



**University of Fort Hare**  
*Together in Excellence*

# **Enhancement of the Usability of SOA Services for Novice Users**

Submitted in the Faculty of Science and Agriculture in fulfilment for the degree

Master of Science

In

Computer Science

**By**

Yalezo Sabelo

**Supervisor:** Prof Mamello Thinyane

**December 2014**

Telkom Centre of Excellence in ICT4D

Computer Science Department

Private Bag 1314

Alice

5700

## **Declaration**

I, the undersigned hereby declare that the work that is contained in this thesis is my own original work, and it has not been submitted to any educational institution for similar or any other degree. Any information that is extracted from other sources is acknowledged.

Name: S Yalezo

Date: December 2014

## **Acknowledgement**

I would like to thank my family especially Mrs Buyiswa Yalezo my mother for supporting me throughout the year, not forgetting my brothers for inspiring me. In addition, I would like to thank Prof M. Thinyane for providing me with the opportunity to do research under his guidance; support and giving me the liberty to explore various research topics. My colleagues and friends thank you for the support, motivation and joy you brought. Finally, I would like to thank Telkom for the funding. Lastly, I would like to thank God that I am serving for having such mercy for me; truly, his grace is sufficient for me.

## Abstract

Recently, the automation of service integration has provided a significant advantage in delivering services to novice users. This art of integrating various services is known as Service Composition and its main purpose is to simplify the development process for web applications and facilitates reuse of services. It is one of the paradigms that enables services to end-users (i.e. service provisioning) through the outsourcing of web contents and it requires users to share and reuse services in more collaborative ways. Most service composers are effective at enabling integration of web contents, but they do not enable universal access across different groups of users. This is because, the currently existing content aggregators require complex interactions in order to create web applications (e.g., Web Service Business Process Execution Language (WS-BPEL)) as a result not all users are able to use such web tools. This trend demands changes in the web tools that end-users use to gain and share information, hence this research uses Mashups as a service composition technique to allow novice users to integrate publicly available Service Oriented Architecture (SOA) services, where there is a minimal active web application development.

Mashups being the platforms that integrate disparate web Application Programming Interfaces (APIs) to create user defined web applications; presents a great opportunity for service provisioning. However, their usability for novice users remains invalidated since Mashup tools are not easy to use they require basic programming skills which makes the process of designing and creating Mashups difficult. This is because Mashup tools access heterogeneous web contents using public web APIs and the process of integrating them become complex since web APIs are tailored by different vendors. Moreover, the design of Mashup editors is unnecessary complex; as a result, users do not know where to start when creating Mashups. This research address the gap between Mashup tools and usability by the designing and implementing a semantically enriched Mashup tool to discover, annotate and compose APIs to improve the utilization of SOA services by novice users. The researchers conducted an analysis of the already existing Mashup tools to identify challenges and weaknesses experienced by novice Mashup users. The findings from the requirement analysis formulated the system usability requirements that informed the design and implementation of the proposed Mashup tool. The proposed architecture addressed three layers: composition, annotation and discovery. The researchers developed a simple Mashup tool referred to as soa-Services Provisioner (SerPro) that allowed novice users to create web application flexibly. Its usability and effectiveness was validated. The proposed Mashup tool enhanced the usability

of SOA services, since data analysis and results showed that it was usable to novice users by scoring a System Usability Scale (SUS) score of 72.08. Furthermore, this research discusses the research limitations and future work for further improvements.

## Table of Contents

Declaration.....	ii
Acknowledgement .....	iii
Abstract.....	iv
Table of figures .....	xi
Table of listings.....	xiii
List of Tables .....	xiv
List of Acronyms .....	xv
Publications.....	xvii
1 Chapter 1: Introduction.....	1
1.1 Introduction.....	1
1.2 Research Rationale .....	1
1.2.1 ICTs .....	2
1.2.2 Mashups .....	3
1.2.3 Significance of Mashups .....	4
1.2.4 Factors Affecting Mashup Usability .....	5
1.3 Research Problem .....	6
1.4 Research Questions.....	6
1.5 Research Objectives.....	6
1.6 Methodology and Research Design .....	7
1.7 Contributions of the Research .....	8
1.8 Chapter Outline.....	8
1.9 Conclusion .....	9
Part I: Research Context and Related Work .....	10
2 Chapter 2: Literature Review .....	11
2.1 Introduction.....	11
2.2 Service Composition.....	11
2.2.1 Portals.....	11
2.2.2 Mashups .....	12
2.3 Mashups as a Service Composition Technique .....	14
2.3.1 Adopted Web Technologies .....	15
2.3.2 Web Services and Semantic Web Technologies .....	16

2.4	EUP in Mashup Editors .....	18
2.4.1	Widgets.....	18
2.4.2	Pipes and Filters .....	19
2.4.3	Data Federation .....	19
2.5	Designing Mashups .....	19
2.5.1	Mashup Engine.....	20
2.5.2	Mashup Editor .....	21
2.5.3	Mashup Tool Architecture.....	21
2.5.4	Mashup Tool Execution Environments.....	22
2.6	Mashup Tools Composition Styles .....	23
2.6.1	Composition Patterns .....	24
2.6.2	Workflows on Mashup process .....	25
2.7	Web content mining.....	26
2.7.1	Web Page.....	26
2.7.2	Web API Documentation .....	26
2.7.3	RPC-style Web APIs .....	26
2.7.4	RESTful Web APIs .....	27
2.7.5	SOAP.....	27
2.7.6	Mashup Repositories .....	28
2.8	Applications of Mashups .....	28
2.9	Challenges Facing Mashups .....	30
2.10	Usability of Mashup Tools .....	31
2.10.1	Definition of Usability .....	31
2.10.2	Usability Goals .....	31
2.10.3	Evaluator-Based Usability Evaluation Methods .....	32
2.10.4	User-based Usability Evaluation Methods .....	33
2.11	Conclusion .....	34
3	Chapter 3: Homogenization of Web Contents.....	36
3.1	Introduction.....	36

3.2	Ontology as a Web 3.0 Service.....	36
3.3	Semantic Annotations .....	37
3.3.1	Service Description .....	37
3.3.2	Service Discovery .....	40
3.3.3	Services Publication .....	40
3.3.4	Service Selection and Matching.....	41
3.4	Related Work .....	41
3.4.1	Usability in Mashup Tools .....	41
3.4.2	Web APIs Homogenization.....	43
3.5	Homogenization Process .....	43
3.6	Conclusion .....	44
Part II: Methodology and System Modelling.....		46
4	Chapter 4: Methodology .....	47
4.1	Introduction.....	47
4.2	Research Methodology .....	47
4.3	Research Approach and Design.....	49
4.3.1	Literature Review .....	49
4.3.2	System Requirements .....	49
4.3.2.1	Requirements Gathering.....	50
4.3.3	Design.....	55
4.3.4	Research Prototyping .....	56
4.3.5	System Evaluation.....	57
4.3.6	Research Conclusion .....	58
4.4	Conclusion .....	58
5	Chapter 5: Mashup Design .....	59
5.1	Introduction.....	59
5.2	Design Process.....	59
5.3	Mashup Engine .....	61
5.4	API Editor.....	62
5.4.1	Discovering Layer .....	62



5.4.2	Annotation Layer.....	63
5.5	Mashup Editor .....	64
5.5.1	Visual Composer Layer.....	64
5.5.2	Service Construction .....	65
5.5.3	Request Handling and Data Passing.....	66
5.5.4	Use Case Diagrams .....	68
5.6	Conclusion .....	68
Part III:	Development and Implementation .....	69
6	Chapter 6: Implementation and Development.....	70
6.1	Introduction.....	70
6.2	Development Process.....	70
6.3	Mashup Engine .....	72
6.4	SerPro API Editor (SAE).....	75
6.5	Mashup Editor .....	82
6.5.1	Connecting Widgets .....	83
6.5.2	Construction Requests.....	84
6.6	Research Implementation Technologies.....	86
6.7	Conclusion .....	88
Part IV:	Experimentation and Research summary.....	89
7	Chapter 7: Experimentation and Discussion .....	90
7.1	Introduction.....	90
7.2	Validation Process .....	90
7.3	Usability Testing.....	91
7.3.1	Methodology .....	91
7.3.2	Findings of SUS .....	91
7.4	Acceptance Testing.....	94
7.4.1	Methodology .....	94
7.4.2	Findings .....	98
7.5	Comparison of Mashup tools.....	99
7.5.1	Research Usability Attributes.....	99

7.5.2	SUS.....	102
7.6	Discussion.....	104
7.6.1	Composition Style .....	104
7.6.2	Semantic Web Technologies .....	104
7.6.3	Usability of SOA Services .....	104
7.7	Conclusion .....	106
8	Chapter 8: Research Conclusion .....	107
8.1	Introduction.....	107
8.2	Research Summary .....	107
8.3	Addressing Research Objectives .....	109
8.4	Research Limitations and Challenges.....	112
8.5	Research Contribution and Recommendation .....	113
8.6	Future Work.....	113
8.7	Conclusion .....	114
Part V:	Reference Material.....	115
	References.....	116
	Appendix A: Ethical clearance .....	124
	Appendix B: SUS questionnaire and Mashup Tool SUS scores.....	125
	Appendix B.1.SUS questionnaire .....	125
	Appendix B.2.Review of Mashup Tools.....	127
	Appendix C: Code Snippet .....	138

## Table of figures

Figure 1-1.Research overview .....	1
Figure 1-2.Yahoo! Pipes (Jones & Churchill, 2009) .....	4
Figure 2-1.Microsoft Popfly (Loton, 2008) .....	13
Figure 2-2.SOA artefacts .....	14
Figure 2-3.SWS formation adapted: (Pedrinaci & Domingue, 2010) .....	16
Figure 2-4.Programmable Web (ProgrammableWeb, 2014).....	28
Figure 3-1.Web evolution (Spivack, 2008).....	36
Figure 3-2.Annotation methods adapted: (Kopeck & Vitvar, 2008) .....	38
Figure 4-1.Research union (Saunders et al., 2011) .....	48
Figure 4-2.Requirements activity.....	50
Figure 4-3.Comparison Of Mashup Tools .....	53
Figure 4-4.Research model .....	56
Figure 5-1.Mashup tool functionality .....	59
Figure 5-2.System architecture .....	60
Figure 5-3.Administrator use case .....	62
Figure 5-4.SerPro API editor flow diagram.....	63
Figure 5-5.Use case diagram for API editor .....	64
Figure 5-6.Widget transformation .....	65
Figure 5-7.Service creation sequence diagram .....	65
Figure 5-8.Request handling sequence diagram .....	67
Figure 5-9.End-user use case diagram .....	68
Figure 6-1.Development process .....	70
Figure 6-2.Mashup tool data flow.....	71
Figure 6-3.Mashup Engine UML class diagram.....	74
Figure 6-4.SerPro annotating tool.....	75
Figure 6-5.API editor workflow .....	76
Figure 6-6.Plain HTML FlickrAddComments service .....	78
Figure 6-7.Sesame server.....	80
Figure 6-8.Mashup Editor.....	82
Figure 6-9.Mashup editor development UML diagram.....	85
Figure 6-10.Development Structure.....	86
Figure 7-1.Validation process .....	90

Figure 7-2.Frequency of users SUS question responses .....	92
Figure 7-3.SUS scores Box of Whisker .....	94
Figure 7-4.Question 1 vs. user's responses.....	95
Figure 7-5.Question 2 vs. user's responses.....	96
Figure 7-6.Question 3 vs. user's responses.....	96
Figure 7-7.Question 4 vs. user's responses.....	97
Figure 7-8.Mashup Tool Comparison.....	103
Figure 8-1.Growth of web APIs (ProgrammableWeb, 2014).....	107
Figure B-1.Microsoft Popfly whisker box .....	128
Figure B-2.Intel mash maker house rentals (Ennals & Shadle 2007).....	128
Figure B-3.Intel MashMaker whisker box .....	129
Figure B-4.Yahoo Pipes whisker box .....	132
Figure B-5.MobiMash (Cappiello et al., 2012).....	133
Figure B-6.MobiMash whisker box .....	134
Figure B-7.Marmite (Wong & Hong, 2007).....	135
Figure B-8.Marmite whisker box.....	136
Figure B-9.Dapper whisker box.....	137

## Table of listings

Listing 5-1. Request handling algorithm .....	67
Listing 6-1.Executing a widget .....	72
Listing 6-2. Adding APIs to Sesame.....	72
Listing 6-3.Saving Mashup details .....	72
Listing 6-4.Populating Mashup details .....	73
Listing 6-5.Executing a published Mashup .....	73
Listing 6-6.Structuring XML to JSON .....	73
Listing 6-7.RDF representation of FlickrAddComments service .....	78
Listing 6-8.Serializes the HTML info to JSON Object .....	79
Listing 6-9.Annotating API details with Ontology.....	79
Listing 6-10.Uploading RDF files to Sesame server .....	79
Listing 6-11.Querying API from Sesame .....	80
Listing 6-12.JSON file Writer.....	81
Listing 6-13.Creates DOJO compatible template .....	81
Listing 6-14.Json file structure .....	81
Listing 6-15.Connecting and wiring two widgets.....	83
Listing 6-16.Unwiring two widgets .....	83
Listing 6-17.Executing request .....	84
Listing 6-18.Instantiating request .....	84
Listing 6-19.Sending request to the Mashup Engine .....	84
Listing 6-20.Response handler in widget piping .....	85

## List of Tables

Table 1-1.Research design .....	8
Table 7-1.SUS score in each question from 12 users .....	92
Table 7-2.SUS scores of proposed Mashup tool.....	93
Table 7-3.Five Whisker terms.....	93
Table 7-4.Comparison Analysis.....	99
Table 7-5.SUS score of Mashup tool.....	102
Table B-1.Microsoft Popfly SUS scores.....	127
Table B-2.IntelMashMaker SUS score .....	129
Table B-3. Yahoo SUS vs. users.....	131
Table B-4.MobiMAsh SUS score .....	133
Table B-5.Marmite SUS score.....	135
Table B-6.Dapper SUS scores .....	137

## List of Acronyms

<b>AJAX</b>	Asynchronous JavaScript And XML
<b>AL</b>	Application logic
<b>API</b>	Application Programming Interface
<b>BPM</b>	Business Process Management
<b>CRUD</b>	Create, Read, Update and Delete
<b>CSS</b>	Cascading Style Sheet
<b>CSV</b>	Comma Separated Value
<b>DA</b>	Data Application
<b>DHTML</b>	Dynamics Hypertext Mark-up Language
<b>DOM</b>	Document Object Model
<b>EUD</b>	End-User Development
<b>EUP</b>	End-User Programming
<b>GUI</b>	Graphical User Interface
<b>hREST</b>	Hypertext Mark-up Language for Representational State Transfer
<b>HTML</b>	Hypertext Mark-up Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>ICT</b>	Information and Communication Technology
<b>JIT</b>	Just In Time
<b>JSON</b>	JavaScript Object Notation
<b>OWL</b>	Web Ontology Language
<b>QoS</b>	Quality of Service
<b>RDF</b>	Resource Description Framework
<b>RDFs</b>	Resource Description Framework Schemas
<b>REST</b>	Representational State Transfer
<b>RO</b>	Research Objective
<b>RQ</b>	Research Question
<b>SAE</b>	SerPro API Editor

<b>SAWSDL</b>	Semantically Annotated Web Service Description Language
<b>SerPro</b>	soa-Services Provisioner
<b>SOA</b>	Service Oriented Architecture
<b>SOAP</b>	Simple Object Access Protocol
<b>SOWER</b>	Sweet is nOt a Wsdl EditoR
<b>SPARQL</b>	SPARQL Protocol and RDF Query Language
<b>SUS</b>	System Usability Scale
<b>SWEET</b>	Semantic Web sERVICE Editing Tool
<b>SWS</b>	Semantic Web Services
<b>UCSD</b>	User Centred System design
<b>UDDI</b>	Universal Description and Discovery Interface
<b>UI</b>	User Application
<b>URI</b>	Universal Resource Identifier
<b>URL</b>	Universal Resource Locator
<b>WADL</b>	Web Application Description Language
<b>WMSL</b>	Web Mashup Scripting Language
<b>WS-BEL</b>	Web Service Business Execution Language
<b>WSDL</b>	Web Service Description Language
<b>WSMF</b>	Web Service Modelling Framework
<b>WSMO</b>	Web Service Modelling Ontology
<b>WSMX</b>	Web Service Modelling Execution
<b>WWW</b>	World Wide Web
<b>XHR</b>	XMLHttpRequest
<b>XML</b>	Extensive Mark-up Language
<b>XML_RPC</b>	XML Remote Procedure Calls



## **Publications**

In the course of the research the following papers were published:

- I. **YALEZO S, THINYANE M (2013).** Enabling Service Composition in ICTD Contexts through GUI Mashup Tool, Southern Africa Telecommunication Networks and Applications Conference (SATNAC 2013), Spier Wine Estate, Stellenbosch, South Africa.
- II. **YALEZO S, THINYANE M (2014).** SerPro: A Mashup tool for enhanced usability for novice users, Southern Africa Telecommunication Networks and Applications Conference (SATNAC 2014), BoardWalk, Port Elizabeth, South Africa.

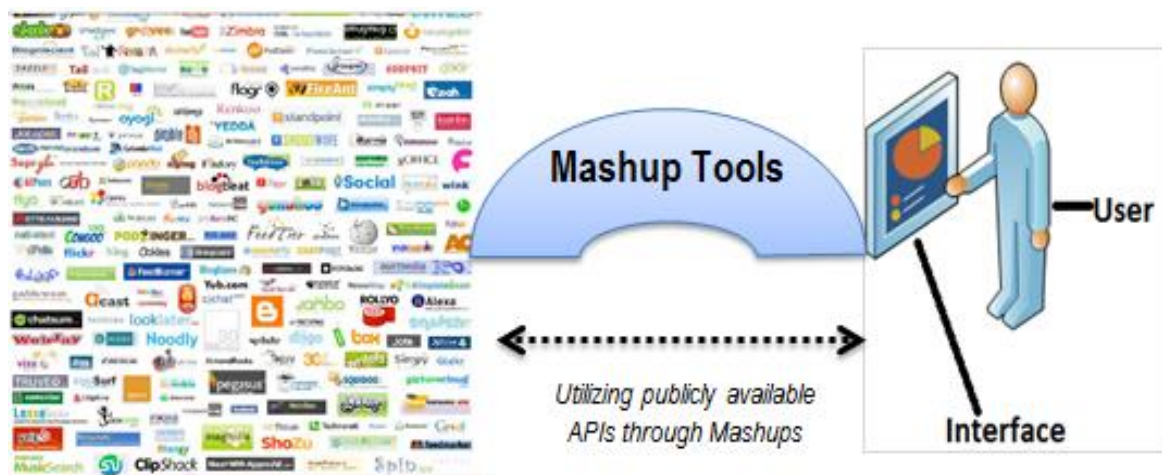
# 1 Chapter 1: Introduction

## 1.1 Introduction

This chapter briefly introduces the research details. It discusses Information and Communication Technologies (ICTs) and background of Mashups. It also covers the thesis statement, research questions, objectives, methodology and chapter outline. In summary, this chapter gives a reader the background and the context of the research project.

## 1.2 Research Rationale

Web applications play an increasingly vital role in the day-to-day lives of millions of people (ranging from social media to online services). This increase has led to web applications overtaking traditional desktop software in terms of the usefulness to end-users (Spivack, 2008). Such a growth in popularity of web applications has been driven by organisations offering a diverse range of online services that provide users with dynamic and feature-rich environments (Halfond, 2010). These trends suggest that the use and importance of web applications will continue to increase significantly in the years to come, and this growth requires high demand of usable web tools. Hence, there is a need for usable web tools to allow novice users to be able to use such technologies.



**Figure 1-1. Research overview**

On the left of Figure 1-1: are the plethora of restful APIs and on the right are the novice users who utilizes the publicly available APIs using Mashup Tools. In this research, novice users are referred to as type of users who are eager to use the web to gain and share information but do not have programming skills (i.e., computer literate).

As mentioned in the research title and as illustrated by Figure 1-1 , this research concerns the evaluation of Mashup tools to enhance the utilization and usability of SOA services by novice users. This research involves usability, which is one of the basic objectives in Human-Computer Interaction (HCI), since HCI is a study that involves the interaction between people and computers. Usability is referred to as the extent to which a user can learn, understand and be satisfied by a system. This means that usability is the measures of how easy the interface is to use and whether the required functionality are provided so that users can do what they want to do. It is difficult to ensure that a system is usable since design goals sometimes conflict one another. The priority of design goals is influenced by the context of use and the targeted group, hence, this research aims to improve the usability of SOA services using Mashup tools with a specific focus on novice users.

The usability of software's is an important aspect for the success of organisation that uses web applications. Moreover, high demands for web applications motivates the development of tools and services that enable web-users to generate web applications that combine content from multiple sources to provide unique services that suit their situational needs. These types of applications are referred to as Mashups (Tuchinda et al., 2008). In Mashup tools, the Mashup Engine that is the software that automates the integration of disparate services to form new services is manages the integration of services then a Mashup editor is used to design and create the Mashup applications.

### **1.2.1 ICTs**

ICT are a diverse set of technological tools and resources used to communicate, and to create, publicize, store, and manage information (Fillip, 2005). Communication and information are at the very heart of web applications, and subsequently the use of ICTs in delivering e-services has a long history. ICTs had been involved in programmes undertaken by governmental agencies, public and private educational institutions and non-profit organisations to provide a better life and improve socio-economic conditions around the world and in Africa (Bertot et al., 2012). This is in the context of 70 % of the population in developing countries having access to a fixed or mobile telephone internet services (Fillip, 2005);and they include the least connected regions of Africa. Such an increased rate of internet connections demands the use of web tools to enhance the utilization of SOA services.

The used communication technologies increase the accountability of online service delivery to the underserved regions, in an innovative, fast, and cost-efficient manner. As ICTs targets

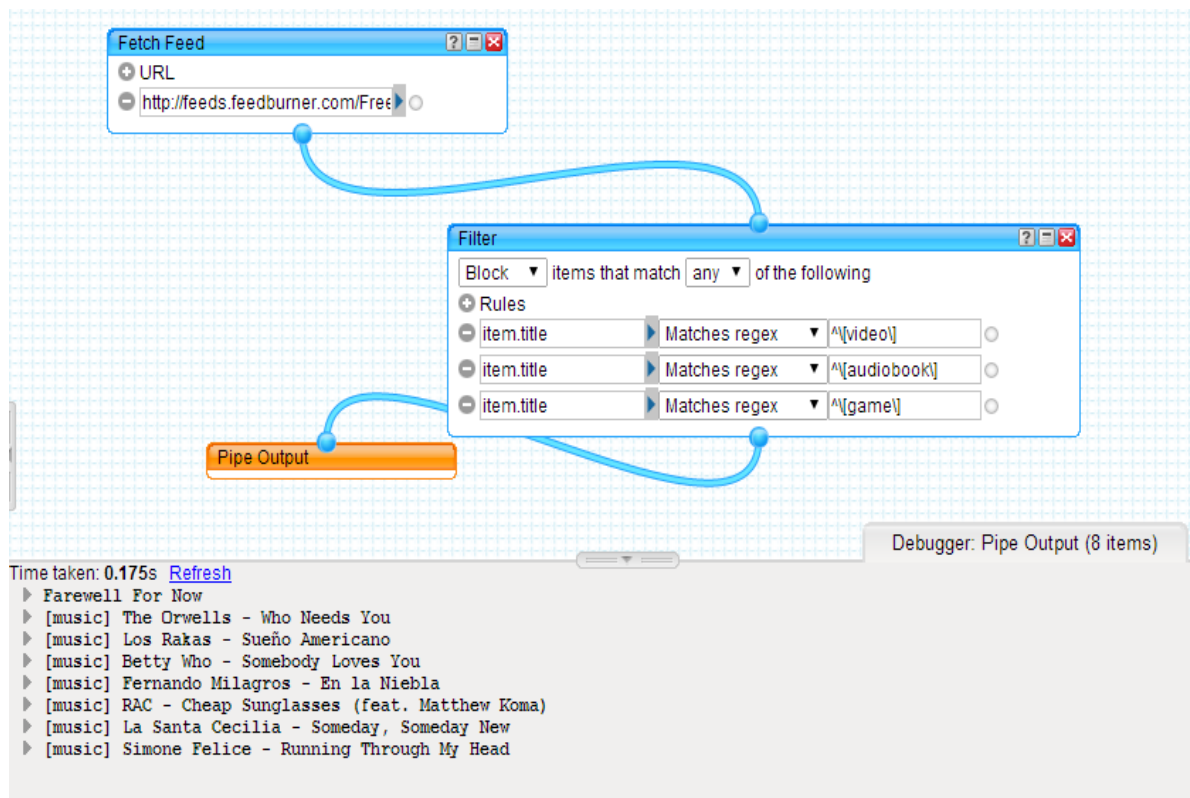
developing countries by creating new ways of doing business and delivering services. Through extending the use of ICTs, this research aim to promote the utilization of SOA services available on the web via a usable Mashup tool, to enable novice users to easily access and benefit from the web. These tools present great opportunities in ICT context since they allow development of web application where there are minimal coding skills. The next subsection briefly discusses Mashups.

### **1.2.2 Mashups**

The dispersed data and services around the web needed to accomplish the user's goals, and are not in a form amenable to the user's needs and capabilities. It is extremely unlikely for the web site developers to fulfil all the user's needs. For this reason, Mashup tools that aid novice users to create web application that suit their needs evolved. One of the earliest and best-known Mashups is housingmaps.com, which crawl rental listings from the Craigslist community web site and puts them on Google Maps (Wong & Hong, 2007; Di Fabbrizio et al., 2009). Rather than having to go through each of the text listings and manually entering each address into a map service, this Mashup makes it easy for end-users to see a map of all the available listings. Mashups combine disparate service to create envisioned personalized services.

There are three types of Mashups namely: Consumer, Data, and Enterprise Mashups (Koschmider et al., 2009). Consumer Mashups (sometimes referred to as Graphical User Interface (GUI) Mashups) are web-based applications that use mainly the presentation layer to interact with the users. Data Mashups are web-based applications that mainly integrate data sources. Enterprise Mashups are web-based applications that are mainly process based Mashups. This research focus is on consumer Mashups to enhance their usability for novice users particular those in rural communities where ICT4D interventions are increasingly being deployed.

Figure 1-2, below illustrates a Mashup application that was created using Yahoo Pipes (i.e. Mashup tool). The purpose of the Mashup designed in Figure 1-2, is to fetch feed and filter results according regular expression, for instance the above example filters everything that is not a video, game and audiobook. At the bottom of the Figure 1-2, results are shown with artist names and the name of the songs or albums.



**Figure 1-2. Yahoo! Pipes (Jones & Churchill, 2009)**

The next subsection discusses the significance of Mashup tools.

### 1.2.3 Significance of Mashups

Mashup applications have recently received a significant attention as one of the supportive web applications. There are three main reasons for this new trend (Latih et al., 2011): The first one is the availability of tools that have enabled end users to develop Mashup applications. The second reason is the availability of web technologies that support this activity. Lastly is the interest and acceptance of Mashup application development by enterprise users. Mashups are the key enablers for a flexible interaction between users and web application since they enhance web contents with cost efficiency and content reuse at the same time they facilitate sharing of resources amongst different businesses and organisations.

Most tools to help novice users create Mashups tend to emphasize programming techniques that are beyond the skills and abilities of average web users. Furthermore, most of the end-user programming tools tend to focus on GUIs (e.g. automating repetitive actions) rather than processing existing web based content and services. Hence, it is inappropriate for end users to carry out online tasks using these tools. One of the challenges experienced by Mashups is heterogeneity of web resources.

#### 1.2.4 Factors Affecting Mashup Usability

This research has identified two factors that affect the usability of Mashups namely: design of interaction platforms and heterogeneity of resources. The design of Mashup editors or interaction techniques affects usability of Mashups by not providing an intuitive design so that users can be able to create Mashups without initial up skilling. Heterogeneity is the measure of degree to which web contents differ. If two agents agree on how to represent and communicate the data, a seamless and conflict-free integration is established. To achieve semantic interoperability two agents (i.e. both humans and machines) need to agree on how to understand the data. The level of understanding differs and depends mostly on the complexity of the formalized knowledge.

Heterogeneity present many challenges to Mashups, since the key guiding principles in Mashups are sharing and reuse. It makes it hard for Mashup to discover web contents and combine them to form new services that satisfies users (Maué et al., 2009). The following listing discusses heterogeneities that hinder flexible integration of web resources:

- i. Application-specific knowledge: there are specific identifiers or terms that are used in an organisations which makes the underlying data hardly usable for other users (Ioannou, 2010). The manner in which such data is represented can only be understood by a certain application, since it was built for a specific domain rather than the entire application domains. Consequently other applications fail to discover such specific data, as it is constrained to a very limited user group.
- ii. Hierarchical problems: probably the most common issue in service discovery is the different level of expertise between the client and the data provider. The way that data is represented often differs in terms of meaning since the order and the perception of using certain words and structure is different (Momeni, 2011). Although some words are spelled the same, they typical differ in meaning according to the contexts of use. Hierarchical challenge makes it impossible to match a user's query and service description even if the chosen languages differ.

Modern web applications have new features and capabilities that significantly increase their complexity. These features and capabilities include dynamically generated HTML contents, interaction with external systems, data structures and data from multiple sources. The dynamic nature of modern web applications limits the use of such techniques. Since the set of generated web contents can vary at runtime, a service composer might be unable to integrate

various web API contents sufficiently; that degrades robustness and result in a cumbersome service combination.

### 1.3 Research Problem

Although there has been an extensive amount of research in Mashup tools (Soriano et al., 2008; Upadhyaya & Zou, 2012; Fisichella & Matera., 2011), many of the techniques developed are not able to adapt to modern web application development approaches. The content used by Mashup tools differ since it is extracted through one or more APIs, for instance the structure or the form of the returned data by web application is a good example of this difference. Eventually these differences cause Mashups to be unusable to novice users since programming skills are needed.

The overall goal of this research is to improve usability of Mashups focussing on novice users. Subsequently, this research focuses on two key parts of this task: the design of a service composer (Mashup tool) and the homogenization of web APIs, in order to allow integration of various public web APIs without the knowledge of programming skills. To address this problem, the thesis statement for the dissertation is as follows: **Semantically enriched web tools to discover transform and compose web contents from public APIs (disparate service) can improve the usability of SOA services to novice users.**

### 1.4 Research Questions

With the stated research problem, the main research question is how can a usable Mashup tool that targets novice users be developed? Following are the sub-questions the research seeks to answer:

- RQ1. What impedes the usability of Mashup tools?
- RQ2. What are the different techniques for Mashup tool service composition?
- RQ3. How can annotation and discovery of APIs be implemented, so as to achieve a smooth composition?
- RQ4. How can semantic web technologies be utilized to improve usability of Mashups?

### 1.5 Research Objectives

The main research objective is to develop a usable Mashup tool for novice users to utilize SOA services, and is accomplished through the following sub-objectives:

- RO1. To identify usability shortfalls of the existing Mashup tools.
- RO2. To investigate service composition on Mashup tools

RO3. To investigate semantic technique on annotating and discovering services.

RO4. To implement a Mashup tool and a Mashup editor

RO5. To implement a homogenization layer and evaluate its effectiveness.

In order to achieve the above research objectives, the next section discusses the methodology used.

## **1.6 Methodology and Research Design**

This section briefly discusses the methodology that was used to achieve the research objectives. It outlines the steps that were taken to the completion the research. Furthermore, it outlines the research design (i.e. how the objectives and question were achieved by mapping them to research methods and chapters used). The following listing discusses the steps that were undertaken in order to complete the research:

- i. Literature review on the background of Mashups, existing Mashup tools, types of Mashups and components of Mashups was undertaken through journals, articles, dissertations, conference proceedings and textbooks.
- ii. The design of Mashup workflow and control flow between Mashup tool and its components was modelled using Unified Modelling Language (UML) diagrams; flow, use-case, and sequence diagrams.
- iii. The Mashup tool was implemented using Java, Eclipse and Tomcat. Mashup editor was developed through GUI libraries such as jsPlumb and API editor was developed using JQuery, and lastly Mashup engine was developed through Java and servlets.
- iv. The Mashup tool was validated through surveys and questionnaires and analyzed through graphs.
- v. Discussion and conclusion was based on how well the analyzed data fit to the research objectives.

The above listing outlined the research methodology, Table 1-1, details how the research questions were addressed in order to accomplish the research.



**Table 1-1. Research design**

Research question	Research objective	Research Method	Chapter
RQ1	RO1	users observations, surveys, focus group	Chap 2
RQ2	RO2 & RO5	Literature and system design	Chap 5 & 6
RQ3	RO4 & RO5	System design and implementation	Chap 3 ,5 & 6
RQ4	RO5	Focus group, Interviews and SUS score	Chap 6 & 7

The following section discusses the contribution of this study.

## **1.7 Contributions of the Research**

As the studies showed that for a massive adoption of Mashups, end-users must be supported by lowering the complexity involved with describing, finding and composing web APIs (Duke et al., 2010; Kenda et al., 2013). A simple Mashup tool, which discovers and compose web APIs was implemented, and it allowed novice users to access and utilise the publicly available online SOA services offered by the businesses and organisations. The improvements and the challenges encountered are discussed in (Section 8.4).

## **1.8 Chapter Outline**

This section presents how the rest of the dissertation chapters are organised.

**Chapter 2:** this chapter focuses on literature review about Mashups. It discusses how to design Mashups and the basic requirements in Mashup developments. Thereafter advantages and benefits are presented followed by the challenges and issues affecting Mashups.

**Chapter 3:** this focuses on the usability of Mashup tools. This chapter discusses the key web technologies used to mitigate the heterogeneity of web APIs. It solely focuses on semantic annotations and ontology developments up to date. Furthermore, the initiatives and approaches developed by other researches are discussed.

**Chapter 4:** this chapter addresses the methodology that was used to achieve the research objectives. Different types of philosophies are discussed and motivation towards the choices made to each research method used.

**Chapter 5:** this chapter focus on the design of the proposed Mashup tool. The system architecture is presented together with its components and modules. Three layers are addressed, namely: discovery, annotation and composition layer. Moreover, user interactions are presented using UML diagram.

**Chapter 6:** this chapter discusses the implementation of the proposed tool with the aid of code snippets. The implementation technologies are also discussed.

**Chapter 7:** this chapter focuses on the evaluation and validation of the system. Two tests are presented (i.e., the usability and the acceptance testing). Focus group, SUS and the interview are then used present the results.

**Chapter 8:** this chapter presents the discussion from the obtain results. It interprets the meaning, significance and importance of the obtained results. Research limitation and recommendations are presented and finally the research summary and the overall conclusion of the research.

## **1.9 Conclusion**

Mashup tools are explicitly designed for sharing and reusing the publicly available web APIs. They are useful for diverse areas and can support a great variety of input and output structured data (i.e. eXtensive Mark-up Language (XML) and JavaScript Object Notation (JSON)). These tools in general have intuitive designs and easy-to-use web interfaces that require little training for beginners. The extraction, recombination and integration of data with these tools are easier than writing code in a particular programming language. In addition, users can share their application amongst one another since the participation of users is a major benefit in web 2.0 tools. As it is only users who know what they want, Mashups allows them to design and create new web applications much more rapidly than with traditional software engineering methods. Despite the strengths some issues arise when using Mashup tools as discussed in the problem statement, nevertheless, they still hold a great opportunity in provisioning service to novice users.

## **Part I: Research Context and Related Work**

## **2 Chapter 2: Literature Review**

### **2.1 Introduction**

The previous chapter discussed the applications and significance of Mashups. This chapter discusses service composition and the types of content aggregation that had been used in the past years namely Portals and Mashups. Content aggregation is the concept of facilitating the sharing and reuse of services or resources on the web. It offers many benefits as it decreases the cost and the time it takes to implement complete web applications. This chapter also focuses on Mashup design, challenges facing Mashups and a brief discussion on usability of Mashups.

### **2.2 Service Composition**

The integration of web services into composite applications is regarded as service composition. Service composition is usually performed on the business logic layer, and results in executable plans that fulfil certain requirements of the newly created web application (Nestler et al., 2009). It refers to a way of delivering enterprise solutions by assembling them from already existing components, instead of building them from scratch. It also includes personalization and customization abilities, so that users can easily and quickly modify specific functionalities. Service composition contains two steps namely: retrieval of data, and combination of web contents. This process of service composition involves the aggregation of content from various web resources used to form a specific service. Web services are used as the basic components (sometimes referred to as APIs) used to form Mashups. There are two flavours of web services: Simple Object Access Protocol (SOAP) and Representational State Transfer (REST), both flavours can be exploited as usable building blocks for the creation of new web applications. There are initiatives aiming at promoting service composition (also known as content aggregators) such as Mashups, Portals and WS-BPEL (Weske et al., 2006). The research does not discuss on WS-BPEL as it is out of the scope. There have been some research inputs towards the combination of various web contents and they include portals and Mashups; the rest of this section details on both approaches.

