for its middleware role in DSNA. The SNSL middleware platform interprets the information of the other SN and that information will become part of the internal data structure of the application. The SNSL middleware platform will use the information at runtime to perform the weaving and binding between the aspects and components, with the aim of updating the SN functionality based on the user’s request.

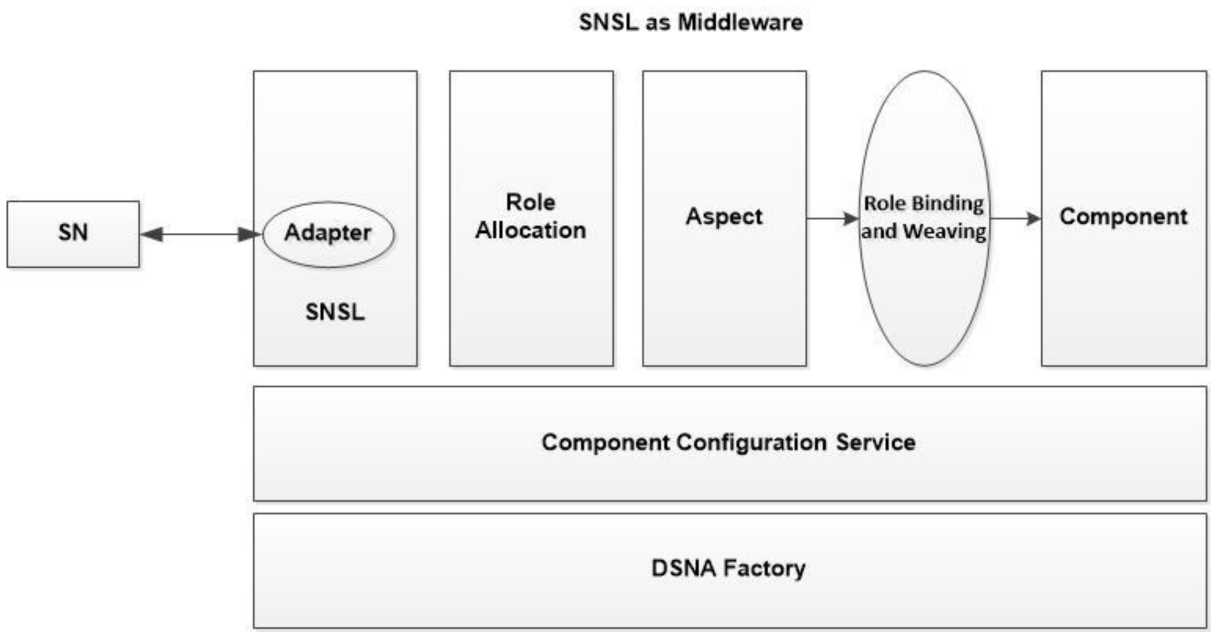


Figure 6-2: SNSL Stage 2 design

6.2.1. Component Configuration Service and DSNA Factory

Several papers including, Schauerhuber at al., (2007), Pessemier et al., (2008), Pinto et al., (2011), have proposed the use of Component Factory and Component Configuration Service in distributed application. The purpose is to avoid data inconsistency and to reduce the information gap between design and implementation. Furthermore, the platform will be able to detect if any component doing design violation.

Aspects and components be can created and destroyed (Pinto et al., 2005). DSNA supports instantiation and deletion of the components by means of he DNSA Factory, which is an altered form of the Component Factory. The DSNA Factory keep track of the components altered to become SN functionalities and their interfaces. The DSNA Factory consists of two methods.
CreateFunctionality(),
DestroyFunctionality().

The syntax of methods are CreateFunctionality(RoleName String, RoleInstance String RoleAllocation String), DestroyFunctionality(RoleName String, RoleInstance String RoleAllocation String)

It should be noted that, Components and Aspects are identified by RoleName, RoleInstance and Role Allocation (See chapter 5). Using this service, a functionality can be called and deployed by giving the string RoleName, RoleIsntance and Allocation. Hence improving data consistency and reusability.

There is an issue regarding recreating and modifying the component based on a user's request at runtime, raised during the SNSL stage one deployment. The second element Configuration Service provides a set of methods to modify the application at runtime, that are stored in Application Logic component of the DSNA. Configuration Services make it possible to add, modify or remove the description of the components, aspects, properties and composition role using the methods in Configuration Service. The information gained from these methods is used to modify or adapt the description of the components and behaviour of the functionalities.

The Configuration Service works in conjunction with Persistence Service. For example, in Iteration one (section 4.4) the messaging application uses the Persistence Service. Few examples of adaptation can be performed in the chat application and the information has to be added or modified in the application logic structure. Adding an image sharing feature to the chat component is another feature of Configuration Service. In the context of application developer, this service gives easy plugging and unplugging of aspects and components into application at runtime. For instance, users connected to the chat applications can be found by changing the Aspect Composition rules and binging aspectual code of tracing to the component.

Configuration Service is composed of three main methods,
addComponentInfo(Name String, Source String, RoleName String)
addAspectInfo(Source String, Target String, RoleName String)
addRoleInfo(RoleName String, TargetAllocation String)

Considering the research design process, with regards to the guidelines of iteration 3, the next stage further investigates the design of SNSL in the context of deployment.

6.3. Deploying SNSL Stage 2

This stage of iteration 3, investigates the deployment of stage 2 of SNSL. In the context of the application developer, design stage has described new features of SNSL based on the lesson learned from Iteration 2. In the light of design challenges and requirements, the deployment stage attempts to solidify the DSNA by demonstrating the deployment of SNSL stage 2.

During deployment, the main problem which is solved concerns the data inconsistency and persistence when importing features from other SNPs. The focus is on component composition and the elements that connect them together and with another SNP.

The current deployment extends the scenario with the additional features of DSNA factory and Component Configuration service. During the prototype implementation in iteration 2, one of the main goals was runtime composition of the functionality from the imported features of other SNPs functions with the help of DSNA components. Role allocation and Adapter were used in conjunction with AO composition features of binding and weaving. The positive aspect of that implementation was that, functionalities were imported successfully, on other hand recomposing and reusability was not possible.

To resolve these issues Component Factory and Component Configuration Service concepts were adopted. The design functions of these two are already explained in the above section 6.2. This section describes deployment SNSL stage 2 deployment for SN functionality reusability.

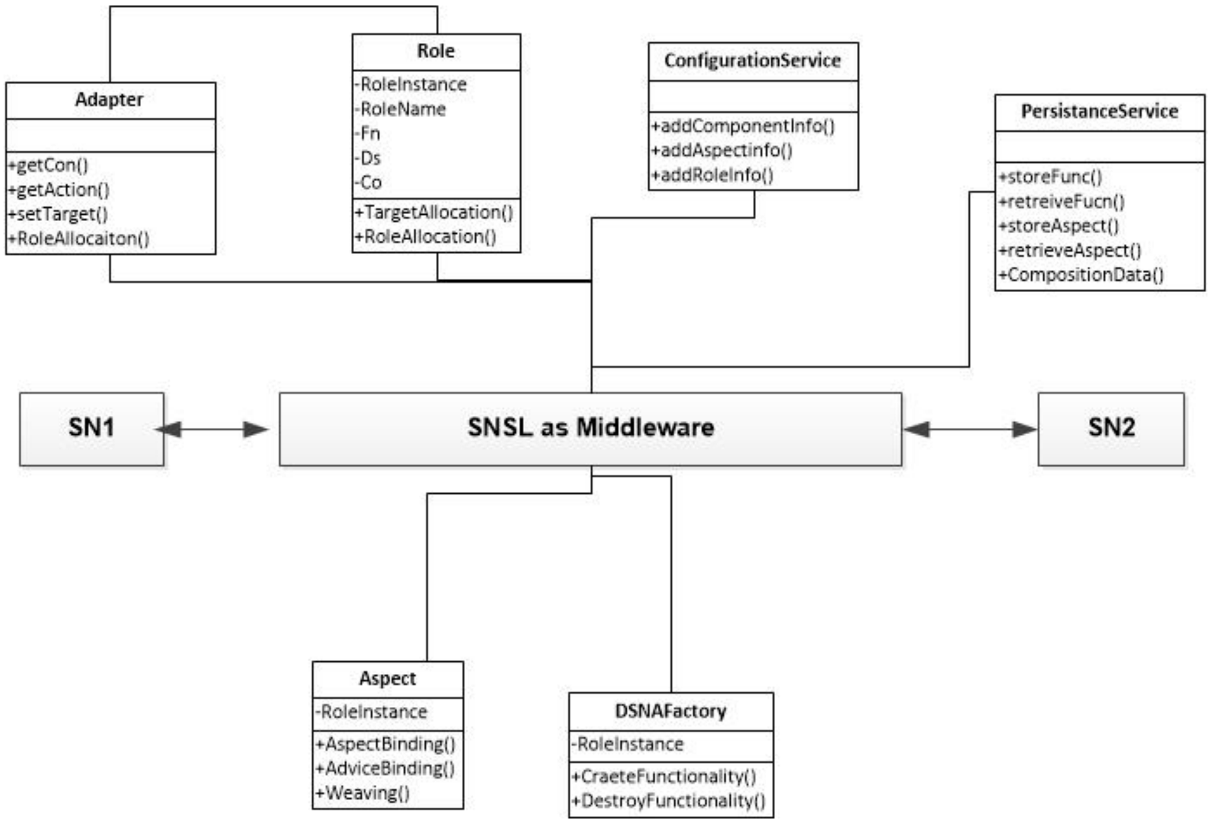


Figure 6-3: SNSL as middleware stage 2 deployment

The figure 6.3 shows how SNSL acts as middleware and key components that work together with DNSA to establish a connection between multiple social networks to procure the SN functionality. To achieve the deployment of the SN functionality within DNSP prototype, Adapter works with Aspect component Role and Role Allocation, DSNA Factory, Configuration Service, and Persistence Service.

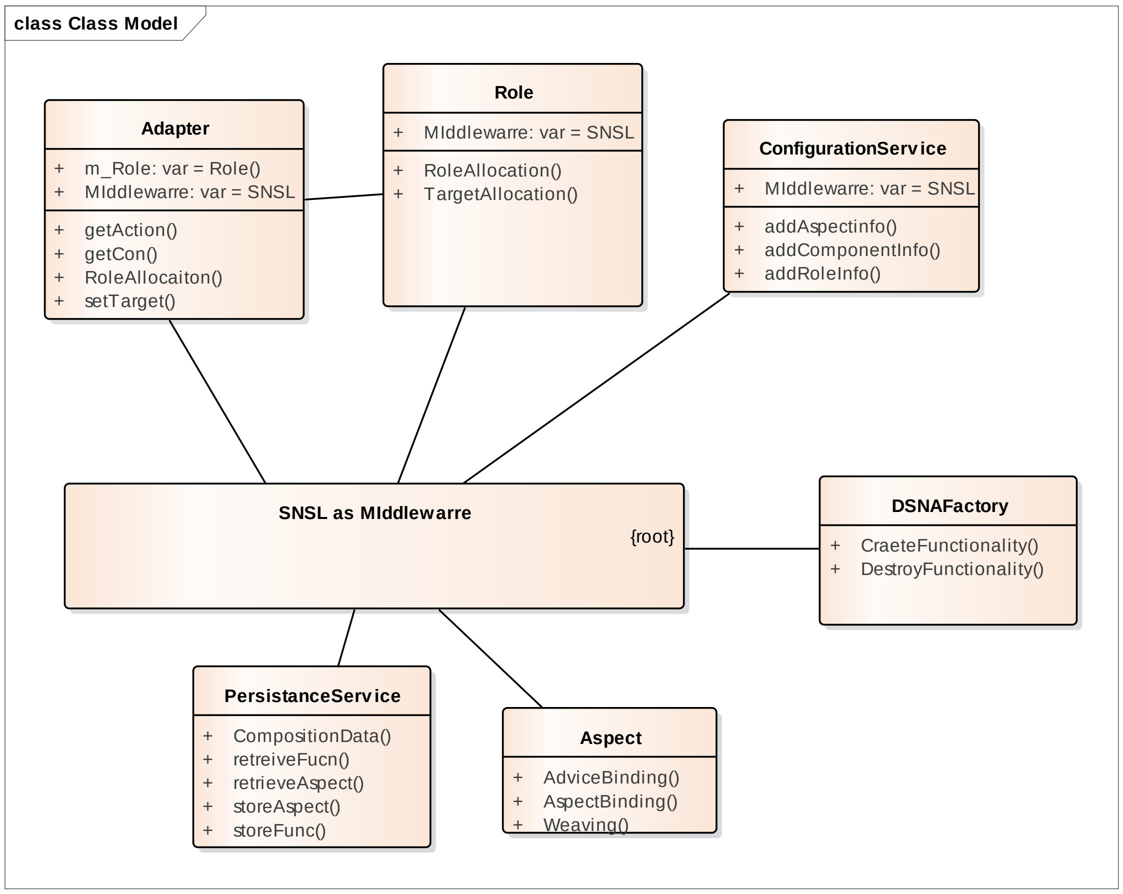
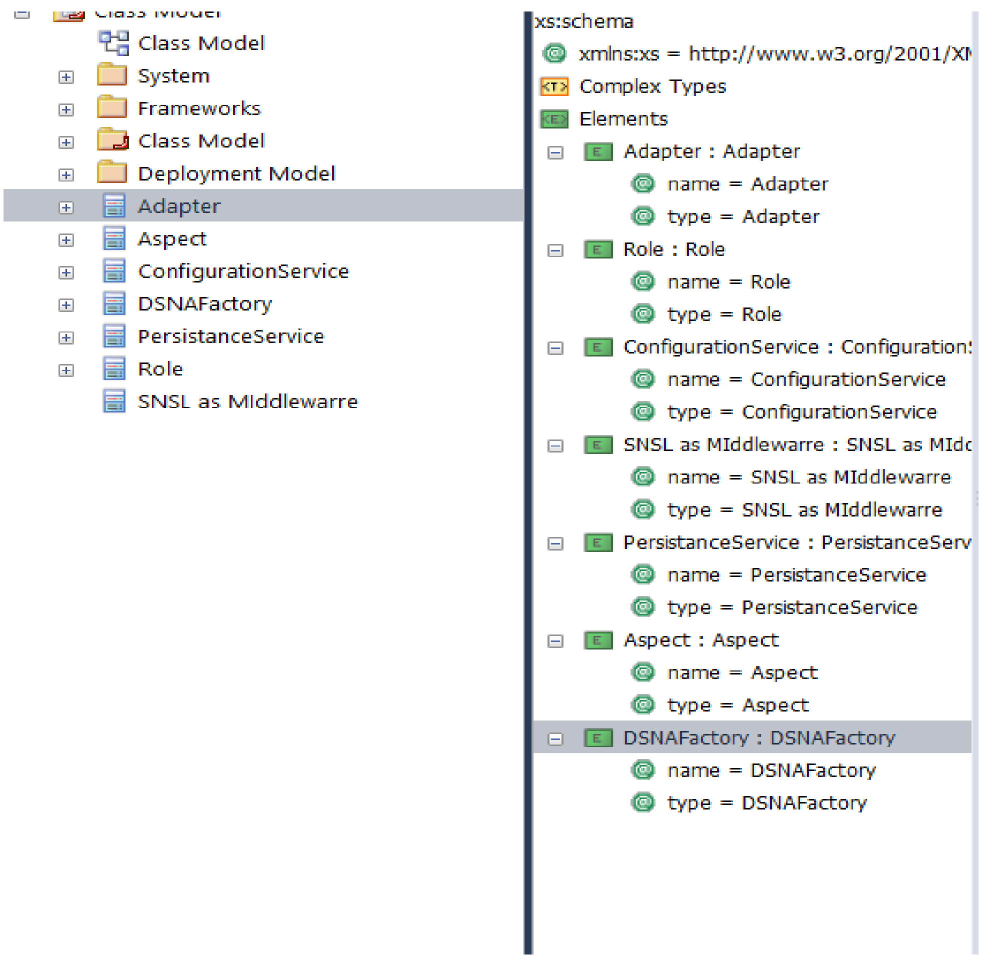


Figure 6-4: SNSL as middleware stage 2 Class Mode in an execution environment

6.3.1. Reusing Functionalities through SNSL Middleware

Traditional middleware platforms such as J2EE Glass Fish, used for AO based distributed applications are equipped with serval services to support various functional and non-functional requirements of the application. The implementation of these services in traditional middleware platforms is often monolithic, due to a high level of coupling between the middleware and the services. As a negative consequence, there is no easy way to remove the unneeded service during the implementation of the middleware and install more suitable third-party services (Landuyt et al. 2011).

