Service Oriented Architecture for a Software Traceability System

Ntheye Lungu
Department of Computer Science
University of Cape Town
Cape Town, South Africa
ntheye@cs.uct.ac.za

Fadzai Muvuti
Department of Computer Science
University of Cape Town
Cape Town, South Africa
fmuvuti@cs.uct.ac.za

ABSTRACT
In order to improve software development and keep up with the fast pace of business, standards and methodologies for determining and endorsing effective software development processes have been introduced and put into effect on software projects. Accordingly, many tools that interpret these standards and methodologies have been developed and employed. Although there is active development and research in the area of requirements traceability, the desired level of acceptance has not been achieved, and the most widely reported reason for this in the industry, is that of: 'poor and immature integration technology'. This has resulted in existing tools often suffering problems due to poor integration and inflexibility with other technologies, which undermines the usefulness, usability and longevity of the Requirements Traceability provided by these tools. The panacea, at least in the confines of this project, is to employ a new technology: 'Web Services' as the underlying framework, to address these problems. The motivation for employing the web services architecture for this project is to allow personalized customization of a traceability solution, hence providing a ubiquitous software development process that incorporates standards as well as software engineering industry best practices.

General Terms
Management, Documentation, Standardization, Verification.

Keywords
Requirement, Traceability, Web Services, Requirements' Traceability

1. INTRODUCTION

1.1 Problem Description
The extent of project failure is well documented. A study from the Standish group (The Chaos Report) indicates that on average, over eighty-three percent of software development projects fail. (Projects considered to have failed are those that are either cancelled before completion, exceed budgeted costs, or overrun project deadlines). In addition to this, given that only 61% of the originally specified requirements ever make it to the final release of the project [1]; to keep up with the fast pace of business, projects must be able to handle the frequently changing goals and needs of the customer through the effective management of requirements. Although this increased awareness to the importance of effective requirements' management has caused a boom in the market for traceability tools [2],most of these tools basically employ the same techniques, and differ mainly in 'cosmetics', time, effort and manual intervention required to achieve their purpose. One of the key shortcomings that these tools all have in common, is that they often suffer problems due to poor integration and inflexibility with other tools and technologies [3][4]. Therefore, despite the growth in the industry, or the advancement in the functionality of the tools, their applicability and widespread adoption will always be undermined for as long as these integration problems exist [5].

1.2 Project Goals
The development and implementation of a complete software traceability solution:

1) Housing industry best practice traceability methodologies, processes, and solution sets;
2) Based on the Web Services' framework;

in response to the existing problems of inflexibility and poor integration, being faced by existing tools. The functionality of this solution can be discovered as a web service; thereby providing an accessible and ubiquitous traceability solution which will allow project stakeholders (project managers in particular) to produce tailor made traceability solutions fitted to the requirements engineering efforts necessary for a particular project.

2. BACKGROUND

2.1 Requirements' Management
To appreciate the concept of 'Requirements Management', it is important to understand what a requirement is. Sommerville [6] defines a requirement as *"a statement, in a natural language of what user services the system is expected to provide…"*. 'Change' is a key characteristic of requirements, and since the fulfilment of requirements constitutes the process of software development, one can conclude that change in software development is inevitable. In fact, *"No matter where you are in the software development process, the system **will** change! [7]"* The primary measure of success of a software system is the degree to which it meets the

purpose for which it was intended [8]. Given that only 61% of the originally specified requirements ever make it to the final release of the project [9]; it is vitally important that 'change' is managed and controlled throughout the development process so that the project is delivered on time, within budget, and according to the requirements. 'Requirements Management' is the field in software development responsible for managing this change, and research by Hofman and Lehner suggest that 'Requirements Management' constitutes the most important part of any software development project [10], and therefore should not be taken lightly.

2.2 Requirements' Management: Challenges

'Rrequirements' Management' may seem fairly straightforward, however, defining and recognising a requirement is an arduous task; hence, many projects experience requirement-related problems during software development. The problems may be categorized as follows [11]:

1. Difficulty to track changes in requirements.
2. The numerous sources from which requirements originate render them naturally inconsistent, and hence not so understandable.
3. Requirements are not always eloquently expressed in words. This lack of clarity makes it difficult to understand requirements statements.
4. There are different types of requirements at different levels of complexity within the software development lifecycle, and
5. Typically, a software project may have a large number of requirements. The more requirements there are, the more arduous the task of managing them.

The challenges identified here can be overcome by 'Requirements traceability'.

2.3 Requirements' Traceability

To address the challenges of requirements management, project managers need to implement the following measures:

- Indicate through identifiers the origins of a requirement, how it is specified and subsequently created, tested and delivered.
- Indicate for each work artifact the requirement(s) this work artifact satisfies.
- Facilitate communications, and customer commitment throughout the software project lifecycle.