#### **2.2.1 Portals**

Portals being an older technology designed as an extension to traditional dynamic web applications, in which the process of converting data content into marked-up web pages is split into two phases: generating fragments and aggregating fragments into pages (Miah et al.,

2012). Each mark-up fragment is generated by a portlet, and the Portal combines them into a single web page. Portlets may be hosted locally on the Portal server or remotely on a separate server. The Portal technology defines a complete event model covering reads and updates. A request for an aggregate page on a portal is translated into individual read operations on all the portlets that form the page. Portal technology is about server-side and presentation-tier aggregation; moreover, it cannot be used to drive more robust forms of application integration such as two-way interaction. The research does not discuss details of portals as they are out of the scope. As of weaknesses they portrayed, portals are now implemented using the Mashup styles.

### **2.2.2 Mashups**

Mashups are web applications that combine presentation, functionality and data from multiple services to help users solve tasks not originally envisioned by the web site designers (Lin et al., 2009). Mashup tools are software tools that assist users in developing Mashup applications. There exist various Mashup tools that are targeted at certain types of users namely; technical users and novice users. Technical users are a group of users that have a skill in programming. End-users are users that do not have programming skill however they are computers literate (Latih et al., 2011).

#### **2.2.2.1 Definition of Mashups**

This subsection defines the building components of a Mashup tool namely: Mashup application, Mashup engine, and Mashup editor.

**Definition 1:** Mashup applications are referred to as Mashups, which are the resultant application created through Mashup editors and/ or Mashup engine.

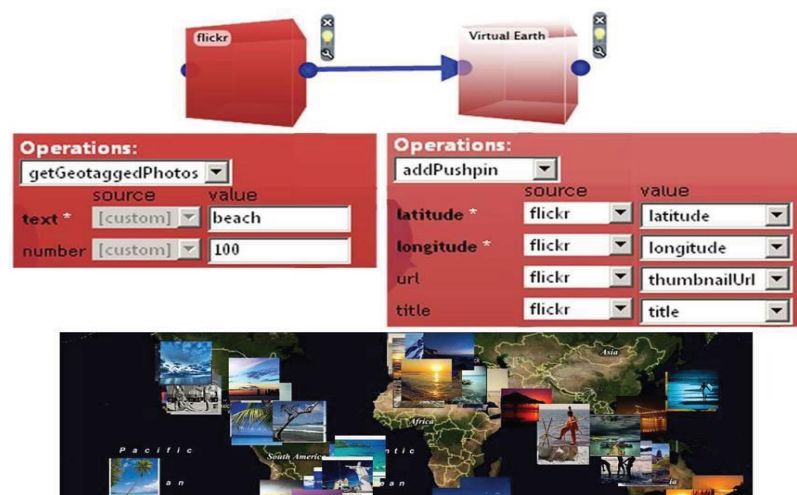
**Definition 2:** Mashup engine is the software program designed to automate the process of service composition and;

**Definition 3:** Mashup editor are the visual support elements used for user interaction. Mashup engine together with Mashup editor form what the research refer to as a Mashup tool.

Different architectures are used by different Mashups, and Mashup architectures are discussed in (Section 2.5.3). The following subsection discusses the background of Mashups.

### 2.2.2.2 Mashup Background

Mashups started in a musical domain by remixing isolated original songs to form new ones (Ogrinz, 2009). With the quick adaptations and advantages offered in the music domain, Mashups were adopted by the web domain. Since the birth of web Mashups, businesses had been creating Mashups and the process was manual according to (Ogrinz,2009). Moreover Ogrinz (2009), claims that Microsoft Excel is arguably the father of the corporate data Mashups, since for years, Excel end-users were duplicating data to feed their calculation engines. Spreadsheet-based Mashups were used throughout the enterprise without the involvement of IT. According to Meditskos and Bassiliades (2011), Daniel et al. (2012a), Mashup tools enabled the automation of integrating resources so that users could easily utilize the available web resources, nevertheless, there are still Mashups that are developed manually which strongly targets programmers. Tuchinda et al. (2008) argue that Mashup editors together with Mashup tools enabled web users to generate web applications that combine content from multiple sources, and provide them as unique services that suite their situational needs. Following Figure 2-1 illustrates a Mashup application that is developed using a Mashup tool:



**Figure 2-1. Microsoft Popfly (Loton, 2008)**

Figure 2-1, shows a Mashup example in which the Flickr block sends a list of images about “beaches” with their geographical coordinates to the Virtual Earth block (Figure 2-1: top and middle) to display them on a map (Figure 2-1: bottom).

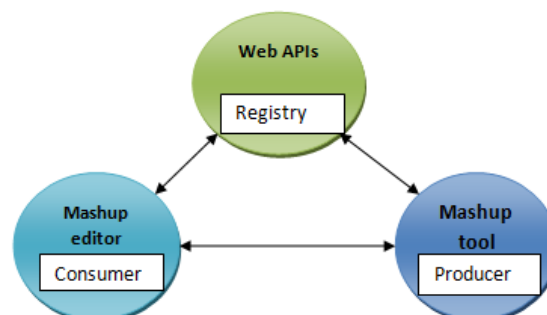
As web Mashups were used to combine data, functionality and presentation to create new web-based applications. Cetin et al. (2007) proposed web Mashups as a one of the approaches that could be used for migration of legacy system to better paradigms, for instance SOA.

Migration is defined as the movement of services from old operating environment to new ones with minimal coding. Mashup facilitate more general platforms and execution environment such as SOA. As the goal of this research is to create a usable Mashup tool that allow novice users to build Mashups easily with no programming experience; Mashup tools are chosen as the key service composition enablers. The following sections give a brief background of concepts and web technologies employed by the Mashup tools.

### 2.3 Mashups as a Service Composition Technique

Mashups are about simplicity, usability, and ease of access. This simplicity has the upper hand over feature completeness or full extensibility. There are two formalities, which lead to identification of heterogeneous web contents as a challenge within Mashup tools namely: sharing and reuse. Sharing of web contents is of great importance as it allows various groups of users to utilize services created by the others. Mashups facilitate sharing by cataloguing newly created services to a repository, which is normally implemented through a middleware. The second focus of Mashups is geared towards web content reuse. Web content reuse is a study that had evolved several technologies in order to gain its true potential. Web contents are derived from web resources in order to simplify the development of Mashups applications.

In order to embrace the primary goals of Mashup tools, studies shows that they are implemented as SOA centric web based application (Benhaddi et al., 2012). SOA facilitates the two formalities through its building blocks and service architecture and is composed of three actors namely service provider, service consumer and repository (Chapin, 2010). With the belief that Mashups implement SOA, Figure 2-2 illustrates that claim and further discusses on what basis does the claim holds.



**Figure 2-2.SOA artefacts**

In web services and SOA as stated in the above paragraph that there are three building blocks known as: producer, consumer and repository of the services. Service producer provides and publishes web services to a registry, so that consumers can be aware of services that are available. Service consumer is the user of the services. Mashup editors correspond to consumers and Mashup tools correspond to service providers and lastly the repository maps to both the databases and the online resources used by the Mashups. The next section discusses the adopted web technologies used by Mashups for added robustness.

### **2.3.1 Adopted Web Technologies**

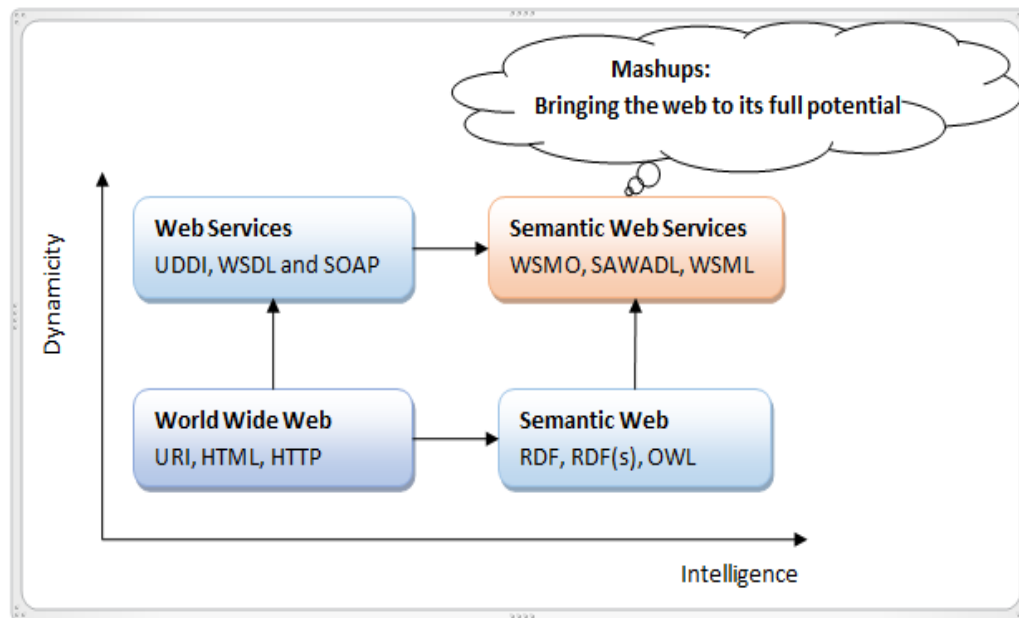
This section presents the web technologies and paradigms that are adopted by the Mashup tools to mitigate the challenges experienced by Mashup developers such as heterogeneity. This subsection does not detail on Mashup challenges rather it focuses on some employed concepts that reduces complexity in Mashup development. In order to homogenize data, Semantic Web Services (SWS) promises to produce web services that are semantically annotated. SWS yields web services that are intelligent (i.e. they can be automatically or semi-automatically described, published, selected and discovered) (Rao, 2004). The aim of the SWS is to create a platform where both humans and web machine agents are able to consume information in the web, using semantic web technologies and web services. There are initiatives that have to be considered in the context of SWS:

- i. It uses the Web Service Modelling Ontology (WSMO) which refines and extends the Web Service Modelling Framework (WSMF) with ontologies. The WSMF is a framework used to model the development and the description of Web services based on two main requirements: maximal decoupling and strong mediation. WSMO also employs a concept known as Web Service Modelling Language (WSML), which is the language that allows the writing of annotations for web services using WSMF.
- ii. Lastly, SWS uses Web Service Modelling Execution (WSMX) as an execution environment in order to discover, select and invoke web services. SWS are capable of referring to the consumer and provider information and perceiving the conditions in order to automatically settle contracts through negotiation (Choi et al., 2006).

There are other annotation-based approaches such as Semantically Annotated Web Service Description Language (SAWSDL), Hypertext Mark-up Language for REST



Services (hRESTS) and WSMO-lite, which enabled the creation of lightweight semantic web services, which are discussed in (Section 3.3). With the introduction of SWS, the Figure 2-3 below shows the key concepts used by Mashups according to (Choi et al., 2006; Pedrinaci & Domingue, 2010).



**Figure 2-3.SWS formation adapted: (Pedrinaci & Domingue, 2010)**

As depicted in Figure 2-3, World Wide Web (WWW) has played a major role in associating web contents with humans. It offered users with limited access due to the scarcity and costs that were involved such that only a certain group of users were able to use the web -: university or college and research labs (Pedrinaci & Domingue, 2010). It mainly used Hypertext Transfer Protocol (HTTP), Hypertext Mark-up Language (HTML) and Universal Resource Identifier's (URI) as the core protocols. This first wave of web evolution is referred to as web 1.0, where users could only read from the Web but they could not do further interactions (O'reilly, 2007). The following subsection details some of the key technologies in web 2.0, which is second wave of the web that allows users to be both producers and consumer of web resources.

### 2.3.2 Web Services and Semantic Web Technologies

Web services are defined as self-contained, modular units of application logic, which provide business functionality to other applications via an internet connection. They can be integrated with each other to fulfil the user's requirement, regardless of the languages that are used to implement them and the platforms where they are executed (Pedrinaci et al., 2012). Web

services present many benefits to the web such that its functionality need to be described with additional pieces of information called semantic annotation which adds meaning to them (Srivastava & Koehler, 2003). Furthermore web services are semantically annotated with ontologies using Semantic web technologies (Feier et al., 2005). In order to establish understanding between that will not restrict the usability of web services mostly to human users and machine user around the web.

According to Koschmider et al. (2009), Mashups implements SOA since most of SOA-centric organisations use web services which are also used by the Mashups. Web services implement SOA, which is a set of well-defined interfaces and guidelines used in software development. Moreover, web services are said to be operations that can be performed on an online resource. They use XML languages such as Web Service Description Language (WSDL) to describe them, SOAP to access them and Universal Description Discovery Integration (UDDI) to discover them (Rao, 2004). These XML languages allow different applications to exchange data regardless of the operating systems, programming languages and execution environments underlying those applications.

Semantic web promises to make web contents machine understandable, allowing machines and applications to access a variety of heterogeneous resources. They also process, integrate web content, and produce added-value output for users. The way in which semantic web technologies work is as such: pages not only store content as a set of unrelated words in a document, but also code their meaning and structure, by so doing they give access to millions of resources, irrespective of their physical location and language (Rao, 2004). In general, the Semantic web languages include the following standards:

- i. The Resource Description Framework (RDF) which is family of standards that leverages URI and XML for meaningful descriptions using triples. Triples allow resources to be described in the form of metadata by means of <subjects, predicates, object>. URI uniquely identifies resources and relations amongst them (Pietro et al., 2008).
- ii. RDF Schema (RDFS) is an extension of RDF, which defines a simple modelling language on top of RDF and it enables the representation of class, property and constraint.
- iii. Ontologies specification languages, such as Web Ontology Language (OWL) as conceptual or data models.

There are many advantages when semantic web technologies are combined with web services to leverage WWW. They result in SWS, which is all about making web services intelligent by annotating them with semantic web technologies (Benjamins et al., 2002).

This section discussed the two ways of aggregating content namely portal and Mashups. Mashups seemed to be the successors of Portals and some of the Mashup key web enabling technologies and concepts were discussed. Before delving into details on how Mashup tools are designed, the research discusses a concept of great importance towards developing web application for novice users. The concept is known as End User Programming (EUP) and is discussed in the following section.

## **2.4 EUP in Mashup Editors**

EUP is playing an increasingly central role in shaping software to meet the broad, varied, rapidly changing needs of the world as the number of users continues to grow in the web (Wong & Hong, 2007). Programming is referred to as the process of transforming a mental plan of desired actions for a computer into a representation that can be understood by the computer. End-user programmers are users who are not formally trained in programming, yet need to program in order to accomplish their daily task (Mehandjiev & Angeli, 2012). EUP enables end users to create their own programs. End-user programmers are also defined as people who write programs, but not as their primary job function i.e. they write programs in support of achieving their main goal allowing diverse communities to use software's and application without knowledge of computer programming. EUP include paradigms such as widgets, Pipes and filters (Bolin, 2005). Programming for end users is one of the important parts of in this research, since the main aim is to allow novice users to access SOA services and create web applications with simplicity. The next sub sections details on the EUP paradigms.

### **2.4.1 Widgets**

A web widget is a small application with limited functionality that can be installed and executed within a web page by an end user (Soylu et al., 2011). A widget occupies a portion of a webpage and does something useful with the information that is fetched from other websites and it can be represented in a certain style. Widgets are typically created in Dynamic HTML (DHTML), JavaScript, or Adobe Flash. A widget is a stand-alone application that can be embedded into third party sites by any user on a page where they have rights of ownership (Batard et al., 2011). They also allow users to turn personal content into dynamic web apps that can be shared on websites where the code can be installed. Widgets are used as small

stand-alone web application in Mashups. They can be connected with each other through Pipes and filters to create meaningful applications.

### **2.4.2 Pipes and Filters**

Many Mashups leverage the Pipes and Filters integration pattern to create the logic for data combination (Latih et al., 2011). The implementation of pipes and filters pattern consists of a series of interconnected components. Each component performs a specific function such as filtering unnecessary data or joining records. The connection between components denotes that the output from one component is the input to the other component (Xuanzhe et al., 2010). The advantage of using the pipes and filters pattern is the simplicity in modelling data flows. Each pipe or filter in the flow is explicitly defined and their integration is driven by the data flow (Cao et al., 2010). Data sources are filtered, processed and combined to produce new mashed-up information. In addition, it is easier to maintain the Mashups built using this pattern since the Mashups execute exactly as they are designed following the flow paths at runtime.

### **2.4.3 Data Federation**

The data federation pattern aims to efficiently federate both structured and unstructured data from multiple disparate sources (Latih et al., 2011). It helps to mitigate the burden of aggregating, correlating and correcting relevant yet heterogonous data. This pattern supports data operations against a virtual view. At design time, views of the data sources are declared and the relationships between such views are defined as operations including joins, unions, projections, selections and aggregations. The data federation pattern can be realized by several pipes and filters, each providing the Mashup component with an interface to access the data (Papadakis et al., 2011). In the context of Mashups, the data redundancy, therefore, this data can be stored and reused later without going through the data processing pipes and filters every time they are needed. Thus introducing some level of redundancy can both simplify the design and improve the responsiveness of Mashups.

In the following sections, a thorough look is conducted on Mashups design, which is the crucial process that determines the success of a Mashup tool.

## **2.5 Designing Mashups**

This section discusses the requirements needed to develop Mashups. The architecture, support and execution environments are discussed. The next subsection presents the design and support provided by the Mashup Engine and the Mashup editor.

### **2.5.1 Mashup Engine**

Mashup tools apply in different domains for different users; therefore, the way they offer support to users varies. According to Latih et al. (2011), Mashup tools can be categorized into three domains automated, semi-automated and manual (scripting tools). This research does not discuss about the Mashup tools that offers manual support to integrate APIs, as they mainly target developers.

#### **2.5.1.1 Automated Mashup Tools**

This category of Mashup tools normally supports the development of situational Mashup applications. A situational application is an application that is developed rapidly to address an immediate need of an individual or small community. It is created for a specific situation and it is utilized only for short periods while the situation exists. It is a Just-in-time (JIT) solution but not necessarily short-lived. Examples of Mashup tool that support the development of situational Mashup applications are MaxMash and Automatic Mashup of Composite Application (Latih et al., 2011).

#### **2.5.1.2 Semi-Automated Mashup Tools**

Semi-automated Mashup tools were developed to support end user in developing Mashup applications. Such tools are developed based on End-User Development (EUD) paradigms (Bolin, 2005). EUD paradigms provide programming capabilities for everyone by pushing on different aspects of computing technologies. One of the EUD paradigms is wiring or dataflow paradigm is where several selected widgets that support particular functions are connect together for example are Yahoo! Pipes and Marmite. There are also tools that use widgets only to create Mashups, each widget represents a particular function, users just select it for use and they get response for example; tools like iGoogle and Netvibes.

#### **2.5.1.3 Scripting Tools**

Lastly, Scripting tools Mashup applications require some form of programming or coding. The coding process is simplified by using scripting languages rather than the normal programming languages. Such Mashup tools are developed for technical users. An example of a Mashup tool in this category is Web Mashup Scripting Language (WMSL) (Sabbouh et al., 2007).

The research adopts the semi-automated style, since it assists users in creating Mashups although the process and tasks are not fully automated. The next subsection details the design

of Mashup editors, since designing a Mashup tool not only needs support offered by the Mashup engine to integrate public web APIs but also need Mashup editor which allows novice users to interact with Mashup tools easily and flexible.

### **2.5.2 Mashup Editor**

Mashup editor is an environment that allows end users to remix the available public web APIs. According to Latih et al. (2011), there are three types of support offered by Mashup editors namely: Visual Mashup editor, Browser extension application and Web portal Mashup tools.

#### **2.5.2.1 Visual Mashup Editor**

A visual Mashup editor allows users to create Mashup application by manipulating program elements graphically rather than by specifying them textually. For instance, Yahoo! Pipes let users to develop Mashup applications via a visual editor by selecting a few modules and wire them together (Latih et al., 2011).

#### **2.5.2.2 Browser Extension**

A browser extension application is developed as a plug-in application to web browsers. By using this tool, users can develop a Mashup while browsing the web for instance Intel Mashmaker. It is a plug-in to Firefox web browser and it suggest to users on what type of data to integrate. Other browser extension Mashup tools are Marmite and Vegemite (Latih et al., 2011).

#### **2.5.2.3 Web Portal Mashup Tools**

Web portal Mashup tools are simple tools that provide users with a dashboard full of widgets ready to be mashed. In this type of Mashup tool, data sources are aggregate into a single layout. They reuse the existing Mashup applications, for example, users can stream their emails from Gmail, get local and world weather information, and pull RSS from their favourite sites. Other examples are MyYahoo and Facebook (Latih et al., 2011; Miah et al., 2012).

The next subsection discusses the components, composition and execution environments of Mashups and the type of support offered by the Mashup tools to integrate public web APIs.

### **2.5.3 Mashup Tool Architecture**

This subsection discusses the two architectures of Mashup application that is server side Mashups and client side Mashup. These architectures enable developers to realise the

lucrative side for mixing contents. Some developers prefer client-side and some server-side depending on resources and situation in hand. Furthermore, this subsection covers the types of support offered by Mashup tools. Noting that Mashup tools objective is to simplify the integration process by automating instead of manually combining various web resources.

Mashups are designed with two different architectures, depending on where process and data integration takes place. If it takes place on the server-side, the Mashup is called a server-side Mashup. In this case, once the result is produced on the server, it will be pushed to the client for the sake of presentation. Alternatively, both integration and visualization tasks can be performed in client browser, which results in a client-side Mashup (Aghaee & Pautasso, 2010). Moreover, in a server-side Mashup all the requests from the client pass through the server, which acts as a proxy to make calls to the other web sites (Aghaee & Pautasso, 2010). Server side Mashups are more secured and support multithreading. Multithreading is when a server execute different threads simultaneously to solve task at hand, and threads are the lightweight processes (i.e. each thread serve a unique function). They are secured since the data transmission between the Mashup client and the server is normally done through client-side XMLHttpRequest (XHR) calls though when dealing with real time data, the communication can be through web sockets (Anttonen et al., 2011). This research focuses on using server side as the location for content mixing.

The integration process side is not enough for the design of a Mashup tool since prior development; it has to be clear what execution environment the Mashup tool targets. The next subsection discusses how Mashups behave at runtime.

#### **2.5.4 Mashup Tool Execution Environments**

A Mashup execution environment defines the device that Mashup application can run on. There are two devices that currently run Mashups applications namely desktop and mobile devices. Desktop Mashups includes PC, laptops and servers while mobile devices include smart phones, tablets and Personal Digital Assistant (PDA). Mobile Mashups serve as an alternative solution as compared to hybrid apps, since they can be accessed from any browsers on any platform that enables portability. Hybrid apps and mobile Mashup combines reusable components from a large pool of web APIs, while mobile Mashups can also be developed using end-user programming Mashup tools such as Yahoo! Pipes. Such end user tools require less effort as opposed to hybrid apps since they are platform-independent and building them requires no programming skills. These observed trends in Mashup domain

increases demands on the standards and the methodologies that are used to build Mashups, so that they can accommodate mobile devices. For instance with the use of Cascading Style Sheet (CSS3) and HTML5, desktop Mashups can be easily ported to mobile browsers and vice versa (Aghaee & Pautasso, 2010).

This section discussed the design and support offered by Mashup tools, the next section discusses the technologies and patterns used to integrate data.

## **2.6 Mashup Tools Composition Styles**

The composition model determines how components are integrated to form the Mashup, assuming components are readily available. To facilitate end-user compositions, the composition model should be as simple as possible. A composition model has several distinct characteristics: output type, orchestration style, and data-passing styles (Benslimane et al., 2008):

- i. First, we distinguish the model's output type. As with components in input, composition output can be of type Data Application (DA), Application logic (AL), or User Interface (UI), depending on whether the composition provides data, programmable APIs, or applications with a user interface.
- ii. The second characteristic is orchestration style. Orchestrating components implies specifying how developers define and synchronize the components' execution. Three main approaches exist:
  - a. Flow-based styles define orchestration as sequencing or partial order among tasks or components and are expressed through flow chart-like formalisms.
  - b. Event-based approaches use publishes–subscribe models. They're particularly powerful for maintaining synchronized behaviour among components.
  - c. In the layout-based style, components (with or without user interfaces) are arranged in the composite application's common layout. Each component's behaviour is specified individually by accounting for the other components' reactions to user interactions.
- iii. Lastly, is the data passing style of a Mashup tool. There are two data-passing approaches: a dataflow approach, in which data flows from component to component; and a blackboard approach, in which data is written to variables, which serve as the source and target of operation invocation on components, much like in programming languages.



This subsection discussed the different composition styles used within Mashup development, and the next subsection presents the data integration patterns.

### **2.6.1 Composition Patterns**

To integrate data in Mashups requires Mashup patterns. Design patterns in software industry are general reusable solutions to some commonly occurring problem. Patterns describe how a certain type of problem can be solved in many different situations. Ogrinz (2009) classifies Mashup patterns into five categories: harvest patterns, enhance patterns, assemble patterns, manage patterns, and test patterns:

#### **2.6.1.1 Harvesting Pattern**

Harvesting pattern is used to extract data from both structured and unstructured sources. Structured sources can be databases, RSS feeds, XML, and other data streams. Unstructured sources can be web sites, binary files, or free form text. Harvest patterns include mechanisms to navigate to important information and then retrieve it. For instance, Ogrinz (2009) identifies two harvest patterns referred to as API Enabler and Infinite Monkeys. The former is used to create REST or other kind of interface for previously closed resources, and the latter is used to extract larger quantities of data to be examined more closely later. Often these two can be combined or executed in parallel.

#### **2.6.1.2 Enhance Patterns**

Enhance patterns make keeping Mashups running less time and money consuming. This happens in five ways: extending Mashups to a wider audience, fixing bugs without touching the underlying code, making software more user friendly, improving the “findability” of data, and incorporating changing business rules. Enhance patterns have emphasis on changing the interaction model of a Mashup to keep it useful.

#### **2.6.1.3 Assemble Patterns**

Assemble patterns can be used when developing the core function of Mashups: adding resources together from multiple sources to create some-thing new. Often the problem to be solved with assemble patterns is availability of source data in suitable format or converting data from a format to another. Sometimes the problem is the amount of data, which is so excessive that presenting it to the user becomes an issue.

#### **2.6.1.4 Manage Patterns**

Management patterns can be used as a mechanism for transmitting and storing data, handling data and securing access to data. Manage patterns include this kind of actions in order to maintain and manage Mashups. Managing data reduces number of bugs that can affect the system performance.

#### **2.6.1.5 Test Patterns**

Test patterns describe mechanisms that make it easy to understand and implement Mashups. Since users can also develop Mashups, there is also an obligation for the users acting as developers to test their Mashups at least in some degree.

Having discussed the patterns used in Mashup tool for data integration, the following subsection focuses on workflow of Mashup applications.

### **2.6.2 Workflows on Mashup process**

Workflow describes the flow of data, and in, Mashup tools it, means being able to specify a control and data flow over human tasks. That is, it requires being able to define sequences, branches, or conditional executions of work items. In the early nineties, establishing such kind of process logic as own modelling concern and detaching its execution from monolithic software systems was one of the major contributions of workflow management systems (Hahmann, 2005).

Typically, modern process modelling formalisms distinguish between control flow (i.e., the order of tasks) and data flow (i.e., the way data is propagated among tasks). The coordination of multiple actors may happen via control flow only, since business data may also flow only in the real world from one actor to another, yet the process needs to progress. Today's data Mashups (e.g., Yahoo! Pipes) or service compositions are already highly process-based without implementing a control flow (Philips et al., 2010). In fact, these types of data Mashups are typically data flow based only. Very likely, the reason for this choice is simplicity and the lack of the need to coordinate multiple actors.

UI-based Mashups are more event-based, yet when it comes to the integration of web services, control flows to express the service orchestration logic is typically needed (Freudenstein et al., 2007). Furthermore, Freudenstein et al. (2007), argue that “Mashing up people, UIs, data, and services and mapping them to an overall workflow requires various techniques such deciding on granularity of actors and services and process state feedbacks”.

The next section discusses the prebuilt resources (components) used by Mashup tools in the process of service composition.

## **2.7 Web content mining**

As outlined in (Section 2.3) that web services are the primary building blocks of Mashups, this section focuses on APIs. Mashup use APIs to extract data from the web. APIs maybe implemented in different flavours including HTML, REST, SOAP and XML Remote Procedure Calls (XML\_RPC). Web APIs are typically a related collection of web services. An example of a web API includes the Google API, which makes various search functions available over the internet for use in third-party applications. Most web APIs have a free limited-use license or trial period, and some of them require payment for the service. These API's are usually contained in Programmableweb website (Lee et al., 2009). The following subsections discuss methods and ways that are used to get data from the public web APIs.

### **2.7.1 Web Page**

Obtaining the necessary information for discovering or invoking the vast majority of Web APIs nowadays requires interpreting highly heterogeneous HTML pages that provides documentation for developers. Tag-based Segmentation approaches are used to analyse the Document Object Model (DOM) tree of web page and automatically divide it into sub trees in order to eventually extract information. These techniques usually rely on specific HTML tags (e.g., <table>). However, this approach does not perform well when dealing with heterogeneous cases where various tags are used as separators.

### **2.7.2 Web API Documentation**

Given a web page documentation and a web API, extracting web contents consists of two main steps: Firstly web pages are pre-processed to i) circumvent any issues with incorrect HTML pages, and ii) to remove scripts and images which will not be taken into account for feature extraction. The second step takes care of processing and analysing the cleaned web pages to extract the main technical features of the web APIs such as the operation names, and URI templates.

### **2.7.3 RPC-style Web APIs**

RPC is a communication style that is centred on the notion of operations or procedures, which essentially define the actions that remote components can trigger. Web APIs adopting this style of communication offer programmatic access to the data and functionality hosted on

the underlying web site by means of an arbitrary number of operations. Additionally, the documentation sometimes indicates further details such as the invocation endpoint (i.e., URL) or the HTTP method to be used. Even though the blocks are both taken from RPC-style web APIs documentation, the underlying structure and their visual appearance differ significantly, providing an example of the heterogeneity among web APIs documentation. In extracting relevant information from RPC-style web APIs the focus is centred on the detection of blocks providing information about the different operations.

#### **2.7.4 RESTful Web APIs**

Web APIs that conform to REST principles are characterised by a communication model whereby requests and responses are built around the transfer of representations of resources and a prefixed set of operations one can carry over these. The documentation of RESTful web APIs is centred on the notion of resource and the parameters required for identifying these resources as well as on the actual operations allowed over the resources which in the case of web APIs is indicated by the HTTP method to be used. Given the characteristics of RESTful interfaces, the main goal is to detect and extract the resources, their URIs (templates) along with the HTTP method to be used and the corresponding description. REST naming conventions are hardly usable since there is no usual criterion for defining resource identifiers. Instead, the most characteristic elements in this case are URI templates and HTTP methods although they occur in several places throughout the documentation (e.g., one may GET and POST a resource) which hampers significantly the segmentation of the web page into coherent sets of resources and operations. The problem often caused by URIs is partially eliminated by adopting the same repetitive pattern for documenting each resource and the operations available (Ly et al., 2012).

#### **2.7.5 SOAP**

Web APIs that conforms to SOAP are said to facilitate the automation of web application. They are described using WSDL and published using UDDI through the ubiquitous HTTP. In order to consume a SOAP web API, developers need to be familiar with XML markup languages. SOAP APIs permits developers to view what types of remote operations are available for a certain service. Even the datatypes returned by the APIs are described to the developer.

### 2.7.6 Mashup Repositories

Programmable web is one of the repositories dedicated to promote Mashup and APIs (ProgrammableWeb, 2014). It allows users to keep track of APIs of their interest and to discover some of the newly created APIs and Mashups. GooglemapsMania is also Mashup repository which is dedicated to promote map-Mashups, those Mashups that are presented on the maps (MapsMania, n.d.).

### 2.8 Applications of Mashups

Mashups have a strong flavour of reuse. They help developers not to reinvent the wheel when someone else has implemented it and provided an API. APIs provide very useful functionality that no longer requires being re-implemented. All of this enables a rapid development when building new web applications, which is perhaps the most compelling reason for using Mashups. There are many categories of Mashup, for instance Figure 2-4, depict the observed trends of Mashups in programmable web:



**Figure 2-4. Programmable Web (ProgrammableWeb, 2014)**

As depicted Figure 2-4 that the number of publicly and commercial APIs are increasing in the recent years. As of 2003 there were few developers who dedicated themselves to provide and share web API, since the infrastructure that existed was not supportive towards rapid development of API and web applications (ProgrammableWeb, 2014).

With so many people using Mashups, there is a demand for API developers to create APIs for other developers as to enable integration of data and services. There are a few limitations to using Mashups and most of these only appear when combining externally hosted services. For example, some services provide limits to the number of calls to an exposed web service, and almost all require some sort of registration. Also when mashing up external APIs, it is demanded that developers keep track of changes (i.e. to know that service is still stable and reliable).

The following listing discusses the benefits and advantages possessed by Mashups and service provisioning platforms:

- i. **Applicability:** the Mashup tools available for Mashups are useful for diverse areas and support a great variety of input and output data types, from Comma Separated Value (CSV) formats to structured XML and semantically rich RDF. Common Mashup tasks such as data integration by mapping identities can be easily performed (e.g. with Yahoo! Pipes even without explicit coding).
- ii. **Simplicity:** the Mashup tools in general have intuitive designs and easy-to-use web interfaces that require little training for beginners. Extraction, recombination and integration of data with these tools is easier than writing code in a particular programming language.
- iii. **Reusability:** Mashup tools are explicitly designed for sharing and reusing. The author of a Mashup can describe and publish his Mashup and others can reuse and add new features or customize and combine it into something new.
- iv. **Interoperability:** various Mashup tools can be easily combined to enhance the Mashup capability. For example, Dapper can be used to fetch data in different formats (Sigelman et al., 2010).
- v. **User participation:** the participation of users is a major benefit in web 2.0 tools. Activities, such as suggesting or even implementing new features by users, facilitate the improvement of applications much more rapidly than with traditional software methods.

Despite these strengths, some issues arise when using Mashup tools. The next section discusses the factors that hinder a wide adoption of Mashups.

## 2.9 Challenges Facing Mashups

In Mashup applications, one of the vital processes that take place is known as service composition. Service composition involves the integration of different web contents to create new services according to user's requests, by refining and combining only the necessary data. This process depends on the support offered by Mashup tool, some Mashup tools are automated and some are semi-automated. The ones that are automated do the process of service composition automatically while the semi-automatic ones automate the process partially. The process of service composition is the one that determines whether a Mashup tool is worth it or not to end-users. According to Tuchinda et al. (2008), Mashups service composition process is much more complex because it needs to deal with retrieval, integration and presentation of data.

One of the key challenges in service composition is the integration of web services and non-web services contents since extra programming is required. Ngu et al. (2010) argue that such a challenge is caused by the components that are developed at different times, by different groups, using different technologies, naming conventions, and structures. Other issues are as follows:

- i. Usability: most Mashups are designed for creativity rather than usability, which impede the fast growth of Mashup users.
- ii. Performance and scalability: Mashing large data using web services from different sources can be very slow. A bandwidth or server bottleneck causing limited speed of the network connections and slow retrieval of the desired data is the result.
- iii. Security: Users have to bear the security risks when uploading their data to web 2.0 sites. Although users may choose not to publish the work to the public, they lose control of the work once it is uploaded (or if directly programmed online) to a server.
- iv. Missing features and instability: Due to the fact that the available tools are relatively new and sometimes still have "beta" status, some essential features may either be not supported or have limited functionality. In addition, bugs can lead to server instabilities and downtimes.
- v. Flexibility: Although these tools are considered to be useful in common data Mashup use cases, they are not as flexible as individually programmed components. For certain tasks, writing custom code may be required.

- vi. **Quality:** Due to the previously mentioned issues, the final output from Mashup tools may not match the quality that “professional” users can achieve with local software development.

This section discussed the challenges affecting a wide adoption of Mashup tools. With the aim of the research being to create a usable Mashup tool that target novice users were there is a less active development of web application; the next section defines usability and outlines the usability requirement.

## **2.10 Usability of Mashup Tools**

This section focuses on the usability amongst the aforementioned challenges. In the previous section (Section 2.9), the advantages and challenges of Mashup tools were discussed. With the usability listed as one of the challenges affecting a wide adoption of Mashup tools, and with the research aim being to create a usable Mashup tool for novice users to utilize the available SOA services. This section discusses the usability and the methods used to evaluate the usability of Mashup tools.

### **2.10.1 Definition of Usability**

According to ISO (1998), usability is defined as “ the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”. The human-centred design is characterised by: the active involvement of users and a clear understanding of user and task requirements (Preece et al., 1994). The web developers need to allocate appropriate function between users and technology, and with user centred design; in order to implement and evaluate different design solutions (Preece et al., 1994).