This lack of adaptability limits the reusability of traditional middleware services. To tackle these limitations, SNSL as middleware is introduced in iteration one. Which is mainly based on the lesson learned in implementation of the messaging in DSNP. SNSL stage 2 accomplishes the goal of reusability and gives flexibility by providing the components to enhance data persistence and consistency between the DSNP and other SNPs. The SNSL also gives flexibility to application developer to adapt according to the changing needs at the network and communication levels.



```
⊟ Class Model
      Class Model
  ⊞   System
  ⊞   Frameworks
  ⊞   Class Model
  ⊞   Deployment Model
  ⊞   Adapter
  ⊞   Aspect
  ⊞   ConfigurationService
  ⊞   DSNAFactory
  ⊞   PersistanceService
  ⊞   Role
      SNSL as MIddlewarre
```

```
xs:schema
  xmlns:xs = http://www.w3.org/2001/X
  Complex Types
  Elements
  ⊟ Adapter : Adapter
       name = Adapter
       type = Adapter
  ⊟ Role : Role
       name = Role
       type = Role
  ⊟ ConfigurationService : Configuration
       name = ConfigurationService
       type = ConfigurationService
  ⊟ SNSL as MIddlewarre : SNSL as MIdc
       name = SNSL as MIddlewarre
       type = SNSL as MIddlewarre
  ⊟ PersistanceService : PersistanceServ
       name = PersistanceService
       type = PersistanceService
  ⊟ Aspect : Aspect
       name = Aspect
       type = Aspect
  ⊟ DSNAFactory : DSNAFactory
       name = DSNAFactory
       type = DSNAFactory
```

Figure 6-5: SNSL as middleware stage 2 execution environment

Figures 6-3, 6-4 and 6-5 describe the deployment of SNSL in the proposed scenario in the context of deployment. The class diagram gives the simplest outlook of SNSL as middleware and the components interaction with each other.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="Adapter" type="Adapter"/>
        <xs:complexType name="Adapter">
                <xs:sequence>
                        <xs:element name="RoleInstance" type="xs:int" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="Role" type="Role" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="SNSL as MIddlewarre" type="SNSL as MIddlewarre"
minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="Role" type="Role"/>
        <xs:complexType name="Role">
                <xs:sequence>
                        <xs:element name="Co" type="xs:int" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="Ds" type="xs:int" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="Fn" type="xs:int" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="RoleInstance" type="xs:int" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="RoleName" type="xs:int" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="SNSL as MIddlewarre" type="SNSL as MIddlewarre"
minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="ConfigurationService" type="ConfigurationService"/>
        <xs:complexType name="ConfigurationService">
                <xs:sequence>
                        <xs:element name="SNSL as MIddlewarre" type="SNSL as MIddlewarre"
minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="SNSL as MIddlewarre" type="SNSL as MIddlewarre"/>
        <xs:complexType name="SNSL as MIddlewarre">
                <xs:sequence>
                        <xs:element name="PersistanceService" type="PersistanceService" minOccurs="1"
maxOccurs="1"/>
                        <xs:element name="Aspect" type="Aspect" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="DSNAFactory" type="DSNAFactory" minOccurs="1"
maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="PersistanceService" type="PersistanceService"/>
        <xs:complexType name="PersistanceService">
                <xs:sequence/>
        </xs:complexType>
        <xs:element name="Aspect" type="Aspect"/>
        <xs:complexType name="Aspect">
                <xs:sequence/>
        </xs:complexType>
        <xs:element name="DSNAFactory" type="DSNAFactory"/>
        <xs:complexType name="DSNAFactory">
                <xs:sequence>
                        <xs:element name="RoleInstance" type="xs:int" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Figure 6-6: SNSL Notation in XML

DSNA Factory and Configuration Service: DSNA Factory provides create and destroy functionality operations. When the create operation is called, the factory looks for the functionality RoleName matching it with Role class according the RoleInstance, binds a component to it, and create a constructor in the SNSL Implementation Class. This constructor can be remote, depending on the TargetAllocation and RoleAllocation of the component.

Configuration service provides set operations to modify the structure of the application at runtime, which is stored in the application logic component of SNSL. Configuration service makes it possible for SNSL to add, modify or remove the description of the components, Aspects and Roles, using the corresponding methods. This service is a crucial part of SNSL stage two implementation, and very helpful in configuring application dynamically, adapting it according the user preferences or any requirement of the connected SNPs.

```
import Adapter
import PersistanceService
import Aspect
import Role
import DSNAFactory
import ConfigurationService

class SNSL as MIddleware:
    m_Adapter= Adapter()

    m_PersistanceService= PersistanceService()

    m_Aspect= Aspect()

    m_Role= Role()

    m_DSNAFactory= DSNAFactory()

    m_ConfigurationService= ConfigurationService()
```

Figure 6-7: SNSL high level functional definition

The code mentioned in above two snippets, is a description of the DNSA architectural components interpretation by the mean of SNSL. Following this procedure of deploying social applications and their functionalities is helpful in testing the application. The execution environment used for this procedure was already explained in section 5-2.

Up to now, this section has given a detailed account of the final extension of the proposed DSNA's SNSL artefact. To assess the practicality and evaluate on the basis of the proposed scenario, the next section performs the final evaluation based on the guidelines explained in section 5.1

6.4. Final Evaluation

The final evaluation of DNSA based prototype is done as part of evolving research process. As research progresses, complexity and challenges concerning the success of the architecture increase as well. In this stage of the development process the same research process pattern is followed as in the previous iterations. The deployment is converted into the final executable prototype and full implementation is carried out to achieve the research objectives 4 and 5.

The evaluation is done in two steps. First by explaining the prototype behaviour and second by explaining the performance evaluation. Performance evaluation is done on the basis of interaction and communication between the users and the DSNSP. SWAT guidelines are used to set the use cases.

6.4.1. SNSL Implementation

Section 5.3.4 of iteration 2 explained, the role of SNSL in the platform during the communication between the SNPs. The SNSL middleware platform is prototyped on Apache WSGI and the combination of JSON and WebSocket. Component interfaces and implementation can be defined in any python-based interpreter. Remote method invocation calls are asynchronous and involve a python web framework built in objects for communication between the components. The interaction between the components is verified by the Cpython Compiler.

The SNSL stage two implementation is tested by importing the messages from other SNPs to the DSNP. The process of application behaviour testing is already explained in section 5.3.3.

Precondition:

- Alice is a member of SN1 and SN2
- Bob is a member of SN3
- Tony is a member of SN3
- They must have access to their accounts.

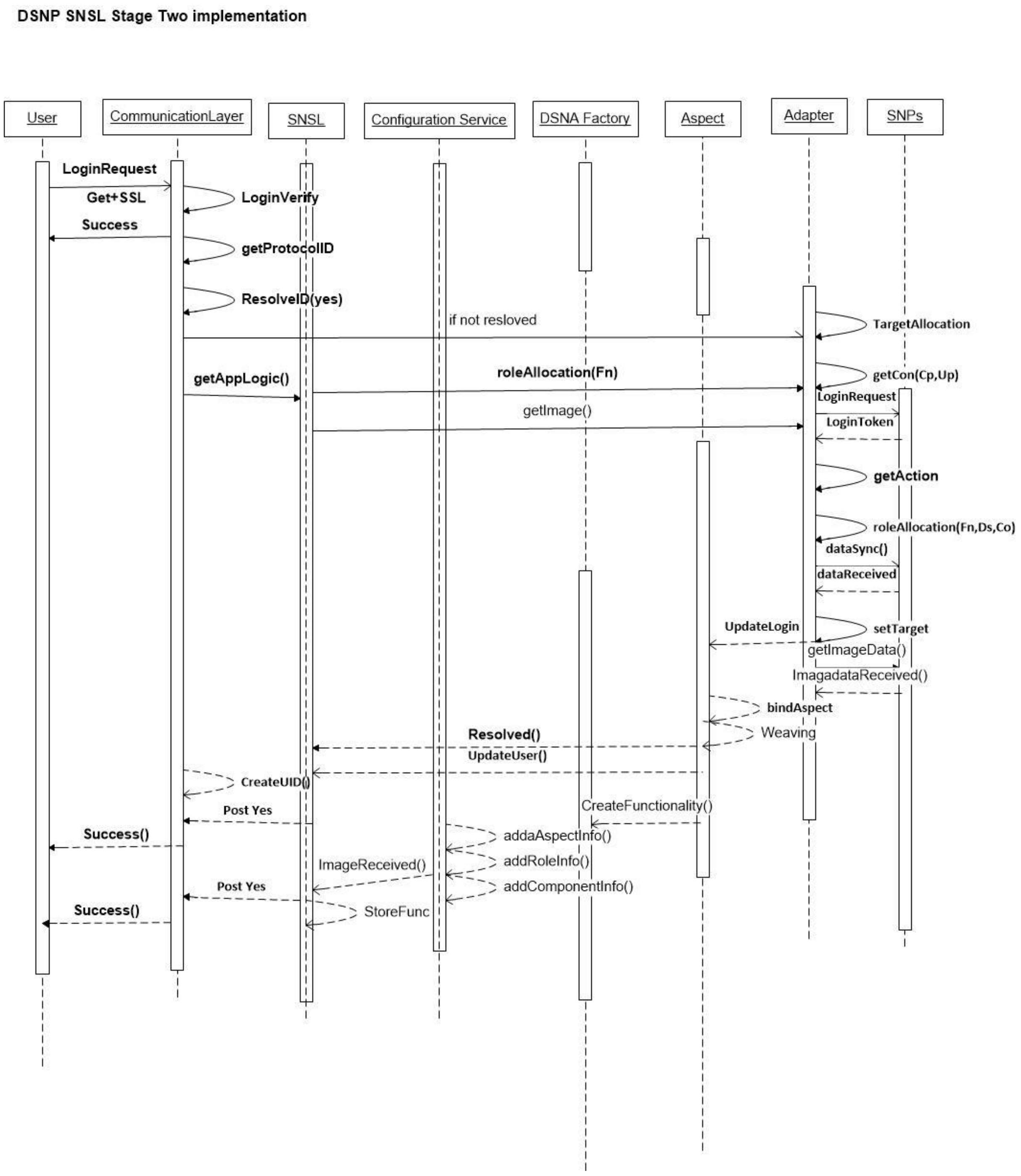
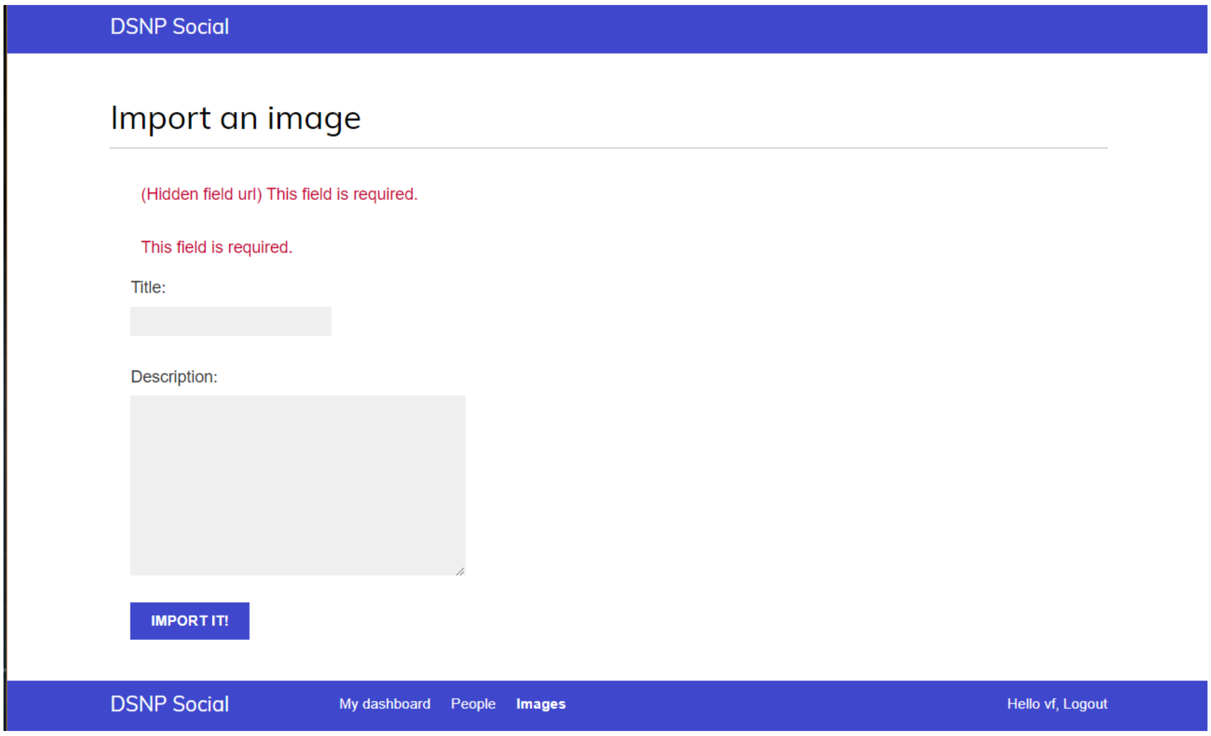


Figure 6-8: SNSL Stage 2 Sequence Diagram

Alice, Bob and Tony are declared as one single entity the "User". In the first step, Alice initiates the request to use DSNP as Alice is not a registered user therefore Alice she either creates new a user account with DSNP, as allowed by the security and privacy policy of the platform or accesses the DSNP with an existing SNP ID. In the next step "Communication Layer" which receives the request looks for the relevant methods of access and security policy. The Protocol Handler call method GetIDProtocl() to know the protocols required and UserID Resolver uses ResolveID() method to give access to the User.


DSNP Social

Import an image

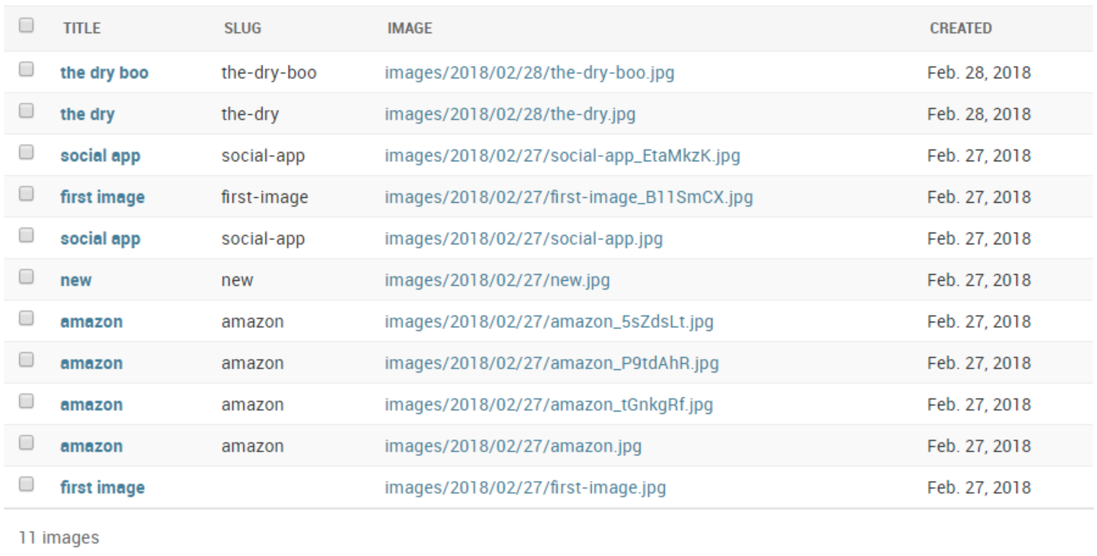
(Hidden field url) This field is required.

