Requirements Traceability on Web Applications

Waraporn Jirapanthong
Department of Information Technology
Dhurakij Pundit University
Bangkok, Thailand
waraporn.jir@dpu.ac.th

***Abstract*— The development of software products is currently being expanded to be based on web applications. Due to the time constraints, software design and development for web applications are driven. It leads the software or applications become more complex. Identifying impacts of changes or relations takes a lot of time and effort. One challenge is to enable traceability on software artefacts created during the development of web applications. Moreover, it is believed that analysis of data obtained from traceability relations is another way to make software process more efficient. However, it is absolutely not an easy task due to lack of guidelines, support, and tools. This research thus aims to enable software traceability on the development of web applications which involve many types of documents.**

Keywords:* *Requirements Traceability, Web Application Development, Requirements Engineering.

I. Introduction

According to the Standish Group [9] found that 53 percent of software projects have failed, and another 31 percent of software projects have spent over budget. Many organizations try to tackle the problem by applying standard software process. Many standards are aimed to standardize the software process. One concern of those standards is to provide the ability to track software artefacts, originally called Requirements Traceability in the software development process.

Traceability relation is a relation that is identified between various artefacts created during the software process. The relations can be forward or backward directions.

1. *Forward traceability relation* is a relation that is built to monitor the development of artefacts created during the software process. In particular, the relations can be traced from requirements forward through system design and so on. This allows analysts and designers to verify if the design is correct and completely implemented the requirements.

2. *Backward traceability relation* is a relation that is built to verify the accuracy of software artefacts backward. In particular, the relations can be traced from such as system design backward through their source or requirements.

Additionally, the development of software products is currently being expanded to be based on web applications. Due to the time constraints, software design and development for web applications are driven. It leads the software or applications become more complex. One challenge is to enable traceability on software artefacts created during the development of web applications. Software artefacts [2][6] are proposed to particularly support the development of web applications. Additionally, the requirements of web applications are rapidly changed. The changes, of course, influence on the relations between software artefacts. Identifying impacts of changes or relations takes a lot of time and effort. It is believed that analysis of data obtained from traceability relations is another way to make software process more efficient. However, it is absolutely not an easy task due to lack of guidelines, support, and tools. Also, existing approaches particularly support the development of conventional software applications. This research thus aims to enable software traceability on the development of web applications which involve many types of documents. Those documents are used as a tool to facilitate change management on web applications. The objective of this research is to present a traceability model for web applications.

II. Background

A. Software Engineering Vs. Web Engineering

According to [3], they have described the different characteristics between web applications and conventional software. [5],[6] described the major differences between web Applications and conventional software. Some aspects are similar but some are different. Moreover, [3] described that the characteristics of advanced web applications are: a) it is dynamic because information rapidly changed due to time and user's needs; b) the volume of information is large; c) it is difficult to navigate information; d) it is integrated with database and tracking systems; e) it likely requires a large development team with various expertise; f) it needs constantly evolution and high performance; and g) it needs promising methodologies and development process.

B. OOWA

According to [1], they proposed the Web Engineering method OOWS (Object Oriented Web Solution) that supports the development of web applications. OOWS is the extension of an object-oriented software production method called OO-Method [2]. OO-Method proposed models to describe the different aspects of a system: *a) class diagram; b) dynamic model;* and *c) functional model.*

The class diagram is used to describe the static structure of a web application. The dynamic and functional models are used to describe the system behavior. OOWS propose the extension of OO-Method with navigational and presentational capabilities. The navigational model is introduced to describe the navigational aspect of a web application that is composed of a set of navigational maps.

According to OOWS, there are two main steps: a) *conceptual modeling* and *b) solution development*. In the conceptual modeling step, there are three sub-steps: i) *Functional requirements elicitation*: use cases and scenarios are used to build up a conceptual schema; ii) *Classical conceptual modeling*: dynamic and functional models are used to capture the system structure and behavior; and iii) *Navigation and presentational modeling*: a navigational model is used to model navigational requirements from the class diagram.