Designers produce primary objectives referred to as usability goals and user experience goals (Abrams et al., 2004). A usability goal addresses the issue of meeting a specific usability criteria (Abrams et al., 2004). A user experience goal is concerned with the quality of the user’s experience with the system.

### **2.10.2 Usability Goals**

Usability goals need to be specified for optimizing user interactions. Quesenberry (2003) argued that the following are the usability dimensions: effectiveness, efficiency, engaging, error-tolerant and easy to learn.



- i. Effectiveness: Most general goal, concerned with whether the system is doing what it generally says it will do.
- ii. Efficiency: Consider saving user's time in performing task. Systems that require the least number of steps are usually considered as being efficient.
- iii. Error-tolerant: It prevents making serious/ unrecoverable errors, provide means of recovering from errors: undo options, confirmation dialogs.
- iv. Engaging: Is sufficient functionality to accommodate range of user's tasks. It deals with how easy to remember once learned important goal to in the systems.
- v. Easy to learn: Is to identify how much time users are willing to spend to learn the system.

The above-mentioned usability goals served as the guide during the design and implementation of the proposed Mashup tool. The process of evaluating system usability can be categorized into two methods: first by evaluators and lastly by users. This classification was introduced by (Nielsen et al., 1994). This classification of usability evaluation methods is the most appropriate for this research work on usability evaluation of Mashup tools, and each mentioned usability methods is discussed.

### **2.10.3 Evaluator-Based Usability Evaluation Methods**

This category includes usability methods that involve evaluators in the process of identifying usability problems. These methods are aimed at finding usability problems that users might encounter while interacting with an interface and making recommendations to improve the usability of the interface. The following subsections discuss some of the usability testing methods.

#### **2.10.3.1 Heuristic Evaluation**

Heuristic evaluation is a usability that involves having a number of evaluators assessing the user interface and judge whether it conforms to a set of usability principles ('heuristics') (Nielsen et al., 1994).

#### **2.10.3.2 Guideline Reviews**

This is a usability method, which contains comprehensive guidelines and involves checking an interface for conformance with these usability guidelines. This method is similar to the heuristic evaluation method, except for the length and details of the guidelines used by evaluators; heuristic evaluators use a short list (of less than a dozen items) while guideline

reviewers use a longer and more detailed list (with several dozen or more guidelines) (Nielsen et al., 1994).

#### **2.10.4 User-based Usability Evaluation Methods**

The user testing method is considered to be the most important and useful approach since it provides direct information regarding how real users use the interface; it illustrates exactly what problems users' encounter in their interaction (Nielsen et al., 1994). The user testing method is a systematic way of observing actual users trying out a product and collecting information about the specific ways in which the product is easy or difficult for them.

##### **2.10.4.1 Questionnaires and Interviews**

Different types of questionnaire (i.e. closed or open) and interviews (i.e. unstructured, semi-structured or structured) are considered useful and simple techniques that collect data regarding users' satisfaction with, or preferences on, a user (Nielsen et al., 1994; Hom, 1998). Hom (1998) also indicated that surveys cannot be used to observe and record actual users' interactions with an interface but can be used to collect information regarding users' opinions, attitudes and preferences, as well as self-reported data concerning behaviour. Therefore, data about users' actual behaviour should have precedence over users' preferences since users' statements cannot always be taken at face value (Hom, 1998).

##### **2.10.4.2 SUS**

The SUS is a simple, ten-item scale giving a global view of subjective assessments of usability. SUS is a likert scale (Brooke, 2013). It is often assumed that a likert scale is simply one based on forced-choice questions, where a statement is made and the respondent then indicates the degree of agreement or disagreement with the statement on a 5 point scale. According Brooke (2013), SUS yields a single number representing a composite measure of the overall usability of the system being studied. The usual threshold for SUS is 68. This means if the system is below 68 then it is not usable.

##### **2.10.4.3 Focus Groups**

This is an informal method for collecting in-depth information regarding the needs, judgments and feelings of typical users about an interface (Nielsen et al., 1994; Hom, 1998). In a focus group, about 5 to 9 users discuss selected topics, such as the different functions and features of an interface, with the assistance of a moderator, and then identify issues during

their interaction. This method allows diverse and relevant issues to be raised; it brings out users' spontaneous reactions, comments and ideas through their interaction (Hom, 1998).

#### **2.10.4.4 KLM**

The Keystroke-Level Model (KLM) is a simplified version of GOMS. It was proposed by (John & Kieras, 1996) as a method for predicting user performance. Using KLM, execution time is estimated by listing the sequence operators and then summing the times of the individual operators. KLM aggregates all perceptual and cognitive function into a single value for an entire task, using a heuristic (John & Kieras, 1996). The total of the operator times is the estimated time to complete the task.

A KLM model provides the designer with a model of a user's behaviour while performing well-known tasks. These models can be used for a variety of purposes, as follows: If the designer has a list of likely user goals, KLM models can be used to verify that a method exists to achieve each of these goals. KLM models can predict the time it will take for the user to carry out a goal (assuming an expert user with no mistakes). This allows a designer to profile an application and to locate bottlenecks, as well as compare different UI designs to determine which one allows users to execute tasks quicker.

In this section, different methods of evaluating usability were discussed and usability principles were outlined. Furthermore, the definitions of usability according to different researchers were presented. The next section summarizes this chapter.

### **2.11 Conclusion**

This chapter discussed the ways of aggregating contents to facilitate service reuse and sharing. Number of benefits and challenges were discussed and the observed trend on the convergence of portals and Mashup was discussed. The aim of the chapter was to give readers and developers the scope of Mashups (.i.e. to what extent had Mashup research been conducted). The basis of SWS is the descriptions of web resources. It creates such descriptions by annotating resources with metadata, resulting in annotations about that resource. The resultant annotations create a relationship between URIs and build up a network of data. They also simplify the integration of annotated data from multiple sources, which results in smooth service composition when properly designed. Service composition using SWS contains the following steps namely: service description, service discovery, service publication and service invocation. These steps can be summarized into selection and

matching of appropriate services. The next chapter address how the proposed Mashup tool addresses the process of services composition using SWS.

## 3 Chapter 3: Homogenization of Web Contents

### 3.1 Introduction

In the previous chapter, a review of literature showed that there are some challenges facing the development of Mashups. This chapter focuses on the techniques of semantic annotation to create meaningful data thereafter further related work is discussed.

### 3.2 Ontology as a Web 3.0 Service

Ontologies are formal explicit descriptions of concepts in a domain using properties of each concept for describing various features and attributes of the concept. Ontologies together with a set of individual instances of classes constitute a knowledge base. In reality, there is a fine line where the ontology ends and the knowledge base begins (Sabou, 2006). The classes are the core building blocks of ontologies, and they describe concepts in the domain.

Ontologies define a common vocabulary for researchers or developers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. Ontologies are developed to share common understanding of the structure of information among people or software's and to enable reuse of domain knowledge. Figure 3-1 shows the evolution of the web:

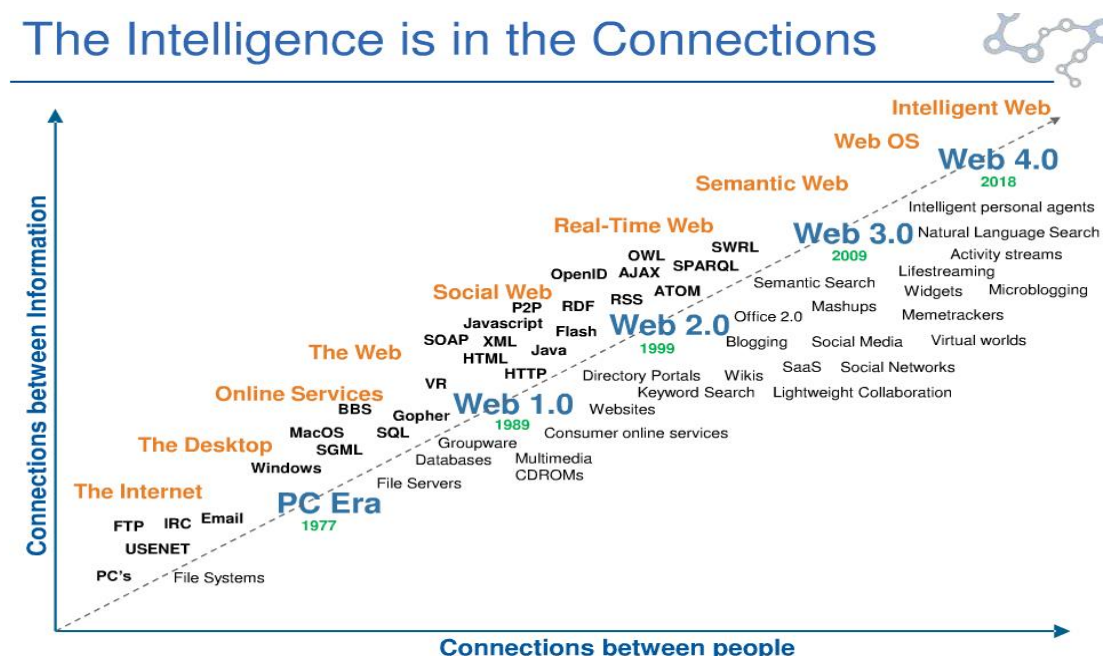


Figure 3-1. Web evolution (Spivack, 2008)

Figure 3-1, shows the evolution of the web from a simple platform that only allowed users to consume web resources, to a platform where users could both provide and communicate information simultaneously. Currently, the web provides users with feature rich web application and platforms together with software programs that can even automate the integration of various web resources. The web resources are moving from meaningless resources to semantically enriched smart agents. This is to allow with other web technologies evolving and being embraced, for instance semantic web and web services.

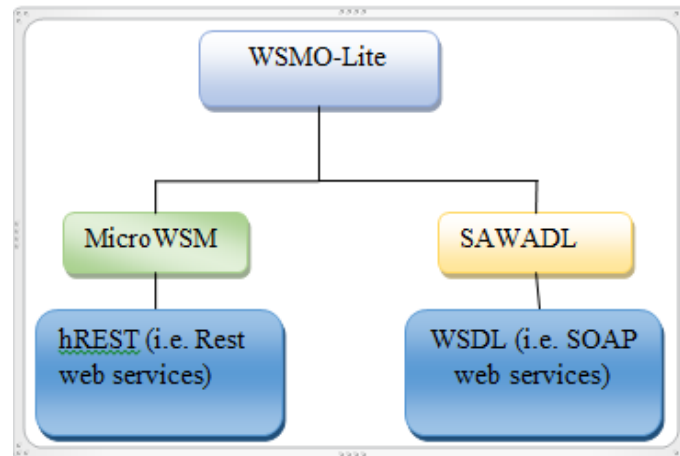
The next section discusses semantic annotations approaches, which are the techniques used to bridge the gap between Mashups and legacy applications.

### **3.3 Semantic Annotations**

Semantically annotating web services results in new concept called SWS. This concept yields web services that are intelligent i.e. they can be automatically or semi-automatically described, published, selected and discovered (Küster et al., 2008). SWS aim at turning the internet from an information repository for human consumption into a world-wide system for distributed web computing by combining semantic web technologies and web services (Zeshan & Mohamad, 2011). There are a number of ways on how services are described, discovered, selected and consumed using SWS.

#### **3.3.1 Service Description**

The problem of ad-hoc semantic annotation of services, have been tackled by the use of WSMO-Lite, minimal extension to SAWSDL. WSMO-Lite provides means to create lightweight semantic service descriptions in Resource Description Framework Schema (RDFS) by annotating various WSDL elements in accordance with SAWSDL annotation mechanism (Loutas et al., 2011). As some services are not described in WSDL the term MicroWSMO was coined to describe such services. MicroWSMO is a micro-format-based language based on the same kinds of annotations as WSMO-Lite, adapted to support the annotation of HTML-based descriptions of web APIs (Ambite et al., 2009). Ultimately the minimal service model evolved, which is defined as a simple RDFS model that provides a model that is able to capture the semantics for both web services and web APIs, consequently allowing both kinds of services to be treated homogeneously when at selection time (Maleshkova et al., 2009; Loutas et al., 2011). Figure 3-2 illustrates how the descriptions of services are made.



**Figure 3-2.Annotation methods adapted: (Kopeck & Vitvar, 2008)**

As illustrated Figure 3-2, the following subsections explain the annotation methods that are used to enhance service description (Kopeck & Vitvar, 2008;Maué et al., 2009).

### **3.3.1.1 SAWSDL**

To date, SAWSDL is the only SWS specification that is a World Wide Web Consortium (W3C) recommendation. It defines a set of extensions to WSDL, as well as rules for linking WSDL elements to semantic information. In particular, SAWSDL supports three kinds of annotations over WSDL and XML Schema, namely model Reference, liftingSchemaMapping and loweringSchemaMapping. These three annotation types enable links to be made from parts of a WSDL document to the associated semantic elements, or to the specifications of data transformations from a syntactic representation to its semantic counterpart and vice versa. In this way, it enables the incremental addition of semantics on top of existing WSDL descriptions, providing a basis for extending results from a well-established approach. However, SAWSDL only provides simple means for connecting service elements to semantic entities and does not define any concrete service semantics such as types, formats or model for the semantic descriptions.

### **3.3.1.2 WSMO-Lite**

WSMO-Lite builds upon SAWSDL, overcoming some of SAWSDL's limitations while remaining lightweight. WSMO-Lite makes explicit the intended meaning for model reference annotations without modifying SAWSDL but rather informing users on how they should structure the models their annotations point-to. The WSMO-Lite ontology can be used to capture four aspects of service semantics:

- i. The *information model* defines the data model. In particular, it describes the model for input and output messages and is represented using domain ontology, along with associated data lifting and lowering transformations.
- ii. The *functional semantics* define what the service does by using functionality classification or preconditions and effects. It describes what the service can offer to its clients when it is invoked by assigning it to a particular class of service functionality, defined in classification ontology.
- iii. The *behavioural semantics* define the sequencing of operation invocations when invoking the service. The behaviour of a service can be described through choreography or a workflow definition. However, behavioural semantics are not represented explicitly in WSMO-Lite.
- iv. The *non-functional semantics* define any service-specific implementation or operational details such as service policies, implementation information or running environment requirements. Non-functional properties can include the price of a service or the Quality of Service (QoS) aspects such as performance and reliability.

### 3.3.1.3 MicroWSMO

MicroWSMO uses micro-format for adding semantic information on top of existing HTML web API documentation. It uses HTML for Restful services (hRESTS) for marking service properties and making the descriptions machine-processable. HRESTS provides a number of HTML classes that enable the marking of service operations, inputs and outputs, HTTP methods and labels, by inserting attributes within the HTML. With the structure of hREST in place, HTML service descriptions can be annotated further by including pointers to the semantics of the service, operations and data manipulated. MicroWSMO adopts the SAWSDL annotation mechanisms and it uses three main types of link relations:

- i. Model, which can be used on any service property to point to appropriate semantic concepts identified by URIs;
- ii. Lifting associate messages with appropriate transformations (also identified by URIs) between the underlying technical format such as XML and a semantic knowledge representation format such as RDF at the lower level.
- iii. Lowering associate messages with appropriate transformations (also identified by URIs) between the underlying technical format such as XML and a semantic knowledge representation format such as RDF at the upper level.



Therefore, MicroWSMO, based on hRESTS, enables the semantic annotation of web APIs in the same way in which SAWSDL, based on WSDL, supports the annotation of web services.

MicroWSMO also adopts the WSMO-Lite ontology as the reference ontology for annotating web APIs semantically. By doing so, both WSDL services and Restful services, annotated with WSMO-Lite and MicroWSMO respectively, can be treated homogeneously.

### **3.3.2 Service Discovery**

Service discovery has been the subject of much research and development. The most renowned work is perhaps Universal Description Discovery and Integration (UDDI), while nowadays Seekda4 provides the largest public index with about 29,000 WSDL Web services (Meditskos & Bassiliades, 2011). Research on SWS has generated a number of ontologies (e.g., hREST, SAWSDL), semantic discovery engines (e.g., Seekda4), and further supporting infrastructure over the years. Despite these advances, however, the majority of these initiatives are predicated upon the use of WSDL web services, which have turned out not to be prevalent on the web where web APIs are increasingly favoured (Küster et al., 2008). As a consequence, there has not been much progress on supporting the automated discovery of web APIs. The main means used nowadays by developers for locating web APIs are the use of traditional search engines like Google or searching through dedicated registries (Daniel et al., 2012b). The most popular directory of web APIs is Programmableweb provides simple search mechanisms based on keywords, tags, or a simple prefixed categorisation. Therefore, despite the increasing relevance of web APIs, in nowadays there is no system that support discovery of APIs. The main obstacles in this regard concern first of all, the automated location of web APIs, and subsequently, the gathering, interpretation and extraction of relevant information concerning these APIs (Ly et al., 2012).

### **3.3.3 Services Publication**

The objective of syntactic and semantic descriptions of web services is to provide information about services in a way that can automatically be processed by machines (Sbodio et al., 2010). However, at present, these descriptions can only be retrieved through the web of documents, which is essentially designed for human beings, or through specific interfaces to silos of services such as UDDI that have failed to see significant uptake. This is particularly true for syntactic descriptions of services (although there is an RDF mapping for WSDL), but also for semantic descriptions, which have remained somewhat disconnected from current practices in the web of data (Brambilla et al., 2006). A fundamental step for bringing services

closer to the web, thus better enabling their discovery and supporting their use, is publishing them based on current best practices on the web such as publishing of service annotations as linked data.

### **3.3.4 Service Selection and Matching**

The objective of creating service descriptions is so that services can be found and discovered. This section focuses on the problem of selecting and matching services. The basic approach is for the client to create a formal description of their requirement, and for the matchmaker to compare this against the stored service descriptions to find the most appropriate matches. Templates can be used operationally in workflows, acting as placeholders for late binding services. It enables automated suggestion of next steps in a pipeline based on the following: type and enabling partial verification of correctness. In discovery engines, it assists to find suitable services. In ranking and selection engines, it allows to order them by suitability; and in execution engines, to supply values to service invocations. Services can be found by creating a SPARQL RDF and Query Language (SPARQL) query, and in particular, queries can be derived from a service template. When there is no explicit mediation in the model, finding exact matches for input and output types is important to connect to the model references from the SAWSDL and MicroWSMO (Pedrinaci et al., 2011).

This section discussed the core concepts of SWS known as WSMO. It consists of ontologies that help to automate services description and service discovery. The next section focuses on further related work.

## **3.4 Related Work**

This section is divided into two sections namely: making content Mashup ready and end-users programmed Mashups (i.e., usable Mashup tools). The research first discusses the usability of Mashup first followed by homogenization of web APIs.

### **3.4.1 Usability in Mashup Tools**

This subsection discusses the studies that focused on encompassing usability principles to Mashup tools. The purpose of the conceptual evaluation model was to serve as a meaningful and useful framework or model for identifying the specific usability impact factors of Mashup makers, as indicated in (Sarraj, 2012). The conceptual evaluation model consisted of three main parts, which represent the three main usability aspects of Mashup makers. These three parts are: visual support, user interaction support and functional support. Firstly, the

visual support part is concerned with the usability of the user interface of the Mashup maker: layout of components, size, colour, and metaphors. Secondly, the user interaction support addresses the usability of the Mashup Maker from a user interaction perspective. It groups usability aspects such as cognitive and intuitive interaction support. Lastly, the functional support part considers how the Mashup maker supports the users' functional requirements. This model was used to serve as the guide for development and evaluation of Mashup tools, but no Mashup tool was implemented to validate the whereabouts of the provided guidelines.

In optimizing Mashup for flexible interaction, Dang et al. (2007) proposed a personalisation technique to enhance the functionality of Mashups interaction with users by presenting a robust and flexible service-oriented architecture of personalised Mashups. While Krummenacher et al. (2009) proposed a project funded by European Union to create applications that can be used by all users and it is known as SOA4ALL. With the aim of providing "a comprehensive framework and infrastructure that integrates complementary and evolutionary technical advances (i.e. SOA, context management, web principles, web 2.0 and semantic web) into a coherent and domain-independent services delivery platform". This work is one of the main projects that have been developed in collaborating end users with Mashups applications. They proposed Semantic Web sERVICE Editing Tool (SWEET) and Sweet is nOt a Wsdl EditoR (SOWER) for annotating web services. SWEET is used to annotate restful web services while SOWER is user to annotate SOAP based web services. To create interactive users interfaces, Daniel et al. (2012a) proposed a domain-specific approach to Mashups that speaks the language of the user. This work focused awareness on the terminology; concepts, rules, and the domain user are comfortable-with. This work enabled end users without programming skills to compose own applications.

Furthermore to give end-users the freedom to customize processes and process activities according to their specific needs, Fisichella and Matera (2011) proposed a reference model. The proposed model allowed for a flexible process definition and process execution to support the dynamic user-based composition process. Yet still there are Mashup tools that use data feeds to create services and uses widgets for user interaction, but still these Mashup tools are not usable to non-technical users. Mashup tools lack the collaborative reasoning emphasized by SOA4All project, though they support both web based data sources and corporate data. Likewise Yahoo Pipes uses widgets and feeds for Mashup creation but still not usable by non-technical users. Yahoo! Pipes are visual tool for data extraction and processing of web sources (Jones & Churchill, 2009).

### **3.4.2 Web APIs Homogenization**

In order to facilitate end-user Mashup based application there is a need to transform heterogeneous structures to a homogenous structure. This subsection discusses some of the studies that were conducted in order to handle heterogeneity of web resources.

In the making of web contents web enabled, Soriano et al. (2008) proposed a flexible presentation layer for human users, using SOA based web-services to enhance interactions and usability of services by developing an open Mashup platform known as EzWeb. This work did not provide a mechanism for making web contents “Mashup-ready”, because in order to simplify the process of developing end user based Mashups, there is need to solve the issues around heterogeneous resource. In solving the issue around heterogeneous, Momeni (2011) proposed a methodology by analysing the components required for combining and integrating information into machine process-able dataset from different web data sources to a homogeneous structure, using the e-commerce ontology. This work partially solved the process of making resource web-enabled, since it provided different adaptors for each service. Yet again, Upadhyaya (2012) proposed a framework to assists end users in the process of discovering and composing services and to simplify the integration of heterogeneous web resources. This work was achieved using the following two goals: first to identify Mashup-ready web resources from different web applications and SOAP-based Web services, and lastly to providing approaches to discover and compose web resources that are capable of accomplishing user’s activities. This work did not solve the issue of making web resources Mashup-ready in a global scale of SOA. Furthermore, in increasing the number of web service based resources used by Mashups, Malki & Benslimane (2012) proposed SAWADL which only allowed the semantic annotation of the REST web Service. This work was used to combine web APIs and REST Web services to develop more agile and flexible Mashup application.

### **3.5 Homogenization Process**

This section highlights the key technologies and requirements for semantic annotation processes. The following listing outlines the requirements for semantic annotation tools (Uren et al., 2006; Maynard, 2005):

- i. User Centerd System Design: the semantic annotation system should include the following aspects in order to allow a diverse range of use and adoption:
  - a. They should be seamlessly integrated in order to allow users to read and write the annotated API documentations.

- b. Ease of installation.
  - c. As the web browser is the mostly used tool to access web resources, semantic annotation tools must be implemented as browser plugin, for ease of use. They should be compatible with more collaborative environments.
- ii. Interoperability: because most of semantic annotation tool support HTML and XML, allowing them to handle different types of media approaches, can result in a more distributed environment. For instance the use of a standard web ontology language for standard consistency can further improve collaborative environments. The format, resulting from annotated documentation of REST services must be usable to other system and applications.
- iii. Annotated API storages: semantically annotated API documentation must employ RDF servers, especially the use of triplestores. Where annotations of APIs are linked with their original documentation.
- iv. Annotation process support: the support offered by the semantic annotation tools should be far more automatic, since the creation, maintenance and navigation of ontology greatly favours specialists ( i.e., knowledge base engineers and experts). If semantic annotation tools support the process of automated annotation and discovery of APIs, then its high likely that their complexity will be hidden to users and that will increase its usability.

The above listing discussed the requirements of semantic annotation tools (i.e., homogenization of web APIs). As the research hypothesis outlined that a semantically enable Mashup tool to discover, annotate and compose web APIs can improve usability of SOA services for novice users; this research use hREST and defined ontology to semantically annotated web APIs, with SerPro API editor being an annotating tool that employs the mentioned semantic annotation approaches. Furthermore the semantic annotation requirements reviewed above are addressed in the proposed Mashup tool. For the discovery SPARQL is used as a query language, to query services from Sesame server. To compose web APIs a survey on the existing Mashup tool was conducted, based on its findings the design architecture was proposed and implemented.

### 3.6 Conclusion

No study has been conducted to create a Mashup tool that could make web contents to be easily combined by providing a Mashup-ready transforming layer. Hence, this research aims

to develop a tool and transform web resources into Mashup-ready contents to improve service composition. To ease service composition for novice end users requires an agile presentation layer. Both sections are applicable to our current research, which aims to distribute web-based applications to rural based areas. Such users need Mashup ready content and user-friendly interfaces.

Having discussed the Mashup landscape, the next chapter presents research methods and the research philosophy used to achieve the highlighted the research challenges.

## **Part II: Methodology and System Modelling**

## **4 Chapter 4: Methodology**

### **4.1 Introduction**

This chapter presents the research philosophy, approach, design and methods used to address the research problem as outlined in (Chapter 1). This research project sought to analyse the usability of Mashup tools, through mainly qualitative and quantitative methods using deductive logic. The research aim is to design and implement a usable Mashup tool that targets novice users to create web applications with ease. This is in line with the overall research problem as identified in (Chapter 1).

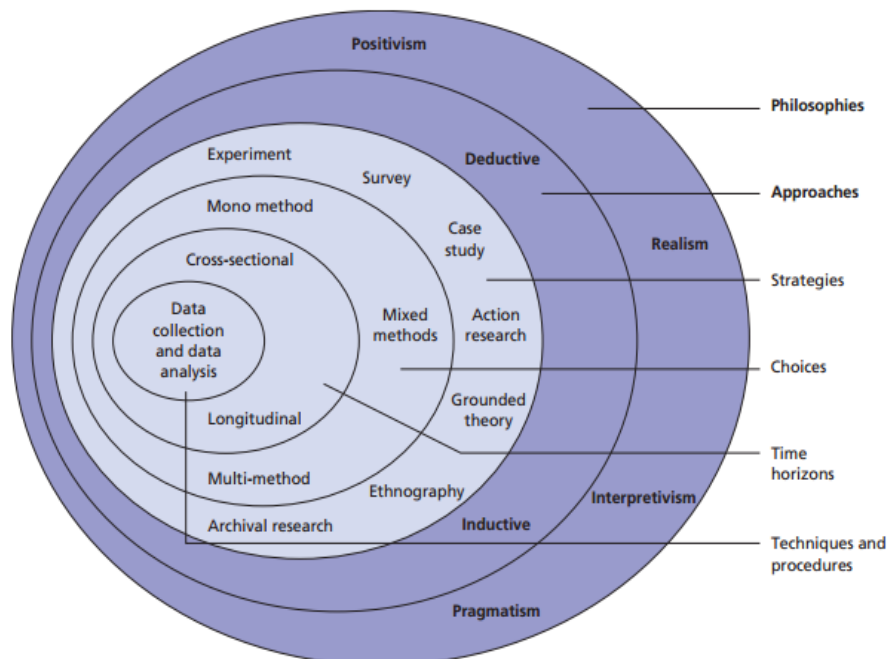
### **4.2 Research Methodology**

Research methodology is a way to systematically solve the research problem (Rajasekar et al., 2006). It is referred to as a science of studying how research is done scientifically. Research methodology consists of various steps that are generally adopted by a researcher in studying the research problem along with the logic behind them and it helps the researcher to know the suitable research methods or techniques for a particular problem (Holden & Lynch, n.d.). Moreover, it helps researchers to know which of the selected methods/ techniques are relevant and what they mean and indicate. According to Williams (2011), research is described as the “traditional assumption that in science the researcher must maintain complete independence if there is to be any validity in the results produced”. This means that in order to attain good results the researcher must be involved but must not have influence on the research outcome in the execution of the fieldwork. The suggestions mentioned in this section were clarified by choosing a research philosophy

The research paradigm/ philosophy offers a framework, consisting of theories, methods and ways of defining data, which explains the relationship between data and theory as suggested by (Collis & Hussey, 2003). There are two main research philosophies or paradigms that guide the design and methods of research. These are positivism and interpretivism (Saunders et al., 2011). These approaches have different propositions regarding common assumptions concerning obtaining knowledge and the process of research. The most common assumptions are termed epistemology, ontology and the logic of the research. Epistemology concerns how a researcher will obtain knowledge during the research; ontology concerns how each paradigm views reality (knowledge), or what is considered reality from the viewpoint of the researcher; and the logic of a research describes the nature of the relationship between research and theory, which could be, according to Saunders et al., (2011) either deductive or



inductive. Saunders et al. (2011) describe scientific research as an onion with multi-layers as shown in Figure 4-1:



**Figure 4-1. Research union (Saunders et al., 2011)**

Following the research union and referring to the aims and objectives of this research as mentioned in (Chapter 1), this research has adopted an interpretivist approach. Interpretivism is an appropriate approach with regard to the research problem. The research problem is related to a positivist approach if it evolves from the literature where variables and theories may exist that need to be tested and verified, while a research problem is related to an interpretivist approach when little information exists on the topic and more exploration is needed since the variables are largely unknown. Therefore, it was clear that the interpretivism approach is an appropriate one to be adopted in this research, as it is not guided by theory that must be tested objectively. Instead, it aimed at finding an understanding regarding which usability methods are the best in evaluating usability issues of the currently existing Mashup tools.

This research combined several research methods for the purpose of facilitating triangulation. Triangulation in the context of this research means the mixing of data or methods used so that diverse viewpoints or standpoints cast light up on a topic (Olsen, 2004). The research employed both qualitative and quantitative research methods. This was performed by the semi-structured surveys performed in the literature study and the empirical investigation of the usability of Mashup tools and it allowed the integration of different research methods to

study Mashups in order to avoid repeating the same weakness possessed by the already existing Mashup tools (Voss et al., 2002). More details about the research design and method are provided in (Section 4.3).

The aim of this section was to highlight the research philosophy related to this research.

### **4.3 Research Approach and Design**

The research approach is a method of producing new knowledge or deepening an understanding of a topic or issue. Research strategies include logical and qualitative research approaches. Research approach and design also helps to establish or confirm facts (Onwuegbuzie & Leech, 2010). The review of literature has produced reoccurring themes emphasizing the importance of Mashup tools in enhancing the utilization of SOA services and service composition is a critical component in preparing Mashups for novice users.

#### **4.3.1 Literature Review**

The literature review presented in (Chapter 2) was used in this research to address the research problem as discussed in the following listing:

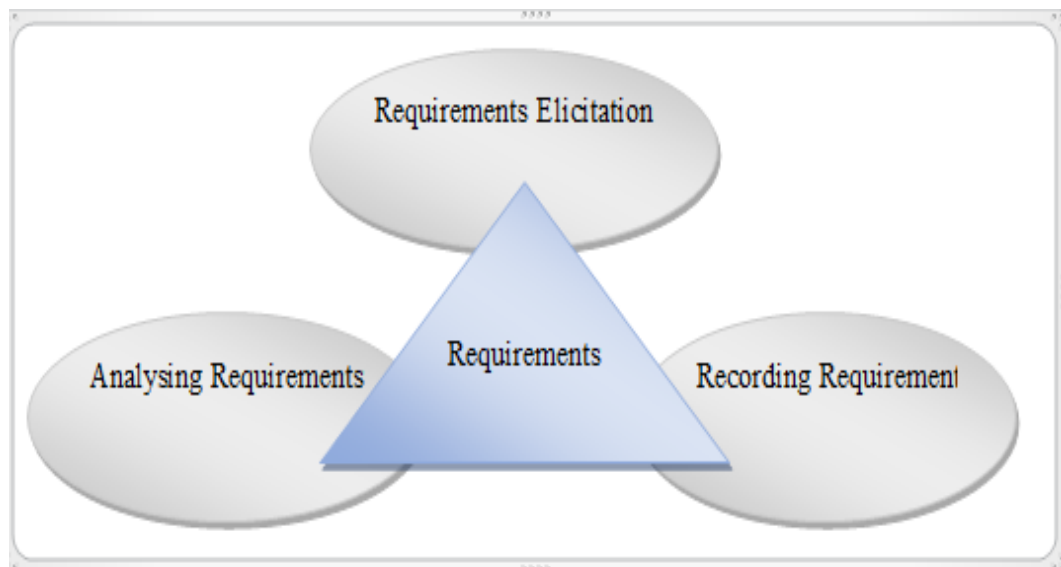
- i. To indicate the key shortfalls of the current Mashup tools (i.e., assess the strengths and weaknesses of previous work including omissions); these were identified in (Chapters 2 and Chapter 3).
- ii. To demonstrate the state of art on service composition platforms, in particular the Mashup usability is up-to-date: as demonstrated in (Chapters 2, Chapter 3 and Chapter 4).
- iii. To show how this research relates to previous published research: as demonstrated in (Chapter 2).
- iv. To enhance and improve the current state of consumer Mashup tools.

The above listing outlined how the review of literature contributed to this research, the following section discusses the system requirements.

#### **4.3.2 System Requirements**

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users (Blanchard et al., 1990). Requirements form the basis for all future work on the project, from design and development to testing and documentation, in order to

provide the best chance of creating a system that fully satisfies the needs of the customers. Conceptually, requirements analysis includes three types of activity:



**Figure 4-2. Requirements activity**

- i. Eliciting requirements: the task of communicating with customers and users to determine what their requirements are. This is sometimes also called requirements gathering.
- ii. Analysing requirements: determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory, and then resolving these issues.
- iii. Recording requirements: Requirements might be documented in various forms, such as natural-language documents, use cases, user stories, or process specifications.

New systems change the environment and relationships between people, so it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implications of the new systems (Blanchard et al., 1990). Analysts employ several qualitative methods to elicit the requirements from their target group. The following subsections explain the methods used to derive the research; functional and non-functional requirements, to inform the design of the proposed Mashup tool.

#### **4.3.2.1 Requirements Gathering**

This section details how other research methods were used to gather usability requirements. However, before getting to details of what methods were used to gather data, the research discusses the ethical issues considered during data collections.

## **Ethical Considerations**

While gathering data using interview, focus groups and SUS to identify short-falls of the current existing consumer Mashup tools, with a special focus on usability; to assist in planning and designing usable mashup tool, as well as to evaluate the effectiveness of the prospective mashup tool. The following ethical issues were taken into consideration:

- Users participation is completely voluntary.
- Respondents are not harmed and embarrass; hence they must feel free about answering questions.
- The collected data is confidential and respondents cannot be identified on the basis of a response (i.e. anonymity).

Participants were allowed to leave at any particular stage if they felt uncomfortable, but their presence was of great importance towards the study. The participants culture and believes were not violated, hence participants were free to go if they felt that the tasks being performed conflicted with their culture. A ethical clearance certificate is attached in (Appendix A). For each survey conducted, a consent form was attached (i.e., users had to feel in the consent forms before completing the surveys and participating in an interview). The next subsection discusses the research subjects (respondents).

## **Research Sample**

In this research two samples were used: one sample was drawn from Faculty of Social Science and the other from the Department of Computer Science. The users from the Faculty of Social Science were regarded as novice users and 15 students were used as the subjects to the study. These users came from different departments: 6 Social Work and 8 from Human and Resources Management and 1 from Political Science, and all of them were post graduates. Their age ranged from 23 to 28. These users were chosen since all of them were computer literate but did not have programming skills. This group of users was chosen when conduction interviews and SUS, to get their behaviour and preference towards Mashup tools.

This research also drawn a sample from Computer Science department and 3 of them were postgraduates and the rest were undergraduate. These users were regarded as technical users since all of them had programming skills. Their age ranged from 19 to 27. The aim using this group of users was to get feedback and opinion of technical users towards Mashup tools, and to identify the usability short-falls and suggest some of the improvement or features that could be increase the usability of Mashup tools. Mostly this group was chosen when

conducting focus group sessions, to get and discuss various ideas that could improve usability of Mashup tools.

The above research samples were used to collect data using the methods discussed in the following subsection.

### **Data Collection Methods**

There are three methods used namely: observation, survey and focus group. This subsection discusses each method and how it was used to collect data in this research

Firstly, participant observation is appropriate for collecting data on naturally occurring behaviours in their usual contexts (Blanchard et al., 1990). Direct observation emphasizes observing and recording actual behaviour, rather than reported or recalled behaviour (Patton & Cochran, 2002). The researcher records as much behaviour as possible, including actions, conversations, and descriptions of the locale and persons.