This field is required.

Title:

Description:

Figure 6-9: Image import page



TITLE	SLUG	IMAGE	CREATED
the dry boo	the-dry-boo	images/2018/02/28/the-dry-boo.jpg	Feb. 28, 2018
the dry	the-dry	images/2018/02/28/the-dry.jpg	Feb. 28, 2018
social app	social-app	images/2018/02/27/social-app_EtaMkzK.jpg	Feb. 27, 2018
first image	first-image	images/2018/02/27/first-image_B11SmCX.jpg	Feb. 27, 2018
social app	social-app	images/2018/02/27/social-app.jpg	Feb. 27, 2018
new	new	images/2018/02/27/new.jpg	Feb. 27, 2018
amazon	amazon	images/2018/02/27/amazon_5sZdsLt.jpg	Feb. 27, 2018
amazon	amazon	images/2018/02/27/amazon_P9tdAhR.jpg	Feb. 27, 2018
amazon	amazon	images/2018/02/27/amazon_tGnkgRf.jpg	Feb. 27, 2018
amazon	amazon	images/2018/02/27/amazon.jpg	Feb. 27, 2018
first image		images/2018/02/27/first-image.jpg	Feb. 27, 2018

11 images

Figure 6-10: Image Selection

After gaining access Alice want to share some photos. SNSL handles the requests for any social sharing activity. roleAllocaiton(fn) and getImage() are called. Getimage() holds the information about the user request whereas roleAllocation(fn) contains interpretation detail regarding the to be imported functionality. The final step of importing is handled by Adapter and which interprets the information and behaviour of the requested functionality. ImageDataReceived() invokes the aspect and Aspectual Binding operations are called to bind and weave the new feature to the existing DSNP functionality. DSNA Factory recreates the functionality and stores it. Configuration Service add the new information to the component, aspect and role. This information is reused for dynamic composition of the components and functionalities. SNSL stores the abstract of the received information and generates a response to the user Alice with new image sharing features. Figure 6-9 shows a description of the information required from the users to import the images. Figure 6-10 shows the imported images from that used for further sharing with other SNP users.

The next section evaluates the performance for the final evaluation of the DNSP application.

6.4.2. Performance Evaluation

To evaluate the performance of the DSNP prototype an overhead thread has been created. Apache JMeter (jmeter.apache.org. 2018) has been used to create the test bed. Overhead thread is evaluated based on the information provided in the test bed. The test bed is used to evaluate the resource usage, increased data access, increased network load, and increase computation resource.

The test bed is based on SWAT use cases of interaction and communication. Interaction and communication between the multiple users attached to different SNP is analysed. The analysis is carried out on JMeter by creating the SWAT scenario.

6.4.3. Method Selection

This section describes the notion of evaluation in web architecture research to explain the evaluation method selection. The research on evaluation methods for purpose of web architecture evaluation is quite vague and mostly evaluation is done to check the quality and performance of the architecture when implemented as a prototype such as attempted in Lundar et al. (2013) and Laine and Säilä, (2012).

The literature on software architecture evaluation methods is mainly focused on the implementation of methods with different criteria to find the weaknesses. The implementation of evaluation methods such as Kazman et al. (2005), Mattsson et al. (2006) provides detailed information on the analysis of the architecture evaluation methods but lacks in providing any guidelines for the selection of architecture evaluation methods, which makes the selection a cumbersome process.

In the context of design science, according to Venable et al. (2012), the artefact designed using DSR must be rigorously evaluated. But how should rigorous evaluation be designed and conducted? What kind of strategies and methods should be used for the evaluation in a project grounded in design science? How can evaluation be designed effectively and efficiently (Venable et al. 2012). As described in chapter 3, the evaluation should have the following purposes, (1) evaluate an instantiation (2) evaluate the formalised knowledge (3) evaluate a designed artefact by comparing it with formalised knowledge to understand whether it achieve the purpose (4) evaluate designed artefact with purpose to know the consequences of evaluation and finally (5) evaluate the designed artefact formatively to identify weakness and areas of improvement for an artefact under development.

Hevner et al. (2004) proposed utility, quality and efficacy as the key aspects of the architecture to be evaluated. In addition, they proposed, the artefact should be evaluated on the basis of functionality, completeness, consistency, accuracy, usability, reliability and performance furthermore the artefact must be adaptable to the functional environment the artefact is intended for (Hvener et al. 2004). Therefore, in the light of research guidelines on artefact evaluation the SWAT process is chosen, because SWAT can be helpful in evaluating the decentralised and distributed social web application.

6.4.4. The Social Web Acid Test (SWAT)

The W3C federated social web group proposed SWAT to test decentralised application at data levels such as data portability, messaging social discovery etc. The evaluation further simplifies the SWAT use case to demonstrate the data portability and focuses on the interaction and communication use case.

Interaction: According to the SWAT test case, there are 3 platforms running on 3 servers.

- Alice -> SN1 or server 1
- Bob -> SN2 or server 2
- Tony -> SN3 or Server 3

Alice has an account on SN1 and she is working with Bob (SN2) and Andy (SN3) on the same project. She would like Bob and Andy to join her on SN1, so they can stay update on the project related notifications. In the context of SWAT for portability to be successful, Bob and Andy should remain friends with her. If any post on Bob's social dashboard on SN2 should be visible on her social dashboard on SN1 and if any post is done by Alice on SN1 then Bob should get a notification on SN2.

Communication: What are the means of communication in DSNP, provides a consistent form of communication to the users to make their interactions feasible and possible?

Alice has an account on SN1 and she is working with Bob (SN2 and Tony (SN3) on the same project. As now Bob and Andy are already on SN1. Alice initiates a group chat and she adds

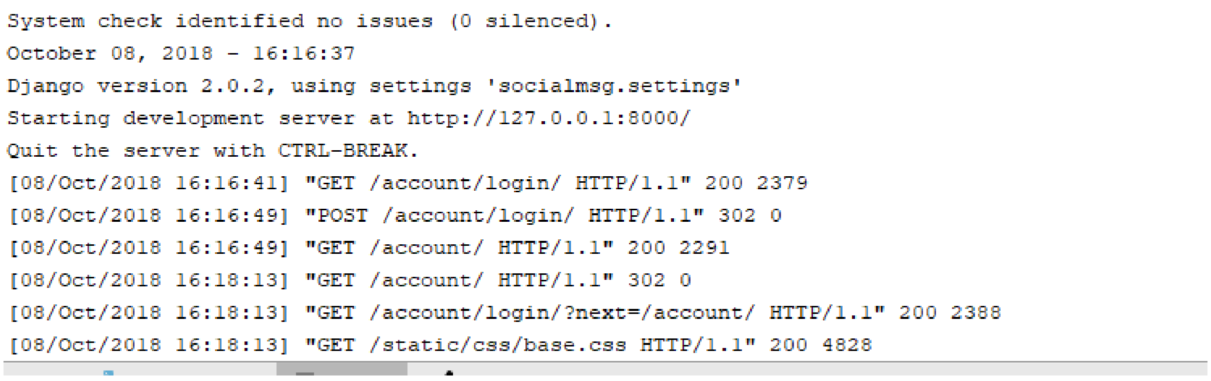
Bob and Andy. Now when carol sends message to Bob and Andy they should receive the message while they are using SN2 and SN3.

The purpose of above-mentioned use cases is to give attributes to test quality and performance of interaction and communication.

The test bed to evaluate the application consist of multiple scenarios as explained in iteration 2 section 5.1, each scenario running the application for max 50 users for the duration of 30 seconds.

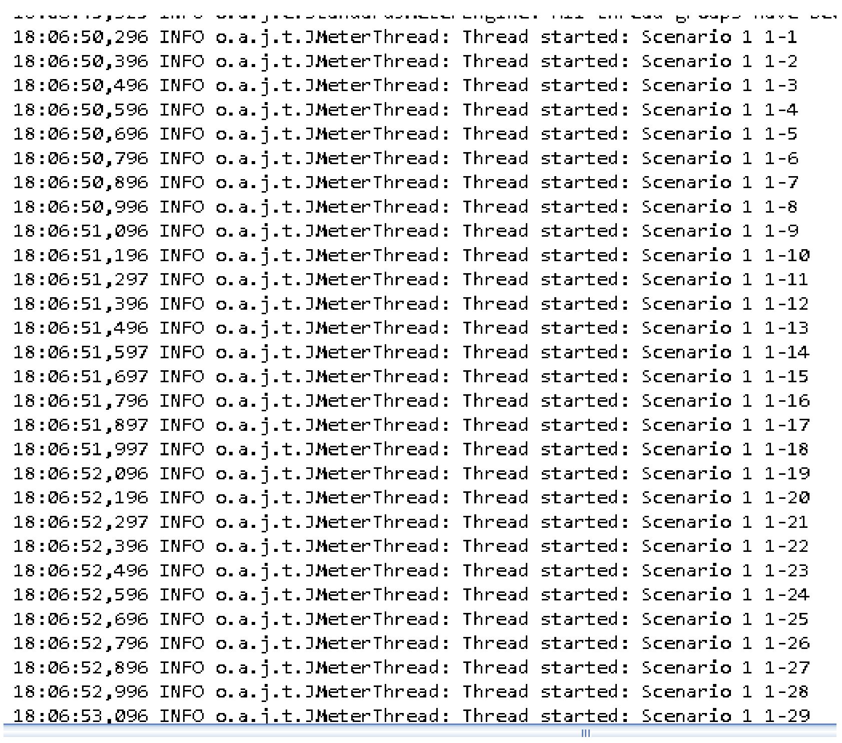
6.4.5. Performance analysis

The runtime overhead DSNP is evaluated in terms of resource usage, increased data access, increased network load, and increase computation resource. The application files and log file are loaded at the start up time of the application. In case of data access, which is initiated when application load is successful. Latency is the main performance bottleneck because its magnitude is larger than other evaluation matrices.



```
System check identified no issues (0 silenced).
October 08, 2018 - 16:16:37
Django version 2.0.2, using settings 'socialmsg.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[08/Oct/2018 16:16:41] "GET /account/login/ HTTP/1.1" 200 2379
[08/Oct/2018 16:16:49] "POST /account/login/ HTTP/1.1" 302 0
[08/Oct/2018 16:16:49] "GET /account/ HTTP/1.1" 200 2291
[08/Oct/2018 16:18:13] "GET /account/ HTTP/1.1" 302 0
[08/Oct/2018 16:18:13] "GET /account/login/?next=/account/ HTTP/1.1" 200 2388
[08/Oct/2018 16:18:13] "GET /static/css/base.css HTTP/1.1" 200 4828
```

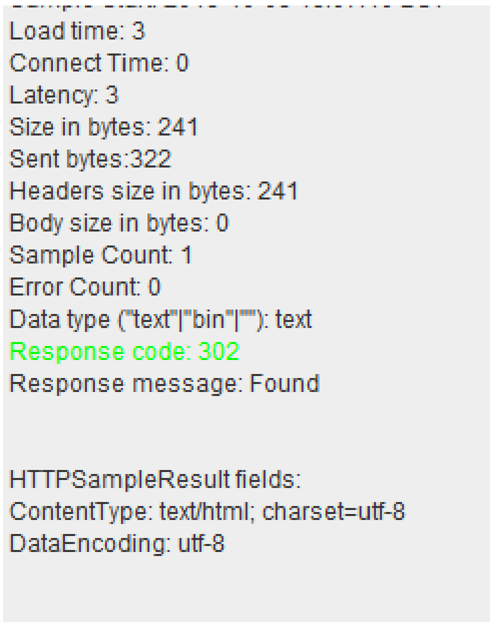
Figure 6-11: Application load test



```
18:06:50,296 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-1
18:06:50,396 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-2
18:06:50,496 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-3
18:06:50,596 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-4
18:06:50,696 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-5
18:06:50,796 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-6
18:06:50,896 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-7
18:06:50,996 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-8
18:06:51,096 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-9
18:06:51,196 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-10
18:06:51,297 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-11
18:06:51,396 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-12
18:06:51,496 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-13
18:06:51,597 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-14
18:06:51,697 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-15
18:06:51,796 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-16
18:06:51,897 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-17
18:06:51,997 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-18
18:06:52,096 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-19
18:06:52,196 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-20
18:06:52,297 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-21
18:06:52,396 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-22
18:06:52,496 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-23
18:06:52,596 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-24
18:06:52,696 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-25
18:06:52,796 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-26
18:06:52,896 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-27
18:06:52,996 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-28
18:06:53,096 INFO o.a.j.t.JMeterThread: Thread started: Scenario 1 1-29
```

Figure 6-12: Application scenario load test

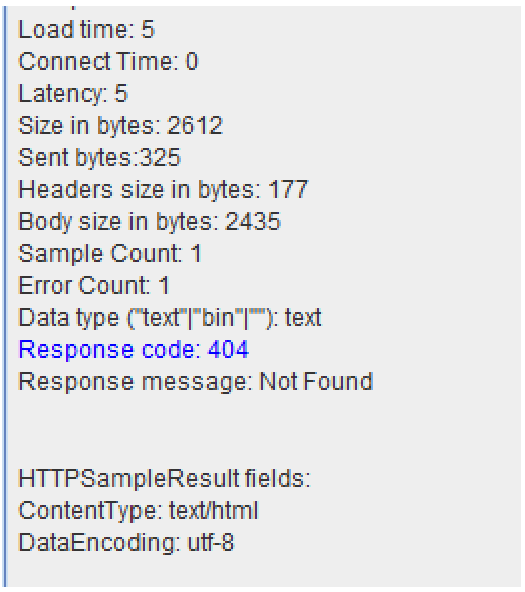
Figures 6-11 and 6 -12 shows the successful application load for test bed and scenarios. After data access the next important overhead thread is network overhead. The role of SNSL middleware evaluation is important to calculate the right result. CAM components such as the AspectBinding, AspectWeaving and AdviceBinding methods cause increase in latency of data access and increase network response time as compared to other DSNA methods. The cause of this significant increase is in the use of external services and APIs used for interaction with SNPs.



```
Load time: 3
Connect Time: 0
Latency: 3
Size in bytes: 241
Sent bytes:322
Headers size in bytes: 241
Body size in bytes: 0
Sample Count: 1
Error Count: 0
Data type ("text"|"bin"|""): text
Response code: 302
Response message: Found


HTTPSampleResult fields:
ContentType: text/html; charset=utf-8
DataEncoding: utf-8
```

Figure 6-13: Successful Test result

For example, figure 6-13 shows the successful attempt to deploy the social service and figure 6-14 shows the failed attempt to deploy the social service.