The steps identified here constitute Requirements' Traceability, which has been defined by Gotel and Finkelstein [12] as: *"The ability to describe and follow the life of a requirement in both forwards and backwards directions (i.e. from its origins, through its development and specification, to its subsequent deployment and uses, and through all periods of on-going refinement and iteration in any of these planes)"*

2.3 Requirements' Traceability: Purpose

The purpose of requirements traceability is:

- Demonstrate to the customer that the developed product conforms to the requested features, and that all changes arising throughout the development phases have been accounted for.
- Ensure that the test plan, test cases and test procedures in general are relevant to the requirements [13].
- Ensure that developers are not creating features that no one has requested – avoid creeping featurism.

From the above-mentioned points, it is clear to see that software development projects, must from inception, accommodate requirements traceability and retain the capacity to handle the frequently changing goals and needs of the user.

2.4 Requirements' Traceability: Techniques

Traceability is achieved through the creation of 'traceability relations'. These are links which define and describe the relationships that exist between the different artefacts (e.g. requirements, designs, assumptions, rationale, system concepts, source code etc [14]) involved in software development. {Gotel, Finkelstein} [15] have proposed several techniques for achieving Requirements Traceability including: 'Traceability Matrices' (a graphical description of which is given below in **Figure 1**), Hypertext, Matrix Sequences, Assumption-Based Truth and Constraint Networks, just to mention a few*.

ID	USER REQUIREMENTS	FORWARD TRACEABILITY
U2	Users shall process retirement claims.	S10 S11 S12
U3	Users shall process survivor claims	S13

ID	FUNCTIONAL REQUIREMENTS	BACKWARD TRACEABILITY
S10	The system shall access requirement data.	U2
S11	The system shall calculate the amount of retirement.	U2
S12	The system shall calculate point-to-point travel time	U2
S13	The system shall calculate the amount of survivor annuity.	U3

Figure 1: Traceability Matrix

(In a traceability matrix, each requirement is required to have a unique identifier that distinguishes it from all the other requirements in the system. The matrix is constructed by the developer by associating a requirement with any other requirement(s) with which it interacts with [16]. This graphical visualization makes it easier for one see which requirement(s) will be affected by a change in the other requirement(s)).

3. WEB SERVICES

In the introductory chapter of this report, it was mentioned that a major requirement of this project is to deploy a web service with traceability capability. This section serves to provide an insight into the web services architecture and web services in particular.

* Due to space limitations, a detailed description of each technique cannot be given.

By definition, *"A Web Service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols."*[W3C]. In essence, Web Services are autonomous constructs whose development, deployment, and operation all vary independently depending on specifics of intended consumption. The functional details of a Service are concealed within that Service; however the functionality is exposed. Moreover, because Web Services are message oriented, the Web Service consumer is significantly insulated from the implementation alternatives available to the developer. This murkiness is critical to service autonomy, and facilitates transparency to programming models, operating systems, and implementation specific details. It may seem that web services are obscure constructs; however, they expose their functionality in machine readable descriptions of the messages a service sends or receives. This functionality is based on the interactions of three roles: service provider, service registry and service requestor. **Figure 2** illustrates how the three roles actually interact.

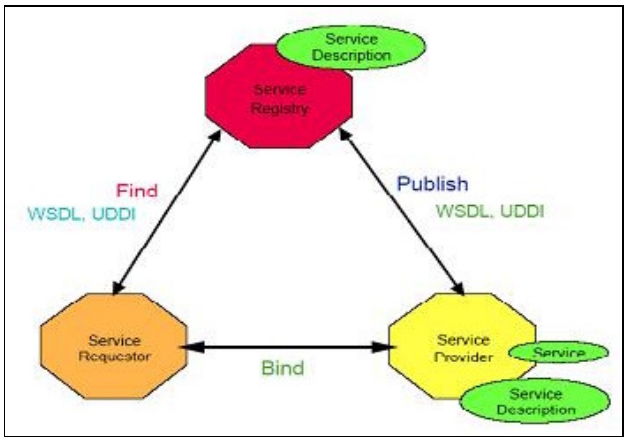

Figure 2: Service Oriented Architecture [17]

The individual responsibilities of each of the roles are as follows [18]:

1. *Service provider:* this is either the platform or owner that publishes the service. The latter applies particularly in the business context.
2. *Service Requestor:* in the framework of this paper, applies to the project team that desires to develop its traceability solution tailored for a particular product development. The service requestor, through an application interface, performs a find operation that retrieves the web service description, and hence functionality. To invoke the web service, the service requestor issues a bind operation.
3. *Service Registry:* this is a searchable registry of service descriptions where service providers publish their service descriptions.