Secondly surveys, according to Williams (2011) surveys are defined as a viable approach to a problem only when there is data to support it. Patton and Cochran (2002) further define survey research as “the process of collecting representative sample data from a larger population and using the sample to infer attributes of the population”. The main purpose of this is to estimate, with significant precision, the percentage of population that has a specific attribute by collecting data from a small portion of the total population. The researcher wanted to find out from members of the population their view on usability of Mashup tools. SUS was used as one of the survey techniques together users’ perception towards the currently existing Mashup tools.

During SUS experiment, the observations were made from the way that most of the respondents behaved towards certain tools. The following listing describes them:

- i. Users did not complete the given tasks; they took a lot of time to complete sub-tasks. They were not satisfied, users were confused (i.e. did not know where to start).
- ii. For some users using the Mashup tool was easy whilst on the other hand it was difficult for remaining group. In order to create or use Mashups most users needed assistance. Six out of twelve of the observed users were not aware of what are Mashups and what is their purpose.

- iii. According to users behaviour the Mashup tools did not map to the real worldwide (i.e. there was too much inconsistency). Majority of users (i.e., 8 out of 12) required more time to get going with the Mashup tools; they did not enjoy using Mashups.
- iv. Although most users (i.e., 10 out of 12) showed that it can be quick and easy to learn Mashup, they were so frustrated since they said it was unnecessary hard to use them.

As it was indicated in the literature review chapter (i.e., Chapter 2), that there are 3 type of support offered by Mashup tools namely: visual, browser extension and web portals support. The existing Mashup tools used in this research were categorised, and the following listing summarizes SUS scores obtained from the pre-evaluation of Mashup tools:

- i. *Widgets-based (visual support)*: Yahoo! Pipes, Microsoft Popfly and MobiMash.
- ii. *Browser extension*: Intel MashMaker and Marmite.
- iii. *Form-based (web portal)*: Dapper.

Type of Mashup Tools	SUS scores	Averages
<b>Widget Based</b>		
Yahoo Pipes	40.21	44.71
Microsoft Popfly	48.33	
MobiMash	45.58	
<b>Browser extension</b>		
Marmite	57.29	55.12
IntelMashMaker	52.92	
<b>Form-based</b>		
Dapper	42.92	42.92

**Figure 4-3.Comparison Of Mashup Tools**

In this research, SUS was used to assist in as far as defining the usability of the existing tools, as well as to evaluate the effectiveness of an implemented tool as compared to its counterparts. The details of the SUS score of the above mentioned Mashup tools are presented in (Appendix B).

Thirdly, Focus groups which are regarded as being effective in eliciting data on the cultural norms of a group and in generating broad overviews of issues of concern to the cultural groups or subgroups represented (Rajasekar et al., 2006). Focus group was used to identify functionality shortfalls of the current existing Mashup tools. As well as recommendation on how Mashup tools can be improved in future. The focus group evaluation techniques are defined as the method used to gather users behaviours, attitudes and reaction towards the system. It was conducted using 6 users, and all the respondents were the same as the one in SUS. In the focus group section, the following questions were discussed:

- i. What is the purpose of a Mashup tools?
- ii. What can be done to improve Mashup tools for end-users?
- iii. How can user freedom be achieved whilst emphasizing on functionality?
- iv. What technologies to use in order to create usable and flexible Mashup tool?

The findings of the focus group signifies that if the proposed Mashup tool can adhere to the following suggested solutions, then it can be usable.

- i. Workflow the way services are executed and composed.
- ii. User feedbacks to inform users about the status of the currently executing tasks.
- iii. User freedom or transparency allow users to know and see what they are doing (Mashing).
- iv. Easy and straight forward documentation tutorial should be easy to learn ( such that it will only take few seconds to get on track).

This subsection discussed and analysed the results obtained from the above usability methods, the findings obtained in pre-evaluation conducted, served as the guide to the design of the proposed tool. Using the above-mentioned methods the following were the two types of requirements needed to be considered when developing the proposed Mashup tool: functional and non-functional requirements.

#### **4.3.2.2 Requirements Analysis**

The previous subsection presented how the requirements were gathered. This subsection analyses the gathered requirements to produce two basic requirement categories on system development: functional and non-functional requirements.

##### **Functional Requirements**

These types of requirement focus on what is offered by the system. The Mashup tool developed offer the following functionalities:

- i. The Mashup editor should interact with users through GUI components for instance widgets and wirings for designing and connecting two or more widgets.
- ii. The Mashup applications created must be published on a repository.
- iii. The Mashup tool must be able to discover web APIs.
- iv. The Mashup tool must be able to semantically annotate web APIs.
- v. Using the Mashup tool users must be able to recommend their services.

## Non-functional Requirements

This section addresses the usability metrics that guided the design and implementation of SerPro. These usability research variables served as the guidelines to create a usable Mashup tool. Based on the observed usability requirements and functional requirements addressed in the previous listing, following are the research usability metrics:

- i. *Familiarity* the Mashup tools should be least surprising and it must use drag and drop functionality for the user interface. The multithreading of request must employed using technology such as JQuery for queuing requests, since user are used to application that multitask. The system should handle number of requests concurrently, without interruptions.
- ii. *Workflow-visibility*: the visibility of task sequence being executed must be shown. The graph showing the mostly used Mashups must provided on the welcome page, which indicate the measure on which users are satisfied.
- iii. *Customizable-design*: users should be able to add they own components (i.e., APIs of their choice), and ambiguous components must be handled. If the user do not find their service they must be able to add their services. The system must be easily integrated (.eg., Adding new services from other Mashup tools).
- iv. *Error-handling* as much as possible errors must be reduced, and in case an error occurs, it must be handled peacefully. The errors must be visible to users although they are handled peacefully
- v. *System-documentation* tutorials should be quick and easy and also API documentation must be clear.

Usability of the system is one of the utmost non-functional requirements. The system should offer services that are useful, enjoyable and supportive to novice users. They should enjoy using the Mashup applications. The services offered must correspond to their location.

### 4.3.3 Design

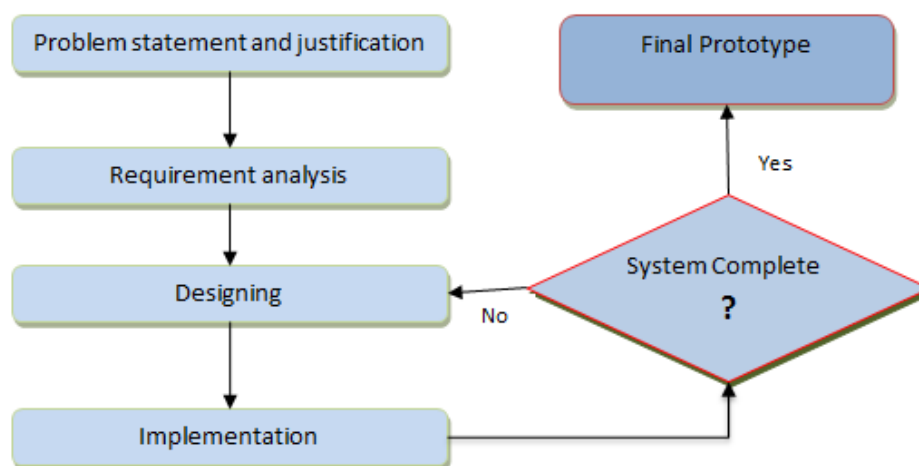
A design to explore the design space was conducted together with a series of small studies to inform the design of the proposed tool. It consisted of tools like Yahoo! Pipes, Dapper, Marmite, IntelMashMaker, MobiMash and Microsoft Popfly, for some of the commercial tools, a blank paper study, and a series of low-fidelity paper prototypes was conducted. *SerPro* (the proposed tool) was inspired by Yahoo! Pipes and Microsoft Popfly, but is focused more on the usability aspects together with the extraction and processing of web



contents. The purpose for conducting the basic usability study with Yahoo! Pipes and other Mashup tools was to understand potential usability problems that should be addressed in the design of SerPro. In Yahoo! Pipes, end-users create a workflow by chaining together a series of operations. End-users can choose operations from desktop applications they are familiar with, such as getting email addresses from the address book application or retrieving a web page with the web browser. The proposed Mashup tool adopted the functionalities like these; and the research design is presented in (Chapter 5).

#### 4.3.4 Research Prototyping

The software development life cycle employed in the study was mixed User Centred System Design (UCSD), and it is depicted in Figure 4-4 (Abrams et al., 2004).



**Figure 4-4. Research model**

The type of model used is known as UCSD. As web 2.0 allows users to become “prosumers” UCSD also have the same vision which is to develop applications that are useful and usable to users (Preece et al., 1994; Gulliksen et al., 2003). As Consumer Mashups allow various range of users to create simple web application with ease so as UCSD. Mashups are said to have adopted the style used by UCSD model. Following are the key feature that UCSD leveraged the research with (Abrams et al., 2004):

- i. It analyses based on users and their world.
- ii. It allows Mashups tool designers and developers to prototype (Väänänen-Vainio-Mattila & Wäljas, 2011).
- iii. The prototype are evaluated and if it does not meet the requirements it iterate to the design stage, until the desired output. The evaluation also allowed researchers to do requirement elicitation using Nielsen’s heuristics (Nielsen et al., 1994). And this

allowed them to better model the users according to what they can do and what they expect of the developed Mashup tool.

If the system can respond in complex ways, it is difficult to appreciate this from static figures in a specification, so the specification phase of projects often uses rapid prototyping tools to construct a functional user interface (Väänänen-Vainio-Mattila & Wäljas, 2011). The developed prototypes can be demonstrated to clients and used as a basis for discussion, as applied in this research where different prototypes were developed to improve the usability of the proposed Mashup tool. UCSD development model prototypes can be refined iteratively until the full system functionality is achieved. Moreover, incremental prototyping requires that the rapid prototyping tool also meets the engineering requirements of the final system (Houde & Hill, 1997). If such a tool is not available, an alternative is deep prototyping, in which one aspect of the system functionality is fully implemented before developing the rest of the interface.

These common approaches to prototyping are quite different to the prototyping techniques that have been found to be successful in developing novel user interfaces. Many product designers believe that creativity in the product design process is directly related to the number of prototypes produced. The objective of building multiple prototypes is to investigate design alternatives through evaluation with actual users to study the mental model that the user develops when interpreting the prototype (Toleman, 1996). The research prototyping is presented in (Chapter 6).

#### **4.3.5 System Evaluation**

System testing was conducted to verify the functional and non-functional requirements of the proposed Mashup tool. As the functional requirements had been outlined, the functional testing and integrating testing is addressed in (Chapter 6). These types of test validate how the system requirements were met. If the proposed Mashup tool achieved the functionalities outlined in (Section 4.3.2.2), then it would mean that the functional requirements were accomplished. Based on the Mashup editor that has already been developed, integration test validates on whether or not if it was possible to integrate Mashup tool with the transformation layer in order to attain the above mentioned functional units.

Furthermore the usability test and acceptance test was conducted to validate the system non-functional requirements. In the usability test, SUS score was used to measure the usability of the system, and for evaluating the acceptability of the system, the focus group and interviews

were used. In the system evaluation (i.e., Chapter 7), the obtained results were analysed, interpreted and discussed.

#### **4.3.6 Research Conclusion**

The research was concluded based on whether the objectives were achieved or not. Furthermore the research limitation and challenges were discussed together with the contribution and achievement made. Finally, future work was suggested to researchers interested in the field of Mashup tools, and is presented in (Chapter 8).

### **4.4 Conclusion**

This chapter summarized the techniques used to carry out a research. It also stated that understanding the philosophy issues in a research study is very useful since, it can help to define the research design in terms of considering what type of evidence is required, how it will be gathered and interpreted, and how this will provide answers to the research questions. Moreover it helped the researcher to determine, and even to develop designs that may be not related to their experience. It also clarified that in order to develop a good Mashup tool and a good Mashup editor, an appropriate model and a good design is required in order to achieve a good productivity. Furthermore, the chapter clarified some of the requirements needed for the system to be functional. The primary goal of methodology was to create a detailed functional specification defining the full set of system capabilities to be implemented, along with accompanying data and process models illustrating the information to be managed and the processes to be supported by the new system (Botta & Bahill, 2008).

This chapter identified the requirements that assisted the designing, implementation and testing of the system. The different prototypes were used to explore different possible solutions, and evaluating the usability of alternatives techniques. The next chapter presents the design of the proposed Mashup tool.

## 5 Chapter 5: Mashup Design

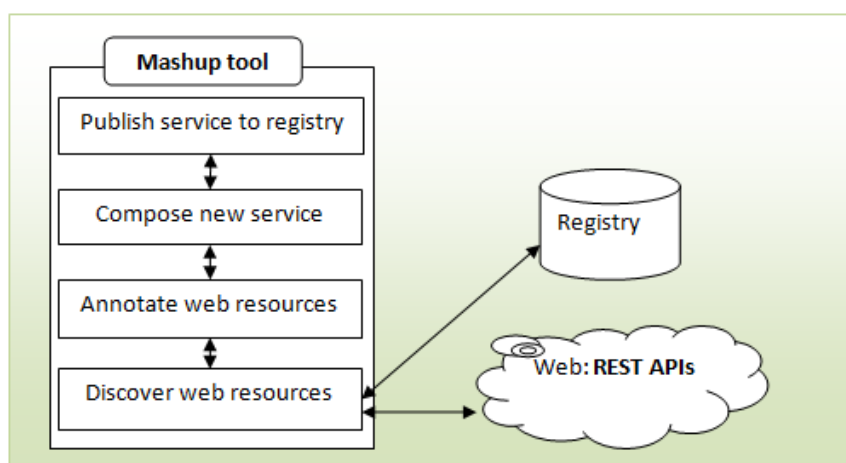
### 5.1 Introduction

This chapter presents the system architecture and system design, and its purpose is to create a technical solution that satisfies the system requirements as an input to the system construction. Furthermore the functional specifications produced during system requirements analysis and findings from the pre-evaluation study are translated into an architecture (Botta & Bahill, 2008). The architecture consists of system components that are designed and prototyped to enable development and testing of the system using languages such as UML (Fowler, 2001).

### 5.2 Design Process

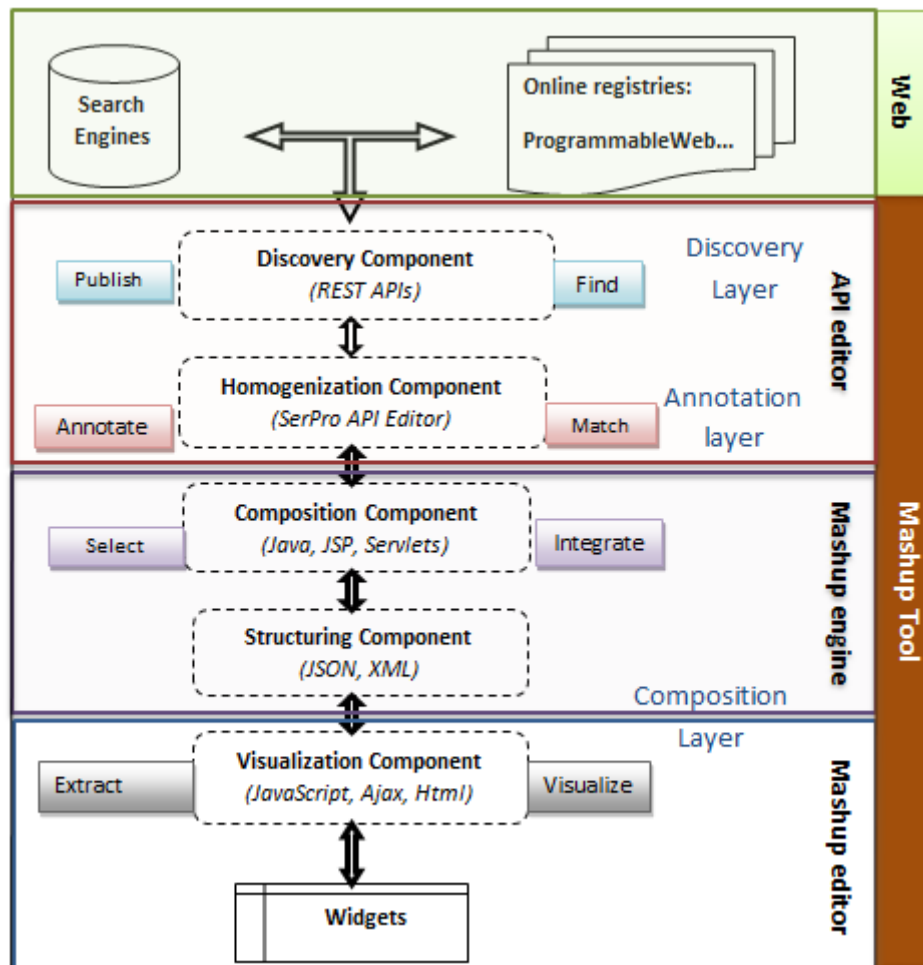
System modelling deals with conversions of requirements to system functionalities. This section deals with how the novice users interact with the system. One of the most interesting, and most difficult, of the tasks that are undertaken in by computer scientists is the design of an entire system (Botta & Bahill, 2008). A system is a set of interacting parts, generally too large to be built by a single person, created for some particular purpose. The way to solve a large problem is to break it into a set of interacting smaller problems.

The proposed Mashup tool is designed as the client-server side processing, that is, the control of data processing from the client to the server. The Mashup Engine is primarily responsible for running Mashup service composition, triggering the component's actions, and managing the communication between the client and server. The Mashup engine provides data flow processing; Figure 5-1 illustrates the Mashup tool functionality:



**Figure 5-1. Mashup tool functionality**

As depicted in Figure 5-1, each module can then be decomposed into even smaller problems, up to a point that a problem can be solved on its own. The decomposition gives a set of components, and how do they fit together. Modelling a system requires a right way to decompose the functionality such that small set of abstractions created can be re-used and re-combined to provide the needed functionality (Hwang, 1979). As Figure 5-1 above illustrates the high-level components of the Mashup Tool; Figure 5-2 presents a detailed architecture of the proposed tool that is broken down into components and layer that operate within:



**Figure 5-2. System architecture**

In Figure 5-2 the following layers are illustrated: firstly, discovery layer, which is the module in charge of retrieving web APIs from online web repository. It is used to store web APIs on the local machine where they can be accessed easily. Secondly, annotation layer as stated in the research hypothesis that a semantically enriched form can improve the way that SOA services are currently provisioned for novice users, this module purpose is to semantically annotate services from the local repository and the web; and SerPro API Editor is used as an annotating tool to achieve the aforementioned purpose. Lastly, composition layer after

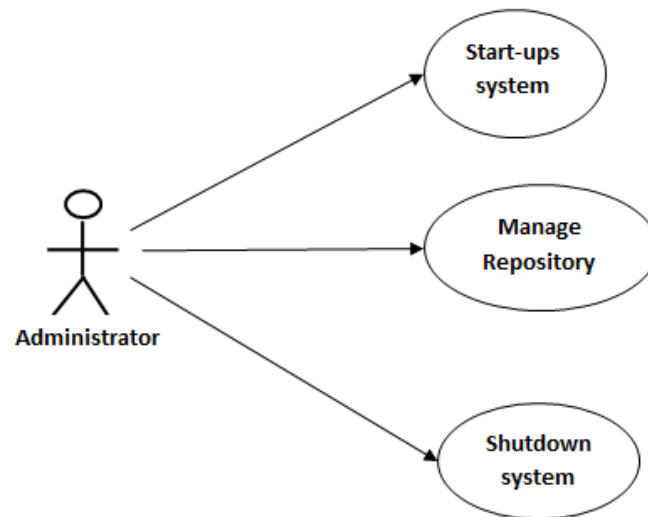
discovering and semantic annotation of publicly available services, this layer aggregates the web contents obtained from such services to create new web applications or Mashups.

As the design of the proposed Mashup tool adapted client-server architecture. The reviewed literature showed that server architectures are more secured than client architecture, moreover client architecture have less payload than server architectures, which results in high performance (.i.e. efficiency). Adopting both architectures yields a well-established system. The following sections describes each of the components illustrated in the system architecture.

### **5.3 Mashup Engine**

The composition layer is where integration of disparate web APIS takes place. This layer is divided into two: visual composition and programmatic composition. Visual composition takes place in the Mashup editor and it is done through GUI widgets and wirings. When end-users finish creating the Mashups through Mashup editor, the final service created is then transferred to the Mashup engine. In the Mashup engine, it is where pragmatic composition takes place and various web contents are combined. Between visual and programmatic composition there is a bi-directional relationship in the sense that every time a user wires or connect two widget the Mashup editor sends request to the server which then connects to the web and send back response to the Mashup editor in order to populate results in the targeted widget.

The Mashup engine parses component details as a JSON object to the composition editor at design time, and bind them in the Mashup engine at run time. This composition layer module implements the REST concept model and supports the checking of data types in the system using the defined ontologies. The Mashup Engine is the key component in the Mashup Tool that establishes communication between different components (i.e., API editor and Mashup Editor). Figure 5-3 shows how users interact with the Mashup engine (only admin users are able to interact with Mashup Engine, novice users only interact with it through the Mashup Editor or the API Editor).



**Figure 5-3. Administrator use case**

The above Figure 5-3 shows the operations of the administrator. The system administrator is the one in charge of frequently updating and maintaining the system. There can be one or more system administrators, and they are required to have technical skills. With the interaction and roles of users being discussed, the next section presents the design of the API editor (annotation browser plugin).

## **5.4 API Editor**

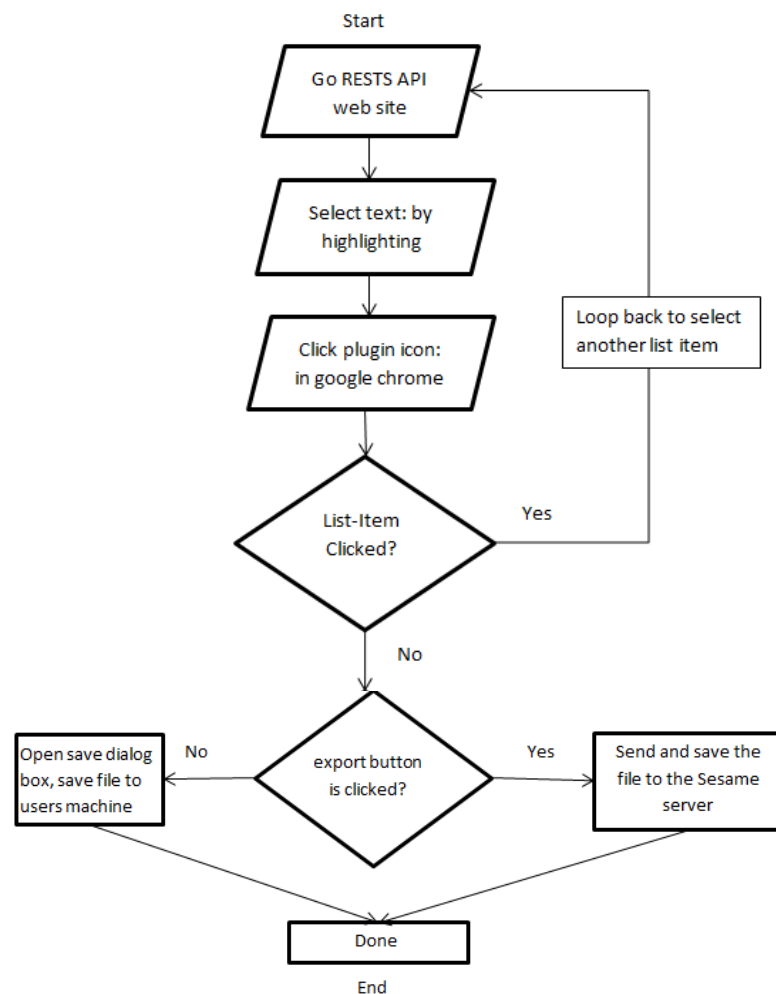
The previous section discussed the Mashup Engine, this section focuses on the key functionalities provided by the SerPro API Editor (SAE) to craft web APIs from the web to the local Sesame server through the Mashup Engine. This component contains two modules: discovery and annotation of web APIs. The discovery layer is partially supported by this component, as its main focus is on the annotation of web APIs.

### **5.4.1 Discovering Layer**

In the discovery layer, is where various web resources are discovered from the web. Discovering resources is one of the major challenges in Mashup development, as a result most of the web resources are manually discovered due to lack of semantic annotation and well-defined structures in the web. This layer uses SerPro API editor which also forms a part of the proposed Mashup Tool. It publishes discovered resources to a repository (i.e., Sesame server), which are later used to populate the API details in the Mashup editor. As resources are discovered they are semantically annotated through the annotation layer.

### 5.4.2 Annotation Layer

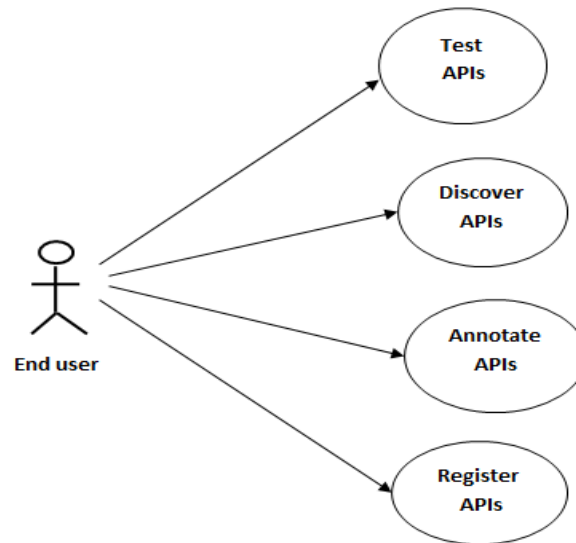
The annotation layer is one of the most crucial parts of the research as it answers the research question (RQ3) in this research. Semantic annotations have been around for quite some time and ever since they evolved, they made life easier for developers. They are said to be the technologies that add meaning and understanding to the web. By semantically annotating resources, the web results in an interoperable environment. On the web, most resources are web services, which are sometimes referred to as APIs. As the literature has showed REST services also form part of these APIs, however in nature they are described in plain HTML. Therefore, this layer describes these APIs in a well-defined structured form like RDF/XML. The annotation layer uses ontology to annotate REST-based web services. These aforementioned web technologies result in a smooth integration of web APIS. When resources are well annotated they undergo the composition process smooth and quick. Figure 5-4, illustrates user interactions with the browser plugin (API editor):



**Figure 5-4.SerPro API editor flow diagram**



The above diagram (i.e., Figure 5-4) illustrates the sequence of steps undertaken when adding APIs from the web. Two listeners are attached to notify and differentiate the system states. In general the use case diagram of the API editor is illustrated by Figure 5-5 :



**Figure 5-5. Use case diagram for API editor**

This section discussed the two layers that operate within the proposed system architecture (i.e., the discovery layer and the annotation layer). Moreover, it discussed how the users interact with the API editor. The API editor component is used to discover APIs on the web and semantically annotate them. Thereafter feed the Mashup editor with all the API details gathered.

## **5.5 Mashup Editor**

A mechanism for discovering and adding APIs to the local Sesame server has been discussed in the previous section. This section presents the component used to create Mashups in the proposed tool. Mashup Editor uses the Mashup Engine to query all the available web API details on the Sesame server. The Mashup engine returns the web API details in a JSON file. Thereafter the Mashup Editor populates widgets based on the obtained web APIs details. This section presents the modules within the Mashup Editor.

### **5.5.1 Visual Composer Layer**

Visual composer editor provides the Mashup design canvas to the user. It shows a Mashup components list, from which users can drag and drop widgets into the canvas in order to connect them (i.e., these list-items become widgets when clicked). Figure 5-6, depicts a pictorial view of the widgets in the context of this research:

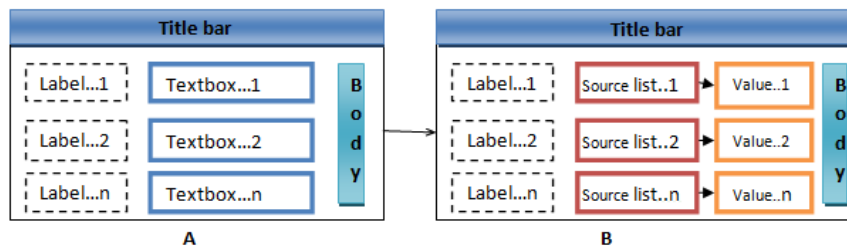


Figure 5-6. Widget transformation

As it was discussed in (Chapter 2), that widgets are used as building blocks of EUD. In Figure 5-6, there's a widget in the left side (labelled **A**) and another in the right hand side (labelled **B**). The one in the left hand side corresponds to the design time while the one on the right side corresponds to runtime. The difference between these binding instances is that in runtime the textboxes that were in design time are converted to two select boxes: first one for source widgets and second one populates response of selected source.

This subsection discussed the design of widgets and the next section focuses on service construction (i.e., how Mashup applications are created).

### 5.5.2 Service Construction

This subsection presents the sequence diagram of how the proposed tool constructs services. The proposed Mashup tool follows both client and server side architecture. The interaction of the Mashup engine, Mashup Editor and the repository is illustrated by Figure 5-7:

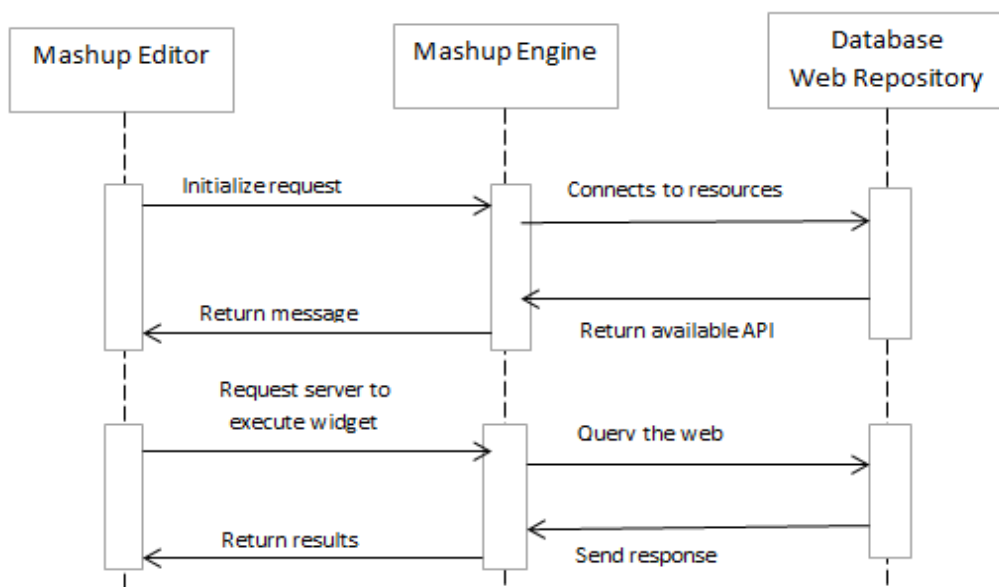


Figure 5-7. Service creation sequence diagram

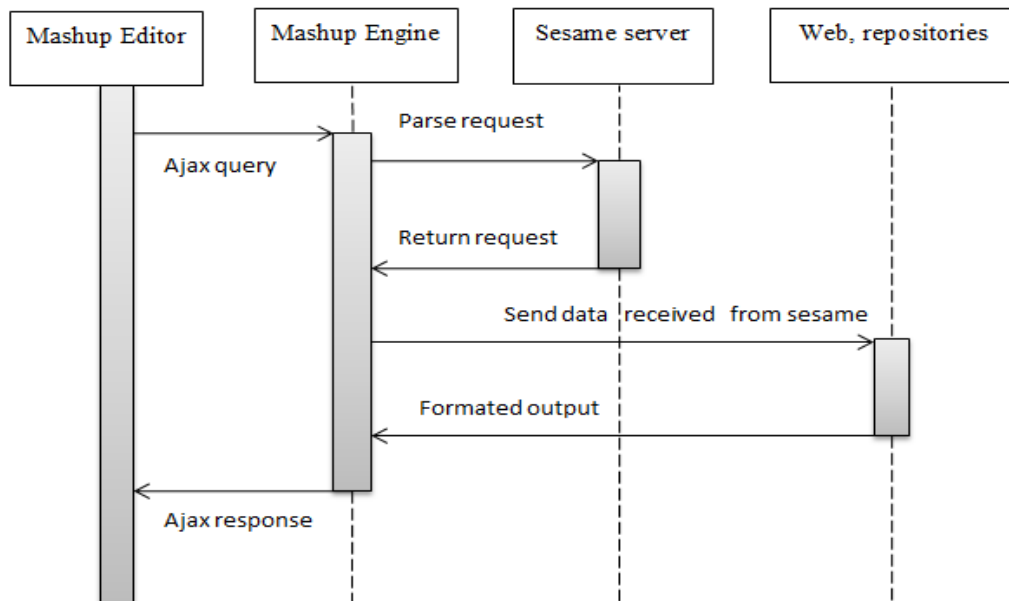
As stated in chapter 2, that a client-server side Mashup tool mixes the content in the web server and transfers it to the client via HTTP Protocol. The following steps are performed during the creation of services:

- i. A user generates an event on the client's web browser. The event triggers a JavaScript function on the client.
- ii. The client makes a request to the server via the ("proxy server") that provides the web site, and the request is typically Asynchronous JavaScript And XML (AJAX) request.
- iii. A web component on the server receives the request and calls a method which encapsulates the code to connect and interact with the other web sites in the Mashup tool.
- iv. The proxy server opens a connection to the web, i.e. the web site that provides the needed service.
- v. The API addresses receives the request, processes the request and then returns data to the proxy server.
- vi. The proxy server receives the response and may transform it to an appropriate data format for the client.
- vii. The proxy server returns the response to the client.
- viii. The AJAX function updates the client's view of the page with the results from the server.

The next subsection discusses request handling and data passing algorithm which was used to integrate web contents form various widgets.

### **5.5.3 Request Handling and Data Passing**

This subsection discusses how different requests are handles within the proposed Mashup tool. For each widget request, JQuery connects to the Mashup engine (i.e. Request handler), as shown by Figure 5-8:



**Figure 5-8. Request handling sequence diagram**

The illustrated sequence diagram in Figure 5-8 above is explained using the following algorithm:

#### **Listing 5-1. Request handling algorithm**

##### **Request handling algorithm**

- A. If (the connection is successfully established) then
  - I. The proxy first query the local server (i.e. Sesame server)
    1. It search requesting widget name against the available APIs
 

If (the widget name matches with one of the defined APIs)

      - a. Sends an object with APIs details to Mashup engines

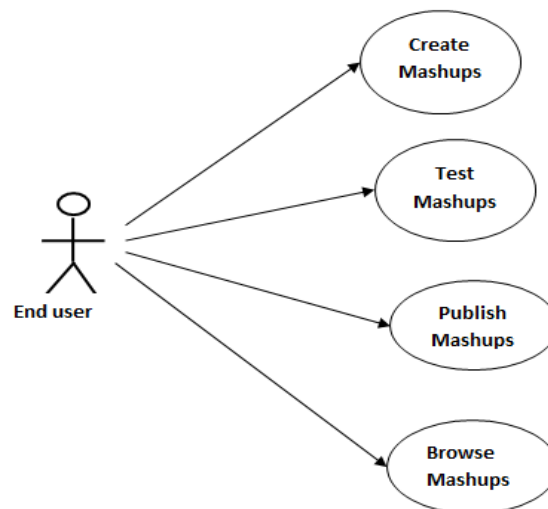
Otherwise

      - b. It sends back the error to Mashup engine
    2. Otherwise
      - a. It sends back error to proxy
      - b. The proxy sends back error to dojohandler
  - II. Send request to the Mashup engine
  - III. The Mashup engine connects to the web and send request to particular API
  - IV. When the response is recieved, it then Mashup editor
  - V. JQuery handler receives the response and passes it as an input to the subsequent widget
  - VI. Then construct the next request
- B. Otherwise
  - I. Request is not sent
  - II. Exceptionhandler notifies client UI that the request was not sent

This subsection briefly outlined how the requests are handled between Mashup Editor and Mashup Engine. Furthermore, it discussed how the data was passed from one widget to the other. Following subsection describes the user interaction with the Mashup Editor.

#### 5.5.4 Use Case Diagrams

Figure 5-9, shows that users can be able to create, update and browse for the available services. In the proposed Mashup tool users are allowed test their Mashup applications before saving them to the repository. The following Figure 5-9, depicts user interaction with the proposed Mashup editor:



**Figure 5-9. End-user use case diagram**

This section briefly outlined how the composition process of both visual and composition process of proposed Mashup tool took place, and it focused on the communication establishment between Mashup editor and Mashup tool. After the details were compiled in the Mashup editor, they are then sent to Mashup tool servlet class. The next section presents the conclusion of the chapter.