Load time: 5
Connect Time: 0
Latency: 5
Size in bytes: 2612
Sent bytes:325
Headers size in bytes: 177
Body size in bytes: 2435
Sample Count: 1
Error Count: 1
Data type ("text"|"bin"|""): text
Response code: 404
Response message: Not Found

HTTPSampleResult fields:
ContentType: text/html
DataEncoding: utf-8

Figure 6-14: Test to connect the social services

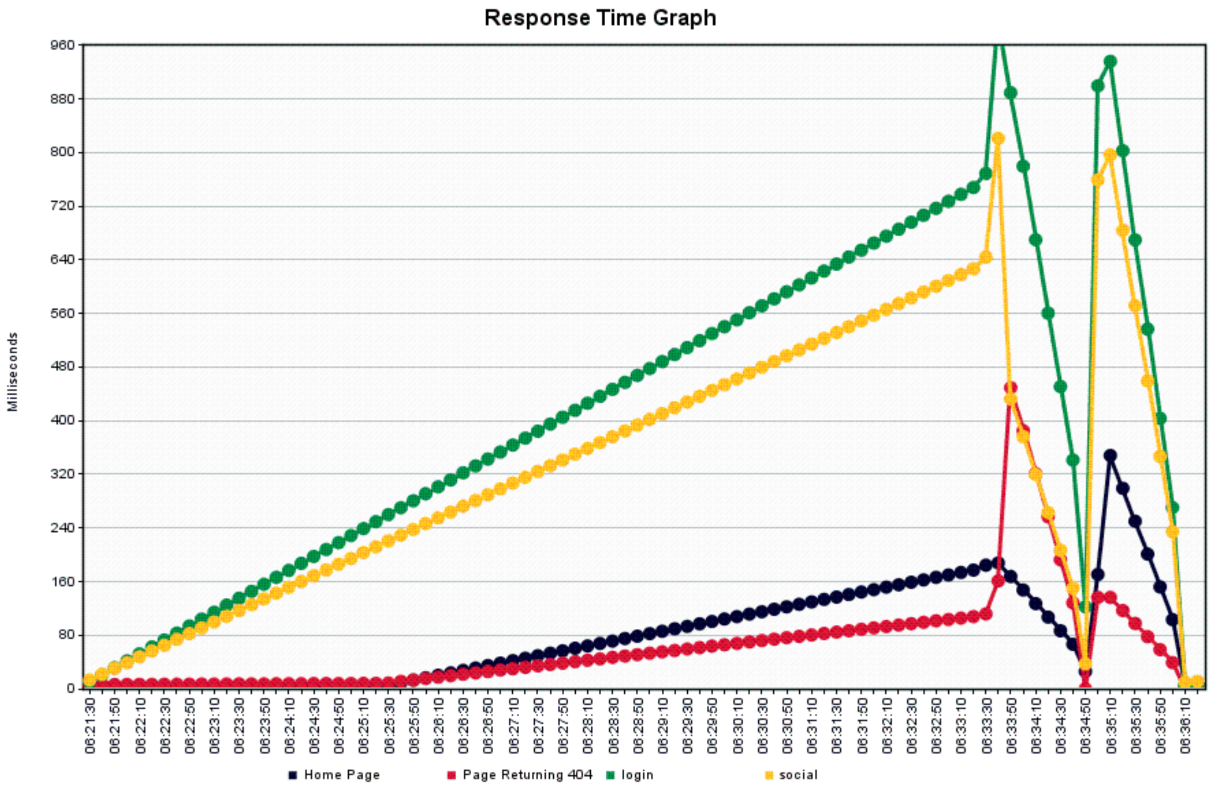
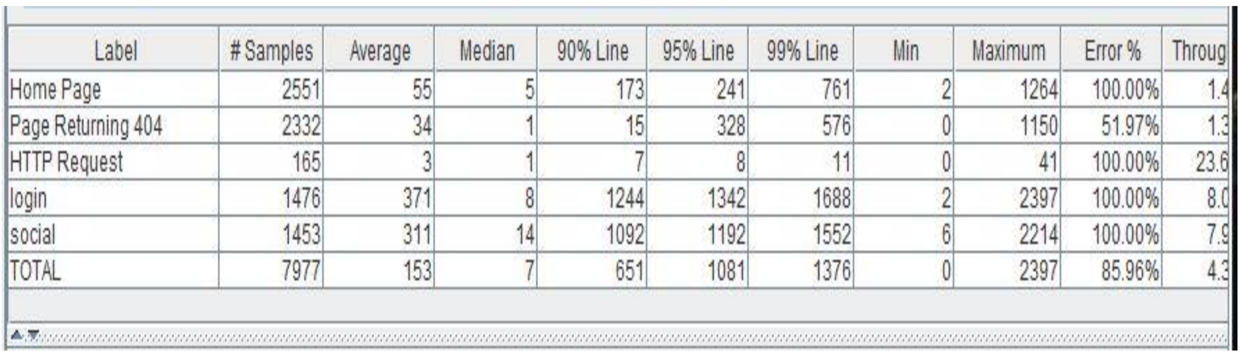


Figure 6-15: Network response time

The graph indicates that as the number of users accessing the application increases the network response time increases as well.



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throug
Home Page	2551	55	5	173	241	761	2	1264	100.00%	1.4
Page Returning 404	2332	34	1	15	328	576	0	1150	51.97%	1.3
HTTP Request	165	3	1	7	8	11	0	41	100.00%	23.6
login	1476	371	8	1244	1342	1688	2	2397	100.00%	8.0
social	1453	311	14	1092	1192	1552	6	2214	100.00%	7.9
TOTAL	7977	153	7	651	1081	1376	0	2397	85.96%	4.3

Figure 6-16: Table showing the overall results.

The domain SNSL includes various algorithms, used for weaving and composition of the components, for example the Adapter. There is a significant increase in the overall access and deployment time of the Adapter during the allocation process. Component composition and the role of Adapter is crucial. Some appropriate measure has to be taken to reduce the overhead time, for example by using better weaving algorithms.

6.5. Chapter Conclusion

In the previous iterations design and implementation of DSNA has evolved the DNSA to be practical and implementable. Such was the goal of this chapter, to demonstrate the results of DNSA when evaluated on reality based scenario. The main goal to achieve is implementation of the SNSL stage two to complete the research process cycle. This chapter proposed a new addition to SNSL the solve the problem of data inconsistency and persistence. DSNA Factory and Component configuration service are introduced. The final application behaviour is explained, and images are successfully imported from another SNP to DSNP.

The characteristic and aspects of the evaluation on the basis of which prescribed DSNA enabled platforms are evaluated. This laid the foundation for the selection of the evaluation method for the final version of DSNA. In the next step a multiple, social networking domain scenario is illustrated to test the scenario and SWAT v1 is used. Which is suggested by the W3C federated social web group. The test use cases are applied to validate the practicality of the DSNA interaction between the user and communication between the platforms. The evaluation of the artefact reflects that the DSNA is practical however more large-scale implementations are required to improve and understand the weaknesses of DSNA. The performance evaluation focuses on the issues related to data access and network usage.

More efficient algorithms should be used for component composition to avoid the data access and composition related issues.

Chapter 7 Discussion and Conclusion

7. Introduction

This chapter summarises the contribution made by this research. A decentralised social network architecture is presented to resolve the problem of portability between different social networking platforms. The architecture is based on a conceptual architecture and architectural style. This chapter discusses, the usefulness of the architecture implementation and results achieved by the evaluation to conceive the final version of the architecture. The remainder of the chapter is organised as follows. Section 7.1 provides an overview of this research and what has been achieved in each chapter. Section 7.2 describes the most important contributions of this research. Section 7.3 describes what objectives have been achieved and as well as the limitations of the current work and plans for future work.

7.1. Research Overview

The research presented in this thesis addresses new challenges and opportunities for the decentralisation in social networking platforms, given by the lack of architectural guidelines, infrastructure, protocols, standards and service provider restrictions. The available research on social web decentralisation is mainly conducted on user privacy, profile data portability, activity and identity related issues. There are three main approaches widely proposed by existing research to decentralise the social web. These include distributed web server hosting, federated layer and P2P approaches. The majority view align itself with the federation of social networking platforms, which is still underdeveloped and is opposed by social network service providers.

The solution envisioned in this research attempts to solve the problem of data portability among social networks at the functional level by using a decentralisation approach. The methodology used to develop decentralised architecture, uses similar standards and protocols to existing architectures, however, it differs on the principles related to, whether decentralisation should be done at the central level such as in the Federated Social Web, which is widely explored or at the functional level, which is under investigated. By using the proposed architecture users will be able to decide which functionality they would like to use across their social network platforms in other words if a user decided to use message related functions then they would able to post to another platform they are registered with.

7.2. Research Achievements

The application of CAM to build web applications is now new (Pinto et al. 2005, Pessemier et al. 2008, Pinto et al 2011, Fuentes et al. 2003), but the use of CAM, AOSD and CBSD for the decentralisation of social web has not been explored by the research community. In the researcher's view the problem with existing social network decentralisation is much broader than normally explained in the literature, and that architectural styles principles and patterns from software engineering play a fundamental role. Adding and mixing some opensource technologies for decentralisation is not sufficient to resolve the problem. Therefore, this research presents a decentralised architecture based on software engineering principles with the goal to propose a way to design and develop decentralised social networking platforms. More specifically;

1. **Use of CAM for SN**

The integration of CAM with very specific component-based architecture PACE, gives the required architectural elements needed for decentralisation. The integration of CAM and PACE provides the model, DSNA style which sets the foundations for DSNA.

2. **DNSA Style**

The DSNA style provides rule and properties on which every component of the DSNA are built. The most important aspect of the style is that its foundation is based on aspect and component composition.

3. **Component Composition**

DNSA supports role-based task division of composition and specification into separate components. The AO composition and role-based division is key to the decentralisation of SNP at the functional levels.

4. **SNSL as middleware**

DNSA is made of four layers of which the social network support layer is the most important. Since it not only acts as bridge between the other SNPs and DNSP but also because this is where the actual runtime composition happens. Adapter, DSNA factory and component configuration services, also represent key architectural elements.

The DNSA based prototype shows promising results but with some setbacks as well. The separation of concerns, AO composition and component customisation based on the proposed feature of the SNSL, successfully decentralised the SN functionalities.

In the analysis of all the iterations, it has been concluded that the addition of AO features to the composed new functionalities is useful. Specifically, the AO functions such as aspect binding and weaving are crucial to change the code.

It has also been concluded that existing AO compositions lacks in methods availability for robust composition, expressive distributed composition and appropriate task division and adaptability of composition logic. These are some of the reasons that have made the current research an immense challenge. SNSL was proposed as middleware to handle some of these challenges. Within this context DSNP communication, interaction, composition and allocation of component further expanded the number of challenges.

To fill these gaps and to achieve all the set objectives a SWAT design scenario and W3C federated social web protocols were used and they played a crucial role in establishing the communication between the DSNP and other SNPs. W3C provides social data syntax also known as JSON, social APIs and other federation protocol. For the sake of implementation these protocols are the part of implementation framework but how they are used is for the user to select. That is one main feature of the DNSA based application. Also, this is the main difference between W3C propositions on their protocols and current research. As compared to their implementation on other peer to peer and distributed applications the DSNA based application runs them dynamically based on the user requirements as prescribed in the component.

7.3. Research Contributions

Novel Contributions to Knowledge

The main contribution of this thesis is an architectural framework, which provides cross-domain social networking functionality to the end users by enabling data portability between different social networking platforms. The proposed architecture addresses new requirements that arise from the recent demands of change in the architecture of the social networking platforms. (1) the shift from a centralised to more open and decentralised architecture and (2) the need of infrastructure that can enable sharing of personal data between different social networking platforms.

List of Main Contributions

Based on DSR, the individual iterations of the research make the following contributions.

DSNA Architectural Style

The most important aspect of the proposed research is the composition of DSNA architectural style. In simple words, the purpose of the DSNA architectural style is to provide rules for relationships between the components of the DSNA. The proposed style is novel and is based on strong and well-established principles that are adopted from the component-based style such as PACE and Aspect Oriented Software development (AOSD) and unified way to obtain them to obtain an architectural style that can solve the complex portability issues related to decentralisation of the social networking platforms.

Contribution

The notion behind the implementation of the DSNA style is to provide guiding principles for the composition of the components during the implementation of the DSNA architecture. It is imperative for the successful implementation of DNSA that the components of the architecture are composed dynamically following the DSNA style.

The proposed research contributes to the software engineering knowledge domain by comparing the CAM and PACE. This comparison brought up components and ideas for the production of decentralised social networking application.

Conceptual Architecture of Decentralised Social Networking Platforms

The conceptual architecture proposed in this research provides guidelines to perspective developers when designing and implementing a decentralised social networking platform. The architecture follows rules set in the DSNA architectural style and specific requirements of decentralisation. The proposed architecture is novel as it provides decentralisation at the functional level of the social networking platforms; moreover, the architecture provides guidelines for the dynamic composition of the components.

Contribution:

- The proposed architecture provides a list of activities and guidelines for leveraging data portability at the functional level of the social networking platforms
- The architecture provides a list of high-level components required to implement decentralisation, to enable data sharing and aggregation between different social networking platforms.

Prototype of DSNA

The outcome of the architecture is shown in the form of a prototype. To verify the practicality of the architecture, the prototype is designed and implemented using different content sharing functional scenarios. The scenarios are designed to verify the social interactions and communication between the different social networking platforms. The prototype implementation shows the protocols, standards used, and the different architectural components used to realise the scenario. The prototype of the proposed architecture shows that the architecture presented in this research is practical and can be implemented as an application in the suitable setting.

Contribution:

- The prototype implementation shows how the proposed architectural components can be used to enable data portability.
- The prototype shows how to enable profile independent content sharing between different social networking platforms.
- The prototype shows how to instantiate the DSNA architectural style and proposed conceptual architecture to enable decentralisation.

SNSL as Middleware

Social network support layer is a DSNA component where dynamic composition of component happens. To achieve the composition a modified Adapter pattern is proposed for DSNP prototype. The novel contribution of which will be the adapter algorithm that allocate the role based on the behavioural description of the SNP functionality and plays an important role for importing the functionality to the DNSP prototype.

- The first stage of SNSL shows the profile reusability by means of a DSNP prototype
- The second stage of SNSL shows successful portability of data

7.4. Conclusion and Future Challenges

The scope of future software development is changing from small network computing to large distributed networks. The decentralised nature of the development and deployment and execution of the systems caused the change in the nature of how the systems are developed. Customisation of large distributed systems such as social networks by means of decentralisation and the proposed platform should be attempted at a large scale with better composition technologies and algorithms.

There is a need to further investigate the adaptation and customisation capabilities of the SNSL middleware. The reliance on the existing open source code for aspectual weaving, binding and advice is not reliable. The outcomes of the adapter algorithm are very close to the expected results, however there are issues with the information and data interpretation at the role allocation levels. The final evaluation shows some of the deficiencies in adaptation and allocation as well as in the effects on composition, which result in issues related to data access.

Privacy and security policies induction into the DSNP is another future challenge that requires attention. A recurring and unsolved problem is SNP's privacy and security policies data import to DSNP. Such policies are not standardised. All SNPs have their own privacy and security management framework user are allowed only to change at their end, but they are not allowed to import their own setting. DNSA provides some initial features related to trust, security and privacy policy of the users. For example, user can select and customise a functionality and share it with another SNP.

Lastly, restricted interfaces and restriction on the queries that are deployed using external APIs is another challenge. External SNPs put restriction on the number of queries a user can make to access their data. This limitation also effects the DNSP features when importing data from the SNPs. There is a need of a mechanism or agreement between a person using DSNP and SNSP to resolve this issue and to have an interface, whereby user can describe their current needs, when connecting to use social networking services.

References

Anderson, G.E., Graham, T.N. and Wright, T.N. (2000). 'Dragonfly: linking conceptual and implementation architectures of multiuser interactive systems', Proceedings of the 22nd International Conference on Software Engineering (ICSE'00), New York, NY: ACM Press, pp.252–261

Alhir, S. (2002), 'Guide to applying the UML', Springer. ISBN 0-387-95209-8.

Auer, S. (2010), 'A Methodology for Enabling Social Semantic Collaboration', Social Computing: Concepts, Methodologies, Tools, and Applications. IGI Global, 2010, 669-692

Ankolekar, A., Krotzsch, M., Tran, T., and Vrande, D. (2008).'The Two Cultures: Mashing up Web 2.0 and the Semantic Web', The Journal of Web Semantics, 6(1), 70–75.

Aiello, L.M. and Ruffo, G., 2010, March. Secure and flexible framework for decentralized social network services. In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on (pp. 594-599). IEEE.

Arenas, M. & P_erez, J. (2011). 'Querying semantic web data with sparql. In Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems', PODS '11 (pp. 305{316). New York, NY, USA:
ACM.

Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M. and Steggles, P., (1999), 'Towards a better understanding of context and context-awareness'. In Handheld and ubiquitous computing (pp. 304-307). Springer Berlin Heidelberg.

Ameller, D. and Franch, X., (2011), 'Ontology-based Architectural Knowledge representation: structural elements module', In Advanced Information Systems Engineering Workshops (pp. 296-301). Springer Berlin Heidelberg.