C. Requirements Traceability

The term traceability has been initially used as requirements traceability and is concerned with the ability to relate requirements with all the other software artefacts generated during the development of software system [7]. It is realized that software traceability should relate between software artefacts created during the life-cycle of software system development such as retrieval documents, requirement specifications, analysis and design models, source codes, and test cases.

Gotel and Finkelstein [7] proposed traceability relations to be bidirectional relations in which requirements can be associated with other software artefacts in both forward and backward directions. They classified the categories associated with traceability, namely (a) *pre-requirements specification (pre-RS) traceability*, in which traceability relations associate requirements with source of requirements, and (b) *pos-requirements specification (pos-RS) traceability*, in which traceability relations associate requirements with other different artefacts generated in different phases of development life-cycle.

Similar to [7], Jarke [8] affirmed that the ability to perform traceability can be accomplished by four different types of relations in forward and backward directions. These relations are (a) *forward from the requirements,* which relate a particular requirement forward to design artefacts, (b) *backward to the requirements,* which relate a particular design artefact backward to requirements, (c) *forward to the requirements,* which relate customer's needs or source of requirements forward to requirements, and (d) *backward from the requirements,* which relate requirements backward to customer's needs of source of requirements.

III. Approach

A. Traceability Model for Web Applications

We propose the traceability model for web applications, particularly supporting object-oriented web applications. The model focuses on the key artefact types for a web application i.e. use case, domain model, activity diagram, navigational context, and presentational context [1]. The relations between artefacts can be different depending on the types of elements and artefacts. Based on our study and analysis of the case studies, we apply the types of traceability relations proposed earlier. We have applied 3 different types of traceability relations for different element in documents used in web applications development. As shown in Table 1, for instance, the relations between activity diagram and navigational context can be classified as *Overlap* and *Refine*. Moreover, the relations between presentational context and domain model can be classified as *Overlap* and *Implement*.

The types of relations are defined according to the semantics of i) *artefact types* that are created in order to capture or specify different types of requirements and design, and ii) *contents* that present the rational of requirements and design.

According to Table 1, the different types of traceability relation between artefacts are: O refers *Overlap*; I refers *Implement*; and R refers *Refine*.

Overlap refers a traceability relation that an element of a document has content overlapping with an element of another document. In this type of relation, an element *e1* overlaps with an element *e2* (and an element *e2* overlaps with an element *e1*) if *e1* and *e2* refer to common aspects of an application or its domain.

Implement refers a traceability relation that an element of a document implements or presents another perspective to be an element of another document. In this type of relation, an element *e1* or a composition of elements implements an element *e2* if *e1* allows for the achievement of *e2*.

Refine refers a traceability relation that an element of a document is elaborated to be another element of another document. This type of relation associates elements in different levels of abstractions.

B. Traceability Rules and Relations

This section presents a template of traceability rules that are used to identify the traceability relations. Figure 1 shows an example of traceability rule. Each rule is built to capture a specific type of traceability relations (*RelType*) based on specific types of documents (*DocType1* and *DocType2*). The logical rule is also described (*Documentation*). The rule consists of two main parts i) <Condition>, and ii)<Consequence>.

In the condition part, it consists of template (<Template>) and check (<Check>) parts. The template has a template name (<TemplateName>) and is used to declare slot(s) (<Slot>). Each slot consists of slot name (<SlotName>) and value (<Value>). The value is composed of variable(s) (<Variable>) which is input from a document.

Moreover, the check part consists of operator (<Operator>) and operand(s) (<Operand>) to declare the operation of identifying a relation. In the consequence part, it also consists of template (<Template>) and assert (<Assert>). The assert declares elements of documents that are identified as a traceability relation.