## 5.6 Conclusion

This chapter discussed the architecture of the system with its components. The workflow management between Mashup editor and Mashup engine was discussed, moreover the libraries and alayers that were used to develop the system were discussed. Lastly the defined user goals in terms of use cases and sequence diagrams were presented. The next section discussees the implementation of the proposed tool.

## **Part III: Development and Implementation**

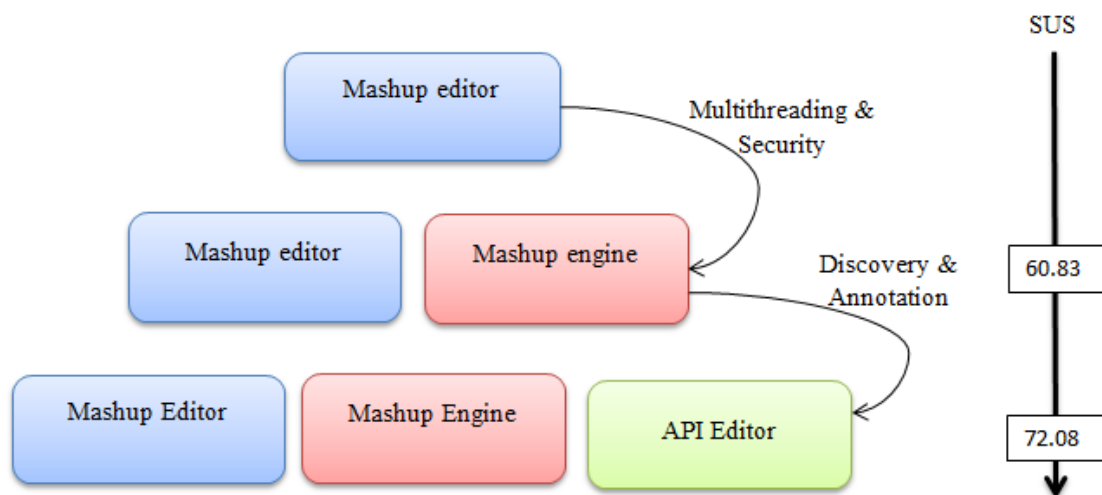
## 6 Chapter 6: Implementation and Development

### 6.1 Introduction

This chapter focuses on how the proposed Mashup tool was implemented using the gathered requirements from the system design (Chapter 5). The purpose of this chapter is to illustrate how the system requirements were converted into a working Mashup tool. The development process is described, and the implementation details for different components are discussed with the aid of code segments. The next section focuses on the development process.

### 6.2 Development Process

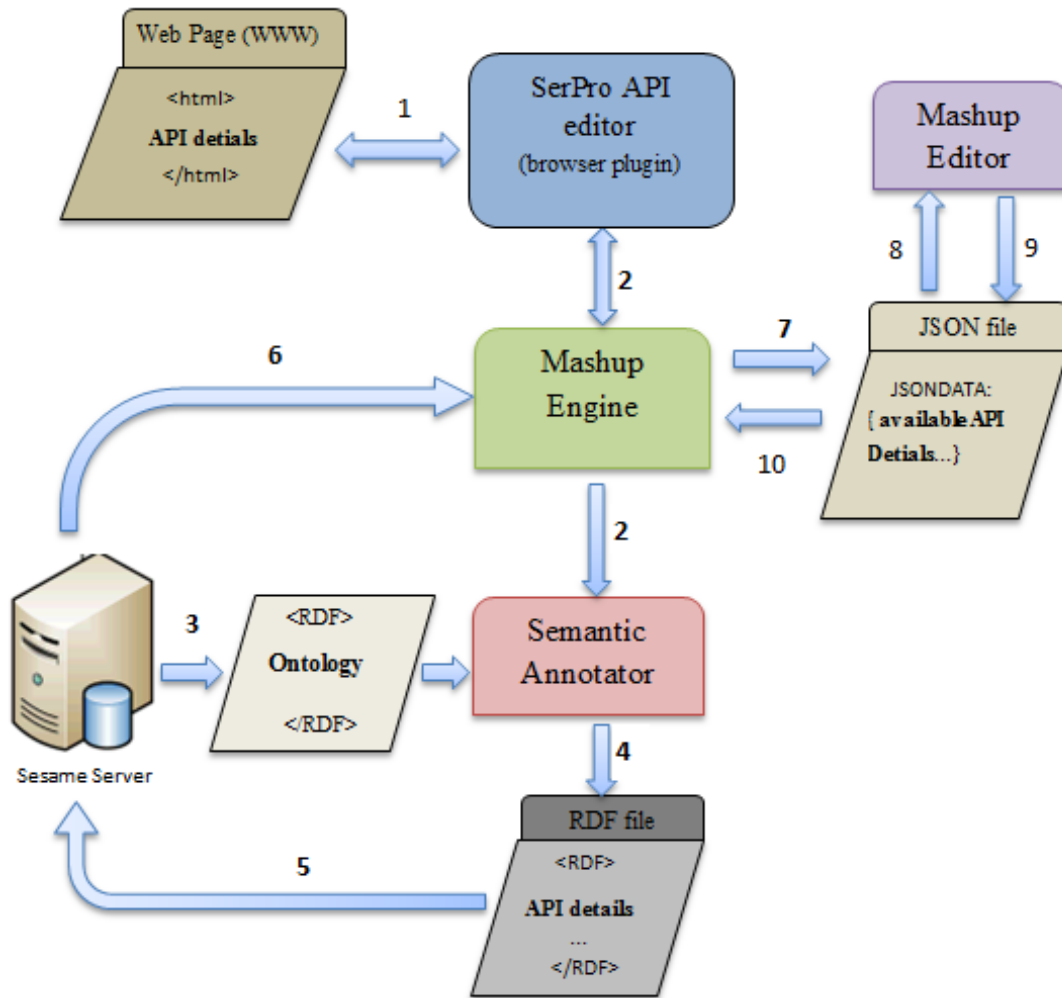
This section outlines the procedure that was used to develop the Mashup Tool, as stated in (Chapter 4), how the system is to be developed. The SLDC used was UCSD as discussed in (Chapter 4). The development process of the proposed tool is iterative in the sense that 3 prototypes were developed as illustrated by the Figure 6-1:



**Figure 6-1. Development process**

The first prototype served as a feasibility study, and only the Mashup Editor was developed. Thereafter a second prototype which included the Mashup Engine was developed to add support for multithreading of processes and security. Lastly, a prototype that included a partial mechanism to discover and annotate web APIs was implemented, to address the manner in which APIs were added to the Mashup Engine. The implementation and the development of the system were executed using three stages. The first stage constitutes the implementation of the Mashup editor and the second phase is the development of the Mashup engine and lastly, is the development of the API editor. As stated in (Chapter 2), that

workflow describes how the data is passed and the sequence in which the data is passed (i.e., control flow), Figure 6-2 illustrates the data flow pattern of the proposed Mashup tool:



**Figure 6-2.Mashup tool data flow**

Figure 6-2 above shows a Mashup engine as a module responsible for establishing communication with the external web resources. It is used to send requests to and from the web. The manner in which APIs are added to Mashup tool is illustrated by (step 1, 2, 3, 4, 5, and 6) which are initiated by the annotation layer that is used to annotate restful services through the API Editor and the Sesame server which is used to publish and query services. For initializing the Mashup Editor with API details, step 6, 7, and 8 are triggered by the Mashup Engine to query Sesame server and create a JSON file that is sent to the Mashup editor in order to initialize widgets. Subsequent to that, (step 9 and 10) are used to construct requests to the Mashup Engine that connects to the web and return API responses. The rest of this chapter discusses how each layer was developed with the aid of code snippets.



### 6.3 Mashup Engine

The aim of the Mashup Engine is to establish communication between Mashup editor, API editor and the web. It controls the flow of data between various components and is the core component of the proposed Mashup Tool. In order to present the development process of the proposed Mashup tool, this research starts by discussing the implementation of the Mashup engine.

As this research has adopted client-server architecture, the following listings present how the Mashup Engine handled the requests and responses coming from different components, and how the Mashup Engine facilitates communication between different components:

#### Listing 6-1.Executing a widget

```
if(isNotEmpty(request,"Single_Widget_Info")){
    String data=request.getQueryString().replaceAll("Single_Widget_Info=", "");
    SolveWidget(data);
    out.println(data);
}
```

Listing 6-1, Illustrates how the various API requests are handled, when executing a single widget. Widget details (such as parameters) are sent from the Mashup Editor to the Mashup Engine, then a method called *SolveWidget* is invoked to parse the request data, construct REST request and connects to the web. Thereafter a response is sent back to the Mashup Editor.

#### Listing 6-2. Adding APIs to Sesame

```
if(isNotEmpty(request,"addingAPI")){
    String param=request.getParameter("addingAPI");
    RDFAnnotator.createJsonStructure(param);
}
```

Listing 6-2, Shows how the Mashup Engine connects with the API Editor. When users finish adding APIs details from the web using the API Editor, the collected data is sent to the Mashup Engine, to invoke a method called *createJsonStructure* of class *RDFAnnotator*. This method annotates API details and saves them to the Sesame server.

#### Listing 6-3.Saving Mashup details

```
if(isNotEmpty(request,"Mashup_Editor_Info")){
    DATA_FROM_CLIENT=request.getParameter("Data");
    MASHUP_NAME=request.getParameter("Name");
    MASHUP_DESCRIPTION=request.getParameter("Desc");
    MashupProxy.MashupInitializer(MASHUP_NAME, MASHUP_DESCRIPTION,DATA_FROM_CLIENT);
}
```

Listing 6-3 shows how the Mashup applications are published to the local server. In order to publish Mashups in the local server, they must have the name and the description to invoke the *MashupInitializer* method of class *MashupProxy*.

#### Listing 6-4. Populating Mashup details

```
if (isEmpty(request, "To_Execute_Mashup_Id")){
    String queryString = request.getQueryString();
    int mashupId=Integer.parseInt(request.getParameter("To_Execute_Mashup_Id"));
    String toremoveFromUrl="http://"+request.getServerName()+":"+request.getServerPort()+request.getContextPath()+"/";
    HashMap sorted=sortQueryString( queryString, "&");
    HashMap mashupResponse=( HashMap)MashupProxy.Find_Mashup_Details_From_Repo(sorted,mashupId, toremoveFromUrl);
    request.setAttribute("response", mashupResponse);
    request.getRequestDispatcher("/home.jsp").forward(request, response);
}
```

Listing 6-5, illustrates how the published Mashup applications are executed. When a certain Mashup is selected for execution, parameters needed in order to complete the request are populated in the home page. Thereafter users fill in the form fields, in order to run a certain Mashup.

#### Listing 6-5. Executing a published Mashup

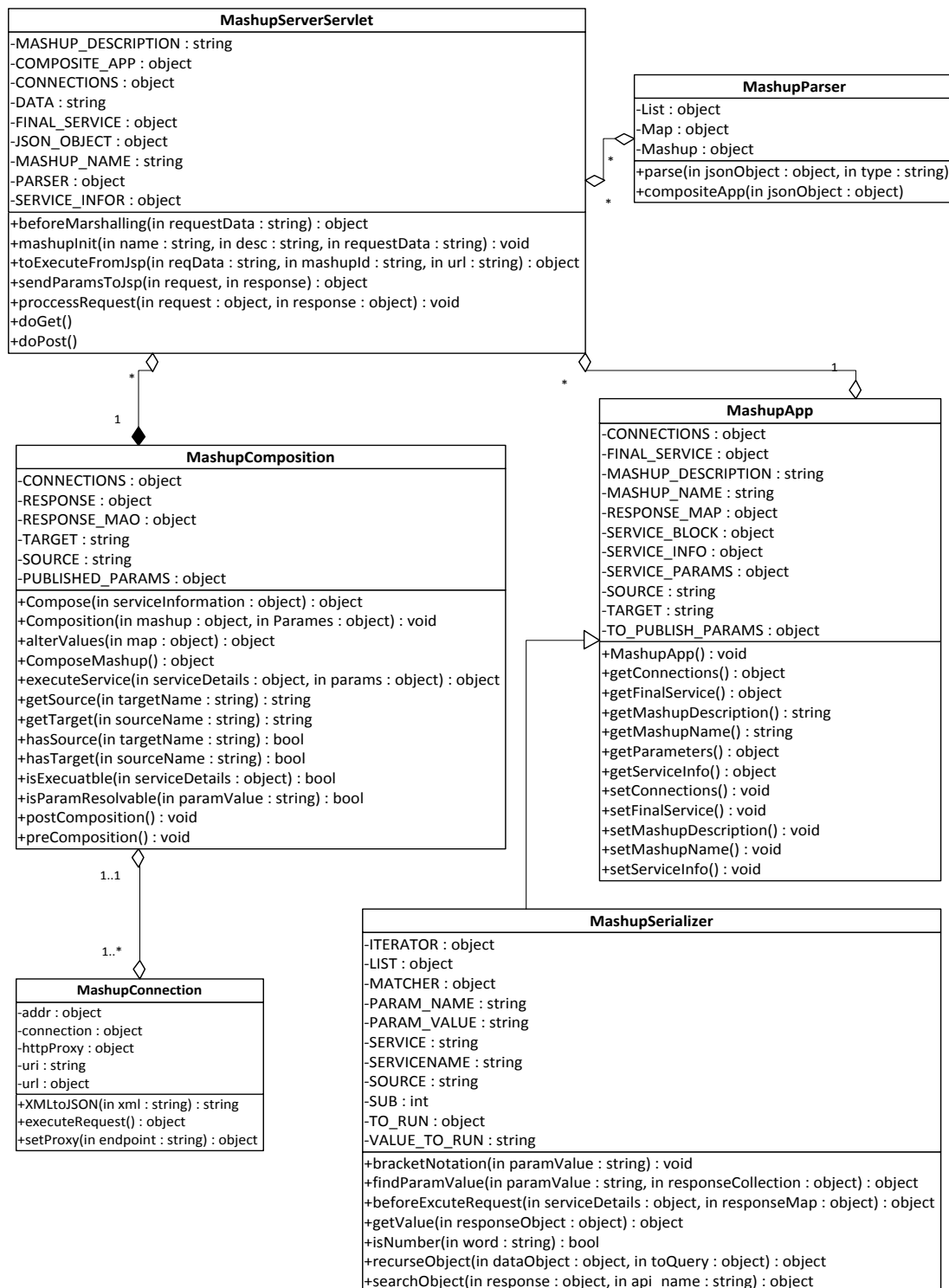
```
if (isEmpty(request, "To_Execute_Mashup_Id")){
    String queryString = request.getQueryString();
    int mashupId=Integer.parseInt(request.getParameter("To_Execute_Mashup_Id"));
    String toremoveFromUrl="http://"+request.getServerName()+":"+request.getServerPort()+request.getContextPath()+"/";
    HashMap sorted=sortQueryString( queryString, "&");
    HashMap mashupResponse=( HashMap)MashupProxy.Find_Mashup_Details_From_Repo(sorted,mashupId, toremoveFromUrl);
    request.setAttribute("response", mashupResponse);
    request.getRequestDispatcher("/home.jsp").forward(request, response);
}
```

The above listings discussed how the Mashup Engine handles the different requests; Listing 6-6, illustrates how the XML responses are converted to JSON format, since the proposed Mashup tool only used JSON as the messaging protocol.

#### Listing 6-6. Structuring XML to JSON

```
public static String XMLtoJSON(String xml) {
    JSONObject jsonObj;
    try {
        jsonObj = XML.toJSONObject(xml);
        String json = jsonObj.toString();
        return json;
    } catch (JSONException ex) {
        Logger.getLogger(MashupConnections.class.getName()).log(Level.SEVERE, null, ex);
    }
    return null;
}
```

As this section has discussed how the composition process takes place in the Mashup engine, the following UML class diagram (i.e., Figure 6-3), illustrates the classes used by the Mashup engine.



**Figure 6-3. Mashup Engine UML class diagram**

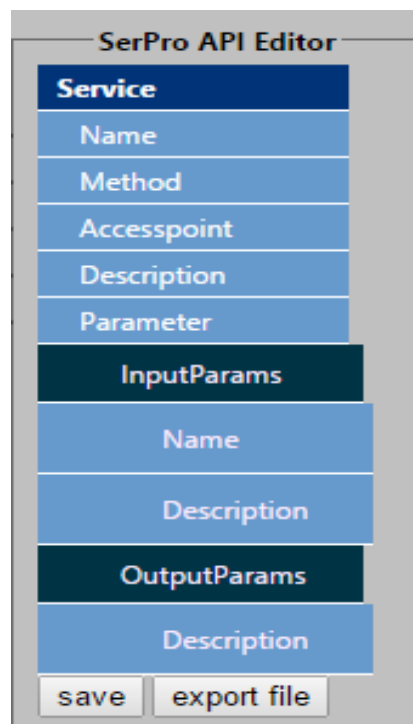
Figure 6-3, shows the classes that extend the Mashup engine in order to maintain sessions and manage requests.

This section outlined that the Mashup Engine maintains the composition life cycle of the proposed tool. Furthermore, this section discussed how the Mashup Engine handles different

requests, and a class diagram showing classes that are used by the Mashup Engine to manage the life cycle of the proposed Mashup tool was presented. Hence, the Mashup engine is the core module of the Mashup tool, as it is in charge of semi-automating the integration of various web contents.

#### 6.4 SerPro API Editor (SAE)

SAE is the tool developed for supporting users in creating semantic Restful services by structuring service descriptions and associating semantic annotations with the aim to support a higher level of automation when performing common tasks, such as their discovery and composition. Its configuration takes the form of a Google chrome plugin displayed within a web browser. It is very lightweight and can be used to directly annotate Restful service descriptions documented in the web site. Figure 6-4 depicts SAE tool:



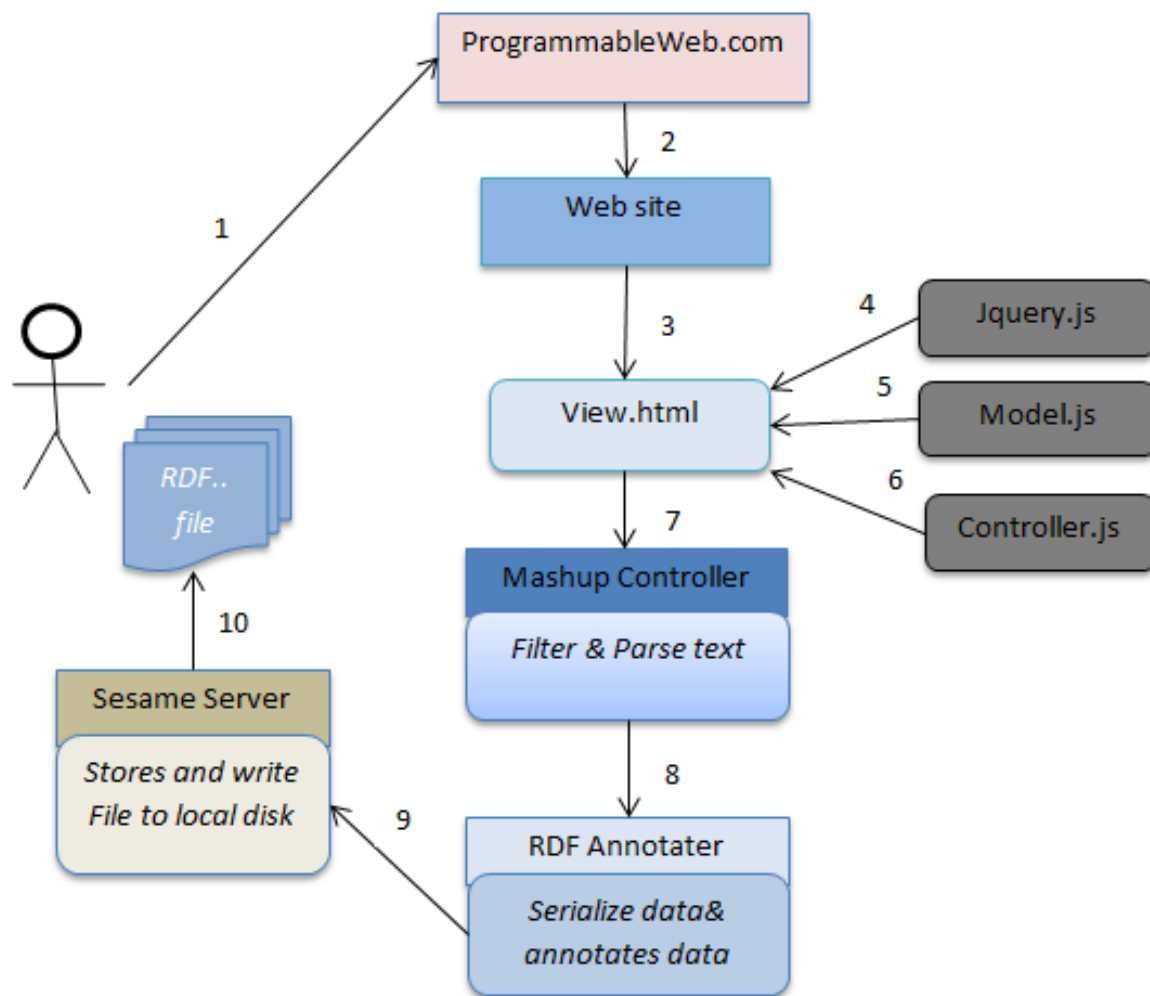
**Figure 6-4.SerPro annotating tool**

Figure 6-4 above is used to discover web APIs from the web by extracting HTML representation and serializing the data as a JSON object which is then semantically annotated using the defined ontology found in (Appendix C). The following listing, describes the role played by the API editor (whose design is inspired by SWEET):

- i. It provides Mapping of HTML service descriptions in order to mark service properties (i.e., operations, address, HTTP method, input, output and labels).

- ii. It includes an ontology to link semantic information to service properties.
- iii. It saves HTML Restful service description, as RDF triples.
- iv. It posts semantic descriptions to Sesame, where they can be browsed, searched and reused.

It can be noted that two layers are included in the API editor (i.e., discovery and annotation layer). The layer that locates API details that are documented using HTML is referred to as discovery component, subsequent to that is the annotation layer which is used to semantically annotate API details in HTML format to RDF format for machine readability and interoperability, as illustrated in Figure 6-5:



**Figure 6-5.API editor workflow**

The following listing explains Figure 6-5 :

1. Users search for rest services through web browsers. They search using search engines and online registries, however the most preferred way is to search using

online registries, such as *programmableweb*, where API details are summarized in details concerning how they can be found and what do they return (i.e., response).

2. As stated in step one above, that online registries are preferred, this step involves visiting the website where the API details reside. Whether a search engine is used or online registry is used, this step remains unchanged.
3. This step is used to map API HTML documentations with the annotation browser plugin (i.e., API editor). Text is highlighted from web site and mapped to the browser plugin by clicking to a corresponding REST property.
4. This step forms a part of (step 3), it attaches a listener to step 3 in order to focus on the currently selected tab. It is also used to simplify JavaScript coding style.
5. Thereafter, this step stores the highlighted text using *LocalStorage*, that is cloned in into a hashmap (i.e. Rest-property --> highlighted HTML text).
6. This step acts as a proxy class between (step 3, 4, and 5). It initiates actions and requests to be taken.
7. The results of the above steps is a JSON data (i.e., the hashmap mentioned step 5), and the Mashup engine filters the request and invoke a class that handles the semantic annotations of services.
8. Uses the data parsed by the Mashup engine and annotates the given data using the defined ontology stored in the Sesame server.
9. The Sesame serverManager class stores the sent data and creates an RDF file based on the triples created from the HTML documentation. It may write the file to the disk or uploads to Sesame server.
10. The RDF file is shown to the users, the user may store it on their own machines or reuse it for different purpose.

As shown by the listing above that the HTML representation of an API documentation can be achieved through the mentioned sequence of steps. The use of SAE produces API details that can be used by different users ranging from human-users to machine-user. The following diagrams present the functionality of SAE; in Figure 6-6 a HTML documentation of Flickr services is shown thereafter a RDF representation obtained using SAE is shown in Listing 6-7:

## flickr.photos.comments.addComment

Add comment to a photo as the currently authenticated user.

**Authentication**

This method requires [authentication](#) with 'write' permission.

**Note:** This method requires an HTTP POST request.

**Arguments**

**api\_key** (Required)  
Your API application key. [See here](#) for more details.

**photo\_id** (Required)  
The id of the photo to add a comment to.

**comment\_text** (Required)  
Text of the comment

**secure\_image\_embeds** (Optional)  
This argument will secure the external image embeds in all the markup and return a secure back in addition to the

**expand\_bbml** (Optional)  
Expand bbml in response

**bbml\_need\_all\_photo\_sizes** (Optional)  
If the API needs all photo sizes added as attributes to the bbml. Use this with expand\_bbml, but dont use it with use\_text\_for\_links. Also when you give this one, you can specify primary\_photo\_longest\_dimension or a default of 240 will be assumed

**primary\_photo\_longest\_dimension** (Optional)  
When used with bbml\_need\_all\_photo\_sizes, it specifies the maximum dimension of width and height to be used as the url

**external\_images\_minimal** (Optional)  
When passed as 1, the external image embeds are minimal and not the big BBML code thats returned otherwise.

**Example Response**

```
<comment id="97777-72057594037941949-72057594037942602" />
```

**API Explorer**

API Explorer : [flickr.photos.comments.addComment](#)

**Figure 6-6.Plain HTML FlickrAddComments service**

Using the API Editor, annotating the above plain HTML file yields the following RDF file:

**Listing 6-7.RDF representation of FlickrAddComments service**

```
<?xml version="1.0" encoding="windows-1252"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:api="http://csc.uhn.ac.za/2013-2014/SerProMashup#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdf:Description rdf:about="http://csc.uhn.ac.za/2013-2014/SerProMashup#flickr.photos.comments.addComment">
    <api:outputNode>&lt;comment id="97777-72057594037941949-72057594037942602" />&lt;/api:outputNode>
    <api:description>Add comment to a photo as the currently authenticated user.</api:description>
    <api:inputParameter>api_key,photo_id,comment_text</api:inputParameter>
    <api:method>HTTP POST</api:method>
    <api:baseUrl>
      https://api.flickr.com/services/rest/?method=flickr.photos.comments.addComment&amp;api\_key=57c6fbfb349da3a1217988416be05ab1&amp;photo\_id={};comment\_text={};format=rest&amp;api\_siq=0ab6e7e851d493f1ff3fd5853f17fa17
    </api:baseUrl>
    <api:Name>flickr.photos.comments.addComment.</api:Name>
  </rdf:Description>
</rdf:RDF>
```

As the diagrams show the representation of both APIs in HTML (Figure 6-6) and RDF (Listing 6-7); that the API details were transformed from plain HTML documentation to RDF

documentation. The following code snippets are used to parse HTML data into JSON object, which is then annotated, using the defined ontology called “*SerPro.owl*” (found in Appendix C). The Mashup engine invokes the *createStructure* method to initiate the annotation process as shown in Listing 6-8:

#### Listing 6-8. Serializes the HTML info to JSON Object

```
OntModel onto= MashupOntology.readOwlFile("SerProOnt.owl");
MashupOntology.createIndividuals(onto,name,method,param,url,node,desc);
SesameServerManager.writeToFile();
```

After the data is parsed as a JSON object, the serializing method is invoked to extract the necessary properties of REST API. After extracting REST API properties, it then stores them into Sesame server by invoking the *createIndividual* method of the *MashupOntology* class. The *createIndividual* method invokes method that saves the details to the Sesame server and that method is called *writeFile*. The above code then call the annotating class and writes the RDF file using *sesameManager* class, using Listing 6-9:

#### Listing 6-9. Annotating API details with Ontology

```
try {
    writeOwlFile( model,"API.rdf");
    RepositoryConnection conn=SesameServerManager.initializeSesameRepository();
    SesameServerManager.LoadOntology(conn);

} catch (IOException e) {
    e.printStackTrace();
}
```

Listing 6-9 shows how the RDF file is created and uploaded to the Sesame server. After such a RDF file is created then it is loaded to the Sesame server using Listing 6-10:

#### Listing 6-10. Uploading RDF files to Sesame server

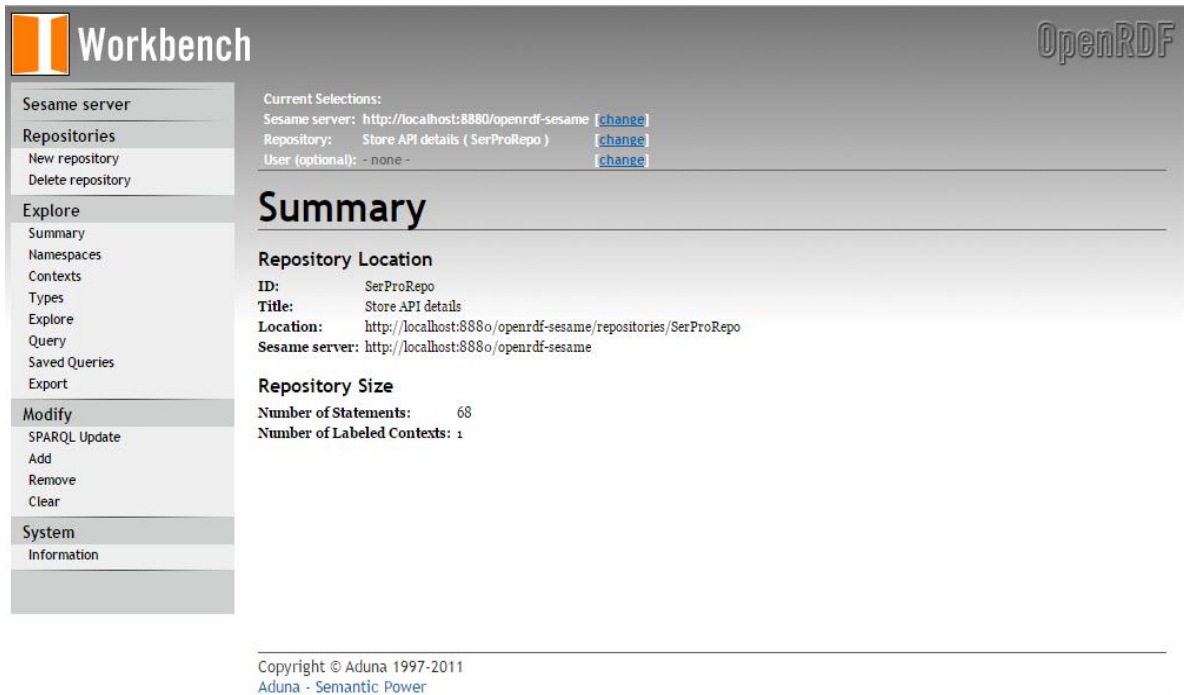
```
try {
    connection.add(_FILE,_NAMESPACE, RDFFormat.RDFXML);
    System.out.println("succesfully added the file to the server:");
} catch (RDFParseException e) {
    e.printStackTrace();
}
```

Listing 6-10 shows how the *sesameserverManager* class establishes connection with the Sesame server and adds the RDF file, using triples mechanism (i.e., subject, predicate, and object).

Sesame is referred to as service warehouse that unifies service publication, analysis, and discovery using lightweight semantics. Sesame provides the features of service registries and



additional functionality that exploits service descriptions and service annotations, and is illustrated by Figure 6-7:



**Figure 6-7.Sesame server**

After the API are annotated and loaded to Sesame server, in order to keep the Mashup Editors API list updated, the *sesameserverManager* queries all the available API triples in its store and the writes a JSON file that is located in the local web server. The created JSON file is then used to populate widgets using DOJO on the Mashup Editor. Listing 6-11 illustrates how the API details are queried and saved to a JSON file:

**Listing 6-11.Querying API from Sesame**

```
String queryString = " PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
    " PREFIX api: <http://csc.ufh.ac.za/2013-2014/SerProMashup#>" +
    "SELECT ?s " +
    "WHERE { ?s ?p ?o . FILTER ((?p = rdf:type) && (?o = api:Service))) } ";
String sub_queryString = " PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
    " PREFIX api: <http://csc.ufh.ac.za/2013-2014/SerProMashup#>" +
    "SELECT DISTINCT ?p ?o WHERE { ?s ?p ?o .FILTER(!isBlank( ?o )&&( ?p != rdf:type))) } ";
TupleQuery tupleQuery = connection.prepareTupleQuery(QueryLanguage.SPARQL, queryString);
TupleQueryResult results, result = tupleQuery.evaluate();
```

Listing 6-11 presents a SPARQL query that is used to select all the API details in the Sesame server and create hashmap object containing all the API details. Listing 6-12, Shows how the object containing API details is then written to a file:

### Listing 6-12.JSON file Writer

```
File jsonFileWriter = new File(jsonFilePath);
if(jsonFileWriter.exists()){
    jsonFileWriter.delete();
}
HtmlParser.createJsonTemplate();
String strFileJson = HtmlParser.getStringFromFile(jsonFilePath);
Object objs = parser.parse(strFileJson);
jsonObj = (JSONObject)(objs);
jsonArr= (JSONArray) jsonObj.get("items");
addToMashupEditor(jsonArr);
HtmlParser.writeJsonFile(jsonFilePath , jsonObj);
```

The JSON file created in Listing 6-12 uses the following template in order to maintain standard form of dojo tree. All the service details are contained in “items” JSONArray object as shown in Listing 6-13:

### Listing 6-13.Creates DOJO compatible template

```
String jsonstring = "{\"label\":\"Name\", \"identifier\":\"Name\", \"items\":[]}\";
```

The output of the Mashup tool after querying Sesame server is a JSON file, which is one of the methods used to create communication between the Mashup Editor and the API Editor, and it has the following structure.

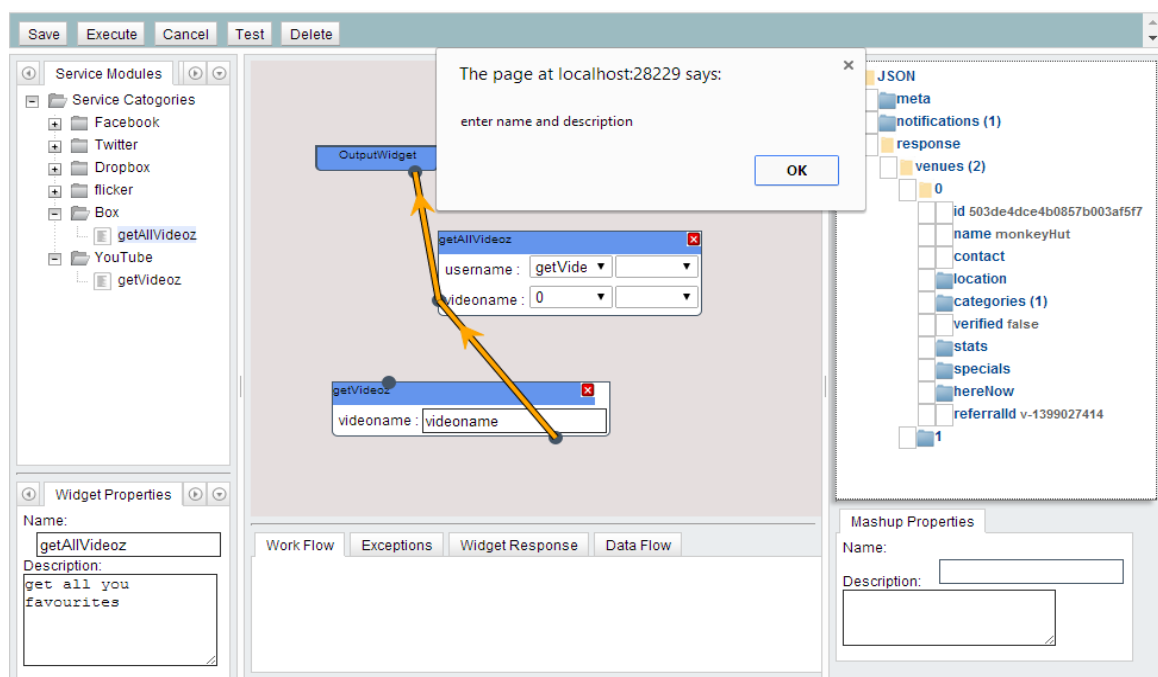
### Listing 6-14.Json file structure

JSON file Structure
<pre>{   "label": "Name",   "identifier": "Name",   "items": [     {       "Name": "",       "category": "",       "operations": [         {           "Name": "",           "Parameters": [],           "Description": "",           "Methods": "",           "Endpoint": ""         }       ]     }   ] }</pre>

This section discussed the role played by the Sesame server in storing and querying API details. The main aim of Sesame server is to store the discovered API details at the same time be able to query (i.e. select and match APIs) locally in order to allow a smooth composition of services. As this section described how the API details were added and retrieved from Sesame server, the next section presents the Mashup Editor.