3.1 Web Services: Protocol Stack

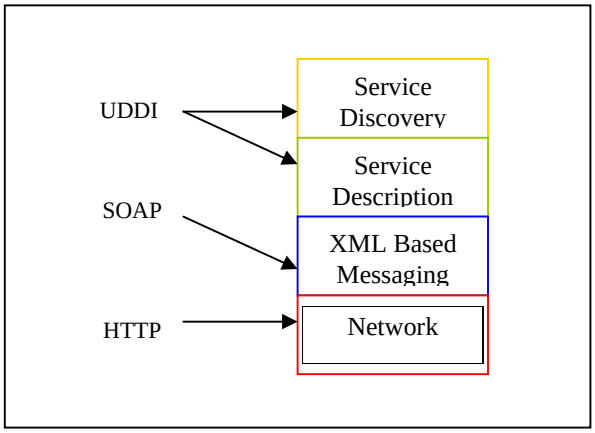

Figure 3: Protocol Stack [19]

The three protocols constitute the web services architecture in the following manner [20]:

- **SOAP:** A lightweight, high level XML-based messaging protocol that defines how Web Services exchange messages with each other. One of the main advantages of SOAP is that it is completely independent of the underlying transport protocol.
- **WSDL**: An XML document that describes a Web Service by providing all the relevant information on how the service communicates, and where it resides.
- **UDDI**: This is essentially the 'yellow pages' of Web Services. It is a public registry in which Web Services and their descriptions (in WSDL) can be registered and retrieved. Through UDDI, one can discover available Web Services.

3.2 Web Services: Message Interactions[21]

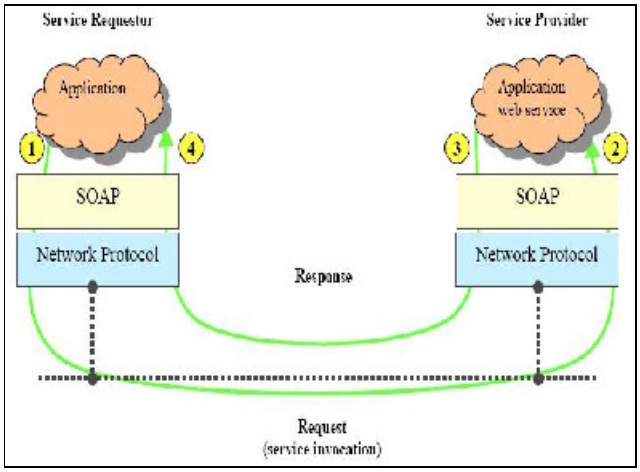

Figure 4: Message Interactions

Figure 4, above illustrates the conceptual interaction between the web service provider of a traceability solution and the consuming application.

1. The service requestor creates a SOAP message that invokes the desired web service operation. The SOAP message has to be consistent with the service description made available for discovery by the web service provider.
2. The initial SOAP message is transported, via the network infrastructure, to the service providers SOAP runtime environment. Typically, the SOAP runtime converts the XML message into programming language specific constructs (objects and types).
3. The web service processes the request and produces a response, which is itself a SOAP message.
4. The response is then routed through the SOAP runtime where the XML message can be converted, yet again, into programming language specific objects, consistent with the application. The application can now use the web services generated response.

4. IMPLEMENTATION

The focus of the whole project i.e. 'Service Oriented Architecture for a Software Traceability System', is the development of a system whose functionality is based on the 6 modules contained within the boundary 'Traceability Web Services' as illustrated in **Fig x** above. Each of these 6 modules will be developed and deployed as independent Web Services, and will therefore be able to communicate with each other through the Web Services'-defined message-passing mechanism; the successful collaboration of which will result in the provision of a complete traceability solution. The overall system was divided into 2 separate sub-systems, responsible for the implementation of 3 modules each. (The subsystem implemented by Fadzai will be called 'SubSystem1' and the subsystem implemented by Ntheye will be called 'SubSystem2' for the remainder of this document).

The development of this project was done using C# (.NET).

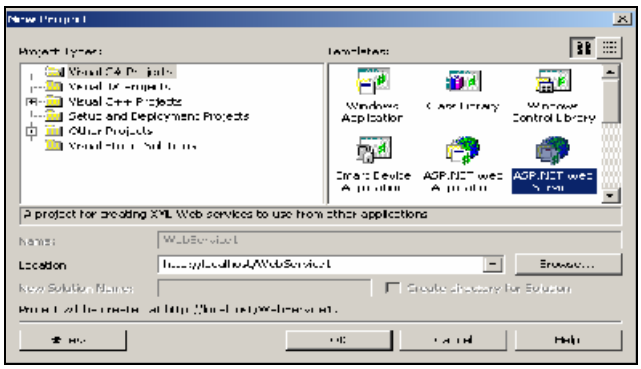

Figure 5: Creation of Web Services in .NET

As shown in **Fig 5** above, .NET provides a predefined template that makes it easy to create Web Services. A Web Service consumer can be any one of the .NET types of applications (i.e. Console, Window Form, Web Form, or another Web Service). However, for this project, the consumer was implemented as a 'Window Form'. Below is a description of the overall implementation of the overall project according to the 2 subsystems:

4.1 SubSystem1

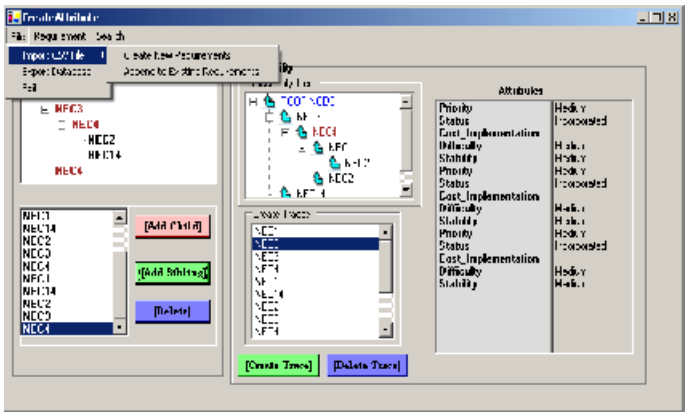

Figure 6:Screenshot of Fadzai's System

Figure 6 illustrates the functionality corresponding to the following modules:

1. Integration with RequisitePro: A user can create and manipulate requirements in RequsitePro, and can then export these requirements into .CSV format file. By clicking *'File ->Import CSV File ->Create New Requirements'* the user opens up a 'File Upload' dialogue box which allows the user to select the location of the file to be uploaded. The system does the first checks which to ensure that the file is in the correct format (i.e. CSV). If it is not, an error is indicated through a message box, and if it is, they system goes into the 'File Process()' method which does a second check on the file. This second check checks that they contents of the file follow the format: *line1->attribute attributes; following lines->attributes and corresponding values'*. If not, another error is indicated else the file is process, and the requirements are uploaded into the database after first truncating the table. 'Append to Existing Requirements' allows the user to upload several files into the repository at once. The same

procedure as in 'Create New Requirements', only that in this case the database is not truncated, it is appended.

2. Impact Analysis: shown on the right side of the figure is the section of the system that implements the traceability aspect of the tool. Traceability is implemented as a Traceability Tree. All roots are hypothetically attached to a ROOT node which 'maintains' the tree structure, but does not participate in the traceability as it is not an actual requirement. A user clicks on a requirement, and selects the second requirement to which the trace should go from the list box below. Various traceability algorithms are done to check for the validity of the trace before the link is created (e.g. duplicate traces: *A->B* when the trace has already been created, similar traces: *A->A;* circular traces: *A->B and B-A* at a later stage. This gets more complex as the height of the tree deepens). On the extreme right-hand-side of the screen, the user can view the attributes and attribute values associated with each requirement clicked on in the list box below. The tree-like structure gives the user a graphical representation of what requirements can be affected by a change in the affected requirement.

ii) Relationship Generation: Shown on the left-hand side of the GUI, the user can select a requirement from the Tree Hierarchy in the tree in the top list box, and then select the child requirement from the bottom list box. This feature makes it easy for the user to manage a requirement because the child requirements are more detailed specifications of the parent (e.g. the requirement 'Plan project' could have as its children: Planning, Requirements' Gathering, etc)

4.2 SubSystem2

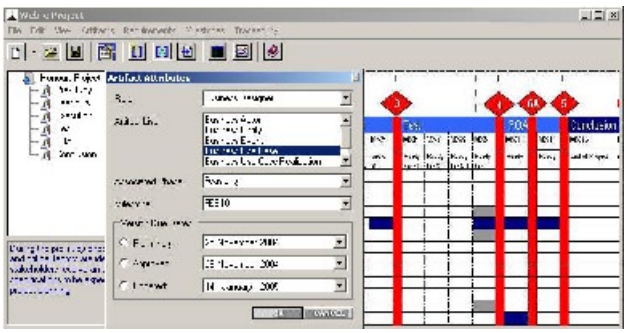

Figure 7: Screenshot of Sub-System2

Consistent with the aims of SubSystem2, figure 7 is a depiction of the main functionalities which are listed as follows:

RUP Harvest

Artefacts, activities, and stakeholder roles typical in a project are harvested from the Rational Unified Process framework (RUP). Depicted in figure 7 is a dialog box showing the link between stakeholder roles and artefacts such that a selected role constructs a list of artefacts that are either produced or consumed by that role. Thus, for example, the Business Designer role is associated with the following artefacts; Business Use-Case, Business Use-Case Realisation, Business Event, among others. The stakeholder role-to-artefact links are derived from RUP.

In reality, projects differ in their detail. Therefore, the tool provides functionality that allows the project initiator (the user), to determine which particular artefacts will be produced, and hence consumed. The artefacts and roles not essential to a particular project may be left out from that project.

Project Setup

Once the stakeholder role-to-artefact relationships have been harvested and the project initiator determines what is essential for the project, the rest of the project properties can be setup. The properties referred to here include all phases essential for project; pre-study, feasibility, execution, test, First Office Application (FOA), and conclusion. Within these phases, milestones as well as the deliverables within those milestones are set.

The project setup information is written to a database, allowing the project initiator and all stakeholders to view the project details. This view is depicted in figure 7 by the thick horizontal lines running through the view.