Abrahamsson, P., Warsta, J., Siponen, M.T. and Ronkainen, J., (2003), 'New directions on agile methods: a comparative analysis', In Software Engineering, 2003. Proceedings. 25th International Conference on (pp. 244-254). IEEE.

Adipat, B., Zhang, D., and Zhou, L. (2011), 'The effects of tree-view based presentation adaptation on mobile web browsing', *MIS Quarterly*, Vol 35 (1), pp. 99-122.

Bass, L., Clements, P. Kezman, R. (2011). 'Software Architectures in Practice' Carnegie Mellon Software Engineering Institute, 2nd Edition, Pearson Education, Boston, MA. US

Ballester, A., Jordan, F., & Pujol, H (2010). 'A semantic approach to security in social networks', Safelayer Secure Communication and Centre of Development and Industrial Technology, Spain. Serial Number E-08039

Berners-Lee, T., Cailliau, R., Groff, J. (1992). 'The World-Wide Web', Computer Networks 25(4-5): 454-459 (1992)

Berners-Lee, T., Hendler, J., And Lassila, O. (2001). 'The Semantic Web', Scientific American, 17 May 2001

Berners-Lee, T., Hall, W., Hendler, J. a., O'Hara, K., Shadbolt, N., & Weitzner, D. J. (2006). 'A Framework for Web Science', 1(1), 1–130. doi:10.1561/1800000001

Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., & Hendler, J. (2007). 'N3Logic: A Logical Framework for the World Wide Web', Theory and Practice of Logic Programming, 8(03), 249–269. Retrieved from http://arxiv.org/abs/0711.1533

Berners-Lee, T., Hollenbach, J., Lu, K., Presbrey, J., Schraefel, M (2008). 'Tabulator Redux: Browsing and writing linked data', In Proceedings of the 1st Workshop on Linked Data on the Web (2008)

Berners-Lee, T. (2009). 'Socially Aware Cloud Storage', W3C Design Note, Available at: http://www.w3.org/DesignIssues/CloudStorage.html, Accessed: 20-12-2015

Béjar, R., Latre, M. A., Nogueras-Iso, J. Muro-Medrano, P. R. and Zarazaga-Soria, F. J. (2009). 'An Architectural Style for Spatial Data Infrastructures', International Journal of Geographical Information Science, 23(3), pp.271-294. doi:10.1080/13658810801905282

Bianco, P. (2007), 'Evaluating A Service Oriented Architecture', SEI Technical Report

Bojars,U., Passant, A., Breslin, J., Decker,S. (2008). 'Social Network and Data Portability using Semantic Web Technologies', DERI, NU Ireland, Galway (1), 5–19.

Bortoli, S., Palpanas, T., Bouquet, P. (2011). 'Decentralised Social Network Management', International journal of Web based communities, 7(3), pp.276-297.

Bizer, C. & Schultz, A. (2009). The Berlin SPARQL Benchmark. International Journal On Semantic Web And Information Systems (IJSWIS), 5(2),

Boyd, D., M. and Ellison, N., B. (2007), 'Social network sites: definition, history, and scholarship', Journal of Computer-Mediated Communication 13: 210–230.

Booch, G. (2001).' The Architecture of the web application', http://www.ibm.com/developerworks/ibm/library/it-booch_web Access Date, 20/11/2012

Boehm, B. Turner, R. (2005), 'Management Challenges to Implement Agile Processes in Traditional Development Organizations', IEEE Software (22)5, 2005, pp. 30-39.

Bonifati, A., Chrysanthis, P.K., Ouksel, A.M. and Sattler, K.U., (2008). 'Distributed databases and peer-to-peer databases: past and present'. ACM SIGMOD Record, 37(1), pp.5-11.

Buchegger S, Schi¨oberg D, Vu L-H, Datta A, (2009), 'Peerson: P2P Social Networking: Early Experiences and Insights'. In: Proceedings of the second ACM Euro Sys workshop on social network systems, SNS '09, pp 46–52

Breslin, J., Harth, A., Bojars, U., & Decker, S. (2005). 'Towards Semantically-Interlinked Online Communities'. In Proceedings of the 2nd European Semantic Web Conference (ESWC05), Heraklion, Greece, LNCS, 3532, 500-514

Brown, A. W., and Mcdermid, J. A. (2007). 'The Art and Science of Software Architecture', International Journal of Cooperative Information Systems, 16(03n04), 439–466 doi:10.1142/S0218843007001718

Burstein, F. and Gregor, S., (1999), 'The Systems Development or Engineering Approach To Research In Information Systems', An Action Research Perspective. In Proceedings of the 10th Australasian Conference on Information Systems (pp. 122-134). Victoria University of Wellington, New Zealand.

Bahri, L., Carminati, B. and Ferrari, E., (2018), Decentralized privacy preserving services for Online Social Networks, Online Social Networks and Media, 6, pp.18-25.

Brickley, D., Miller. L. (2007), 'FOAF Vocabulary Specification', Available at: http://xmlns.com/foaf/spec/. Access Date, 14-2-2014

Breslin, J. and Decker, S. (2007). 'The Future of Social Networks On The Internet': The Need For Semantics, IEEE Internet Computing, pp: 86 – 90

Brambilla, M., Ceri, S., Fraternali, P., and Manolescu, I. (2006). 'Process modelling in Web applications', ACM Transactions on Software Engineering and Methodology, 15(4), 360–409. doi:10.1145/1178625.1178627

Brown, S. A., Dennis, A. R., & Venkatesh, V. (2010). Predicting Collaboration Technology Use: Integrating Technology Adoption and Collaboration Research. Journal of Management Information Systems, 27(2), 9–54.

Bojars, U., Breslin, J. G., Finn, a, & Decker, S. (2008).'Using the Semantic Web For Linking And Reusing Data Across Web 2.0 Communities', The Journal of Web Semantics, Retrieved from http://www.sciencedirect.com/science/article/B758F-4R8MDS6-1/2/339e88a4ebcde208f23e0332bbdbc725

Bosch, J., (2000), 'Design & Use of Software Architectures – Adopting and Evolving A Product-Line Approach', ISBN 0-201- 67494-7, Pearson Education, 2000.

Breslin, J., Passant, A. and Decker, S., (2009), 'The Social Semantic Web', 20 Springer Science & Business Media.

Brichau, J., Chitchyan, R., Rashid, A. and D'Hondt, T., (2008), 'Aspect-Oriented Software Development: An Introduction. Wiley Encyclopedia of Computer Science and Engineering.

Bai, X., White, D. and Sundaram, D., (2013), 'Multi-Methodological Approaches in Design Science', A Review, Proposal and Application. PACIS, 159.

Creswell, J. W. (2005), 'Educational research', Planning, conducting, and evaluating quantitative and qualitative research, 2nd edition.

Cao, L., Mohan, K., Xu, P. and Ramesh, B., 2009. A Framework For Adapting Agile Development Methodologies. European Journal of Information Systems, 18(4), pp.332-343.

Cutillo, L.A., Molva, R. and Önen, M., (2011), June. Safebook: A Distributed Privacy Preserving Online Social Network. In World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a (pp. 1-3). IEEE.

Conboy, K., Gleasure, R. and Cullina, E., (2015), 'Agile Design Science Research', In International Conference on Design Science Research in Information Systems (pp. 168-180). Springer International Publishing.

Chen, L., Wei, S. and Qingpu, Z., (2010), 'Semantic Description of Social Network Based on Ontology', E-Business and E-Government (ICEE), 2010 International Conference on, 1936–1939. doi:10.1109/ICEE.2010.489

Conti, M., Hasani, A., & Crispo, B. (2013), 'Virtual private social networks and a facebook implementation', ACM Transactions on the Web, 7(3), 1–31, doi:10.1145/2516633.2516636

Calegari, S. Pasi, G. (2013), 'Personal ontologies: Generation of user profiles based on the YAGO ontology', Journal of Information Processing and Management, Volume 49 Issue 3, May, 640 – 658

Catanese, S.A., De Meo, P., Ferrara, E., Fiumara, G. and Provetti, A., (2011), 'Crawling Facebook For Social Network Analysis Purposes'. In Proceedings of The International Conference On Web Intelligence, Mining And Semantics (p. 52). ACM.

Constantindes, E., and Fountain, J.S. (2008), 'Web 2.0. Conceptual Foundations and Marketing Issues', Journal of Direct Data and Digital Marketing Practice, pp. 231 – 244

Catanese, S., De Meo, P., Ferrara, E., Fiumara, G. and Provetti, A., (2012). 'Extraction and analysis of facebook friendship relations.' In Computational Social Networks (pp. 291-324). Springer London.

Chen, D., Doumeingts, G. and Vernadat, F., (2008). 'Architectures for enterprise integration and interoperability': Past, present and future. Computers in industry, 59(7), pp.647-659.

Chen, W., Wang, Y. and Yang, S (2009).'Efficient Influence Maximization in Social Networks', In Proceedings of The ACM Conference On Knowledge Discovery And Data Mining
Conallen, J. (1999), 'Building Web application with UML', Rational Software Corp, pp 133-160, Addison Wesley, Boston, MA, USA

Clements, P. (2003), 'Documenting Software Architectures', Views and Beyond, SEI Series in Software Engineering Addison-Wesley

Challenger, M. (2012), 'The Ontology and Architecture for an Academic Social Network', International Journal of Computer Science, 9(2), 22–27.

Cena, F., Dattolo, A., Lops, P., & Vassileva, J. (2013), 'Perspectives in Semantic Adaptive Social Web', ACM Transactions on Intelligent Systems and Technology, 4(4), 1–8, doi:10.1145/2501603

Crocker, D. (2009),'Internet mail architecture', W3C Network Working Group

Cadima, R., Ferreira, C., Monguet, J., Ojeda, J. and Fernandez, J., (2010). 'Promoting Social Network Awareness: A Social Network Monitoring System'. Computers & Education, 54(4), pp.1233-1240.

Chowdhury, S.R., Roy, A.R., Shaikh, M. and Daudjee, K., (2015), 'A Taxonomy of Decentralized Online Social Networks', Peer-To-Peer Networking And Applications, 8(3), pp.367-383.

Coulouris, G.F., Dollimore, J. and Kindberg, T., (2011), 'Distributed systems: concepts and design'. Pearson education.

Datta A, Buchegger S, Vu L-H, Rzadca K, Strufe T., (2010), 'Handbook of Social Network Technologies and Applications'. Decentralized Online Social Networks. Springer

Cadima, R., Ferreira, C., Monguet, J., Ojeda, J. and Fernandez, J., (2010). 'Promoting social network awareness: A social network monitoring system'. Computers & Education, 54(4), pp.1233-1240.

Cuppens, F., Cuppens-Boulahia, N. and Vina, E.P., (2012), 'April. Adaptive Access Control Enforcement In Social Network Using Aspect Weaving', In International Conference on Database Systems for Advanced Applications (pp. 154-167). Springer, Berlin, Heidelberg.

Cross, R., Parker, A., Prusak, L., & Borgatti, S. P. (2001). Knowing What We Know: Supporting Knowledge Creation and Sharing In Social Networks. Organizational Dynamics, 30, 2, 100–120.

Dooren, M.V., Lagaisse, B. and Joosen, W., (2013). 'Modularity and Variability of Distributed Software Architectures through Multi-view Refinement of AO-Connectors'. In Transactions on Aspect-Oriented Software Development X (pp. 109-147). Springer Berlin Heidelberg.

Chicaiza, J., López, J., Piedra, N., Martínez, O., & Tovar, E. (2010), 'Usage Of Social And Semantic Web Technologies To Design A Searching Architecture For Software Requirement Artefacts', IET Software, pp.4(6), 407, doi:10.1049/iet-sen.2010.0046

Celino, I., Dell'Aglio, D., Valle, E.D., Balduini, M., Huang, Y., Lee, T., Kim, S.H. and Tresp, V., (2011). 'Bottari: Location based social media analysis with semantic web'. ISWC.

García-Castro, R., Gómez-Pérez, A., Muñoz-García, Ó. and Nixon, L.J., (2008), 'Towards A Component-Based Framework For Developing Semantic Web Applications'. In the Semantic Web (Pp. 197-211). Springer Berlin Heidelberg.

Cardoso, J. (2007). 'The Semantic Web Vision': Where Are We? IEEE Intelligent Systems, 22, 84{88.

Cunha, L. M. & de Lucena, C. J. P. (2006),' Cluster The Semantic Web Challenges Applications: Architecture and Metadata Overview'. Technical report, Ponti_cia Universidade Catolica do Rio de Janeiro

Dijkstra, Edsger W., (1982). 'On the role of scientific thought'. Selected writings on Computing: A Personal Perspective. New York, NY, USA: Springer-Verlag. pp. 60–66. ISBN 0-387-90652-5.

Duclos, F., Estublier, J. and Morat, P., (2002), 'Describing and Using Non-Functional Aspects In Component Based Applications', In Proceedings Of The 1st International Conference On Aspect-Oriented Software Development (Pp. 65-75). ACM.

Datta, A., Buchegger, S., Vu, L.H., Strufe, T. and Rzadca, K., 2010. Decentralized online social networks. In Handbook of Social Network Technologies and Applications (pp. 349-378). Springer, Boston, MA.

Dadzie, A.-S. and Rowe, M. (2011), 'Approaches to Visualising Linked Data': A Survey. Semantic Web, 2(2), 89{124.

Dasgupta, S. (2010), 'Social Computing: Concepts, Methodologies, Tools, and Applications' IGI Global, USA.

Duong, T.H., Uddin, M.N. and Jo, G.S., (2009), 'Collaborative Web For Personal Ontology Generation And Visualization For A Social Network', The 1st International Conference On Knowledge And Systems Engineering, 237–242. doi:10.1109/KSE.2009.32

Decker, S., Bernardi, A., Van Elst, L., Grimnes, G., Groza, T., Handschuh, S., Jazayeri, M., Mesnage, C., Moeller, K., Reif, G. and Sintek, M (2010),'The Social Semantic Desktop' A New Paradigm Towards Deploying The Semantic Web On The Desktop, IGI Global, Doi:10.4018/978-1-60566-112-4.Ch012

Ellis, T. J., and Levy, Y. (2008), 'Framework of problem-based research: A guide for novice researchers on the development of a research-worthy problem', Informing Science, The International Journal of an Emerging Trans-discipline,

Ellis, T.J. and Levy, Y., (2010), A guide for novice researchers: Design and development research methods. In Proceedings of Informing Science & IT Education Conference (InSITE) (pp. 107-118). https://en.oxforddictionaries.com/definition/design

Fielding, R. T., & Taylor, R. N. (2002), Principled Design of the Modern Web Architecture

Ferreira, J. C., & Porflirio, F. P. (2009), Academic Ontology to Support the Bologna Mobility Process, International Conference on Adaptive Science and Technology 2009

Ferrara, E. (2012), 'Community structure discovery in Facebook', International Journal of Social Network Mining, 1(1), 67, doi:10.1504/IJSNM.2012.045106

Foaf-Project.org, (2013),'Friend of a Friend Project' Available at:http://www.foaf-project.org/ (Accessed: 20-10-2013)

Fielding, R. T (2000), ‘Principled Design of the Modern Web Architecture’, ACM Transactions on Internet Technology (TOIT), Volume 2 Issue 2, May 2002

Fazio, M., Celesti, A., Villari, M. and Puliafito, A., (2014), ‘August. Resource Management in Cloud Federation Using XMPP’, In Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on (pp. 67-70). IEEE.

Famulari A, Hecker A., (2013), ‘Mantle: A Novel DOSN Leveraging Free Storage and Local Software. In: Advanced Infocomm Technology, pp 213–224. Springer

Fuentes, L., Pinto, M. and Vallecillo, A., (2003). ‘How MDA Can Help Designing Component and Aspect-Based Applications’. In Enterprise Distributed Object Computing Conference, 2003. Proceedings. Seventh IEEE International (pp. 124-135). IEEE.

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). ‘The many faces of publish/subscribe’. ACM Computer Survey., 35, 114{131.