## 6.5 Mashup Editor

In this research Mashup editor is referred to as the main module that users require in order to compose new services. This is because most of the time, the users design and compose Mashup through it and novice users do not care about the server-side technicalities; they are only concerned about what they see and do. The review of relevant literature and pre-evaluation study showed that when Mashup tools are evaluated, the main tests lie on the usability of the Mashup editor (i.e., so the easier it is to use a Mashup editor is the more a Mashup tools becomes usable). In other words, the simplicity of the Mashup editor is directly proportional to the usability of the Mashup tool. A Mashup editor requires an intuitive design as illustrated in pre-evaluation study for exiting Mashup tools. As stated in (Section 5.2) that the composition of Mashup applications involves both Mashup editor and Mashup tool, Figure 6-8 shows that with the Mashup editor users are able to design their application by wiring widgets and pipes:



**Figure 6-8. Mashup Editor**

Figure 6-8, depicts the functional view of how the users compose services using the Mashup editor together with the Mashup engine. Each module of the Mashup editor is discussed as to how it aids the creation of Mashups with novice users. The file structure labelled “Service Modules” that is shown on the left of Figure 6-8, is produced from the local JSON file created from the local repository to initialize the widgets.

### 6.5.1 Connecting Widgets

In the centre of Figure 6-8 is the canvas which allow the dragging and dropping of widgets. The widgets use information that is attached to dojo tree using JSON file. When the item from the dojo-tree is clicked, dojo attaches service details to the click-listener object, then when creating widgets JavaScripts simple iterate through the event object to populate the necessary data on the widget. These widgets are designed using HTML form elements, and when connecting two or more widgets the code in Listing 6-15 is used:

**Listing 6-15.Connecting and wiring two widgets**

```
jsPlumb.bind("connection", function(connection) {  
    if(connection.targetId!="Results"){  
        removeGoodInputs(connection.targetId,connection.sourceId);  
        sorter(connection.targetId);  
    }  
});
```

Listing 6-15 shows that when two or more widgets are connected two method are invoked (i.e., *removeGoodInputs* and *sorter*). The first function is used to remove any input fields in widget and replace them with two select boxes as shown in Figure 5-6 in (Section 5.5.1). When users unwire widgets, Listing 6-16 is used to attach the detach event listener:

**Listing 6-16.Unwiring two widgets**

```
jsPlumb.bind('dblclick', function (connection,event) {  
    jsPlumb.detach(connection);  
  
});  
jsPlumb.bind('beforeDetach', function (connection) {  
    var s=confirm("Detach connection?");  
    if(s){  
        var selectBoxExist=beforeConnectOnRunTime(connection.targetId);  
        var formname=connection.targetId;  
  
        if(selectBoxExist){  
            $(document).ready(function() {  
                $("#"+formname+" select ").each(function() {  
                    $(this).remove();  
                });  
                $("#"+formname+" label").each(function() {  
                    var labelText=$(this).text();  
                    $(this).append("<input type='text' name='"+labelText+"id="+formname+"-"+labelText+" value='"+labelText+">");  
                });  
            });  
        }  
    }else{  
        alert("it was a mistake, i have changed my option");  
    }  
});
```

Furthermore, it shows how the select boxes are removed and replaced with the input fields. From the editor, it is also possible to execute Mashups and widgets, the next section presents how the requests are constructed.

### 6.5.2 Construction Requests

The construction of requests takes place using two type of default widget namely: output-widget and user input-widget. The output-widget is use to signal the completion of Mashup design (i.e. the final service) whilst the input-widget signals that for a particular widget to execute, a user is prompted to enter a certain value.

#### Listing 6-17.Executing request

```
(function($) {  
    $(document).ready(function() {  
        $('#Results').click(function(Event) {  
            var name=$('#name').val();  
            var description=$('#description').val();  
  
            if(name!=null&&description!=null){  
                getCompositeApp(name,description);  
            }else{  
                alert("enter name and description");  
                Event.preventDefault();  
                Event.stopPropagation();  
            }  
        });  
    });  
})(jQuery);
```

Furthermore, each time users save Mashups, they are required to fill in the name and description of the Mashup, and if the user does not do so an alert box is shown as illustrated in the Mashup editor. After the output-widget is clicked, Listing 6-18 is used to collect widget details:

#### Listing 6-18.Instantiating request

```
clientSideInfo={Mashup_Editor_Info:JSON.stringify(clientSideInfo),name:mashupNam,desc:mashupDescriptio};  
InstatiateRequest(clientSideInfo);
```

Using the code in Listing 6-18 to formulate API details into JSON object, then Listing 6-19 is then used to send a request to the Mashup engine:

#### Listing 6-19.Sending request to the Mashup Engine

```
$.ajax({  
    type: 'GET',  
    url: 'http://localhost:8680/Tools4DaWebApps/MashupController',  
    data: dataObject,  
    success: function(html, textStatus) {  
        html=research2;  
        var response=new Array();  
        try{  
            $.fn.exists = function(){return $(this).length>0;}  
            JSONtree.init('jsonTree','jsonForm',html);  
            response=structureData(html);  
            responseHandler(response,idResponseSelectBox);  
        },error: function(xhr, textStatus, errorThrown) {  
            alert('An error occurred! ' + ( errorThrown ? errorThrown : xhr.status ));  
        }  
    });
```

Listing 6-19 shows how requests are constructed. The web technologies that are used to construct request are AJAX and JQuery, and the response on success, is handled by *JSONtree* function, which populates the response data in a tree view. On the right side of Figure 6-8, the display module, which shows the widget responses returned from the Mashup engine, is shown in a tree view mode with variable-name and values. The purpose of the tree is to make it easier for the users to see and know what they mashing.

In order to maintain data passing between widgets a *responseHandler* function is invoked to populate response data in select boxes of the targeted widget. Listing 6-20 shows how the response data is passed from one widget to the other:

### Listing 6-20. Response handler in widget piping

```
function responseHandler(responseOptions,targetedId){
    (function($) {
        $(document).ready(function() {

            $("#"+targetedId+" option").each(function() {
                $(this).remove();
            });

            $("#"+targetedId).append("<option >val</option>");

            $.each(responseOptions,function (idx, connection) {
                var html = '<option value=' + connection+'>' + connection + '</option>';
                $("#"+targetedId).append(html);
            });
        });
    })(jQuery);
}
```

The *responseHandler* function uses JQuery to traverse through the JSON object and populate option inside select boxes. Different classes are used to initialize and manage events in the Mashup Editor, and the class diagram (i.e., Figure 6-9) below depicts the classes and methods involved in the implementation of Mashup Editor:

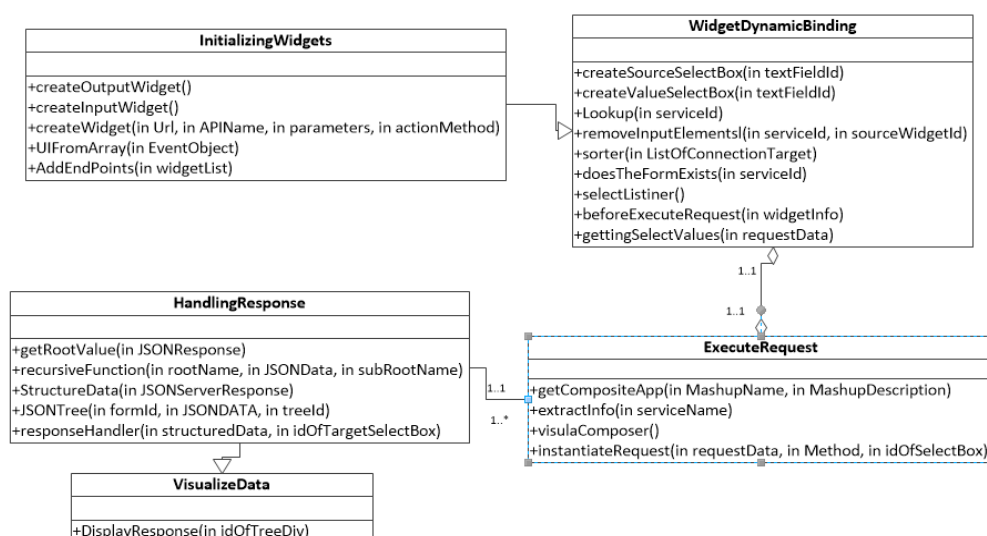


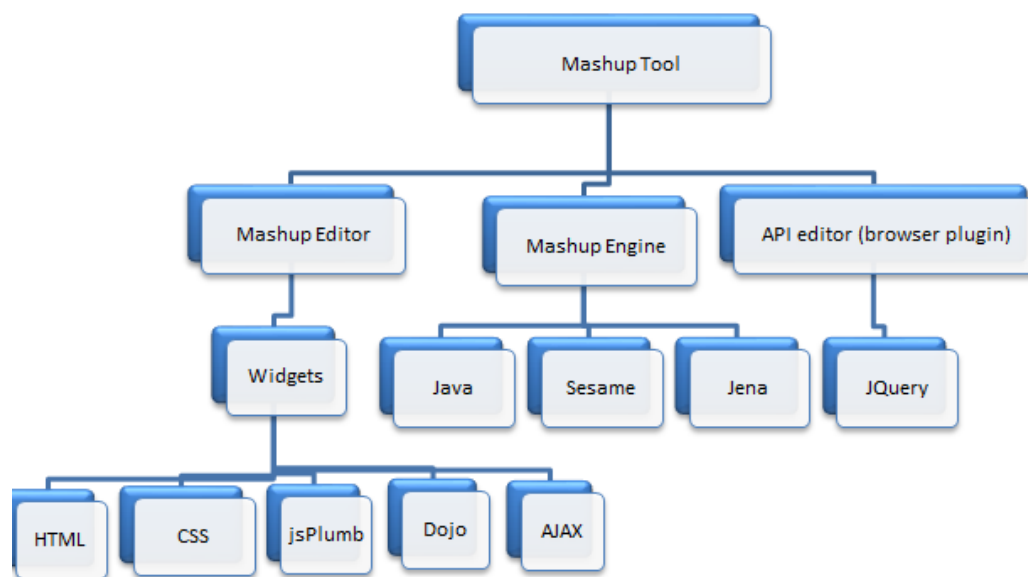
Figure 6-9. Mashup editor development UML diagram

Figure 6-9, explains the policies used in the class diagram: Each widget uses two libraries (i.e. Dojo library and Jsplumb). Dojo library is used for data extraction. Jsplumb is used for widget manipulation (User interface). Each widget uses events handlers to establish communication between one another.

This section presented the visual composition layer (i.e., Mashup Editor) together with its components; screenshots were presented to show the user interfaces and the pictorial view of the proposed system. As the Mashup engine is in charge of executing request made in client side and return responses back to the client side in an amenable way (i.e. JSON format). The next section presents the research tools and libraries that were used to develop the proposed Mashup Tool.

## 6.6 Research Implementation Technologies

The Mashup tool consists of three layers represented by SAE, Mashup engine and the Mashup editor, with a specific focus on discovery and semantic annotation, composition, and UI provisioning respectively (see Figure 5-2). The project was implemented in Java. On the server side JSE and Java Servlets were used to create the server. On the client side JSP, CSS, HTML and JavaScript's were used. The discovery and semantic annotation layer were developed using the server side technologies, while composition layer was developed using both server-side and client-side technologies. The rest of this chapter focuses on implementation on each of the aforementioned layers.



**Figure 6-10. Development Structure**

- i. Java is an object-oriented computer programming language, and it is specifically designed to have as few implementation dependencies as possible. The code that runs on one platform does not need to be recompiled to run on another. Java is simple, familiar, secure, architecture-neutral and portable.
- ii. Ontology tools: Jena is a Java framework for building semantic web applications. It provides extensive Java libraries for helping developers develop code that handles RDF, RDFS, RDFa, OWL and SPARQL. Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies, and a variety of storage strategies to store RDF *triples* in memory or on disk.
- iii. Sesame is an open source Java framework for storing, querying and reasoning with RDF and RDF Schema. It can be used as a database for RDF and RDF Schema, or as a Java library for applications that need to work with RDF internally. Sesame provides the necessary tools to parse, interpret, query and store information. More generally; Sesame provides an application to developers as a toolbox that contains useful functionalities to manipulate RDF.
- iv. CSS: is a simple mechanism for adding style into web pages (e.g., position, colours, size), and it is a way to style and present HTML. HTML is the meaning or content, which is presented using style sheet.
- v. AJAX: is an important front-end technology that lets JavaScript communicate with a web server. It allows website to load new contents without leaving the current page, by that it creates faster and better experience for user.
- vi. JavaScript (JS) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. It is also being used in server-side programming, game development and the creation of desktop and mobile applications.
  - a. Dojo toolkit is an open source modular JavaScript library designed to ease the rapid development of cross-platform, JavaScript/Ajax-based applications and web sites.
  - b. jsPlumb provides a means for a developer to visually connect elements on their web pages. It uses SVG in modern browsers, and VML on IE 8 and below.



- c. JQuery jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is the most popular JavaScript library in use today.

This section discussed the technologies that were used to implement the proposed Mashup tool. The next section presents the chapter summary.

## **6.7 Conclusion**

This chapter detailed on how the system components were implemented. Furthermore, the section also outlined how the composition modules were integrated (i.e. how the communication was established). Three components were addressed which played an important role in the establishment of communication with both the web and the Mashup tool. Lastly this section presented the technology review to describe what software's and libraries were used to implement the proposed Mashup tool.

The next part of the research focuses on the evaluation and conclusion of the research.

## **Part IV: Experimentation and Research summary**

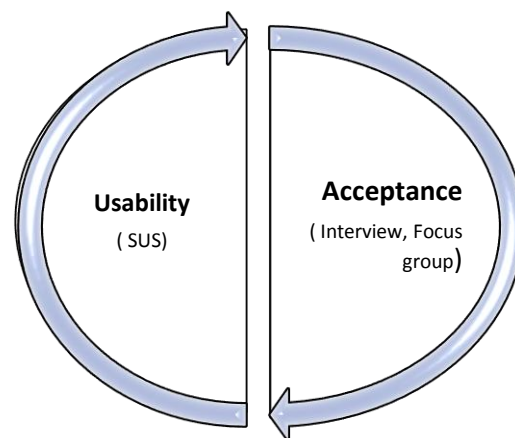
## 7 Chapter 7: Experimentation and Discussion

### 7.1 Introduction

This chapter presents the tests that were conducted to evaluate the effectiveness of the proposed Mashup tool. Furthermore, it details on the types of tests that were conducted (i.e., usability testing and acceptance testing). Thereafter the results obtained from the tests are analysed, interpreted and discussed.

### 7.2 Validation Process

As it is already outlined that the validation process undertakes two types of tests to evaluate the system, this section presents how the evaluation of the proposed Mashup tool was conducted. With the main research question being how to design a usable Mashup tool that targets novice users, usability testing is conducted to determine whether the implemented Mashup tool is usable and suitable for novice users. While acceptance testing is conducted to evaluate novice user's attitude, preferences and behaviour towards the Mashup tool proposed. Figure 7-1 illustrates a condensed view of the evaluation process:



**Figure 7-1. Validation process**

For each test, the following aspects are discussed:

- i). Why choose the type of test?
- ii). What methodology was used to achieve it?
- iii). What were the results?

Thereafter the research compares the results obtained from the two conducted tests with the findings obtained from the pre-evaluation study presented in Chapter 4 and in Appendix B. Furthermore the results obtained are analysed and interpreted.

The next section focuses on the usability tests conducted, as it is the main objective of this research to implement a usable Mashup tool.

### **7.3 Usability Testing**

This section presents the usability of the proposed tool using the SUS to determine the success of the proposed Mashup tool, where all of its modules had been integrated. This type of test is conducted to evaluate the usability of the system, as the research topic stated that the aim of this research is to enhance the usability of SOA services for novice users. Usability test is conducted since it measures the extent to which users can use a system. The proposed Mashup tool usability was measured using a SUS. Furthermore the research employed a UCSD, that means that the Mashup tool was design with users in mind, and also the user's needs towards usability of SOA services are the central focus of the research (Preece et al., 1994).

#### **7.3.1 Methodology**

The SUS scores are between 0 and 100, and they are analysed using percentiles. Percentiles are useful for giving the relative standing of an individual in a group, and they provide normalized ranks. A SUS score of 68 (a usability threshold) signifies that a system is usable to its users (Brooke, 2013). The SUS is a simple, ten-item scale giving a global view of subjective assessments of the system's usability. The respondents were trained on how to use the proposed Mashup tool (i.e., they had an opportunity to interact with the system before the evaluation process). During the evaluation process 12 respondents were allowed to interact with the proposed Mashup tool, thereafter before any discussion took place, the respondents were asked to record their immediate response to each item (a SUS questionnaire found in Appendix B). The 12 postgraduate students were from the same institution as researchers and the ethical clearance certificate was obtained from the university (found in Appendix A). The respondents used were from Faculty of Social Sciences (with 4 of them masters, 8 of them being honours students). The purpose of the SUS was to determine the usability and learnability of the Mashup tools.

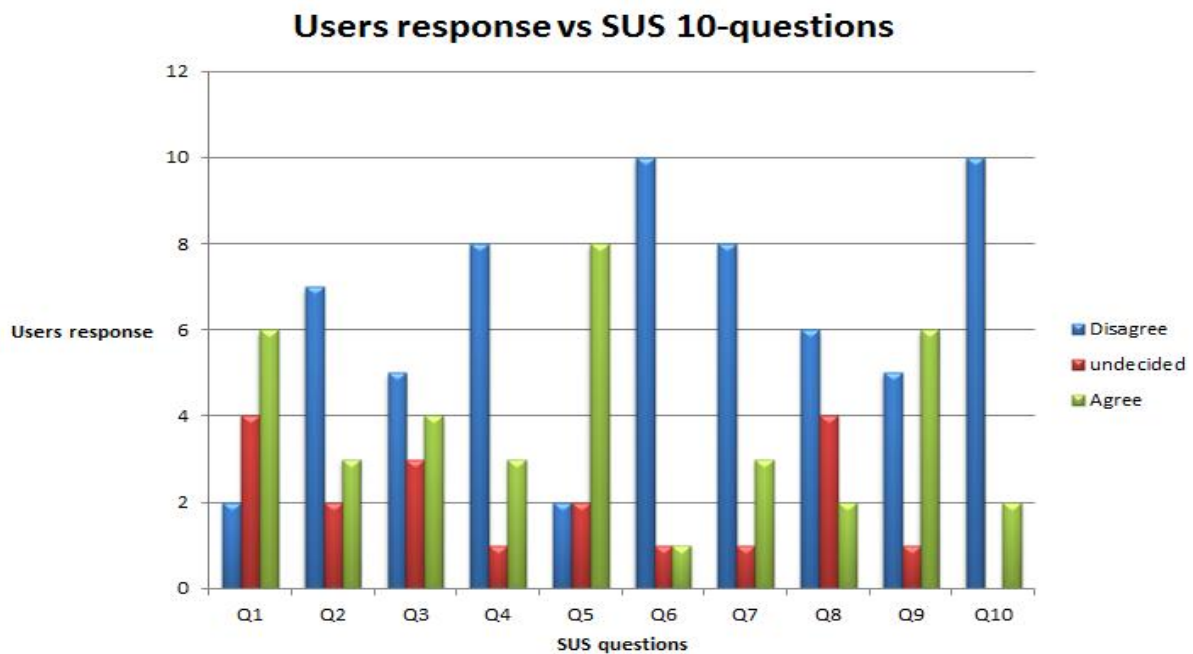
#### **7.3.2 Findings of SUS**

Using twelve respondents and normalizing user's response, the following were the obtained SUS scores.

**Table 7-1.SUS score in each question from 12 users**

	Disagree	Undecided	Agree
Q1	2	4	6
Q2	7	2	3
Q3	5	3	4
Q4	8	1	3
Q5	2	2	8
Q6	10	1	1
Q7	8	1	3
Q8	6	4	2
Q9	5	1	6
Q10	10		2

The table above shows the response from the users, for each question in the SUS. Since the SUS has ten question, each question is denoted by  $Q_n$  for  $n= 1...10$  (i.e., it can be found in appendix B). The purpose of this table is to map user's attitudes and behaviours towards the proposed Mashup tool. Figure 7-2, shows the frequency of users who disagreed, in-between and agreed, for each question:



**Figure 7-2.Frequency of users SUS question responses**

In Figure 7-2 above, it can be seen that there were users who were in-between (i.e., undecided) for all the questions except last question (i.e., question 10) while in other ones

they agreed or disagreed. For instance question 1, it can be noted that 6 of the users showed that they would enjoy using the system frequently and that they were satisfied with how the system operated. Moreover, for question 2, only 3 users disagreed that the proposed tool was simple and intuitive, the rest found the system simple while 7 of the users found the system simple and properly designed. Furthermore, for all the question users responses were positive; they showed that the proposed Mashup tool was usable and acceptable to them. For question10, which is the question on learnability of the system, the respondents showed that there was not much to learn recording a total of 10 out 12. Table 7-2 presents the normalized SUS scores of the proposed Mashup Tool:

**Table 7-2.SUS scores of proposed Mashup tool**

Users	Score	SUS score=score*2.5
1	24	60
2	29	72.5
3	33	82.5
4	32	80
5	30	75
6	27	67.5
7	33	82.5
8	29	72.5
9	29	72.5
10	25	62.5
11	21	52.5
12	34	85
sum		865
Overall SUS score		72.08

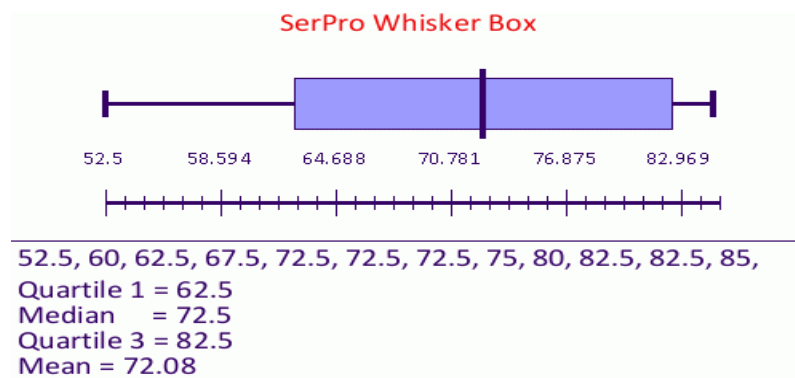
The obtained SUS scores is 72.08 as presented in Table 7-2. The obtained SUS score is above the general threshold of SUS (i.e., 68), and that signifies that the proposed Mashup tool is usable to novice users. Furthermore the obtained SUS scores are analyzed using the following five-number summary:

**Table 7-3.Five Whisker terms**

Standard Deviation	Mode	Median	Min	Max	First Quartile(Q1)	Third Quartile(Q3)
9.99	72.5	72.5	52.2	85	62.5	82.5

Table 7-3 above table shows the five numbers for plotting a whisker. It can be noted that the median is 72.5, which means the middle number of the score is 72.5. The median show that most of the scores were positive. The standard deviation was calculated to be 7.71, which

signifies the spread of scores from the mean (72.08). It means that the scores can range from (62.09, 82.07) which shows positive results since most of SUS fall inside the box. They yield the following Whisker and a Box:



**Figure 7-3.SUS scores Box of Whisker**

The above diagram (Figure 7-3) shows the measure of usability of the proposed Mashup tool against users SUS scores. It can be noted from the diagram 25<sup>th</sup> percentile is 62.5, which means that 25% of the individual SUS scores are below 62.5 and 75% of the scores are above 62.5. It shows that most users were happy with system usability. The interquartile range is 20, which signify the difference between 25<sup>th</sup> and 75<sup>th</sup> percentile. The lower quartile range shows that 10 out of 12 scores were inside the box (i.e. they were between 25<sup>th</sup> and 75 percentile). As it was discussed in the literature that for system to be usable it must have attained a SUS score of 68 and above, the SUS score of the proposed tool was 72.08 which above the average SUS score. These results signify that the proposed Mashup tool was usable to novice users. The following section evaluates the acceptability of the system.

## 7.4 Acceptance Testing

This section presents the acceptance testing that was used to evaluate whether the proposed system was acceptable to novice users. The acceptability of the system was evaluated using interviews and focus groups to describe the user's behaviours, desires and reactions to the proposed Mashup tool. These research methods were used as a justification technique towards the usability of the proposed Mashup tool.

### 7.4.1 Methodology

This section presents the methods that were used to test the acceptability of the system. In the evaluation process users were given the following scenario:

**Case scenario:** As a user who is computer literate, but not having programming skills but you are eager to use the internet to gain and share information. You have heard about the type of web application that makes accessing different web resource easy, namely Mashups. You are then asked to create a Mashup application. You have been promised that an initial training will be conducted to get you started. How will you go about creating your Mashup Application?

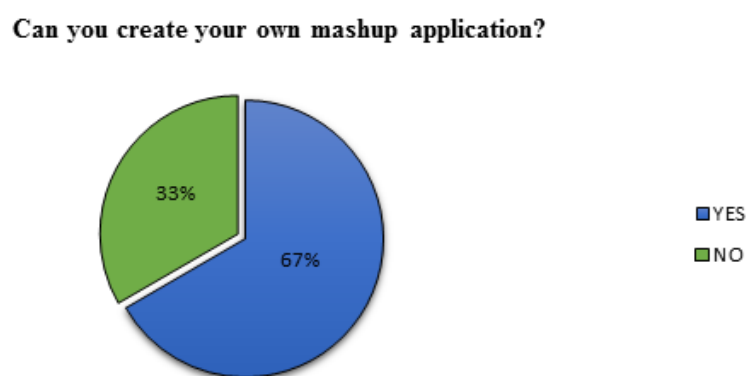
The purpose of this study was to validate the system acceptability and usability. The interview questions are discussed in the next subsection.

#### 7.4.1.1 Interviews

For the interview 15 respondents were used (including the 12 students from the Faculty of Social Sciences that were used in usability test) there 3 students were given training. As all of the users in the interview sessions were trained prior the interview the following question were asked during the sessions:

- (Q)1 Can you create your own Mashup application?
- (Q)2 Are you able to discover web APIs? If yes then how?
- (Q)3 Can you publish and semantically annotate Mashups?
- (Q)4 Are you satisfied with design and usability of the system?

The above listing served as the follow-up questions of the SUS testing, the following results were obtained:



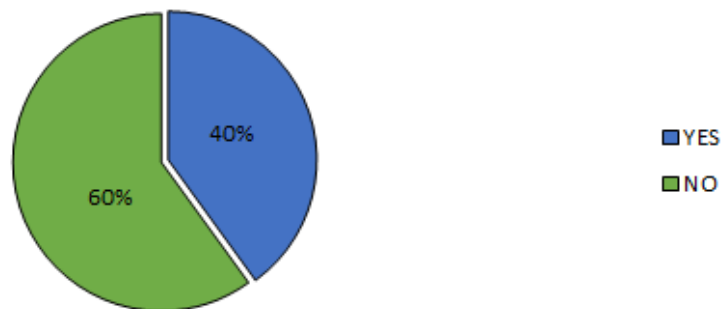
**Figure 7-4. Question 1 vs. user's responses**

The pie chart in Figure 7-4 above shows the results of the people that were involved during interview sessions for question 1 (Q1). This was done to investigate the usability of the interface. The results were positive, 67% of the interviewees demonstrated that they were able to create Mashup applications while 33% found it difficult to create Mashup



applications. As the system was developed for novice users, with main purpose of the Mashup tool being to allow them to create web application of their choice, the user interface (i.e., Mashup editor) was easy and usable, however 33% of the users found the Mashup Editors interface unnecessary complex.

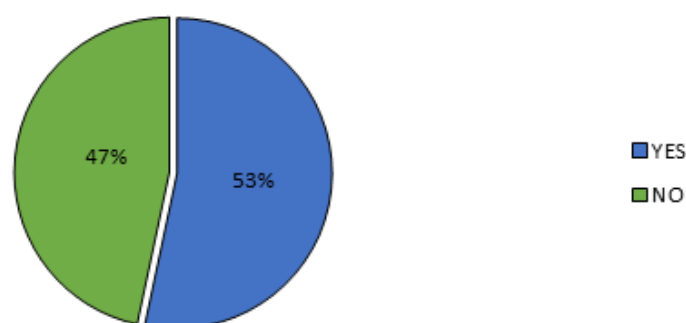
**Are you able to discover web APIs? If yes then how?**



**Figure 7-5.Question 2 vs. user's responses**

Figure 7-5 gives an explicit detail of users' responses based on how they discovered APIs. The results depicted by the pie chart shows that it was not easy for users to discover APIs on the web. As a result 60 % of the respondents found the process time consuming and frustrating whilst 40% of the respondents were able to discover APIs. However even though 40% of the respondents were not satisfied, the majority of the users were confused and not satisfied (i.e. did not know where to start).

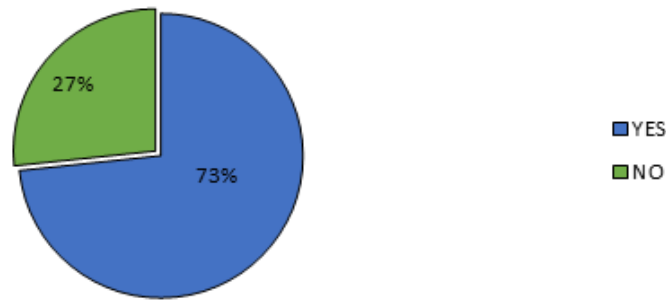
**Can you publish and semantically annotate Mashups?**



**Figure 7-6.Question 3 vs. user's responses**

This question was used to check whether the users were able to publish and semantically annotate web APIs using SerPro API editor. According to users response annotating web APIs was easy and enjoyable as compared to discovering APIs, and 53 % of the respondents found annotating web API easy while 47% could not semantically annotate web APIs.

**Are you satisfied with design and usability of the system?**



**Figure 7-7. Question 4 vs. user's responses**

This was the last question and it was used to check the overall response of the users towards the proposed Mashup tool. The user's response showed that the system was effective and satisfying, 73% of users were happy with the design of the system. Although 27% of users were not satisfied since they found the API editor more error-prone (i.e., they were making many mistakes).

This subsection presented results from the interview; the next subsection focuses on the focus group session.

#### **7.4.1.2 Focus Group**

In the focus group, the above scenario was given to 6 students (of 3 were undergraduates and the rest postgraduates). All the above subjects were from the Computer Science department. They were asked to create a simple Mashup application using iTunes and foursquare APIs, thereafter to discover their APIs of choice and annotate them. The following aspects were used as the key points in each task to evaluate their effectiveness in the proposed Mashup tool:

- i. *Mashup editor*: widgets, piping, tree-views, select boxes.
- ii. *SerPro API Editor*: highlighting, button-click, chrome-plugin, compatibility.
- iii. *Discovery*: programmable web, web engines, ontology-base repositories.

In the focus group the following questions were used:

- (Q)1 Is the proposed tool conforming to what it promised?
- (Q)2 Do the added support improve usability of Mashup tools?
- (Q)3 To further improve usability of Mashups, what can be added?

The results obtained from the focus group are presented using two sections; Mashup Editor, and API Editor and discovery.

## **Mashup editor**

All the users enjoyed drag and drop together with wiring widgets, as they mapped it to a game (“referred to as snake”). Several users (i.e., 4 out of 6) also recommended the use of select boxes to populate widgets response when piping. As they insisted that, it was easy and less error-prone since users only had to choose from what exists than having to use filters that were used by its counterparts. For all of them (6 user) it was the first time they created a web application in a simple and intuitive manor.

## **API Editor and Discovery**

The entire focus group respondents enjoyed annotating web API using SerPro API editor, as it was configured as a Google chrome plugin; they found it well designed in terms of extracting HTML information to a well-defined RDF form that is populated in the Mashup editor. Although they were fascinated by SerPro API editor the use of click-button was error-prone to them, as they were arguing on what if a certain users highlights wrong items or inappropriate web content and add it as API property (i.e., name, method, etc..) since most of them were not aware of REST style architecture. Furthermore (4 of 6) respondents recommended the use of a smart and intelligent discovery of web APIs. Moreover, they insisted on providing the platform in other execution environment such as mobile devices, since they believed that such initiative can enable a wide adoption of Mashups.

### **7.4.2 Findings**

This subsection interprets and summarizes the results obtained from the interview processes and focus group session. In general, respondents were satisfied with their experience in creating Mashups (as a results 73% of them were satisfied with how the Mashup Tool was designed), and they were eager to share their observations. As the main research hypothesis has stated that a semantically enriched Mashup tool that can compose, discover and annotate publicly available web APIs, can actually improve the usability of SOA services for novice users. The acceptability tests conducted showed that the proposed system is acceptable to novice users, which served the notion that: if the novice users can use a system then all types of users can.

This section discussed the acceptability of the proposed Mashup tool. Two methods were used: interview and focus group. The aim of using interview was to get the quantitative measure on the system acceptance while focus group was used to obtain the users attitudes towards the system components. This section showed that the proposed Mashup tool was

acceptable to novice users although they were not satisfied with the design of the other components (such as discovery component).

The next section presents the comparison of Mashup tools.

## 7.5 Comparison of Mashup tools

This section presents a comparison study of the proposed Mashup tools together with the existing Mashup tools (discussed and summarised in Appendix B). This section presents the evaluation of results presented in the aforementioned tests. The comparison is conducted using the usability metrics formulated in (Chapter 4), also referred to as system usability non-functional requirements and the SUS scores. Thereafter an evaluation of functional requirement is also discussed.

### 7.5.1 Research Usability Attributes

This section presents the comparison study of Mashup tool based on the formulated research usability attributes namely: familiarity, workflow-visibility, customizable-design, error-handling and system documentation. Table 7-4 presents a comparative table based on the analyses conducted amongst various Mashup tools together with the proposed Mashup tool and the formulated usability attributes presented in (Chapter 4). In Table 7-4 below “X” signifies that the Mashup tool supports a usability attribute and “-” means that the Mashup tool does not supports a usability attribute.

**Table 7-4.Comparison Analysis**

	Familiar	Workflow-visibility	Customizable-design	Error handling	System-documentation
<b>Microsoft Popfly</b>	X	-	X	X	-
<b>Yahoo Pipes</b>	-	-	X	X	-
<b>Intel Mashmaker</b>	X	-	X	-	X
<b>MobiMash</b>	X	-	-	-	X
<b>Marmite</b>	-	X	X	X	X
<b>Dapper</b>	X	X	X	-	X
<b>SerPro</b>	X	X	-	X	X

According to literature survey in (Section 4.3.2.1), Table 7-4 presents the usability attributes that are the key determining factors of usability in Mashup tools, and each attribute is discussed in the following subsections.

#### **7.5.1.1 Familiarity**

The proposed Mashup tool was least surprising since the users understood the wiring of the different widgets, and enjoyed using the proposed to an extent they thought of it as game. The drag and drop user interface functionality was enabled, and users were able to drag items from the JSON tree view to the widgets (as shown in Figure 6-8). Its user interface was easy to use and unnecessary components were not provided in the Mashup editor. Similarly, Dapper and Microsoft Popfly are familiar, since they were form-based and widget-based, respectively. Moreover Dapper used a form wizard to create Mashups. Although Yahoo! Pipes and Marmite were widget-based and form-abased, respectively; they were not familiar. In Yahoo! Pipes the dragging and dropping of widgets is appealing but when it comes to the process of creating Mashups, users found it unnecessary complex; since users had to fill in URLs before they could execute the Mashup applications.

Furthermore, Intel MashMaker and MobiMash are familiar to users although Intel MashMaker is browser extension and MobiMash is template-based. Moreover users found MobiMash to be familiar since it was a mobile based Mashup and since mobile devices and smartphones are gaining wide popularity around the world. While Intel Mash Maker gathered user preference based on the site users visit in order to extract and compose services, the use of the “already gathered info” looked appealing to users.

#### **7.5.1.2 Workflow-visibility**

The visibility of task being executed was provided in the Mashup editor. Users are able to see all the tasks that are currently being executed. Moreover, a GUI web interface to add services and annotate service was provided using SAE and Sesame server, where users were able to see the results of annotated web API documentation form plain HTML documentation. Similarly, Dapper and Marmite provided the workflow, since it displayed to users all the tasks that were executed, although some of the tasks were hidden, and also since they have form-based fields and buttons; every task executed was visible to the users, as they knew where button clicks would direct them.

Whilst Yahoo Pipes and Microsoft Popfly did not provide a sequence of tasks being executed and only displayed visual elements that represent its components - users enjoyed using them. Furthermore, Intel MashMaker was not visible since even the manner it created Mashup applications was based on user preferences that is usually derived from the users web browser history, users only see buttons to execute the Mashup without knowing the design of the

underlying Mashup application. Lastly, MobiMash was not visible because of its template-based nature; the composition process did not provide sequence of tasks being executed.