TABLE 1: TRACEABILITY MODEL FOR WEB APPLICATIONS

	Use Case	**Domain Model**	**Activity Diagram**	**Navigational Context**	**Presentational Context**
Use Case		O	O	O	O
Domain Model	O I R		O	O	O
Activity Diagram	O I R	O I R		O R	O R
Navigational Context	O R	O R	O R		O
Presentational Context	O	O I	O	O I	

For instance, a traceability rule in Figure 1 (RuleID = “1”) is used for identifying a traceability relation entitled overlap between documents typed domain model and navigational map. In the condition part (<Condition>), it declares two variables x1 and x2 that occur in slots, SlotOne and SlotTwo accordingly. The condition is to capture the variables as the operands (<Operand>) and identify if they are satisfied by the operator (<Operator>). If the condition is satisfied, the consequence part (<Consequence>) is then activated. In the consequence part, two slots (SlotThree, and SlotFour) are declared as output variables (y1 and y2). The two variables are identified as two output artefacts and the traceability relation will be created between those two variables in assert part (<Assert>).

In the following section, we describe a case study of web application development and illustrate examples of the traceability relations through the case study.

C. *A Case Study of a Web Application*

The objectives of the built-up of the case study are: (i) to study the activities during the development of a web application, particularly applying with OOWS; and (ii) to demonstrate and evaluate our approach.

A case study has been developed based on study, analysis, and discussions on online shopping domain. We have adopted the case study of web application development, online shopping [4]. The system is a common web-based application based on marketing demands which requires online trading. We have studied the case on behalf of stakeholders ranging from market researchers, to requirements engineers, system analysts, and developers. Conceptual modeling of the web application is performed according to OOWS. In this way, a number of artefacts are created by a developer team. The list of functionalities of the system in our case study is shown in Table 2. The web application of online shopping is expected to be a trendy web site that provides the functions of user authentication, displaying product catalog, making online purchase, processing credit card payment gateway, checking out selected items, and managing administration issues i.e. logging, orders, customers, and warehouse and inventory.

```
<TraceRule ID="1" RelType="Overlap” DocType1=”Class
Diagram” DocType2=”Navigational Map” Documentation=”if
a (SlotOne ?x1) (SlotTwo ?x2) then then b (SlotThree ?y1)
(SlotFour ?y2)">
<Condition><Template><TemplateName>a</TemplateName>
<Slot><SlotName>SlotOne</SlotName><Value><Variable>x1
</Variable></Value></Slot>
<Slot><SlotName>SlotTwo</SlotName><Value><Variable>x2
</Variable></Value></Slot></Template>
<Check><Operator>=</Operator><Operand no=”1”>x1
</Operand><Operand no=”2”>x2</Operand></Check>
</Condition>
<Consequence>
<Template><TemplateName>b</TemplateName>
<Slot><SlotName>SlotThree</SlotName><Value><Variable>y
1</Variable></Value></Slot>
<Slot><SlotName>SlotFour</SlotName><Value><Variable>y
2</Variable></Value></Slot></Template>
<Assert>
<Slot><SlotName>SlotThree</SlotName><Value><Variable>y
1</Variable></Value></Slot><Slot><SlotName>SlotFour</Slo
tName><Value><Variable>y2</Variable></Value></Slot>
</Assert> </Consequence>
</TraceRule>
```

Fig. 1: An example of Traceability Rule

Table 2: Functionalities of Online Shopping System

Functionality	**Description**
F1	Make purchases online
F2	Provide credit card processing system (credit card payment gateway)
F3	Manage logging and website
F4	Display product catalog
F5	Check out items
F6	Authenticate members
F7	Manage orders
F8	Manage customers
F9	Manage warehouses and inventory

The requirements is modeled and described in term of a use case diagram. The main functional requirements are composed as use cases *View Items, Make Purchase, Checkout,* and *Client Register*.