Gerber, A., Van der Merwe, A. and Barnard, A., (2008), ‘A Functional Semantic Web Architecture’, European Semantic Web Conference 2008, LNCS, Springer 5021, pp. 273–28

Gruber, T. (2007), ‘Collective Knowledge Systems: Where the social Web meets the Semantic Web’, Journal of Web Semantics, 2007

Goodhue, D., & Thompson, R. (1995). Task-Technology Fit and Individual Performance. MIS Quarterly, 19(2), 213–236.

Griffin, K. & Flanagan, C. (2010). ‘Evaluation of asynchronous event mechanisms for browser-based real-time communication integration’. In Technological Developments in Networking, Education and Automation (pp. 461{466). Springer Netherlands.

Gregg, D.G., Kulkarni, U.R. and Vinzé, A.S., (2001). ‘Understanding The Philosophical Underpinnings Of Software Engineering Research In Information Systems’, Information Systems Frontiers, *3*(2), pp.169-183.

Gjoka et al., (2010), 'Walking In Facebook: A Case Study Of Unbiased Sampling Of Osns', In Proceedings Of The 29th Conference On Information Communications, pages 2498{2506. IEEE Press

Grarland, J., and Anthony, R. (2003), 'Large Scale Software Architecture', New York Wiley, 2003

Grace, P. (2009), 'Dynamic Adaptation'. In Middleware for Network Eccentric and Mobile Applications, B. Garbinato, H. Miranda and L. Rodrigues (Eds.), pp. 285-304, Springer.

Group Report, Available at: http://www.w3.org/2005/Incubator/socialweb/XGR-socialweb-20101206/#ref-eduserv (Accessed: 6/8/2013)

Gacenga, F., Cater-Steel, A., Toleman, M. and Tan, W.G., (2012), 'A Proposal And Evaluation Of A Design Method In Design Science Research', Electronic Journal of Business Research Methods, 10(2), pp.89-100.

Gleasurea, R., (2015), 'When Is A Problem A Design Science Problem? ', Systems, Signs & Actions, 9(1), pp.9-25.

Gleasurea, R., (2015), 'When is a Problem a Design Science Problem?', International Journal on IT, Action, Communication and Work practices 9(1), pp.9-25.

Hanington, B. and Martin, B., (2012), 'Universal methods of design: 100 Ways to Research Complex Problems, Develop Innovative Ideas and Design Effective Solutions', Rockport Publishers.

Hill, D. (2009), 'Microsoft Application Architecture Patterns and Practices' (2nd Ed) Microsoft

Holm, M. (2011), Available at: http://timelessrepo.com/json-isnt-a-javascript-subset (Accessed: 10-11-2013)

Hendler, J., & Berners-Lee, T. (2010), 'From the Semantic Web to social machines: A research challenge for AI on the World Wide Web', Journal of Artificial Intelligence, 174(2), pp 156–161. doi:10.1016/j.artint.2009.11.010

Han, L., Nath, B., Iftode, L. and Muthukrishnan, S., (2011), 'Social butterfly: Social Caches For Distributed Social Networks'. In Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on (pp. 81-86). IEEE.

Halpin, H. (2013), ‘Social Semantics: The Search for Meaning on the Web’, Semantic Web and Beyond Vol. 13, DOI 10.1007/978-1-4614-1885-6 2

Hu, P., and Lau, W. C. (2013), ‘A Meta Social Networking Approach Towards Decentralisation‘, W3C Workshop on Social Standards, 7-8 August 2013, San Francisco, USA,

Halpin, H., and Tuffield, M., (2010), ‘A Standards-based, Open and Privacy-aware Social Web’, W3C Social Web Incubator Group Report 6th December 2010 Report., W3C Incubator

Heitmann, B. (2010), ‘Architecture and Methodologies for Adaptive Personalisation on The Web of Data’, DERI Technical Report 2010. DERI Galway

Heidemann, J., Klier, M., & Probst, F (2012), Online social networks, ‘A survey of a global phenomenon’, Journal of Computer Networks, 56(18), 3866–3878, DOI: 10.1016/j.comnet.2012.08.009

Hirschheim R, and Klein HK (1989), ‘Four paradigms of information systems development’, Communications of the ACM 1989; 32(10):pp 1199–1216.

Heitmann, B. (2014), ‘An Open Framework for Multi-source, Cross-domain Personalisation with Semantic Interest Graphs’, DERI Technical Report 2014. DERI Galway

IETF.Org (2011), ‘The Web Socket Protocol’, Available at: http://tools.ietf.org/html/rfc6455 (Accessed 12-10.-2013)

IETF RFC6120, (2015). ‘XMPP Core’, Access on 02-06-2015, Available at: https://tools.ietf.org/html/rfc6120

Ionita, M.T., Hammer, D.K. and Obbink, H., (2002),’Scenario-based Software Architecture Evaluation Methods’: An overview. ICSE/SARA.

IEEE 1472, (2000), ‘IEEE Recommended Practice for Architectural Description of Software-Intensive Systems’: IEEE Std 1472000. 2000.

IEEE P42010/D9, (2011), ‘Systems and Software Engineering- Architecture Description’, ISO/IEC JTC 1/SC 7N

ISO/IEC 24760-1, (2011), ‘A Framework for Identity Management - Part 1: Terminology and Concepts’, ISO. 2011. Retrieved December 2015.

Irfan, R., Bickler, G., Khan, S.U., Kolodziej, J., Li, H., Chen, D., Wang, L., Hayat, K., Madani, S.A., Nazir, B. and Khan, I.A., (2013), ‘Survey on Social Networking Services’, IET Networks, 2(4), 224–234. doi:10.1049/iet-net.2013.0009

Internet World Stat, (2012), Available at: http://www.internetworldstats.com/facebook.html (Accessed: July 2013)

Jorge, A. L. & Porflirio, F. P. (2010), Building an Academic Social Network for Bologna Mobility, 3rd International Conference on Social Network Systems, 2010, ACM

Jalali, S. and Wohlin, C., 2012. Global Software Engineering and Agile Practices: A Systematic Review. Journal of Software: Evolution and Process, 24(6), pp.643-659.

Juste, P.S., Wolinsky, D., Boykin, P.O., Covington, M.J. and Figueiredo, R.J., (2010), SocialVPN: Enabling Wide-Area Collaboration with Integrated Social and Overlay Networks’. Computer Networks, 54(12), pp.1926-1938.

Jiang et al., (2013) ‘Understanding Latent Interactions in Online Social Networks’, ACM Transaction on the web, Vol. 7(4), Article 18

Jones, W. (2007), ‘Personal Information Management’, Annual Review of Information Science and Technology, 41(1), 453–504. doi:10.1002/aris.2007.1440410117

Natarajan, R. and Rosenblum, D.S., (1998), ‘Merging Component Models and Architectural Styles’. In Proceedings of the Third International Workshop on Software Architecture (pp. 109-111). ACM.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W. (2001). ‘Getting started with AspectJ’, Communications of the ACM, Vol. 44, No. 10, pp.59–65.

Kiczales, G. and Mezini, M. (2005), ‘Aspect-oriented programming and modular reasoning’, ICSE ‘05: Proceedings of the 27th International Conference on Software Engineering, ACM Press, New York NY, USA, pp.49–58.

Kazman, R., Bass, L., Klein, M., Lattanze, T., and Northrop, L., (2005). ‘A Basis for Analysing Software Architecture Analysis Methods’, Software Quality Journal, 13(4):329-355

Khare, R. and Taylor, R.N., (2004). ‘Extending the Representational State Transfer (rest) Architectural Style for Decentralized Systems’. In Proceedings of the 26th International Conference on Software Engineering (pp. 428-437). IEEE Computer Society.

Kazman, R., Bass, L., Abowd, G., and Webb, M., (1994), ‘SAAM: A Method for Analysing the Properties of Software Architectures,’ Proc. 16th International Conference of Software Engineering, pp. 81-90, 1994.

Kosanke, K., (2006), ‘ISO Standards for Interoperability a comparison’, In Interoperability of Enterprise Software and Applications (pp. 55-64). Springer London.
https://www.iso.org/obp/ui/#iso:std:24020:en

Kwong, C.K., Mu, L.F., Tang, J.F. and Luo, X.G., (2010). Optimization of software components selection for component-based software system development. *Computers & Industrial Engineering*, *58*(4), pp.618-624.

Kiczales, G. and Mezini, M., (2005), ‘Aspect-Oriented Programming and Modular Reasoning’, In Proceedings of the 27th international conference on Software engineering (pp. 49-58). ACM.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G., (2001), An overview of AspectJ. In European Conference on Object-Oriented Programming (pp. 327-354). Springer Berlin Heidelberg.

Kolp, M., and Wautelet, Y. (2010), ‘A Social Framework for Software Architectural Design’, 490–511. doi:10.4018/978-1-60566-264-0.ch025

Kruchten, P. (1995), ‘Architectural Blueprints’, The “4 + 1 ” View Model of Software Architecture, 12(November), pp 42–50

Kietzmann, J.H., Hermkens, K., McCarthy, I.P. and Silvestre, B.S., (2011), ‘Social media? Get Serious! Understanding the Functional Building Blocks of Social Media’, Business Horizons, 54(3), 241–251. doi:10.1016/j.bushor.2011.01.005

Ko, M. N., Cheek, G. P., Shehab, M., and Sandhu, R. (2010), ‘Social Networks Connect Services’, IEEE Computer Society, 10(August), pp 37–43.

Kagal, L., Berners-lee, T., Connolly, D., and Weitzner, D. (2006),'Using Semantic Web Technologies for Policy Management on the Web ', Journal of Artificial Intelligence

Kayes, I. and Iamnitchi, A (2017), 'Privacy and Security in Online Social Networks', A survey, Online Social Networks and Media, 3, pp.1-21.

Lundar, J., Grønli, T.M., and Ghinea, G. (2013) 'Performance Evaluation of a Modern Web Architecture', International Journal of Information Technology and Web Engineering, 8(1), 36–50. doi:10.4018/jitwe.2013010103

Lin, J.W. and Lai, Y.C., (2013). 'Online Formative Assessments with Social Network Awareness', Journal of Computers & Education, 66, pp.40-53.

Laplante, P., (2007).' What Every Engineer Should Know About Software Engineering', CRC Press. ISBN 0849372283.

Larman, C., (2012), 'Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development', Pearson Education India.

Lundar, J.A., Grønli, T.M. and Ghinea, G., (2013). 'Performance evaluation of a modern web architecture'. International Journal of Information Technology and Web Engineering (IJITWE), 8(1), pp.36-50.

Laine, M. and Säilä, K., (2012). 'Performance Evaluation of XMPP on the Web', Aalto University Technical Report.

Langegger et al., (2005), 'Simplifying a Web Application´s Architecture ' , The DaVinci Framework in: The Seventh International Conference on Information Integration and Web-based Applications and Services (IIWAS 2005), Vol. 2, G

Sommerville, I., (2007). 'Introduction to software Engineering', Addison-Wesley.

Morandi, B., West, S., Nanz, S. and Gomaa, H., (2013). 'Concurrent Object-Oriented Development with Behavioural Design Patterns', In Software Architecture (pp. 25-32). Springer Berlin Heidelberg.

Mattsson, M., Grahn, H. and Mårtensson, F., (2006), 'Software Architecture Evaluation Methods for Performance, Maintainability, Testability, And Portability'. In Second International Conference on the Quality of Software Architectures.

Maier, M.W., Emery, D. and Hilliard, R., (2001), 'Software architecture: introducing IEEE Standard 1471', *Computer*, *34*(4), pp.107-109.

Masuhara, H. and Kiczales, G., (2003), Modelling Crosscutting in Aspect-Oriented Mechanisms. In European Conference on Object-Oriented Programming (pp. 2-28). Springer Berlin Heidelberg.

Martin, A., Mazalu, R. and Cechich, A., (2010), 'Supporting an aspect-oriented approach to web accessibility design'. In Software Engineering Advances (ICSEA), 2010 Fifth International Conference on (pp. 20-25). IEEE.

Morrison, J. and George, J.F., (1995), 'Exploring the Software Engineering Component in MIS Research', Communications of the ACM, 38(7), pp.80-91.

Mouheb, D., Debbabi, M., Pourzandi, M., Wang, L., Nouh, M., Ziarati, R., Alhadidi, D., Talhi, C. and Lima, V., (2015), 'Aspect-Oriented Security Hardening of UML Design Models', Springer.

Murugesan, S. (2007),' Understanding Web 2.0', IT Pro, IEEE, 1520-9202/07/2007 IEEE

Meier et al., (2008), 'Microsoft Patterns and Practics', Architecture Guide 2.0 Microsoft

Miller, J. & Smith, S. M. (2010), 'The Locker Project', Lockerproject.org, 2010

McLeod, D. and Heimbigner, D., (1980), ' A Federated Architecture for Database Systems'. In Proceedings of the May 19-22, 1980, national computer conference (pp. 283-289). ACM.

Maurer, F. and Labitzke, S., (2014). 'FOSP: Towards A Federated Object Sharing Protocol That Unifies Operations on Social Content'. In DFN-Forum Kommunikationstechnologien (pp. 57-66).

Narayanan, A., Toubiana, V., Barocas, S., Nissenbaum, H. and Boneh, D. (2012), 'A Critical Look at Decentralized Personal Data Architectures', arXiv preprint arXiv:1202.4503.

Oinas-Kukkonen, H., Lyytinen, K. and Yoo, Y., (2010).' Social Networks and Information Systems: Ongoing and Future Research Streams'. Journal of the Association for Information Systems, 11(2), p.3.

Noran, O. (2003), 'An Analysis of the Zachman Framework for Enterprise Architecture from the GERAM Perspective', Annual Review in Control, vol, 27(2003), pp 163- 183, Elsevier

Netter, M., Hassan, S. and Pernul, G., (2012),' An Autonomous Social Web Privacy Infrastructure with Context-Aware Access Control', (pp. 65-78). Springer Berlin Heidelberg.

Ng, J. W., & Lau, D. H. (2013), Social Ontology and Semantic Actions, Enabling Social Networking Services for Distributed Web Tasking, 2013 IEEE Ninth World Congress on Services, (c), 131–135. doi:10.1109/SERVICES.2013.23

Nilizadeh, S., Jahid, S., & Borisov, N. (2012), 'Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching', ACM CoNext 12, Nice France, 337–348.

Netter, M., Hassan, S., and Pernul, G. (2012), 'An autonomous social web privacy infrastructure with context-aware access control' (pp. 65-78), Springer Berlin Heidelberg.

NEPOMUK, (2014), 'The Social Semantic Desktop', Available at: http://nepomuk.semanticdesktop.org/ (Accessed: 06-05-2014)

Nathan, B. Graham,M. , Talapatra, S. Dale,S. (2015), 'Social Network Application Programming Interface', U.S. Appl. No. 13/244,942, filed Sep. 26, 2011,

OpenID.net, (2013), Available at http://openid.net/developers/specs/

Oren, E., Heitmann, B., & Decker, S. (2008),' ActiveRDF: embedding Semantic Web data into object-oriented languages'. Journal of Web Semantics

Ozturk, O. (2010). 'Introduction to XMPP protocol and developing online collaboration applications using open source software and libraries', In Collaborative Technologies and Systems (CTS), 2010 International Symposium on (pp. 21-25). IEEE.

OpenSocial, (2013), Available at https://code.google.com/apis/opensocial/

O'Reilly,T. (2005), 'What is web 2.0', Available at: http://oreilly.com/web2/archive/what-is-web-20.html (Accessed 29/10/2013)

Otte, E. Rousseau, R. (2002),' Social Network Analysis: A Powerful Strategy', Also for The Information Sciences', Journal of Information Science 28: 441–453. doi:10.1177/016555150202800601, Retrieved on 22-11-2015

Perry, D. E., & Wolf, A. L. (1992). Foundations for the Study of Software Architecture, ACM SIGSOFT, 17(4)

Piirainen, K.A. and Gonzalez, R.A., (2014), 'Constructive Synergy in Design Science Research: A Comparative Analysis of Design Science Research and The Constructive Research Approach', Liiketaloudellinen Aikakauskirja, pp.3-4.