5. TESTING

5.1 SubSystem1

The following tests were conducted on the system in order to verify and validate the system:

5.1.1 Desk Checking

This type of testing involved the actual matching of each of the test cases that had been specified, against the requirements, in order to ensure that the system not only met the requirements, but that it also fulfilled them in the way in which it was expected to.

5.1.2 Unit Testing

The 'Unit Tests' were based on the results obtained from 'Desk Checking'. The modules were tested separately to ensure that they all met the specified requirements according to the specifications given at the start of the project. It is only after a module had passed all the tests specified by its corresponding Desk Check, was it deemed to have 'passed'.

5.1.3 Integration Testing

The power of Web Service is drawn from the fact that they do not suffer from integration as they use a message-passing framework (this forms the basis of the choice to implement this project upon a Web Services technology framework). Therefore no integration testing was done.

5.1.4 User Testing

A mixed sample of 10 candidates was chosen to come and test the system. The main aspects which were being test for were usability of the system, and applicability to the relevant market. The sample consisted of 3 individuals who are currently working in the I.T industry, 3 ordinary PC users, and 4 students from the Honours class.

5.2 SubSystem2

Two main factors influenced the testing of subsystem 2.

Factor 1: An appropriate test case for the web services is for an application that consumes the web services to be created. This in fact was achieved, however, because the testing application was built and designed by the authors of the web service, testing presented a challenge. To mitigate this challenge, the application was designed to test all the available web services. This was the basis of the unit tests where all web service method calls where tested by the developed application.

Factor 2: Because the tool was designed to perform specific user tasks, the most appropriate usability testing was the cognitive walkthrough. The cognitive walkthrough requires an *analyst* acting as a user and performing *a set of obligatory listed tasks*, a persona or *user profile*, and a *cognitive walkthrough checklist*. As the analyst walks through the designated tasks, the following questions are addressed as a requirement for the test:

1) Can the user reasonably perform each task in this set?
2) Are there enough clues on the interface to provide guidance on how the user should go about conducting a task?
3) Once an action is performed, does the interface provide reasonable feedback?

The above test questions were invariably answered by the cognitive walk through.

Other than the aforementioned factors, and the mitigation techniques thereof, user testing was conducted. Considering the domain of the application being used and the intended user, user testing was confined to technology aware individuals who are well versed with software development. To evaluate the intended benefits of the system, the sample of users included (in categories):

1) Users unfamiliar with traceability tools and have been part of small (less than 5) software development teams only
2) Users unfamiliar with traceability tools and have managed development teams with at least 20 different roles.
3) Users familiar with traceability tools and have been part of small (less than 5) software development teams only
4) Users familiar with traceability tools and have managed development teams with at least 20 different roles

6. RESULTS

6.1 SubSystem1

The results below are in relation to the user testing that was done on the system:

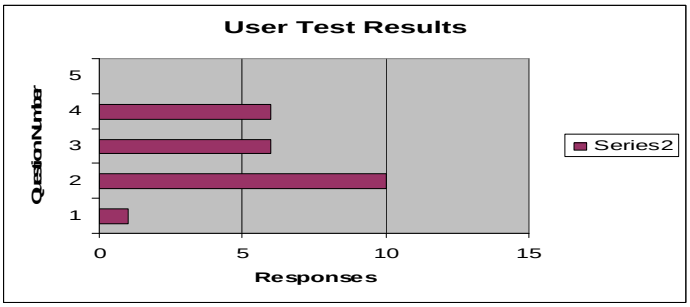

Figure 8: Results from User Testing

The table above graphically depicts the following results: only 1 individual (one of the 3 working in industry), could fully explain what 'Traceability' was. The rest of the candidates either did not know, or managed to give vague descriptions. 6/10 of the candidates said that they preferred free software (in the questionnaire, open source is assumed to be also free). With regards to the interface itself, all 10 candidates arrived at the general consensus that the green and pink colors used for the buttons on the interface were too 'flashy' and rendered the system 'unprofessional'. None of the candidates agreed to buy application if it was sold in its present state, but gave the view that if more features and functionalities were added, then perhaps, this opinion would change.

6.2 SubSystem2

The results are listed below according to test category described under testing in section 5.2.

Unit testing: The developed application successfully invoked all the implemented web services.

Cognitive walkthrough: The iterative cognitive walkthrough tests identified a number of improvement areas, not much with the interface, but more so with the functionality. In particular, it was identified that the project setup ought to support flexibility, hence the final prototype had this functionality incorporated.

User Testing: The results of table 1 are illustrated in table 1 below. Out of a total of 16 individuals that tested the tool, 11 found the user interface easy to use. The same number identified with the relevance of the application, whereas just over half felt that they would apply such a tool to a familiar project. The number of those that would actually apply the tool to a project was significantly smaller compared to the other sets of results.