#### **7.5.1.3 Customizable-design**

A partial technique to make the system customizable was included in design of the proposed Mashup tool, moreover users were able to add their own components, and ambiguous components were handled. However the users found the proposed tool not customizable since it was unnecessary hard for them to discover and annotate web API, as a results in question 3 and 4 of the interview (in Section 7.4.1.1), 60% user could not discover web APIs and 43% of the respondents could not annotate web API. While Dapper and Microsoft Popfly are customizable since the Mashups created using them can be crafted to other platforms, for instance, Mashup applications created using Dapper could be used in the Netvibes and iGoogle. Moreover, Microsoft Popfly is customizable since it is also able to use Mashup created using other Mashup tools as its components.

Similarly, Yahoo! Pipes is customizable since it could use variety of web resources such as feeds (e.g. ATOM, RSS) and web sites, although it is not compatible with Mashup applications created using other Mashup tool. Furthermore, Marmite and Intel MashMaker are customizable in the sense that users were able to do different operations since their process of creating Mashups was semi-automated and browser based. Moreover they relied on the community defined extractors to extract data from various sources (i.e., structured and unstructured). Lastly, MobiMash is not customizable, since its templates are predefined, which reduced it dynamic adaptability of data from different web contents.

#### **7.5.1.4 Error-handling**

A minimally error-prone and error handling design was addressed in the proposed Mashup tool and errors were reduced. In case an error took place, it is handled peacefully by alerting the users that an error has occurred, and what the cause of the error was (as shown in Figure 6-8). In Dapper and Marmite there was no error handling technique employed, since it was less error-prone according to users. Similarly, Microsoft Popfly employed error-handling mechanism since users did not have to fill in anything rather they were selecting, dragging and dropping, and wiring blocks. Moreover, Yahoo Pipes had a mechanism of handling exception. When an error occurred, it provided a feedback to the user. Even during the piping of widgets, it alerted users of compatible and incompatible widgets when connecting them. In Intel MashMaker no error handling mechanism was employed, and even if an error handling

mechanism was employed, it is unlikely that users could make errors as it is browser-based extension.

#### **7.5.1.5 System-documentation**

Documentations deals with how long a user's is required to read before using the proposed Mashup tool. During the evaluation trainings, respondents were able to grab the sequence of steps needed to complete a task. Therefore, the proposed tool was easy to learn. Moreover, Dapper provided documentation, since the users were guided throughout the process of creating Mashups without having to read a tutorial. Similarly, Marmite, MobiMash and Intel MashMaker were documented with short and brief instruction since users had to do little to get going. Whilst Yahoo! Pipes provided the documentations, respondents were still confused; as a result, it was useless to users (not because of the functionality offered rather the complexity that was involved). This is in contrast to Microsoft Popfly that did not provide documentation but users found it easy to use.

This subsection discussed the comparison of Mashup tools based on the usability attributes formulated in this research; the next subsection presents a comparison of SUS scores.

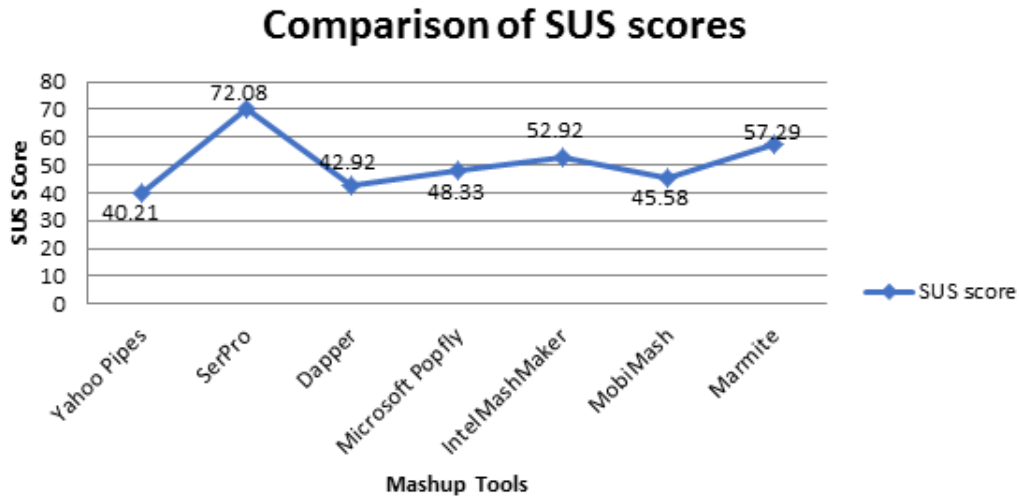
#### **7.5.2 SUS**

The above subsection (Section 7.5.1) discussed the usability dimensions of the research; this section presents a comparison of the studied Mashup tool using tables and graphs. Table 7-2, summarizes the obtained SUS scores in this research:

**Table 7-5.SUS score of Mashup tool**

<b>Mashup tool</b>	<b>SUS score</b>
Microsoft Popfly	48.33
Yahoo Pipes	40.21
Intel Mashmaker	52.92
MobiMash	45.58
Marmite	57.29
Dapper	42.92
SerPro	72.08

Table 7-2 depicts that the proposed Mashup tool scored a greater SUS than its competitors, since its design and was adopted from its counterparts. As it has already been outlined that the minimum threshold SUS score= 68 (the SUS scores for the mentioned tool can be found in Appendix B), Figure 7-8 presents a line graph of SUS scores:



**Figure 7-8. Mashup Tool Comparison**

From Figure 7-8 it can be noted that the browser based extension (i.e., Intel Mashmaker and Marmite) scored a great SUS as compared to widget based (Microsoft Popfly: 48.33, MobiMash: 45.58 and Yahoo Pipes: 40.21) and form based (Dapper: 42.95) as categorized in (Section 4.3.2.1). This is because browser extension Mashup Tools were less error-prone, as compared to widget based where users encountered a lot of errors especially Yahoo! Pipes. The proposed Mashup tool scored a great score as compared to its counterparts, since different prototypes were developed and a SUS score of 60.83 was obtained for the second prototype while a SUS score of 72.08 was obtained for the final prototype. The proposed tool was also widget based, but why did the users enjoy it as compared to the other ones? This question is answered by the following paragraph.

It can be noted that between the SUS scores and research variables there was insignificant correlation. Some of the Mashup tools posed more than one research variable but their SUS scores were low (e.g., Dapper, posed 4 research variables but SUS score was 42.92). It can be concluded that the ease of use in a Mashup tool depended on its interaction technique. Furthermore, the proposed tool was familiar to respondents as Microsoft Popfly and Yahoo! Pipes inspired it; the researchers adapted some of the design techniques from them. For instance, the use of select boxes in the proposed Mashup tool was adapted from Microsoft Popfly, moreover the drag and drop functionality of widgets was adapted from Yahoo! Pipes. Therefore, it was expected for the proposed tool to be usable since before it was designed a pre-evaluation study was conducted to identify strengths and weaknesses encountered by the currently existing Mashup tools. The next sections further discuss the research findings.



## **7.6 Discussion**

As the research hypothesis stated that a semantically enabled Mashup tool that can compose, annotate and discover web APIs can improve the usability of SOA services for novice users, the research is discussed using two sections: composition style, and semantic web technologies, to interpret and determine effectiveness of the system components.

### **7.6.1 Composition Style**

This section discusses how the users perceived the composition layer. The respondents enjoyed the drag & drop functionality, particularly the information on what to see and do. Several respondents as shown in (Section 7.4.1.1) liked the piping of widgets and the visibility of the response, which was displayed in select boxes. Furthermore, 67 % of the respondents were able to create their Mashup applications. The composition layer was easy and appealing to users, because of the widget design and response handling mechanism. Moreover, the users liked the display of pre-defined APIs as shown in left side of Figure 6-8.

### **7.6.2 Semantic Web Technologies**

In order to discover or semantically annotate web APIs most users needed assistance. Half of the observed users were not aware of what Mashups are and what their purpose is. According to users behaviour the Mashup discovery process was inconsistent. The results showed that most users did not prefer the technique used to discover APIs in the web, as they were saying it was not easy to use. Even if the users showed that it can be quick and easy to learn Mashup, they were so frustrated since they said it was unnecessary hard to use them. Although it was simple to add APIs to the Mashup tool, 60% of the users encountered errors and were complaining that it was time consuming to discover and annotate web APIs. As a result, they preferred using programmable web to discover web APIs since it provides them with all the API details needed in order to add APIs to Mashup editor, rather than using search engines. This is because the annotation and discovery layer were designed with novice users being the central focus, and that might have limited the functionality and resulted in complexity. Furthermore, the discovery and annotation of web resources is still a major challenge that affects web developers, especially when the targeted group are non-technical users. The research limitations and challenges are discussed in (Section 8.4).

### **7.6.3 Usability of SOA Services**

The main aim of this research is to enhance the usability of SOA services for novice users, this section outline the key factor of Mashup tools usability. As this research employs

Mashups as the transport layer to enhance the usability of SOA services for novice users, the findings in (Section 7.5) revealed the following attributes as the most influential to the system usability:

- i. Customizable: transparency and visibility were the key requirements observed. Allowing users to interchange Mashup applications greatly improves usability.
- ii. Familiarity: deals with consistency of the Mashup tools from one Mashup tool to the other. Since Mashups are web application, they should be adopt the similar style even though they create a composite applications.
- iii. Documentation: having a easy to learn Mashup documentation and less complex process during the composition of Mashup tool.

Furthermore, according to the findings from the focus group and interviews (in Section 7.4), participants pointed programming skills as the key determinant to the creation of the Mashups. While participants had their different values and behaviours towards system components, the collected data also suggested the explored factor that leads usability of Mashup tools were the following:

- i. Discovery: which deals with how the users discover APIs.
- ii. Annotation: discovering APIs is not enough without given them a meaning and a common understanding, especially REST services. Allowing API to have a common understanding amongst each other can result in a smooth integartion of services.
- iii. Composition style: the composition style used like widgets and use of select box inside widgets, was also key to the succes of Mashups

Even though the proposed Mashup tool did not address some usability attributes such as customizability, it scored a SUS score of 72.08, which is greater than threshold score to regard a system as usable to user. One of the challenge that made the proposed Mashup tool not usable, was the discovering technique used. It was manual and needed technically equipped users. The focus group was conducted using respondents from the Computer Science department (i.e., users with technical skills) and the results showed that the Mashup Editor was usable but the discovery layer and annotation method used was unnecessary complex and error-prone according to their response. They recommended an intelligent discovering tool that will hide the complex of semantic web technologies at the same facilitate ease of use. Does the proposed Mashup tool enhance the usability of SOA services? This question is answered by the following paragraph.

This research enabled novice users to utilize SOA services (with a particular focus on REST services) via the proposed Mashup tool. The key role played by the Mashup tool was to hide the complexity of SOA services so that, novice users are able to utilize SOA services with ease. That allowed novice users to use SOA services without worrying about the underlying technicalities. In the future, SOA services such as (SOAP) will be included in the proposed tool, to enhance the use of Mashups in the global scale of SOA.

Overall the findings signify that the proposed Mashup tool was usable to novice users and enhanced the usability of SOA services, since it was efficient, effective, less error-prone, easy to learn and engaging with the users as argued by Quesenbery (2003), in the literature survey (Section 2.10.2 ), although challenges and limitation were encountered.

This section discussed and interpreted the results obtained from usability, acceptance testing, and comparison study of the proposed Mashup tool. The next section summarizes this chapter.

## **7.7 Conclusion**

This section presented the testing processes undertaken to validate the proposed system. The usability and acceptance test were conducted, with the purpose of validating whether the different components mentioned in the design architecture and implementation adheres to the research goals. The comparison study was conducted to compare the proposed Mashup tool to its counterparts. Focus group was used to gather user's preferences and behaviour towards the Mashup tools. Furthermore, the interviews were conducted to test on whether the users accept the proposed tool or not. Lastly based on different methods used to evaluate the proposed Mashup tool, the results obtained were analysed, interpreted and discussed. The next section presents research summary, research challenges and the conclusion of this research.

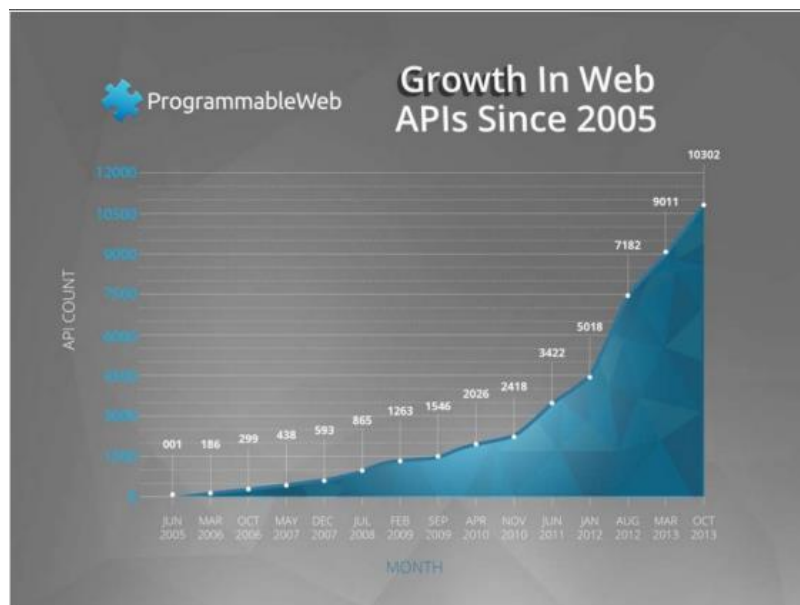
## 8 Chapter 8: Research Conclusion

### 8.1 Introduction

This chapter discusses and concludes the usability of Mashups for novice users. Furthermore, it reviews some of the limitations and problems that were encountered during research and discusses the research achievements and future work. Moreover, it presents the overall conclusion about the research. The next section discusses the use of Mashup and Restful services, as compared to their alternatives.

### 8.2 Research Summary

*Why the research has chosen Mashups as the key enablers of service provisioning?* Mashup tools are explicitly designed for sharing and reusing. Mashup tools are useful for diverse areas and can support a great variety of input and output data types, from structured XML and semantically rich RDF. These tools in general have intuitive designs and easy-to-use web interfaces that require little training for beginners, and creating web applications using them is easier than writing a code in a particular language. Following is the diagram showing how APIs have emerged in the past 8 years.



**Figure 8-1. Growth of web APIs (ProgrammableWeb, 2014)**

With plethora of web APIs, *why the research has chosen Restful web services over SOAP and others?* Web services are there to give web developers the ability to create the best web-based applications in the shortest amount of time. There are two approaches for interfacing to the web with web services, namely SOAP and REST. Both approaches work, both have

advantages and disadvantages to interfacing to web services, but it is up to the web developer to make the decision of which approach may be best for each particular case. One of the benefits of using RESTs in the research was due to its ability of exposing public API over the internet and the handling CRUD operations on data. REST is focused on accessing named resources through a single consistent interface. SOAP brings its own protocol and focuses on exposing pieces of application logic (not data) as services. SOAP exposes operations. SOAP is focused on accessing named operations, each implement some business logic through different interfaces. REST provides true web services based on URIs and HTTP methods, but both technologies can be mixed. REST is very easy to understand although it lacks the standards for documenting its services (restful services.).

Since REST uses standard HTTP, it is much simpler in just about every way. Creating clients, developing APIs, the documentation is much easier to understand and there much that REST does not do easier/ better than SOAP. REST permits many different data formats whereas SOAP only permits XML. While this may seem like it adds complexity to REST because there is a need to handle multiple formats, from this research point of view REST has actually been quite beneficial since JSON usually is a better fit for data and parses it much faster. REST allows better support for browser clients due to its support for JSON. REST reads can be cached, SOAP based reads cannot be cached. For instance, Yahoo uses REST for all their services including Flickr. Amazon and EBay provide both though Amazon's internal usage is nearly all REST. Google used to provide only SOAP for all their services, but in 2006, they deprecated in favour of REST. As shown above, each technology approach has their uses. They both have underlying issues around security, transport layers, and the like, but they both can get the job done and in many cases, they each bring something to the web.

*Why incorporating SOA and SWS technology in the proposed tool?* As SOA, advocates the development of complex distributed applications based on the reuse and composition of existing functionality offered as services. Essential to this vision are the publishing and discovery of services (i.e. Restful services). Number of repositories has been created to this but they have failed to provide suitable support for publishing and querying services in expressive and extensible manner. The web of data had proposed a set of principles that actually represent the current best practices for publishing data on the web in a machine processable manner. IServe builds upon these principles and uses them as its core what we refer to as the Minimal Service Model, a minimal vocabulary for describing services in RDF, which abstracts away the original approach used for annotating the services, such as

SAWSDL, WSMO-Lite and MicroWSMO. The Minimal Service Model provides a simple common ground for describing both WSDL services and Web APIs in RDF in a way such that developers for simple service matchmaking based on SPARQL can directly use it.

In further realizing the vision and the main aim of this research, developers design less complex and less programming skills demanding tools. Moreover giving the services available online back to novice users in different parts of the country can improve the socio-economic development in South Africa, Africa and the world as the whole. The next section present on how the research objectives were accomplished.

### 8.3 Addressing Research Objectives

This section reflects on the research and outlines answers to the questions addressed in this research. The main research question was how a usable Mashup tool that targets novice users can be developed. The research was completed using the following main research hypothesis: **Semantically enriched web tool to discover transform and compose web contents of public APIs (disparate service) can improve usability of SOA service for novice users.** With the main research question being stated, the following sections addresses how each sub research questions together with sub research objectives was accomplished.

*What impedes the usability of Mashup tools?* **This research question was answered by the identification of usability shortfalls of the currently existing Mashup tools.** Different tools were compared in terms of their design, visualization and composition model. The design referred to their architecture style, for instance client architecture or server architecture. The visualization comprised of how the data was presented to end-users. Lastly, composition model entailed workflow processes and how the composition was done. From the pre-evaluation analysis model in (Section 4.3.2.1.), the findings showed that the users were not happy with the user interface design of the existing Mashup tools. Six usability metrics were formulated using the observed weaknesses of the current Mashup tool. Based on the mentioned issues the first research question was answered. Research method like SUS and focus group were used. The next question discusses how the identified issues were addressed in the design of the proposed Mashup tool.

*What are the different techniques for Mashup tool service composition?* **This research question was answered by investigating service composition using Graphical Users Interface.** Using the comparative analysis of the existing Mashup tools (Section 4.3.2.1.), it was realised that most of the tools used widgets as presentation modules to allow user to

remix web contents. The review of literature showed that there were three techniques used when integrating web contents (Latih et al., 2011) namely: Visual Mashup editor, Browser extension application and Web portal Mashup tools. A visual Mashup editor is a tool that lets users create a Mashup application by manipulating program elements graphically rather than by specifying them textually, for instance Yahoo Pipes let users to develop Mashup applications via a visual editor by selecting a few modules and wire them together. This research adopted the visual Mashup editor since novice users find these tools easy if they are using graphics. Visual Mashup tool are the easiest type of tools and they involve users throughout the process of service composition. This research question was answered through the design and the implementation of the Mashup tool, which was presented in (Chapter 5 and Chapter 6), respectively. In the proposed architecture different modules were encompassed to attain a usable Mashup tool (as discussed in Section 5.1). Lastly, design principles were also used as guidelines to attain a simple and intuitive Mashup tool.

As stated in the previous research question that most tools used widgets. The integration of services contained web contents from different web resources to create new services according to user's requests, by refining and combining only the necessary data. This process depends on the support offered by Mashup tool, some Mashup tools are automated and some are semi-automated. Literature shows that semi-automated composition was developed to support End User Development of Mashup applications. EUD paradigms provide programming capabilities for everyone by pushing on different aspects of computing technologies. One of the EUD paradigms employed was wiring or dataflow. Dataflow paradigm is where several selected widgets that support particular functions are connect together for example in Yahoo Pipes. Each widget represents a particular function, users just select it for use and they get response for example tools like iGoogle and Netvibes. The use of widgets was appealing to users as the results showed from the focus group in (Section 7.4.1.2.)

This research question was answered by incorporating semi-automatic manner in the proposed tool, since it supported widgets, which were used in the Mashup editor for designing Mashup applications.

*How can annotation and discovery of APIs be implemented, to achieve a smooth composition? This question was answered by investigating a semantic technique to match, select and annotate services.* Most of the approaches that have been used to annotate services resulted in models that were more complex, when using artificial branch to select

services that better fit a certain context. The use of SWS mitigates such challenges by providing semantic annotations of services to support the application of automated machinery to be able to reason about the functional and non-functional characteristics of services. Literature mentioned semantic annotation of services, which deals with universal description amongst different services can result in a smooth integration of services. Service composition using SWS included the following steps namely: service description, service discovery, service publication and service invocation, as discussed in (Section 2.3.1). These steps can be summarized into selection and matching of appropriate services. The semantic approach of the proposed tool is referred to as the homogenization layer and is presented in (Chapter 3). The SerPro API editor was designed and implemented as discussed in (Section 5.3.). SerPro API editor was used as a semantically annotating tool and Sesame repository was used as a local discovery repository for the proposed tool. In (Chapter 3) different service description, selection, and matching approaches were discussed, as a result, the API editor used SPARQL to select and match services in the local Sesame server. The ontology to semantically annotate REST web APIs was defined and created, and it can be found in (Appendix C).

*How can semantic web technologies be utilized to improve usability of Mashups?* **This question was answered by the implementation of API Editor and evaluating its effectiveness.** After integrating the API editor to the proposed tool there was an improving although the composition process of the proposed tool was not that smooth, since the discovery process was still manual and complex to novice users, nevertheless the SUS score increased from 60.83 to 72.08 as discussed in (Section 5.1). Furthermore in (Section 7.4.1.1.), interview question number two (i.e., Q2), showed that 60% of the user could not discover REST web APIs whilst 40% of the users were able to discover API, this signifies that the discovery of web APIS was a key component in determining the usability of Mashups. Moreover interview question number three (i.e., Q3), showed that 57% of users were happy with semantic annotation process and only 43% of users found the annotation tool unnecessary complex.

In (Section 3.5), the semantic annotation tools requirements were reviewed from the literature namely; usability, interoperability, annotation storages and support offered for annotation process. This research semantic annotation tools (SAE) was interoperable in the sense that it used RDF and Owl, which are semantic ontologies that construct data and services that are machine-readable. The storage for annotated document is Sesame server, which is triple store database that can store and query RDF data. Furthermore, the SAE offered manual support



for the process of annotating API documentation. As the previous paragraph outlined that 57% of the users enjoyed the semantic annotation tool, although there were challenges encountered. This question was answered and showed that semantic web technologies to discover and annotate web APIs can improve usability of SOA services for novice users, on the bases that they are autonomous and support EUD.

The next section addresses some of the limitations and challenges encountered during the research process.

#### **8.4 Research Limitations and Challenges**

A partial method to address the discovering and publishing web APIs was used and its limitations and challenges are discussed. Furthermore, the rights or permission granted for external users for the iServe and SWEET external applications were limited. As a results annotating and discovering web APIs was done manually. The following were the challenges encountered:

- i. Due to time constraints learning (SWS) is the project on its own, because there is no standard way of adding APIs, errors may arise.
- ii. Constructing requests from API base URLs was a limitation since there was no mechanism to keep track of the changes made to API documentation (i.e., there was lack of a mechanism to link semantically annotated APIs documentation to original API documentation). Security policies some site don't allow displaying their APIs inside frame.
- iii. Measuring usability also depends upon the users or the respondents used during the evaluation process, the chosen respondents may have effect on the obtained results. Furthermore the traing given to users, might have been limited as compared to their mental capabilities.
- iv. The research mostly employed user oriented evaluation methods (as it was trying to target novice users). Not exploring other usability testing methods (such as Keystroke Level Model) may have effect.

Mostly the usability was prioritized over other system testing methods, in the near future performance and bandwidth tests might be taken into consideration. Testing the proposed Mashup tool in different types of novice users was a challenge as some of novice users were not aware of Mashups, so this might have limited the research findings in terms of the system usability.

## **8.5 Research Contribution and Recommendation**

The research provided a platform for provisioning service in ICT context. The Mashup editor produced showed that novice users are able to create their own Mashup application with easy and flexibility. Primarily APIs interfaces are designed and developed for programmers or advanced users (i.e. users sophisticated with computer science skills), with the aim of allowing web developers to utilize the already developed services. The web developer's design what application logics to include in APIs and they do that differently especially REST APIS, and that makes it difficult for machine users to use them since it is difficult generalised the pattern in which they are deigned. In this research, a partial general pattern (or standard) was designed and used to discover and annotate REST web APIs. Moreover, a Mashup Editor that encapsulated API details inside widgets was developed and enhanced the usability of SOA services for novice user.

With the rise of web 2.0 application, such Mashup, being applications that allow novice users to share and communicate information on the web freely, demands are mounting to a level that leaves web developers confused. All that is left to be done from the present moment is to design and implement web tools that are customizable; in the sense that users are able to use each other web applications and services. While services created could be accessed and executed in different environments (for instance mobile phone and sharing amongst the already existing Mashup Tools). Currently, the studies and tests conducted in this research showed that not only novice users are not able to use the existing Mashup tools even the advanced users with programming skills that reside in other domain within computer science were not able to use them.

Furthermore, because novice users are aware of platforms and technologies used to create online services, there is huge demand for developers to not only target developer when developing APIs but to include designs that makes it easy for novice users to utilize their services.

## **8.6 Future Work**

The proposed Mashup tool enabled novice users to create their situational web applications, which served a good purpose since the main aim of this research was to enable novice users to create Mashups with no coding skills being required. Nevertheless, some of the users encountered challenges when it comes to annotate and discover web APIs. As automated annotation, tools to discover and select APIs are needed to inspire and attract wide audience

in Mashup tools for novice users. For the future work, an investigation to extend the API editor to include automated features for extracting and semantically annotating APIs will be conducted.

The next section discusses the overall conclusion of the research.

## **8.7 Conclusion**

The different service provisioning techniques were reviewed. The research problem together with research question and objectives were addressed. Solution to the research hypothesis were formed, structured and analysed. Different prototypes (i.e., 3 prototypes) were developed to achieve the research goal. A research philosophy was used together with research methods as the technique of the methodology. This research was interpretive, in the sense that it interpreted the already working Mashup tools by identifying their shortfalls and later those weaknesses were addressed to the proposed tool. The shortfalls assisted in the design and implementation of usable tool such as the proposed one. The results showed that the proposed tool was usable and acceptable to novice users. Furthermore, the proposed tool does not require any skill of coding. The main research question was achieved through its sub-questions, the proposed Mashup tool was usable to novice users, and that enhanced the usability of SOA services for novice users.

The research showed that the usability of any web tool has to be viewed in terms of the context in which it is used, and its appropriateness to that context. In general, it is impossible to specify the usability of a system (i.e., its fitness for purpose) without first defining who are the intended users of the system, the tasks those users will perform with it, and the characteristics of the physical, organisational and social environment in which it will be used. This research shows that well designed Mashup tool are usable to novice users. And that the utilization of SOA services could be increased through Mashup tools for novice users, since they hide complexity of APIs and focus on GUI elements to represent services. The overall findings suggested that Mashup developer need to develop web tools that are usable, simple and appealing to novice users in order to create web application (i.e., composite applications) where there is a minimal development of web application.

## **Part V: Reference Material**

## References

- Abras, C., Maloney-Krichmar, D., & Preece, J. (2004). User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, 37(4), 445-56.
- Aghaee, S., & Pautasso, C. (2010). Mashup development with HTML5. In *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups* (p. 10). ACM.
- Ambite, J. L., Darbha, S., Goel, A., Knoblock, C. A., Lerman, K., Parundekar, R., & Russ, T. (2009). *Automatically constructing semantic web services from online sources* (pp. 17-32). Springer Berlin Heidelberg.
- Anttonen, M., Salminen, A., Mikkonen, T., & Taivalsaari, A. (2011). Transforming the web into a real application platform: new technologies, emerging trends and missing pieces. In *Proceedings of the 2011 ACM Symposium on Applied Computing* (pp. 800-807). ACM.
- Barbara F. (2005). ICT4D - Information and Communication Technologies for Development. Available at [http://ictlogy.net/lo/01001/ict4d\\_course\\_barbara\\_fillip\\_at\\_courses.ictlogy.net.pdf](http://ictlogy.net/lo/01001/ict4d_course_barbara_fillip_at_courses.ictlogy.net.pdf) [Accessed 13 February 2014].
- Batard, F., Boudaoud, K., & Riveill, M. (2011). A middleware for securing mobile mashups. In *Proceedings of the 20th international conference companion on World wide web* (pp. 9-10). ACM.
- Benhaddi, M., Baïna, K., & Abdelwahed, E. H. (2012). A user-centric Mashuped SOA. *International Journal of Web Science*, 1(3), 204-223.
- Benjamins, R., Contreras, J., Corcho, O., & Gomez-Perez, A. (2002). The six challenges of the Semantic Web. Available at <http://oa.upm.es/5668/1/Workshop06.KRR2002.pdf> [Accessed 20 April 2014].
- Benslimane, D., Dustdar, S., & Sheth, A. (2008). Services Mashups. *The New Generation of Web Applications*, 13-15. Available at [http://www.infosys.tuwien.ac.at/staff/sd/papers/ServicesMashups\\_IC.pdf](http://www.infosys.tuwien.ac.at/staff/sd/papers/ServicesMashups_IC.pdf) [Accessed September 2013].
- Bertot, J. C., Jaeger, P. T., & Grimes, J. M. (2012). Promoting transparency and accountability through ICTs, social media, and collaborative e-government. *Transforming Government: People, Process and Policy*, 6(1), 78-91.
- Blanchard, B. S., Fabrycky, W. J., & Fabrycky, W. J. (1990). *Systems engineering and analysis* (Vol. 4). Englewood Cliffs, New Jersey: Prentice Hall.
- Bolin, M. M. T. (2005). *End-user programming for the web*. Doctoral dissertation. Cambridge: Massachusetts Institute of Technology.
- Botta, R., & Bahill, R. (2008). Fundamental principles of good system design. *Engineering Management Journal. Citeseer*.

- Brambilla, M., Celino, I., Ceri, S., Cerizza, D., Della Valle, E., & Facca, F. M. (2006). A software engineering approach to design and development of semantic web service applications. In *The Semantic Web-ISWC 2006* (pp. 172-186). Springer Berlin Heidelberg.
- Brooke, J. (2013). SUS : A Retrospective. *Journal Of Usability Studies*, 8(2): 29–40.
- Cao, J., Riche, Y., Wiedenbeck, S., Burnett, M., & Grigoreanu, V. (2010). End-user mashup programming: through the design lens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1009-1018). ACM.
- Cappiello, C., Matera, M., Picozzi, M., Caio, A., & Guevara, M. T. (2012). MobiMash: end user development for mobile mashups. In *Proceedings of the 21st international conference companion on World Wide Web* (pp. 473-474). ACM.
- Cetin, S., Altintas, N. I., Oguztuzun, H., Dogru, A. H., Tufekci, O., & Suloglu, S. (2007). A mashup-based strategy for migration to service-oriented computing. In *Pervasive Services, IEEE International Conference on* (pp. 169-172). IEEE.
- Chapin, N. (2010). Software characteristics of SOA projects. In *Proceedings of the 11th International Conference on Product Focused Software* (pp. 152-157). ACM.
- Choi, O., Moon, S. H., Han, S., & Abraham, A. (2006). Intelligent Web Services System based on Matchmaking Algorithm. *WSEAS Transactions on Circuits and Systems*, 5(8), 1166.
- Collins, J., & Hussey, R. (2003). Business Research: a practical guide for undergraduate and postgraduate students. *New York: Paigrave Macmillan*.
- Dang, M. T., Dimitrova, V., & Djemame, K. (2007). Personalised mashups: opportunities and challenges. *User modelling. LNCS*, 4511.
- Daniel, F., Imran, M., Kling, F., Soi, S., Casati, F., & Marchese, M. (2012). Developing domain-specific mashup tools for end users. In *Proceedings of the 21st international conference companion on World Wide Web* (pp. 491-492). ACM.
- Daniel, F., Rodríguez, C., Roy Chowdhury, S., Motahari Nezhad, H. R., & Casati, F. (2012). Discovery and reuse of composition knowledge for assisted mashup development. In *Proceedings of the 21st international conference companion on World Wide Web* (pp. 493-494). ACM.
- Di Fabbri, G., Okken, T., & Wilpon, J. G. (2009). A speech mashup framework for multimodal mobile services. In *Proceedings of the 2009 international conference on Multimodal interfaces* (pp. 71-78). ACM.
- Di Pietro, I., Pagliarecci, F., Spalazzi, L., Marconi, A., & Pistore, M. (2008). Semantic web service selection at the process-level: The ebay/amazon/paypal case study. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on* (Vol. 1, pp. 605-611). IEEE.
- Duke, A., Stincic, S., Davies, J., Rey, G. Á., Pedrinaci, C., Maleshkova, M., & Mehandjiev, N. (2010). Telecommunication mashups using RESTful services. In *Towards a Service-Based Internet* (pp. 124-135). Springer Berlin Heidelberg.

Ennals, R., Brewer, E., Garofalakis, M., Shadle, M., & Gandhi, P. (2007). Intel Mash Maker: join the web. *ACM SIGMOD Record*, 36(4), 27-33.

Feier, C., Polleres, A., Dumitru, R., Domingue, J., Stollberg, M., & Fensel, D. (2005). Towards intelligent web services: The web service modeling ontology (WSMO). Available <http://oro.open.ac.uk/23147/1/> [ Accessed 16 June 2014].

Fisichella, M., & Matera, M. (2011). Process flexibility through customizable activities: A mashup-based approach. In *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on* (pp. 226-231). IEEE.

Fowler, M. (2001). Is design dead?. *SOFTWARE DEVELOPMENT-SAN FRANCISCO-*, 9(4), 42-47.

Freudenstein, P., Buck, J., Nussbaumer, M., & Gaedke, M. (2007). Model-driven Construction of Workflow-based Web Applications with Domain-specific Languages. In *MDWE*.

Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., & Cajander, Å. (2003). Key principles for user-centred systems design. *Behaviour and Information Technology*, 22(6), 397-409.

Hahmann, T.(2005). Workflow / Web Service Composition. Available at <http://www.spatial.maine.edu/~torsten/presentations/WorkflowComposition.pdf> [Accessed 23 August 2013].

Halfond, W. G. (2010). Program analysis to support quality assurance techniques for web applications. unpublished Doctoral dissertation, Atlanta: Georgia Institute of Technology.

Hom, J. (1998). The usability methods toolbox handbook. *Fuente: http://www.idemployee.id.tue.nl/gwm-rauterberg/lecturenotes/UsabilityMethodsToolboxHandbook.pdf* (Consultado el 23-09-12).

Houde, S., & Hill, C. (1997). What do prototypes prototype. *Handbook of human-computer interaction*, 2, 367-381.

Hwang, K. (1979). *Computer arithmetic: principles, architecture and design*. John Wiley & Sons, Inc.

Ioannou, E. (2011). *Entity linkage for heterogeneous, uncertain, and volatile data*. Doctoral dissertation, University of Hanover.

ISO, W. (1998). 9241-11. Ergonomic requirements for office work with visual display terminals (VDTs). *The international organization for standardization*.

John, B. E., & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4), 320-351.

Jones, M. C., & Churchill, E. F. (2009). Conversations in developer communities: a preliminary analysis of the yahoo! pipes community. In *Proceedings of the fourth international conference on Communities and technologies* (pp. 195-204). ACM.