Paul, T., Famulari, A. and Strufe, T., (2014). 'A Survey on Decentralised Online Social Networks', Journal of Computer Networks, 75, pp.437-452.

Davis, F. Perceived Usefulness, Perceived Ease of Use, And User Acceptance of Information Technology. Mis Quarterly 13, 3 (1989), 319-340.

Pessemier, N., Seinturier, L., Duchien, L. and Coupaye, T., (2008). 'A Component-Based and Aspect-Oriented Model for Software Evolution'. International Journal of Computer Applications in Technology, 31(1-2), pp.94-105.

Pettenati, M.C., Chini, D., Parlanti, D. and Pirri, F., (2007), 'InterDataNet: A Web of Data foundation for the Semantic Web vision'. extended version) IADIS Int. Journal on WWW/Internet, 6(2), pp.16-30..

Peña, P., Del Hoyo, R., Vea-Murguía, J., González, C. and Mayo, S., (2013), 'Automatic Ontology User Profiling for Social Networks from URLs Shared', LNCS, 'Conference of Spanish Association for Artificial Intelligence, CAEPIA', Springer, pp. 168-177

Pettenati, M.C., Cio_, L., Parlanti, D., Pirri, F., Giuli, D (2011), 'An Overlay Infrastructural Approach for a Web-Wide Trustworthy Identity and Profile Management', In: Salgarelli, L., Bianchi, G., Blefari-Melazzi, N. (eds.) Trustworthy Internet, pp. 43- 58. Springer (2011)

Pinto, M., Fuentes, L. and Troya, J.M., (2003), 'DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development', In International Conference on Generative Programming and Component Engineering (pp. 118-137). Springer Berlin Heidelberg.

Pinto, M., Fuentes, L. and Troya, J.M., (2011), 'Specifying Aspect-Oriented Architectures In AO-ADL'. Information and Software Technology, 53(11), pp.1165-1182.

Peffers, K., Tuunanen, T., Rothenberger, M.A. and Chatterjee, S., (2007), 'A design science research methodology for information systems research', Journal of management information systems, 24(3), pp.45-77.

Pai, P. and Arnott, D.C., (2013), User adoption of social networking sites: Eliciting uses and gratifications through a means–end approach. Computers in Human Behavior, 29(3), pp.1039-1053.

Python-aspectlib, (2016), 'AspectLib – 1.4', Available at: http://python-aspectlib.readthedocs.io/en/latest/index.html, Access date, 6/11/2016

Piessens, F. (2009). 'A Comprehensive Integration of AOSD and CBSD concepts in Middleware', Ph.D. Thesis, Department of Computer Science, K.U.Leuven, Belgium, Dec 2009

Pinto, M., Fuentes, L. and Troya, J.M., (2001),'Towards an Aspect-Oriented Framework In The Design Of Collaborative Virtual Environments', In Distributed Computing Systems, 2001. FTDCS 2001. Proceedings. The Eighth IEEE Workshop on Future Trends of (pp. 9-15). IEEE.

Pinto, L. Fuentes, J. M. Troya (2001), Specifying aspect-oriented architectures in AO-ADL, Information and Software Technology, Volume 53, Issue 11, Pages 1165-1182 (2011)

Reenskaug,T. and Coplien,J. (2009), The DCI Architecture: A New Vision of Object-Oriented Programming, http://www.artima.com/articles/dci_vision.html access date 24-11-2013

Richter, A. and Riemer, K., (2009), 'December. Corporate Social Networking Sites–Modes of Use And Appropriation Through Co-Evolution'. In 20th Australasian Conference on Information Systems (pp. 2-4).

Rathfelder, C., Klatt, B., Sachs, K. and Kounev, S., (2014). ‘Modeling Event-Based Communication In Component-Based Software Architectures For Performance Predictions’. Software & Systems Modeling, 13(4), pp.1291-1317.

Rad, M.S., Dahlan, H.M., Iahad, N.A., Nilashi, M. And Zakaria, R., (2014), Assessing The Factors That Affect Adoption Of Social Research Network Site For Collaboration By Researchers Using Multi-Criteria Approach. Journal of Theoretical & Applied Information Technology, 65(1).

Rohani, V., and Hock, O. (2010), ’On Social Network Web Sites: Definition, Features, Architectures and Analysis Tools, Journal of Advances in Computer Research, 1, 3–1

Sioc-project.org, (2013), ‘Semantically-Interlinked Online Communities’, Available at: http://rdfs.org/sioc/spec/ (Accessed: 10/12/2013)

Sutcliffe et al., (2011),’ Social Mediating Technologies: Social Affordances and Functionalities’, International Journal of Human Computer Interaction, 27(11), 1037–1065. doi:10.1080/10447318.2011.555318

Shaw, M. (1990), ‘Toward higher-level abstractions for software systems’, Data & Knowledge Engineering, 5, pp. 119–128

Sjoberg, D. I. K., Dyba, T., & Jorgensen, M. (2007), ‘The Future of Empirical Methods in Software Engineering Research’, Future of Software Engineering, (pp. 358{378).

Sommerville, I. (2007), ‘Software Engineering’, 8th edn. International Computer Science Series. Addison-Wesley, Reading (2007)

Suryanarayan, G., Erenkrantz, J.R. and Taylor, R.N., (2005). ‘An architectural approach for decentralized trust management’. Internet Computing, IEEE, 9(6), pp.16-23.

Suryanarayana, G., Diallo, M.H., Erenkrantz, J.R. and Taylor, R.N., (2006), ‘Architectural Support for Trust Models In Decentralized Applications’. In Proceedings of the 28th international conference on Software engineering (pp. 52-61). ACM.

Suryanarayana, G., Diallo, M.H., Erenkrantz, J.R. and Taylor, R.N., (2006). ‘Architecting Trust-Enabled Peer-To-Peer File-Sharing Applications’. Crossroads, 12(4), pp.5-5.

Szyperski, C. (2002). ‘Component Software. Beyond Object-Oriented Programming ‘(2nd edn). Addison-Wesley

Skaf, H., Rahhal, C., & Molli, P. (2008). Peer-to-Peer Semantic wikis. Research Report RR-6714, INRIA.

Suryanarayana, G., Erenkrantz, J.R., Hendrickson, S.A. and Taylor, R.N., (2004). ‘PACE: An Architectural Style for Trust Management In Decentralized Applications’, In Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on (pp. 221-230). IEEE.

Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer, M. And Kappel, G., 2007. A Survey on Aspect-Oriented Modeling Approaches. Relatorio Tecnico, Vienna University of Technology, 36, Pp.41-42.

Suryanarayana, G. and Taylor, R.N., (2004). ‘A Survey of Trust Management And Resource Discovery Technologies In Peer-To-Peer Applications’.

Smith, D.A., Van Kleek, M., Seneviratne, O., Schraefel, M., Bertails, R., Berners-lee, T., Hall, W. and Shadbolt, N.,(2012),’ WebBox: Supporting Decentralised and Privacy-respecting Micro-sharing with Existing Web Standards’.

Shaw,M. and Garlan, D. (1996), ‘Software Architecture’, Perspectives on an Emerging Discipline, Prentice Hall

Sauermann, L. (2006), ‘The Gnowsis Semantic Desktop for Information Integration’, German Research Centre for Artificial Intelligence DFKI GmbH. Germany

Sfakianakis, S, (2010) ‘Social Semantic Web and Semantic Web Services’, ICS Fourth Greece, pp.350-368. IGI Global, USA
Slater, J. A. Venkatraman, R. Topi, H. (2015), ‘Modern Database Management’, Prentice Hall

Sharman R., Kishore R., and Ramesh R. (2007), ‘Ontologies, A Handbook of principles, concepts and applications in information systems’, Springer’s Integrated Series in IS, Published by Springer 2007, NY, USA.

Sun Systems, Logical Architectures http://docs.oracle.com/cd/E19263-01/817-6096/architecture.html Access Date, 20-10-2013,

Seong, S., Jiwon, S., Matthew, N., Debangsu, N., Sengupta, D., Hangal, S., Teh, K.S., Chu, R., Dodson, B., and Lam, M. S. (2010), 'PrPl : A Decentralized Social Networking Infrastructure', ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond (MCS).

Sintek, M., Handschuh, S., Scerri, S., & Elst, L. V. (2009), 'Technologies for the Social Semantic Desktop', Springer LNCS 5689, pp. 222-254.

Sloni, D.K. and Sharma, V.K., (2011), 'Safe Semantic Web and Security Aspect Implication for Social Networking', International Journal of Computer Applications in Engineering Sciences, 1(2), pp.141-149

Surajbali, B., Grace, P. and Coulson, G., (2014), 'An AO-Middleware Architecture Supporting Flexible Dynamic Reconfiguration', In Proceedings of the 17th international ACM Sigsoft Symposium on Component-Based Software Engineering (pp. 75-84). ACM.

Turnquist, G.L., (2010), 'Spring Python 1.1', Packt Publishing Ltd.

Tandukar, U. and Vassileva, J. (2012), 'Selective Propagation of Social Data in Decentralized Online Social Network' UMAP 2011 Workshops, LNCS 7138, pp. 213–224, Springer-Verlag Berlin Heidelberg 2012

Traz, W (1994), 'DSSA frequently asked questions', ACM Software Engineering Notes 19, 52–56 (1994)

Tiwana, A., & Bush, A. (2001), 'A social exchange architecture for distributed Web communities', Journal of Knowledge Management, 5(3), 242–249. doi:10.1108/13673270110401220

Taylor, R.N., Medvidovic, N., Anderson, K.M., Whitehead Jr, E.J., Robbins, J.E., Nies, K.A., Oreizy, P. and Dubrow, D.L., (1996). 'A component-and message-based architectural style for GUI software'. Software Engineering, IEEE Transactions on, 22(6), pp.390-406.

Trams, S., Frischmuth, P., Arndt, N., Ermilov, T., & Auer, S. (2011). Waeving a distributed, semantic social network for mobile users. In Extended Semantic Web Conference.

Valencia-García, R., García-Sánchez, F., Castellanos-Nieves, D., Fernández-Breis, J. T., and Toval, A. (2010),'Exploitation of social semantic technology for software development team configuration'. IET Software, 4(6), 373. doi:10.1049/iet-sen.2010.0043

Van Landuyt, D., Truyen, E. and Verbaeten, P., (2010), 'Building a digital publishing platform using aosd', LNCS Transactions on Aspect-Oriented Software Development, 9, pp.1-34.

Vannoy, S.A. and Palvia, P., (2010), 'The social influence model of technology adoption', Communications of the ACM, 53(6), pp.149-153.

Venkatesh, V, Morris, M., Davis, G., & Davis, F. (2003). User Acceptance Of Information Technology: Toward A Unified View. Mis Quarterly, 27(3), 425–478.

Van Landuyt, D., Truyen, E. And Verbaeten, P., 2011. Building A Digital Publishing Platform Using Aosd. In Transactions on Aspect-Oriented Software Development Viii (Pp. 163-195). Springer, Berlin, Heidelberg.

Van Landuyt, D., Truyen, E. and Verbaeten, P., (2011). 'Building a digital publishing platform using AOSD'. In Transactions on Aspect-Oriented Software Development VIII (pp. 163-195). Springer Berlin Heidelberg.

Venable, J., Pries-Heje, J. and Baskerville, R., (2012). 'A comprehensive framework for evaluation in design science research', In Design Science Research in Information Systems, Advances in Theory and Practice (pp. 423-438). Springer Berlin Heidelberg.

Vidgen, R., Donnellan, B., Matook, S. and Conboy, K., (2011), 'Design Science Approach to Measure Productivity in Agile Software Development', In *European Design Science Symposium* (pp. 171-177). Springer Berlin Heidelberg.

Fowler, M. and Highsmith, J., (2001), 'The agile manifesto'. *Software Development*, *9*(8), pp.28-35.

W3. Org, (2013), 'DOM document object model', http://www.w3.org/DOM/ Access date 20-10-2013

W3.org, (2015), ‘Social Web Acid Test’, Available at: https://www.w3.org/2005/Incubator/federatedsocialweb/wiki/SWAT1_use_cases, Access date, 5/11/2015

Webber, J., Parastatidis, S. and Robinson, I., (2010), ‘REST in Practice’, Hypermedia and System Architectures, O`Reilly Media Inc. 2010

Wasserman, S., and Faust, K. (1994), ‘Social Network Analysis in the Social and Behavioral Sciences’, Social Network Analysis: Methods and Applications, Cambridge University Press. pp. 1–27. ISBN 9780521387071.

Wilson, C., Sala, A., Puttaswamy, K.P. and Zhao, B.Y., (2012), ‘Beyond Social Graphs User interactions in Online Social Networks and their Implications’, ACM Transactions on the Web, 6(4), 1–31, doi:10.1145/2382616.2382620

Wu, P., & Li, S. (2009), ‘Social Network Visualization via Domain Ontology’, International Conference on Information Engineering and Computer Science, 1–4. doi:10.1109/ICIECS.2009.5362898

Wieringa, R., (2009),’ Design science as nested problem solving’, In Proceedings of the 4th international conference on design science research in information systems and technology (p. 8). ACM.

Yeung, C., M., A., Liccardi, I., Lu, K., Seneviratne, O., and Berners-Lee, T. (2008), ‘Decentralization: The Future of Online Social Networking’, In W3C Workshop on the Future of Social Networking

Zhong, C. and Sastry, N., (2017), Systems applications of social networks. ACM Computing Surveys (CSUR), 50(5), p.63.

Zhang, C., Cheng, C., & Ji, Y. (2012), ‘Architecture design for social web of things’, Proceedings of the 1st International Workshop on Context Discovery and Data Mining - ContextDD ’12, 1. doi:10.1145/2346604.2346608

Appendix 1. Iteration 1

Aspect.py, is a CAM based version of Aspect, which is adapted from aspectlib module of python. This library is modified based on the need of the DSNA.