Table 1: user testing results

	1	2	3	4	Total

The application was easy to use	2	2	4	3	11
The relevance of the application was clear	1	3	3	4	11
The application could be applied to a familiar project	0	4	1	4	9
I could apply the application for a project	0	1	1	3	5

7. EVALUATION OF RESULTS

7.1 SubSystem1

A worrying result was that only 1 person knew what traceability was and neither of the other 2 candidates in the industry had an idea. Although one might argue that the sample size was far to insignificant to reach a conclusion it can still be drawn from this that is clear to see that although Traceability is still a relatively new concept, there is still a lot of ground to be covered before it permeates through all facets of life. A second observation was that people will not easily part with their money, and if they can get something for free, then they will. Although it was obvious that no one would want to pay for the tool, a point that stands out here is that in the development of any commercial tools, several aspects of the tool have to be taken into consideration in order to not only convince the user to part with their money, but to also beat the competition, thus increasing market share and return on investment.

7.2 SubSystem2

An interpretation of the results for subsystem 2 is classified below:

Unit testing: The web service were consumed as was intended

Cognitive Walkthrough:

1) The user was able to perform the set out tasks with reasonable ease and within acceptable time.
2) A majority of the icons and menus provided sufficient clues as to their functionality (affordance). Thus, users where able to navigate the application and recognize functionality afforded by the application interface.
3) The view provided feedback on the actions performed by the user. However, feedback in this area suggests that the view ought to be a continuous improvement area.

8. RELATED WORK

8.1 TraceM [22]

Tool Category: Research-Based

Key Researchers: Susanne A. Sherba, Kenneth M. Anderson, Maha. Faisal

Research Lab: Department of Computer Science, University of Colorado

Contact Details: {sherba, kena, faisal}@cs.colorado.edu

Problem Definition: There are many different types of relationships that can exist between artefacts (e.g. requirements -> design, design -> code etc). In isolation, these traces are easy to capture, however they provide limited information with regards to how these artefacts interact within the overall system, leaving the developer with the burden of having to manually follow each of these relationships in order to get the overall picture.

Description of Solution: TraceM is based on the theory that a requirements traceability tool should be able to *derive* implicit relationships already represented in the system (i.e. using the above example, TraceM would allow the developer to see which part of the code the requirement actually traces to). TraceM uses 2 tools to achieve its purpose:

1.Open Hypermedia Systems: these enable the creation and viewing of relationships in heterogeneous applications, as well as the traversal of those relationships within and between applications.

2. Information Integration Systems: these are systems which provide the services to automate the process of discovering, creating, maintaining and evolving these relationships [23].

8.2 Generation of Traceability Based on the 'Dempster-Shafer Theory' [24]

Tool Category: Research –Based

Key Researchers: Andrea Zisman, George Spanoudakis; Elena Minana, Paul Krause

Research Labs: Software Engineering Group, Department of Computing, City University; Software Engineering Applications Group, Phillips Research Laboratories, UK

Contact Details: {a.zisman | gespan} @ soi.city.ac.uk

{Elena.perez-minana | krause}@phillips.com

Problem Definition: Motivated by the realisation that existing traceability tools expect developers to create and maintain relations manually, making this process error-prone and time-consuming; this is a tool which was developed to automatically generate and maintain traceability relations between any artefacts expressed in structured forms of natural language (e.g. requirements statements, use-cases, UML-based object models etc). This tool builds on the functionality presented by a tool previously developed by Zisman et al [25]. Zisman's tool generates traceability relations based on heuristic traceability rules which specify ways of matching syntactically related terms in the textual parts of the artefacts, with the related elements in the corresponding object model, and then automatically creates traceability relations of different types when a match is found. Although this approach has generally high precision in terms of

creating correct relations, one of the flaws discovered is that, since the rules it uses are based on syntactic and grammatical relations in the text, ambiguities will arise from time to time, and since the tool was not developed to resolve these, the result is that some traces are either insufficiently defined, or left out all together.

Proposed Solution: To overcome this problem, {Spanoudakis, Garcez and Zisman} [26] [27] have introduced 'belief' functions based on the 'Dempster Shafer Theory of Evidence'[†] to Zisman's tool. These measure the extent to which it may be *believed* that a rule is correct (given all the ambiguities) and then updates the measures that these functions generate, to reflect the confirmation or disconfirmation of the traceability relations. Not only does this tool confirm and disconfirm the existence of the relations, extra functions have been defined to measure the correctness of the relations based on the extent to which a rule is satisfied by its artefacts [28].

Although this approach has been shown to work, Alexender [29] believes that because the whole process is fully automated, this introduces the possibility of machine-made errors which reduce the quality of the resulting traceability model. In his paper, he postulates the theory of 'Semi-Automatic Traceability', in which a balance should be found between human intervention and automation, whereby human skill and experience should be used to ensure complete correctness of the automatically generated relations.