- Kenda, K., Fortuna, C., Moraru, A., Mladenicić, D., Fortuna, B., & Grobelnik, M. (2013). Mashups for the Web of Things. In *Semantic Mashups* (pp. 145-169). Springer Berlin Heidelberg.
- Kopecky, J., & Vitvar, T. (2008). Wsmo-lite: Lowering the semantic web services barrier with modular and light-weight annotations. In *Semantic Computing, 2008 IEEE International Conference on* (pp. 238-244). IEEE.
- Koschmider, A., Torres, V., & Pelechano, V. (2009). Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW* (pp. 1-9).
- Krummenacher, R., Norton, B., Simperl, E., & Pedrinaci, C. (2009). Soa4all: enabling web-scale service economies. In *Semantic Computing, 2009. ICSC'09. IEEE International Conference on* (pp. 535-542). IEEE.
- Küster, U., Lausen, H., & König-Ries, B. (2008). Evaluation of semantic service discovery—a survey and directions for future research. In *Emerging Web Services Technology, Volume II* (pp. 41-58). Birkhäuser Basel.
- Latih, R., Patel, A. & Zin, A.M. (2011). The Design Of Block-Based Mashup Tool For End-Users Mashup Applications Development. *JATIT*, 34(2): 215–225.
- Lee, Y. J. (2013). Algorithm for Automatic Web API Composition. In *WEB 2013, The First International Conference on Building and Exploring Web Based Environments* (pp. 57-62).
- Lin, J., Wong, J., Nichols, J., Cypher, A., & Lau, T. A. (2009). End-user programming of mashups with vegemite. In *Proceedings of the 14th international conference on Intelligent user interfaces* (pp. 97-106). ACM.
- Loton, T. (2008). *Introduction to Microsoft Popfly, No Programming Required*. Lotontech Limited.
- Loutas, N., Peristeras, V., & Tarabanis, K. (2011). Towards a reference service model for the Web of Services. *Data & Knowledge Engineering*, 70(9), 753-774.
- Ly, P. A., Pedrinaci, C., & Domingue, J. (2012). Automated information extraction from Web APIs documentation. In *Web Information Systems Engineering-WISE 2012* (pp. 497-511). Springer Berlin Heidelberg.
- Maleshkova, M., Pedrinaci, C., & Domingue, J. (2009). Supporting the creation of semantic restful service descriptions. Available at <http://oro.open.ac.uk/23106/1/paper6.pdf> [ Accessed 31 March 2014].
- Malki, A., & Benslimane, S. M. (2012). Building Semantic Mashup. In *ICWIT* (pp. 40-49). Available at <http://ceur-ws.org/Vol-867/Paper5.pdf> [Accessed 12 November 2013].
- MapsMania. Available at <http://googlemapsmania.blogspot.com/> [Accessed 5 June 2014].
- Maué, P., Schade, S., & Duchesne, P. (2009). Semantic annotations in OGC standards. Available at



[https://portal.opengeospatial.org/modules/admin/license\\_agreement.php?suppressHeaders=0&access\\_license\\_id=3&target=http://portal.opengeospatial.org/files/%3fartifact\\_id=34916](https://portal.opengeospatial.org/modules/admin/license_agreement.php?suppressHeaders=0&access_license_id=3&target=http://portal.opengeospatial.org/files/%3fartifact_id=34916)[Accessed 16 July 2014].

Maynard, D. (2005). Benchmarking ontology-based annotation tools for the semantic web. In *UK e-Science Programme All Hands Meeting (AHM2005) Workshop on Text Mining, e-Research and Grid-enabled Language Technology, Nottingham, UK*.

Meditskos, G., & Bassiliades, N. (2011). A combinatory framework of Web 2.0 mashup tools, OWL-S and UDDI. *Expert Systems with Applications*, 38(6), 6657-6668.

Mehandjiev, N., & De Angeli, A. (2012). End user mashups: analytical framework. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups* (pp. 36-39). ACM.

Miah, S. J., Debusse, J. C., Gammack, J., & Pigott, D. (2012). A Portal-Based Web Service Development Using a Mashup Approach. In *Proceedings of the Conference on Information Systems Applied Research ISSN* (Vol. 2167, p. 1508).

Momeni, E. (2011). Semantic Web-Based Integration of Heterogeneous Web Resources. In *AAAI Spring Symposium: AI for Business Agility*.

Nestler, T., Feldmann, M., Preussner, A., & Schill, A. (2009). Service composition at the presentation layer using web service annotations. In *Proc. of the 1st Intl. Workshop on Lightweight Integration on the Web (ComposableWeb'09)*.

Ngu, A. H., Carlson, M. P., Sheng, Q. Z., & Paik, H. Y. (2010). Semantic-based mashup of composite applications. *Services Computing, IEEE Transactions on*, 3(1), 2-15.

Nielsen, J. (1994). Usability inspection methods. In *Conference companion on Human factors in computing systems* (pp. 413-414). ACM.

Ogrinz, M. (2009). *Mashup patterns: Designs and examples for the modern enterprise*. Pearson Education.

Olsen, W. (2004). Triangulation in social research: qualitative and quantitative methods can really be mixed. *Developments in sociology*, 20, 103-118.

Onwuegbuzie, A. J., & Leech, N. L. (2010). Generalization practices in qualitative research: a mixed methods case study. *Quality & Quantity*, 44(5), 881-892.

O'Reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications and Strategies*, 65(1), 17-37.

Papadakis, G., Ioannou, E., Niederée, C., & Fankhauser, P. (2011). Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the fourth ACM international conference on Web search and data mining* (pp. 535-544). ACM.

Patton, M. Q., & Cochran, M. (2002). A Guide to Using: Qualitative Research Methodology. *Medicins Sans Frontieres, UK*.

Pedrinaci, C., & Domingue, J. (2010). Web services are dead. long live internet services. *SOA4All White Paper*. Available at <http://soa4all.eu/file-upload.html> [Accessed 26 June 2013].

Pedrinaci, C., Lambert, D., Maleshkova, M., Liu, D., Domingue, J., & Krummenacher, R. (2011). *Adaptive service binding with lightweight semantic web services* (pp. 233-260). Springer Vienna.

Pedrinaci, C., Liu, D., Lin, C., & Domingue, J. (2012). Harnessing the Crowds for Automating the Identification of Web APIs. In *AAAI Spring Symposium: Intelligent Web Services Meet Social Computing*.

Philips, E., Carreton, A. L., Joncheere, N., De Meuter, W., & Jonckers, V. (2010). Orchestrating nomadic mashups using workflows. In *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups* (p. 1). ACM.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T. (1994). *Human-computer interaction*. Addison-Wesley Longman Ltd.

ProgrammableWeb (2014). Fastest Growing Web API Categories. Available at <http://www.programmableweb.com/api-research> Accessed [Accessed 5 June 2013].

Quesenbery, W. (2003). The five dimensions of usability. *Content and complexity: Information design in technical communication*, 81-102.

Rajasekar, S., Philominathan, P., & Chinnathambi, V. (2006). Research methodology. *arXiv preprint physics/0601009*. Available at <http://arxiv.org/pdf/physics/0601009.pdf> [Accessed 23 November 2013].

Rao, J. (2004). Semantic web service composition via logic-based program synthesis. unpublished doctoral dissertation. Norwegian: Norwegian University of Science and Technology.

Sabbouh, M., Higginson, J., Semy, S., & Gagne, D. (2007). Web mashup scripting language. In *Proceedings of the 16th international conference on World Wide Web* (pp. 1305-1306). ACM.

Sabou, M. (2006). Building web service ontologies. *SIKS Dissertation Series*, (2004-4).

Sarraj, weal al. (2012). *A Usability Evaluation Framework for Web Mashup Makers for End-Users*.unpublished doctoral disseration. Belgium: Vrije Universiteit Brussel. Available at [https://wise.vub.ac.be/sites/default/files/theses/Wael\\_Al\\_Sarraj\\_PhD\\_WISE\\_VUB\\_2012.pdf](https://wise.vub.ac.be/sites/default/files/theses/Wael_Al_Sarraj_PhD_WISE_VUB_2012.pdf) [Accessed 19 January 2014].

Saunders, M. N., Saunders, M., Lewis, P., & Thornhill, A. (2011). *Research methods for business students*, 5/e. Pearson Education India.

Sbodio, M. L., Martin, D., & Moulin, C. (2010). Discovering Semantic Web services using SPARQL and intelligent agents. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4), 310-328.

Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S. & Shanbhag, C. (2010). Dapper, a large-scale distributed systems tracing infrastructure. *Google research*.

Soriano, J., Lizcano, D., Hierro, J. J., Reyes, M., Schroth, C., & Janner, T. (2008). Enhancing user-service interaction through a global user-centric approach to SOA. In *Networking and Services, 2008. ICNS 2008. Fourth International Conference on* (pp. 194-203). IEEE.

Soylu, A., Wild, F., Mödritscher, F., Desmet, P., Verlinde, S., & De Causmaecker, P.(2011). Mashups and widget orchestration. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems* (pp. 226-234). ACM.

Spivack N., (1998). *Web Evolution*. Available at <http://www.slideshare.net/novaspivack/web-evolution-nova-spivack-twine> [Accessed 12 October 2013].

Srivastava, B., & Koehler, J. (2003). Web service composition-current solutions and open problems. In *ICAPS 2003 workshop on Planning for Web Services* (Vol. 35, pp. 28-35).

Toleman, M. A. (1996). *The design of the user interface for software development tools* (Doctoral dissertation, University of Queensland).

Tuchinda, R., Szekely, P., & Knoblock, C. A. (2008). Building mashups by example. In *Proceedings of the 13th international conference on Intelligent user interfaces* (pp. 139-148). ACM.

Upadhyaya, B., & Zou, Y. (2012). Integrating heterogeneous web services from an end user perspective. In *Proceedings of the 9th Middleware Doctoral Symposium of the 13th ACM/IFIP/USENIX International Middleware Conference* (p. 7). ACM.

Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., & Ciravegna, F. (2006). Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: science, services and agents on the World Wide Web*, 4(1), 14-28.

Väänänen-Vainio-Mattila, K., & Wäljas, M. (2011). Towards user-centered mashups: exploring user needs for composite web services. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems* (pp. 1327-1332). ACM.

Voss, C., Tsikriktsis, N., & Frohlich, M. (2002). Case research in operations management. *International journal of operations & production management*, 22(2), 195-219.

Weske, M., Vossen, G., & Puhlmann, F. (2006). Workflow and service composition languages. In *Handbook on Architectures of Information Systems*(pp. 369-390). Springer Berlin Heidelberg.

Williams, C. (2011). Research methods. *Journal of Business & Economics Research (JBER)*, 5(3).

Wong, J., & Hong, J. I. (2007). Making mashups with marmite: towards end-user programming for the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1435-1444). ACM.

Xuanzhe, L., Zhao, Q., Huang, G., Jin, Z., & Mei, H. (2010). iMashup: assisting end-user programming for the service-oriented web. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (pp. 285-288). ACM.

Zeshan, F., & Mohamad, R. (2011). Semantic web service composition approaches: Overview and limitations. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 1(3), 640-651.

## Appendix A: Ethical clearance

### ETHICAL CLEARANCE CERTIFICATE REC-270710-028-RA Level 01

Certificate Reference Number: TH1011SYAL01

Project title: **Enabling service composition in ICTD context through GUI Mashup**

Nature of Project: Masters

Principal Researcher: Sabelo Yalezo

Supervisor: Prof M Thinyane

Co-supervisor:

On behalf of the University of Fort Hare's Research Ethics Committee (UREC) I hereby give ethical approval in respect of the undertakings contained in the above-mentioned project and research instrument(s). Should any other instruments be used, these require separate authorization. The Researcher may therefore commence with the research as from the date of this certificate, using the reference number indicated above.

Please note that the UREC must be informed immediately of

- Any material change in the conditions or undertakings mentioned in the document
- Any material breaches of ethical undertakings or events that impact upon the ethical conduct of the research

The Principal Researcher must report to the UREC in the prescribed format, where applicable, annually, and at the end of the project, in respect of ethical compliance.

- Withdraw or amend this Ethical Clearance Certificate if
  - Any unethical principal or practices are revealed or suspected
  - Relevant information has been withheld or misrepresented
  - Regulatory changes of whatsoever nature so require
  - The conditions contained in the Certificate have not been adhered to
- Request access to any information or data at any time during the course or after completion of the project.
- In addition to the need to comply with the highest level of ethical conduct principle investigators must report back annually as an evaluation and monitoring mechanism on the progress being made by the research. Such a report must be sent to the Dean of Research's office

The Ethics Committee wished you well in your research.

Yours sincerely

  
Professor Gideon de Wet  
Dean of Research

06 June 2014

## Appendix B: SUS questionnaire and Mashup Tool SUS scores

### Appendix B.1.SUS questionnaire



University of Fort Hare  
Together in Excellence

## System Usability Scale: Mashup tools

---

I am **Sabelo Yalezo** currently a master student in Computer Science. I am conducting this survey in the fulfilment of my master's degree, under the topic: “**Enabling Service Composition in ICTD context through a GUI Mashup Tool**”. The aim of this SUS is to identify short-falls of the current existing consumer Mashup tools, with a special focus on usability. The survey outcomes will be used to assist in planning and designing usable Mashup tool, as well as to evaluate the effectiveness of the prospective Mashup tool. The following ethical issues were taken into consideration:

- Users participation is completely voluntary.
- We do not harm and embarrass respondents; hence they must feel free about answering questions.
- The collected data is confidential and respondents cannot be identified on the basis of a response (i.e. anonymity).

Participants may leave at any particular stage if they may feel uncomfortable, but their presence is of great importance towards our study.

Supervisor:

Prof M Thinyane

*Telkom Centre of Excellence in ICT for Development,*

*Department of Computer Science, University Of Fort Hare,*

*Private Bagx1314, Alice*

**Key:** 1= Strongly-disagree, 2= Disagree, 3= In-between, 4= Agree, 5=Strongly-disagree

- 1) I think that I would like to use this Mashup tool frequently
 

1	2	3	4	5
- 2) I found the Mashup tool unnecessarily complex
 

1	2	3	4	5
- 3) I thought the Mashup tool was easy to use
 

1	2	3	4	5
- 4) I think that I would need the support of a technical person to be able to use this Mashup tool
 

1	2	3	4	5
- 5) I found the various functions in this Mashup tool were well integrated
 

1	2	3	4	5
- 6) I thought there was too much inconsistency in this Mashup tool
 

1	2	3	4	5
- 7) I would imagine that most people would learn to use this Mashup tool very quickly
 

1	2	3	4	5
- 8) I found the Mashup tool very cumbersome to use
 

1	2	3	4	5
- 9) I felt very confident using the Mashup tool
 

1	2	3	4	5
- 10) I needed to learn a lot of things before i could get going
 

1	2	3	4	5

Tool: [pipes.yahoo.com](http://pipes.yahoo.com)

Tool: [open.dapper.net](http://open.dapper.net)

## Appendix B.2.Review of Mashup Tools

The SUS scale is generally used after the respondent has had an opportunity to use the system being evaluated, but before any debriefing or discussion takes place (Brooke, 2013). Respondents should be asked to record their immediate response to each item, rather than thinking about items for a long time. All items should be checked. If a respondent feels that they cannot respond to a particular item, they should mark the centre point of the scale.

- **Scoring SUS**

Note that scores for individual items are not meaningful on their own. To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1, 3, 5, 7, and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU. SUS scores have a range of 0 to 100.

### Microsoft Popfly

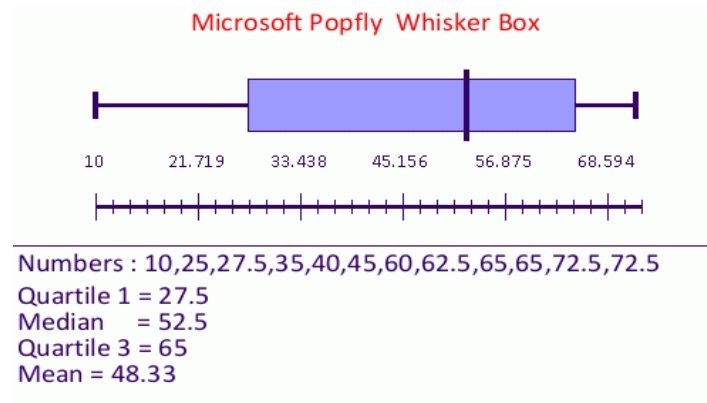
In Popfly, users build Mashups using basic programming constructs called *blocks*. Each block performs a set of *operations* such as data retrieval and data display. Each operation takes *input* parameters to allow customization. Blocks are connected to form a network in which the output of a block can be used as input for adjacent blocks. In Popfly, blocks are listed in different categories, which users can search. Additionally, users may share their Mashups with others for reuse and modifications. Shared Mashups can be retrieved using a textual search. Microsoft Popfly was show in Figure 2-1.

**Table B-1.Microsoft Popfly SUS scores**

Users	Score	SUS score=score*2.5
1	25	62.5
2	29	72.5
3	14	35
4	26	65
5	16	40
6	26	65
7	29	72.5
8	11	27.5
9	18	45
10	10	25
11	4	10
12	24	60



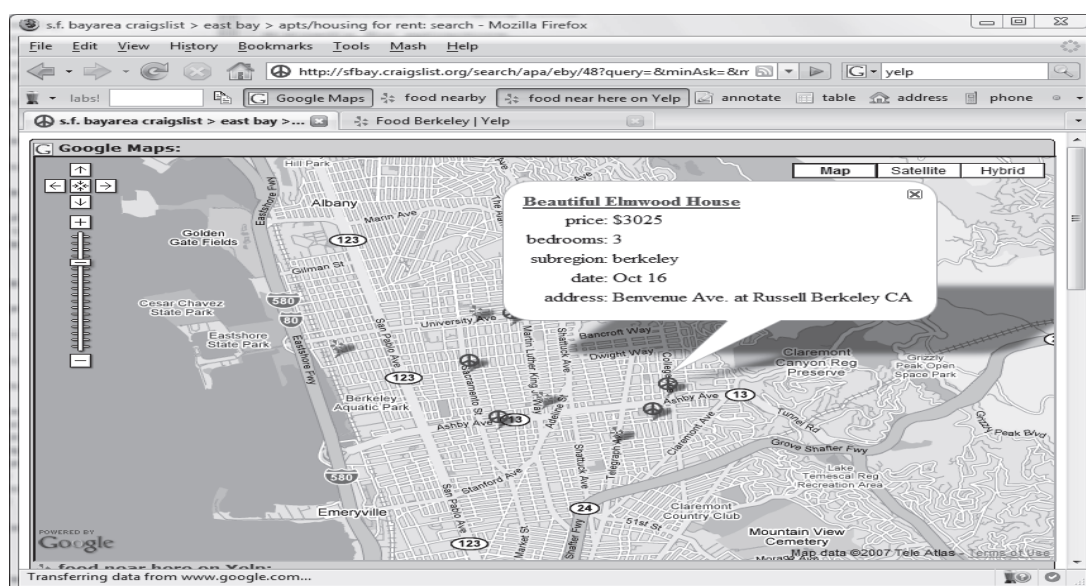
Average	48.33
---------	-------



**Figure B-1. Microsoft Popfly whisker box**

## Intel MashMaker

Intel® Mash Maker is an interactive tool that tracks what the user is doing and tries to infer what information and visualizations they might find useful for their current task (Ennals et al., 2007). Mash Maker uses structured data from existing web sites to create new “mashed up” interfaces combining information from many sources. The Intel® Mash Maker client is currently implemented as an extension to the Firefox web browser. Mash Maker adds a toolbar to the browser that shows buttons representing enhancements that Mash Maker believes the user might want to apply to the current page. An enhancement might combine the data on the page with data from another source, or visualize data in a new way.

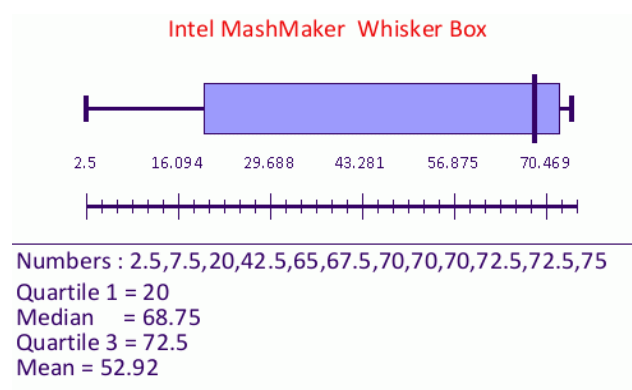


**Figure B-2. Intel mash maker house rentals (Ennals & Shadle 2007)**

Composition: Mash Maker is intended to be an integral part of the way the user browses information, rather than being a special tool that a user uses when they want to create Mashups. In order to create Mashups from normal websites, Mash Maker must first extract structured data from them. If the web site does not provide RDF data, then Mash Maker extracts structured data from the raw HTML using a community-maintained database of extractors, where each extractor describes how to extract structured data from a particular kind of web site. The next diagrams depict the SUS scores of respondents and a box of whisker.

**Table B-2.IntelMashMaker SUS score**

Users	Score	SUS score=score*2.5
1	26	65
2	27	67.5
3	30	75
4	29	72.5
5	29	72.5
6	1	2.5
7	3	7.5
8	28	70
9	17	42.5
10	8	20
11	28	70
12	28	70
<b>Average</b>		52.92



**Figure B-3.Intel MashMaker whisker box**

## Yahoo! Pipes

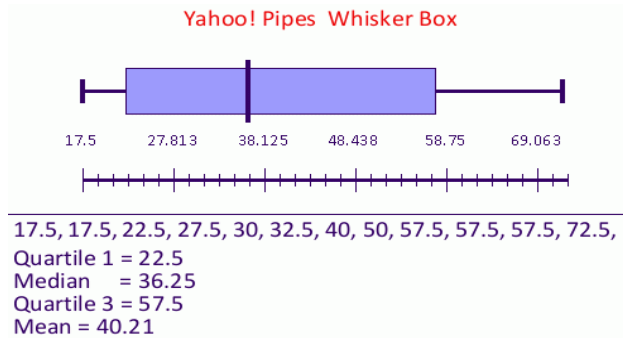
Yahoo! Pipes is a web-based visual programming language for constructing data Mashups. Yahoo! Pipes was originally developed as a tool to make extracting, aggregating, and republishing data from across the web easier (Jones & Churchill, 2009). Since its launch in February 2007, over 90,000 developers have created individual pipes on the Yahoo! Pipes platform, and pipes are executed over 5,000,000 times each day. The Yahoo! Pipes editing environment consists of four main regions: a navigational bar across the top, the toolbox on the left, the work canvas in the centre, and a debug-output panel at the bottom. The toolbox contains modules, the building blocks of the Yahoo! Pipes visual language. In the graphical language of Yahoo! Pipes, modules (operators) are laid out on a design canvas. Modules may have zero or more input ports, and all have at least one output port; additionally, modules may have parameters which can be set by the programmer, or themselves wired into the output of other modules so that the value of the parameter is dependent upon a runtime value specified elsewhere.

**Composition:** The input and output ports are wired together, representing the flow of data through the application. Selecting an output port, highlights all the compatible input ports to which the user may connect it. There are a number of data types within Yahoo! Pipes which determine what inputs and outputs are compatible. In the most general terms, there are simple scalar data *values*, and *items*, which are sets of data objects (e.g., items in an RSS feed, or nodes in an XML document). Values include types like text, urls, locations, numbers, dates, and times. In Yahoo! Pipes, data flows from the initial module(s), where user-data are input, or external data are retrieved, through subsequent modules in the pattern and order dictated by the wiring diagram. All applications in Yahoo! Pipes have a single output module, which is wired to the end of the execution sequence, and collects the final data stream for distribution via RSS, JSON (JavaScript Object Notation), or a variety of other formats. Yahoo! Pipes allows users to define complex branching, and looping structures, have multiple sources and execution paths executing in parallel, and in general, create programs of arbitrary complexity. There is no direct method for writing recursive functions, and Yahoo! Pipes does not allow for cycles in the program structure (i.e., where the output of a module is fed back to the input of a module further „upstream“). This enforces a mostly linear execution flow to the applications which is bounded by the amount of data being processed.

**Pipes:** Each pipe application is individually addressed by a unique ID and URL. Users may publish their pipes in the public directory, where they can be searched, browsed, and viewed by anyone. When a pipe application is viewed by someone who is not the owner of the pipe, a local copy is made in the viewer's browser. Any changes that are made by the viewer are saved to a new copy on the server, preserving the original pipe. In addition to entire pipes being copy able and modifiable, pipes can be embedded within one another as a "sub-pipe". This allows developers to create and share reusable components, and generate intermediate levels of abstraction in their applications. An embedded sub-pipe is represented as a module in the Yahoo! Pipes interface, which can be wired to other modules, or other sub-pipes. Users can drill-down into embedded sub-pipes, to inspect and modify the included functionality. The diagram of Yahoo Pipes is shown in Figure 1-2; the next diagrams depict the SUS scores of respondents and a box of whisker.

**Table B-3. Yahoo SUS vs. users**

<b>Users</b>	<b>Score</b>	<b>SUS score=score*2.5</b>
1	7	17.5
2	7	17.5
3	9	22.5
4	11	27.5
5	12	30
6	13	32.5
7	16	40
8	20	50
9	23	57.5
10	23	57.5
11	23	57.5
12	29	72.5
<b>Average</b>		<b>40.21</b>



**Figure B-4. Yahoo Pipes whisker box**

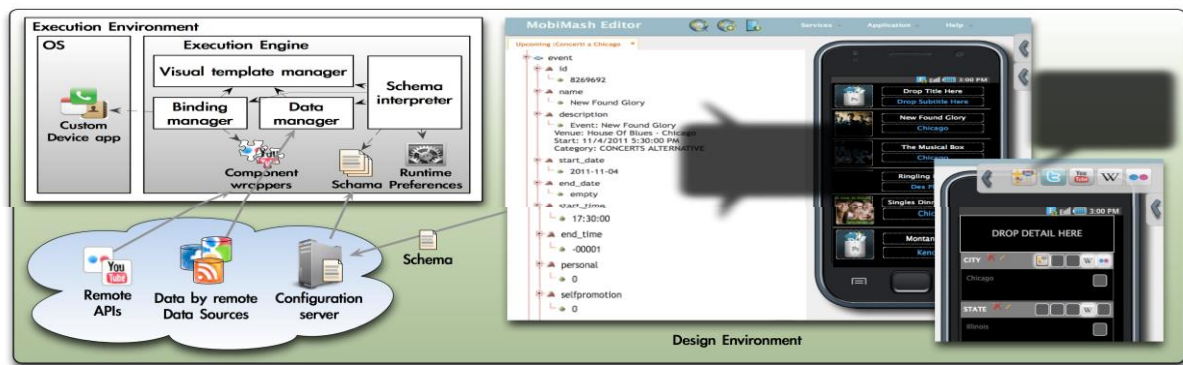
## MobiMash

The need to self-create applications is now even stronger for the mobile users: simple applications addressing very contingent and situational requirements, as Mashups are, can solve several users' information needs arising in mobile usage contexts (Cappiello et al., 2012). MobiMash is a platform for the construction of mobile Mashups, characterized by a lightweight composition paradigm, mainly guided by the notion of visual templates. The composition paradigm generates an application schema that is based on a domain specific language addressing dimensions for data integration and service orchestration, and that guides at runtime the dynamic instantiation of the final mobile app.

MobiMash is characterized by an End-User Development (EUD) Web environment, where a visual composition paradigm, based on the completion of visual templates, allows the users to easily configure the fusion of contents coming from different data sources, and the synchronization of such core contents with both remote APIs and local services available on the mobile device. The so-created applications are device's native applications that, in contrast with Web Mashups, do not need the Web browser as execution environment - the access to mobile device services is therefore enhanced.

**Visual composition:** The right hand side of Figure B-5 exemplifies the DE visual composition paradigm based on the completion of visual templates. The composition canvas consists of two main panels: the data panel on the left displays the data retrieved by querying some selected data services; the visual template panel on the right shows a selected visual template, i.e., a representation of the UI of the final app. As soon as data items selected in the data panel are associated to UI elements, the visual template panel is updated with a preview of the association effect. In this way, the user constructs the presentation layer - and

implicitly integrates data. In fact, in case of data items deriving from different sources, the user actions are targeted towards the construction of unified data views. In this integration process, the local schemas of the different sources are reshaped depending on the user selection of data items, while the global schema is derived by the structure of the visual template and the user-defined mappings between its elements and the items of each single source.

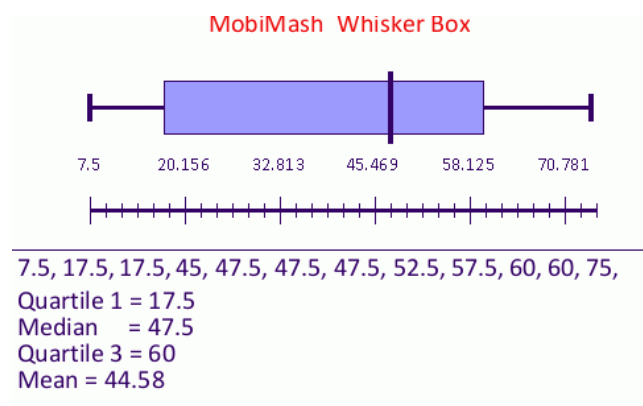


**Figure B-5.MobiMash (Cappiello et al., 2012)**

The DE also allows the user to synchronize the integrated data view with the invocation of UI components and local services. The aim is to enrich the core data with contents of different nature. The available services are shown to the user through an icon menu in the upper part of the screen, which adapts its listed items by showing the only APIs and local services that are compatible with the data associated to the selected visual element. The next diagrams depict the SUS scores of respondents and a box of whisker.

**Table B-4.MobiMash SUS score**

Users	Score	SUS score=score*2.5
1	24	60
2	19	47.5
3	19	47.5
4	18	45
5	30	75
6	23	57.5
7	24	60
8	3	7.5
9	19	47.5
10	21	52.5
11	7	17.5
12	7	17.5
<b>Average</b>		<b>44.58</b>



**Figure B-6.MobiMash whisker box**

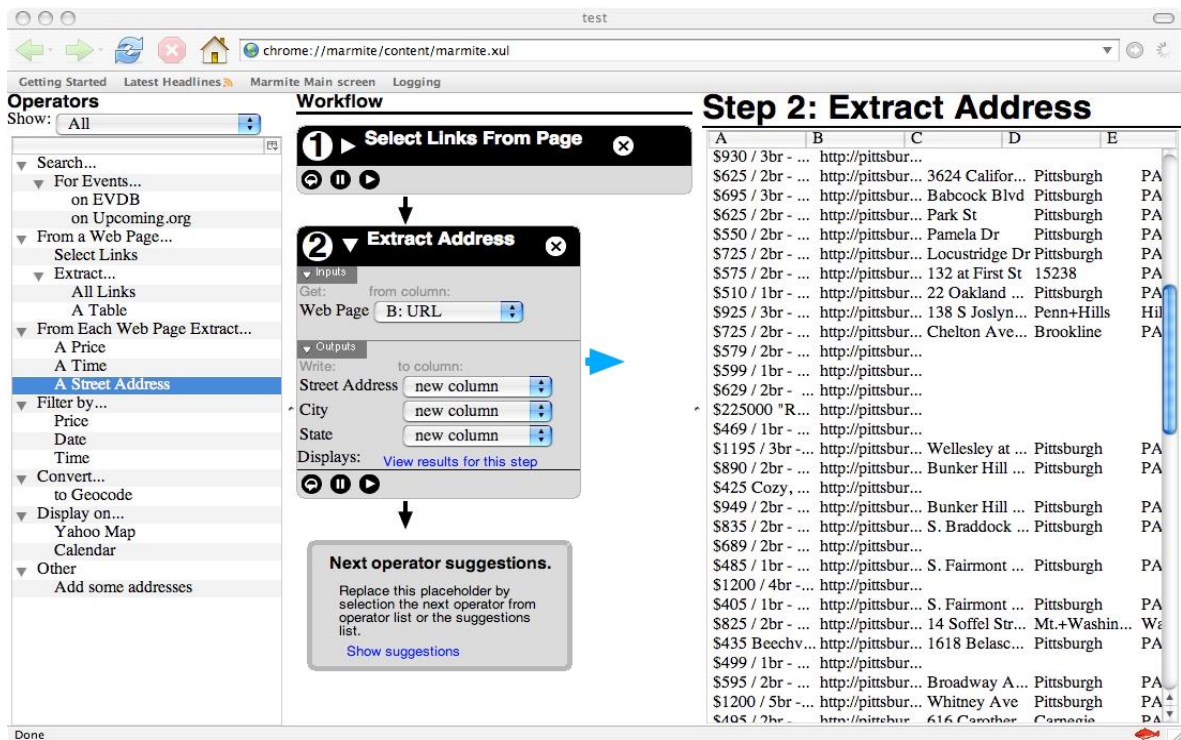
## **Marmite**

Marmite lets end-users create so-called Mashups that repurpose and combine existing web content and services(Wong & Hong, 2007). Marmite supports a data flow architecture, where data is processed by a series of *operators* in a manner similar to Unix pipes.

More specifically, Marmite lets end-users:

- Easily extract interesting content from one or more web pages.
- Process it in a data flow manner, for example filtering out values or adding metadata
- Integrate it with other data sources, either from local or remote databases, from other existing web pages or services.
- Direct the output to a variety of sinks, such as databases, map services, text files, web pages, or composable source code that can be further customized

In other words, Marmite's linked view shows both the program and the data simultaneously. With some users are able to use this system and to construct programs with web services quickly without difficulties.



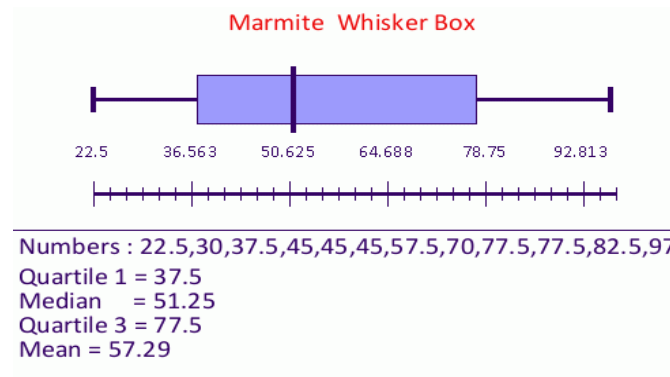
**Figure B-7.Marmite (Wong & Hong, 2007)**

The next diagrams depict the SUS scores of respondents and a box of whisker.

**Table B-5.Marmite SUS score**

Users	Score	SUS score=score*2.5
1	39	97.5
2	12	30
3	18	45
4	23	57.5
5	15	37.5
6	18	45
7	18	45
8	28	70
9	31	77.5
10	9	22.5
11	31	77.5
12	33	82.5
Average		57.29





**Figure B-8.Marmite whisker box**

## Dapper

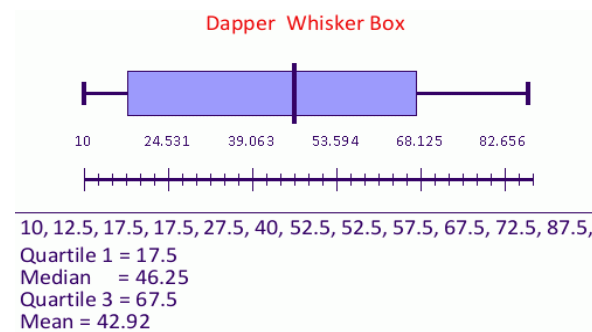
Dapper stands for Data mapper. The main purpose of the service is to convert any type of content into a standard form that can be reused (i.e., RSS, XML) (Sigelman et al., 2010). It also has a set of publishing features that turn that content into Google Gadget, Netvibes Module, Flash widgets and so on. It is a web application that visually runs is a wizard mode asking the user to fill-in some field at each step in order to create a “dapp” (data imported).

The user interface is very minimalist, but it gets the things done. Dapper can be made public and indeed for popular services like YouTube and Flickr there is a huge collection of dapps available. Typically there is no need to create a separate dapp. In many cases dapp is a good candidate to be tuned into a map Mashup or image loop. The user defines the output format or visualization type to use. The next level of development is to combine those dapps into an aggregator service. The typical example is to combine search result from several search engines or video clips from alternative video services similar to a movie aggregator.

The next diagrams depict the SUS scores of respondents and a box of whisker.

**Table B-6.Dapper SUS scores**

Users	Score	SUS score=score*2.5
1	21	52.5
2	23	57.5
3	7	17.5
4	7	17.5
5	29	72.5
6	5	12.5
7	16	40
8	21	52.5
9	27	67.5
10	4	10
11	35	87.5
12	11	27.5
<b>Average</b>		42.92



**Figure B-9.Dapper whisker box**

## Appendix C: Code Snippet

```
<?xml version="1.0" encoding="windows-1252"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:MashupComponent="http://csc.ufh.ac.za/2013-2014/SerProMashup#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <owl:Ontology rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#">
    <owl:imports rdf:resource="file: C:/Users/sabelo/Dropbox/Ontologies/hrest.rdf"/>
    <rdfs:comment> This an example of a usable Mashup Tool for Novice Users</rdfs:comment>
    <rdfs:label>SerPro Mashup Tool REST Ontology</rdfs:label>
  </owl:Ontology>
  <owl:Class rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#ServiceParameterMap">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
          >1</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#paramValue"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
          >1</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#paramName"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#Operation">
    <rdfs:subClassOf>
      <owl:Class rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#serviceOutputMap">
    <rdfs:label>Web Service response</rdfs:label>
  </owl:Class>
  <owl:Class rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
          >1</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#Name"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
          >1</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#outputNode"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </rdfs:subClassOf>
</rdf:RDF>
```

```

    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#baseUrl"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:label>REST Web Services</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#inputParameter">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#ServiceParameterMap"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#OutputParameter">
  <rdfs:range rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#serviceOutputMap"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#baseUrl">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#paramName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#accessPath">
  <rdfs:range rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#baseUrl"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#outputNode">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#description">
<owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#method">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://csc.ufh.ac.za/2013-2014/SerProMashup#Name">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://csc.ufh.ac.za/2013-2014/SerProMashup#Service"/>
</owl:DatatypeProperty>
</rdf:RDF>

```