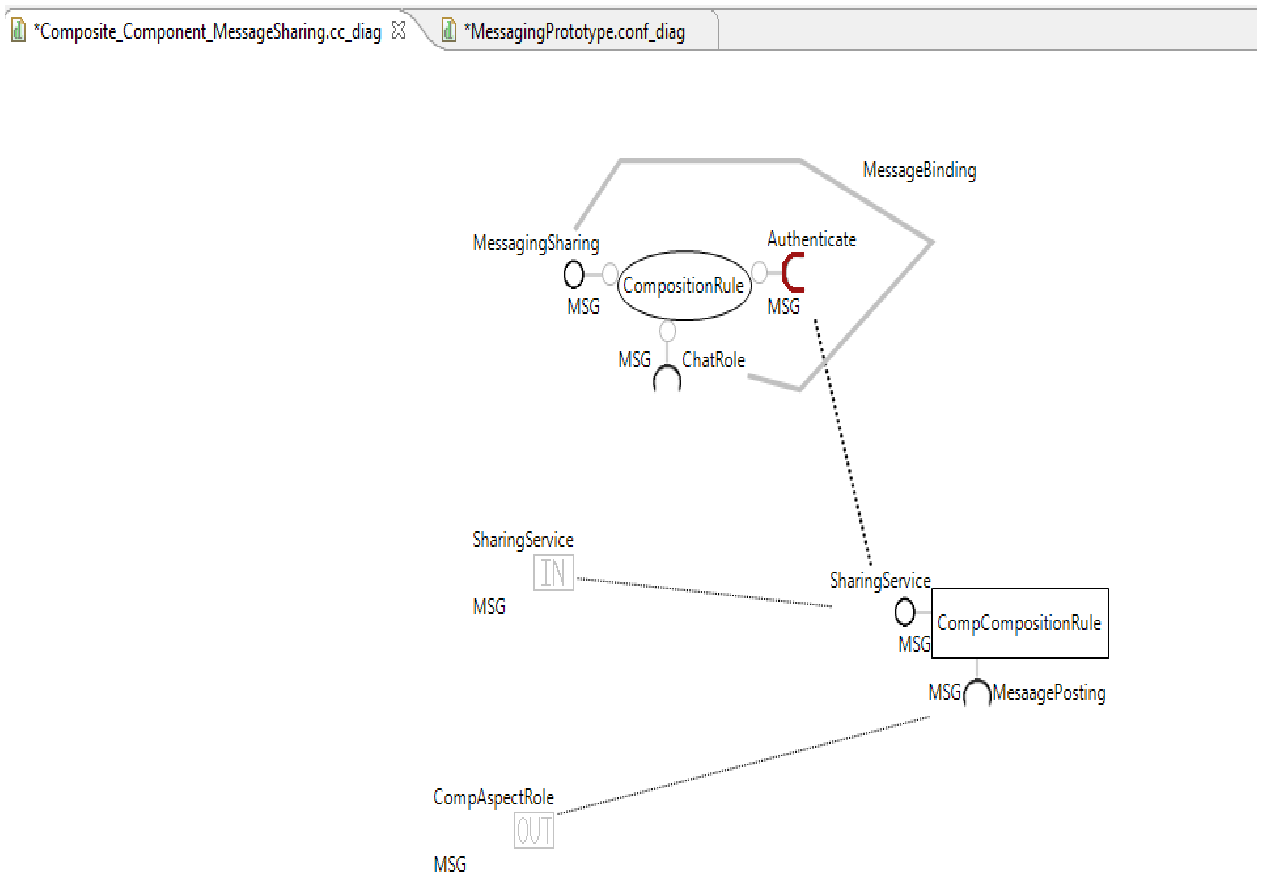
```
Import  re

Class  Aspecter ( type ):

  "" "

    Meta class used by classes that will have methods oriented

    The aspect (with join-points and cross-cut points and etc.



    The aspect_roles object contains all the aspect roles

  "" "

  Aspect_roles  =  []

  Wrapped_methods  =  []

  Def  __new__ ( cls ,  name ,  bases ,  dict ):

    "" "          Class initialization that contains aspect-oriented methods.

      It basically annotates all methods of the class so that every

      Call can be checked if there is a corresponding role

      To them:

    "" "

    For  key ,  value  in  dict . Items ():

      If  hasattr ( value ,  "__call__" )  and  key  ! =  "__metaclass__" :

        Dict [ key ]  =  Aspecter . Wrap_method ( value )

    Return  type . __new__ ( cls ,  name ,  bases ,  dict )


  @classmethod
```
n

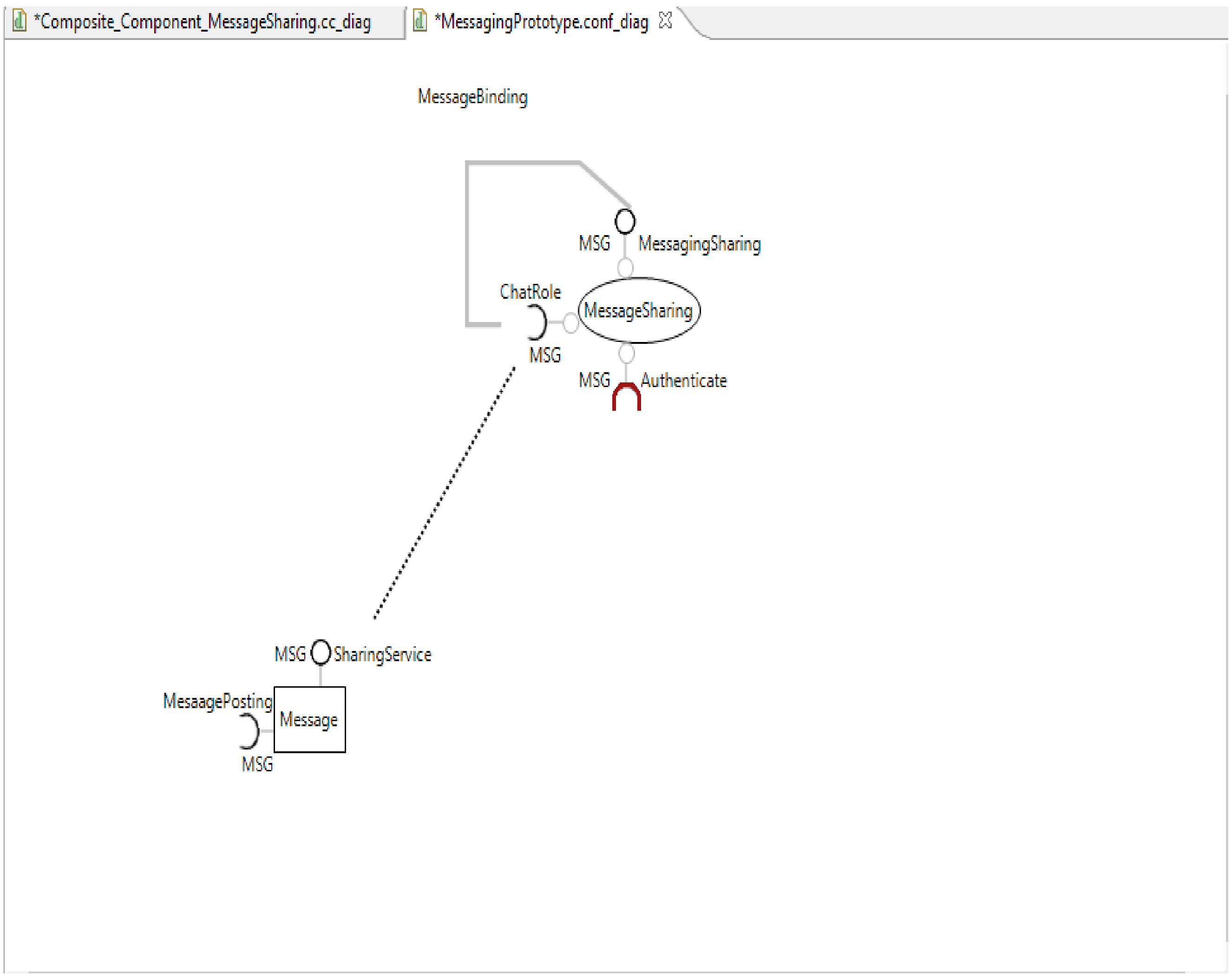
```
   Def  register ( cls ,  name_pattern = "" ,  in_objects = (),  out_objects = (),
          Pre_function = None ,
          Post_function = None ):
   "" "Method used to register a new aspect role.
       Logging can be done dynamically at run time
       Name_pattern: is a regular expression that matches the names of the
       Methods. Blank, home with all methods.
       In particular, note that this simplified scheme does not account for
       To call a pre_function based on out_objects
   "" "
   # So simple method that could be used a direct append in
   # "Aspect roles"
   role  =  { "name_pattern" :  name_pattern ,  "in_objects" :  in_objects ,
          "Out_objects" :  out_objects ,
          "Pre" :  pre_function ,  "post" :  post_function }
   Cls . Aspect_roles . Append ( role )
@classmethod
Def  wrap_method ( cls ,  method ):
   Def  call ( * args ,  ** kw ):
      Pre_functions  =   cls . Matching_pre_functions ( method ,  args ,  kw )
      For  function  in  pre_functions :
         Function ( * args ,  ** kw )
      Results  =  method ( * args ,  ** kw )
      Post_functions  =  cls . Matching_post_functions ( method ,  results )
      For  function  in  post_functions :
         Function ( results ,  * args ,  ** kw )
```

```
  Return  results
   Return  call
  @classmethod
Def  matching_names ( cls ,  method ):
   Return  [ role  for  role  in  cls . Aspect_roles
           if  re . Match ( role [ "name_pattern" ],  method . Func_name )
             Or  role [ "name_pattern" ]  ==  ""
        ]
  @classmethod
Def  matching_pre_functions ( cls ,  method ,  args ,  kw ):
   All_args  =  args  +  tuple ( kw . Values ())
   Return  [ role [ "pre" ]  for  role  in  cls . Matching_names ( method )
           If  role [ "pre" ]  and
              ( Role [ "in_objects" ]  ==  ()  or
               Any (( type ( arg )  in  role [ "in_objects" ]  for  arg  in  all_args )))
        ]
@classmethod
Def  matching_post_functions ( cls ,  method ,  results ):
   If  type ( results )  ! =  Tuple :
      Results  =  ( results ,)
   Return  [ role [ "post" ]  for  role  in  cls . Matching_names ( method )
           If  role [ "post" ]  and
             ( Role [ "out_objects" ]  ==  ()  or
              Any (( type ( result )  in  role [ "out_objects" ]  for  result  in  results )))
        ]
```

Appendix 2. Iteration 2

Design and Deployment of the DSNA prototype, based on AOADL tools. Section 5.2 and 5.3




*Composite_Component_MessageSharing.cc_diag
*MessagingPrototype.conf_diag
MessageBinding
MSG
MessagingSharing
ChatRole
MessageSharing
MSG
MSG
Authenticate
MSG
SharingService
MesaagePosting
Message
MSG

Deployment of Messaging App

Message Sharing Interface

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<interface xmlns="http://caosd.lcc.uma.es/AO-ADL/AO-ADLSchema"
name="SharingComponent">
      <operation name="Initiate">
            <parameter name="UserID" type="String" direction="IN"/>
            <parameter name="ContentType" type="Array" direction="IN"/>
            <parameter name="DestinationID" type="String"
direction="IN"/>
            <parameter name="Token" type="String" direction="IN"/>
            <returnType>String</returnType>
      </operation>
      <operation name="Exit">
            <parameter name="UserID" type="String" direction="OUT"/>
            <parameter name="DestinicationID" type="String"
direction="OUT"/>
            <parameter name="ContentType" type="Array" direction="OUT"/>
            <parameter name="Token" type="String" direction="OUT"/>
            <returnType>String</returnType>
      </operation>
      <description>Description of sharing component</description>
</interface>
```

Messaging Connector

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<connector xmlns="http://caosd.lcc.uma.es/AO-ADL/AO-ADLSchema"
name="MessagingConnector" type="Connector">
      <provided_role roleName="MessagingSharing"
role_specification="//interface[@name='SharingComponent']" type="MSG"
minOccurs="1" maxOccurs="1"/>
      <required_role roleName="ChatRole"
role_specification="//interface[@name='SharingComponent']" type="MSG"
minOccurs="1" maxOccurs="1"/>
      <componentBindings>
            <binding name="MessageBinding">

      <source>//provided_role[@name='MessagingSharing']</source>
                  <target>//required_role[@name='ChatRole']</target>
            </binding>
      </componentBindings>
      <description>Triggers messaging functionality</description>
      <aspectual_role roleName="Authenticate"
role_specification="//interface[@name='SharingComponent']" type="MSG"
minOccurs="1" maxOccurs="1"/>
      <aspectualBindings>
            <aspectual_binding name="StartMessageSharing">
                  <pointcut_specification>

      <pointcut>(//provided_role[@name='MessagingSharing']) and
(//operation[@name='Initiate'])</pointcut>
                  </pointcut_specification>
                  <binding operator="after" order="first">
                        <aspectual_component
aspectual_role_name="Authenticate">
                              <advice label="Initiate">
                                    <attachment>
                                          <argument_binding
target="UserID [String]"/>
                                          <argument_binding
target="ContentType [Array]"/>
                                          <argument_binding
target="DestinationID [String]"/>
                                          <argument_binding
target="Token [String]"/>
                                          <argument_binding
target="String [returnType]"/>
                                    </attachment>
                              </advice>
                        </aspectual_component>
                  </binding>
            </aspectual_binding>
            <aspectual_binding name="EndMessageSharing">
                  <pointcut_specification>

      <pointcut>(//provided_role[@name='MessagingSharing']) and
(//operation[@name='Exit'])</pointcut>
                  </pointcut_specification>
                  <binding operator="after" order="last">
                        <aspectual_component
aspectual_role_name="Authenticate">
                              <advice label="Exit">
                                    <attachment>
```

```
                                          <argument_binding
target="UserID [String]"/>
                                          <argument_binding
target="DestinicationID [String]"/>
                                          <argument_binding
target="ContentType [Array]"/>
                                          <argument_binding
target="Token [String]"/>
                                          <argument_binding
target="String [returnType]"/>
                                    </attachment>
                              </advice>
                        </aspectual_component>
                  </binding>
            </aspectual_binding>
      </aspectualBindings>
</connector>
```

Messaging Component

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<component xmlns="http://caosd.lcc.uma.es/AO-ADL/AO-ADLSchema"
name="Message" type="String">
      <provided_interface portName="SharingService" type="MSG"
uri="//interface[@name='SharingComponent']"/>
      <required_interface portName="MesaagePosting" type="MSG"
uri="//interface[@name='SharingComponent']"/>
      <description>description about the message, displayed and
stored</description>
</component>
```

Messaging Prototype Component

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration xmlns="http://caosd.lcc.uma.es/AO-ADL/AO-ADLSchema"
name="MessagingPrototype" uri="" description="Archtiecture of the
message sharing">
      <component instance_name="Message" multiplicity="1"
uri="//component[@name='Message']"/>
      <connector instance_name="MessageSharing" multiplicity="1"
uri="//connector[@name='MessagingConnector']"/>
      <attachments>
            <attachment component="Message" component_number="1"
connector="MessageSharing" connector_number="1">

      <provided_interface>SharingService</provided_interface>
                  <required_role>ChatRole</required_role>
            </attachment>
      </attachments>
</configuration>
```

Component Composition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<compositeComponent xmlns="http://caosd.lcc.uma.es/AO-ADL/AO-ADLSchema"
name="Composite_Component_MessageSharing">
      <provided_interface portName="SharingService" type="MSG"
uri="//interface[@name='SharingComponent']">
            <attachment component="CompCompositionRule"
role="SharingService"/>
      </provided_interface>
      <required_interface portName="CompAspectRole" type="MSG"
uri="//interface[@name='SharingComponent']">
            <attachment component="CompCompositionRule"
role="MesaagePosting"/>
      </required_interface>
      <configuration>
            <component instance_name="CompCompositionRule"
multiplicity="1" uri="//component[@name='Message']"/>
            <connector instance_name="CompositionRule" multiplicity="1"
uri="//connector[@name='MessagingConnector']"/>
            <attachments>
                  <attachment component="CompCompositionRule"
component_number="1" connector="CompositionRule" connector_number="1">

      <provided_interface>SharingService</provided_interface>
                        <aspectual_role>Authenticate</aspectual_role>
                  </attachment>
            </attachments>
      </configuration>
</compositeComponent>
```

Appendix 3. Iteration 3

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="Adapter" type="Adapter"/>
	<xs:complexType name="Adapter">
		<xs:sequence>
			<xs:element name="RoleInstance" type="xs:int" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="Role" type="Role" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="SNSL as MIddlewarre" type="SNSL as
MIddlewarre" minOccurs="1" maxOccurs="1"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="Role" type="Role"/>
	<xs:complexType name="Role">
		<xs:sequence>
			<xs:element name="Co" type="xs:int" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="Ds" type="xs:int" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="Fn" type="xs:int" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="RoleInstance" type="xs:int" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="RoleName" type="xs:int" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="SNSL as MIddlewarre" type="SNSL as
MIddlewarre" minOccurs="1" maxOccurs="1"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="ConfigurationService" type="ConfigurationService"/>
	<xs:complexType name="ConfigurationService">
		<xs:sequence>
			<xs:element name="SNSL as MIddlewarre" type="SNSL as
MIddlewarre" minOccurs="1" maxOccurs="1"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="SNSL as MIddlewarre" type="SNSL as MIddlewarre"/>
	<xs:complexType name="SNSL as MIddlewarre">
		<xs:sequence>
			<xs:element name="PersistanceService" type="PersistanceService"
minOccurs="1" maxOccurs="1"/>
			<xs:element name="Aspect" type="Aspect" minOccurs="1"
maxOccurs="1"/>
			<xs:element name="DSNAFactory" type="DSNAFactory"
minOccurs="1" maxOccurs="1"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="PersistanceService" type="PersistanceService"/>
	<xs:complexType name="PersistanceService">
		<xs:sequence/>
	</xs:complexType>
	<xs:element name="Aspect" type="Aspect"/>
	<xs:complexType name="Aspect">
```