8.3 TOOR [30]

Tool Category: Research -Based

Key Researchers: Bashar Nuseibeh;Steve Easterbrook

Research Labs: Software Engineering Laboratory Department of Computing, Imperial College London;

Department of Computer Science, University of Toronto

Contact Details: ban@doc.ic.ac.uk; sme@cs.toronto.edu

Pinheiro and Goguen have developed a system – TOOR, for tracing requirements through-out the project development lifecycle. Within the TOOR framework, all artefacts that are produced or consumed during the software development lifecycle are considered as objects. Thus, vision documents, use case diagrams, test cases, along with many other artefacts, are recognized as objects. Moreover, relationships between these objects are themselves considered objects, and form the basis of requirements tracing. As an inherent requirement, TOOR uses the Functional Object-Oriented Programming System (FOOPS)[‡] to specify requirement classes from the identified objects, and the entire system as well. Thus, where a vision document is identified as an object, its specification is formulated by FOOPS. TOOR provides traceability setup when initiating a project. The first steps involve defining templates (automatically) for either objects, or relations between objects. These templates are forms containing fields for class and attribute name. Verification of whether the attribute value entered conforms to the stipulated axioms (of traceability) is performed by FOOPS. This verification requires a means by which TOOR should communicate with FOOPS, and this is achieved through UNIX sockets. Like TraceM, TOOR also has hyper-media capabilities to allow both forwards and backwards tracing. In addition, requirements are modularized allowing for both formal and informal axioms – an advantage for integration between phases through out the project development lifecycle. The study identified a number of disadvantages with TOOR:

1. Using UNIX sockets as a means of communication between TOOR and FOOPS renders the solution platform dependent (UNIX base). This presents major portability challenges.

2. FOOPS, the basis of TOORS, is not a widely available programming language. This makes it difficult to tailor TOORS to address a specific traceability problem unique to a particular project.

Future work on TOOR involves the representation of objects in two independent windows, one graphical, and the other employing a host of other visual criteria.

8.4 DOORS [31]

Tool Category: Commercial

Company Name: Telelogic

Contact Details: http://www.telelogic.com

The Dynamic Object Oriented Requirements System (DOORS) is a requirements management tool developed by Telelogic. It provides document-like modules (typically Microsoft Word and Excel) containing objects (artefacts) that can be linked. Three main features are supported; Use Case Modelling, Project Dictionary Construction, and an Exporter module. Also incorporated within the tool is a programming language DXL which facilitates modelling and full access to the available data structures. Traceability is a built in feature of DOORS. The Project Dictionary Construction feature searches through text in a selected module for phrases that match with terms in the project dictionary, if it exists, and links these. If there are no matches, a new dictionary may be created with stakeholder specified terms and definitions. The links produced are modelled into use cases depending on the context specified within the DOORS framework. After dictionary construction and modelling, a specialised exporter feature traverses a module to produce a series of web pages. The pages contain generated list of actors, associated with the use cases in which they act. Traces in between artefacts are represented as hyperlinks within a web page. These are bidirectional, thus, two hyperlinks are required for any single trace. Standard point-and-click controls are used to navigate within the DOORS environment and to follow through traces. The traceability capabilities of DOORS notwithstanding, the tool itself does not provide a structured way of representing traces into groups such as by Goals or Use Cases. Traces are seen as belonging purely to atomic artefacts – individual requirements

[†] More information on the Dempster-Shafer Theory can be found at: http://www.glennshafer.com/assets/downloads/article48.pdf

[‡] Even though TOOR is built on top of FOOPS, the specifications of FOOPS are beyond the scope of this text.

or test cases. The manual creation of structure and later export to web pages are not only time consuming, but render traceability unintuitive.

9. CONCLUSIONS

Thus far, it is apparent that commercial tools and ongoing research offer a suitable platform for the aims of this project. In fact, academic tools are continually addressing the pitfalls of the commercial tools, predominantly by providing automated traceability. Whereas, work in improving efficiency of traceability relationship management is vital, contemporary work on traceability does not particularly take into account particular traceability needs of individual software projects.

From the related works, with both advantages and disadvantages, it is apparent that a unique opportunity exists in this domain. The current tools lack of focus on an overall traceability solution, and more so the challenge of platform independence, requires new approaches. Therefore, rather than continually defining traceability albeit make traceability elaboration "more capable", this project proposes made available, through the web services architecture, the programmatic interface of a traceability solution, such that individual project teams account for their unique project details when defining a traceability solution. The intended consequence is the ability of all project teams to discover and implement a traceability solution specific to a project and its development lifecycle.

10. REFERENCES

[1] http://www.projectsmart.co.uk/docs/chaos_report.pdf

[2] Ramesh, B., "Towards Reference Models for Requirements Traceability", *IEEE Transactions on Software Engineering*, 2001, Vol. 27, No. 1

[3] Nuseibeh, B., Easterbrook, S.: "Requirements Engineering: A Roadmap," *International Conference on Software Engineering Proceedings of the conference on The future of Software engineering*, Limerick, Ireland, 2000, pp. 35 - 46

[4] Gotel, O., Finkelstein, A.: "Analysis of the Requirements Traceability Problem," *Proc. Of st Int. Conf. On Requirements Engineering*, April 1994, pp. 94 - 101.

[5] Gotel, O., Finkelstein, A.: "Analysis of the Requirements Traceability Problem," *Proc. Of st Int. Conf. On Requirements Engineering*, April 1994, pp. 94 - 101.

[6] Sommerville

[7] http://www.kellen.net/SysDev.htm

[8] Nuseibeh, B., Easterbrook, S.: "Requirements Engineering: A Roadmap," *International Conference on Software Engineering Proceedings of the conference on The future of Software engineering*, Limerick, Ireland, 2000, pp. 35 - 46

[9] http://www.projectsmart.co.uk/docs/chaos_report.pdf

[10] Hubert F. Hofmann, Franz Lehner

[11] Pinheiro, F., TOOR: A System for Tracing Object-Oriented Requirements

http://www.cs.ucsd.edu/users/goguen/sys/toor.html#tstat

[12] Gotel, O., Finkelstein, A.: *"An Analysis of the Requirements Traceability Problem,"* 1st International Conference on Requirements Engineering (ICRE'94), Colorado Springs, April 1994, pp. 94-101

[13] Ramesh, B., Jarke, M., "Towards Reference Models for Requirements Traceability", *IEEE Transactions on Software Engineering*, Vol. 27, No. 1, 2001.

[14] Egeyd, A.,Grunbacher,P.: "Towards Understanding Implications of Trace Dependencies among Quality Requirements",*2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '2003)*, Montreal, Canada, 10-2003

[15] Gotel, O., Finkelstein, A.: "An Analysis of the Requirements Traceability Problem," *1st International Conference on Requirements Engineering (ICRE'94)*, Colorado Springs, April 1994, pp. 94-101

[16]http://www.jiludwig.com/Traceability_Matrix_Structure.htm

[17] IBM Web Services Conceptual Architecture

[18] Schmelzer, R., Vanderspyn, T., Bloomberg, J., Siddalingaiah, M., Hunting, S., Qualls, M., Houlding, D., Darby, C., Kennedy, D.: "XML and Web Services" *SAMS Publishing*

[19] IBM Web Services Conceptual Architecture

[20] Schmelzer, R., Vanderspyn, T., Bloomberg, J., Siddalingaiah, M., Hunting, S., Qualls, M., Houlding, D., Darby, C., Kennedy, D.: "XML and Web Services" *SAMS Publishing*

[21] IBM Web Services Conceptual Architecture

[22] Sherba, S., Anderson, K., Faisal, M.: "A Framework for Mapping Traceability Relationships," *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, Montreal, CA, Oct. 7th, 2003, pp. 32-39

[23] Sherba, S., Anderson, K., Faisal, M.: "A Framework for Mapping Traceability Relationships," *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, Montreal, CA, Oct. 7th, 2003, pp. 32-39

[24] Zisman, A., Spanoudakis, G., Perez-Minana E., Krause P.: "Towards a Traceability Approach for Product Families Requirements," *Proc. Of 3rd ICSE Workshop on Software Product Lines: Economics, Architectures and Implications*, 2002

[25] Zisman, A., Spanoudakis, G., Perez-Minana E., Krause P.: "Towards a Traceability Approach for Product Families Requirements," *Proc. Of 3rd ICSE Workshop on Software Product Lines: Economics, Architectures and Implications*, 2002

[26] Spanoudakis, G., d'Avila, G., Zisman, A.: "Revising Rules to Capture Requirements Traceability Relations: A Machine Learning Approach," *15th International Conference in Software Engineering and Knowledge Engineering (SEKE 2003)*, San Francisco, CA, USA, 1-3 July 2003

[27] Spanoudakis, G.: "Plausible and Adaptive Requirement Traceability Structures," *ACM International Conference Proceeding Series Proceedings of the 14th international conference on Software engineering and knowledge engineering*, Ischia, Italy, 2002, pp.135 - 142

[28] Spanoudakis, G.: "Plausible and Adaptive Requirement Traceability Structures," *ACM International Conference Proceeding Series Proceedings of the 14th international*

conference on Software engineering and knowledge engineering, Ischia, Italy, 2002, pp.135 - 142

[29] Alexander, I.: "SemiAutomatic Tracing of Requirement Versions to Use Cases," *Second International Workshop on Traceability,* Montreal, October 2003

[30] Pinheiro, F., TOOR: A System for Tracing Object-Oriented Requirements

http://www.cs.ucsd.edu/users/goguen/sys/toor.html#tstat

[31]www.telelogic.com/

[25] Zisman, A., Spanoudakis, G., Perez-Minana E., Krause P.: "Towards a Traceability Approach for Product Families Requirements," *Proc. Of 3rd ICSE Workshop on Software Product Lines: Economics, Architectures and Implications,* 2002

[26] Spanoudakis, G., d'Avila, G., Zisman, A.: "Revising Rules to Capture Requirements Traceability Relations: A Machine Learning Approach," *15th International Conference in Software Engineering and Knowledge Engineering (SEKE 2003),* San Francisco, CA, USA, 1-3 July 2003

[27] Spanoudakis, G.: "Plausible and Adaptive Requirement Traceability Structures," *ACM*