Each use case is further described its behavior in activity diagrams. As shown in Figure 2, a partial activity diagram of online shopping system, particularly use case *View Items*, shows system activities after user requested to view items. There are two possible activities i.e. search or browse items. In a case of *Search Items*, the system will perform searching and return a list of found items in order to be viewed, otherwise return to the initial state of *View Items* again.

The domain model is designed in term of a class diagram. For example, the activities *Search Items, Browser Item,* and *View Item* are related to classes, for instance, *Product* and *LineItem*.

Moreover, we proposed to apply navigational map and navigational context [1] to describe the wireframe which shows navigational views depending on different types of users.

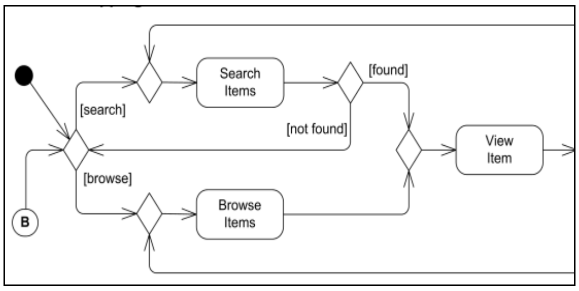

Fig. 2: Example of a partial activity diagram [7]

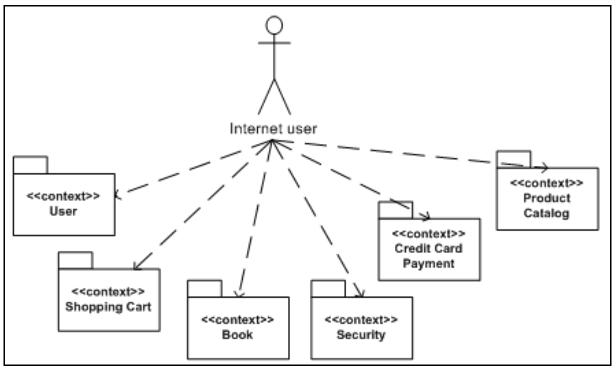
Internet user
<<context>> User
<<context>> Product Catalog
<<context>> Credit Card Payment
<<context>> Shopping Cart
<<context>> Book
<<context>> Security

Fig. 3: User navigational map for Online Shopping system

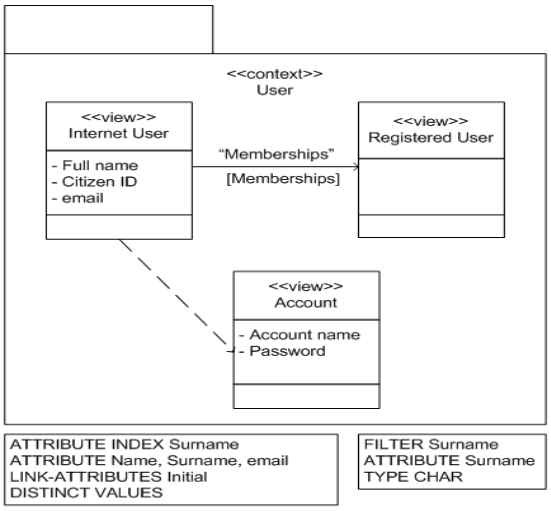

Fig. 4: *User* naviational context for Online Shopping system

The navigation model is used to model navigational requirements from the class diagram. A set of classes are clustered based on business logic. Figure 3 and figure 4 show a part of the navigational model of online shopping system. Particularly, figure 3 shows that a navigational map *User* that consists of a set of navigational contexts i.e. *user, shopping cart, book, security, credit card payment,* and *product catalog*. The *user* navigational context is elaboratedly described in terms of classes which are sterotyped with the *<<view>>*. As shown in Figure 4, it consists of classes *entitiled, internet user, registered user,* and *account*. The domain classes are elaborated and developed as a complete design class diagram.

In addition to, we apply presentational context [1] to capture the presentational requirements. As shown in figure 5, *User* presentational context shows the presentational patterns of each view.

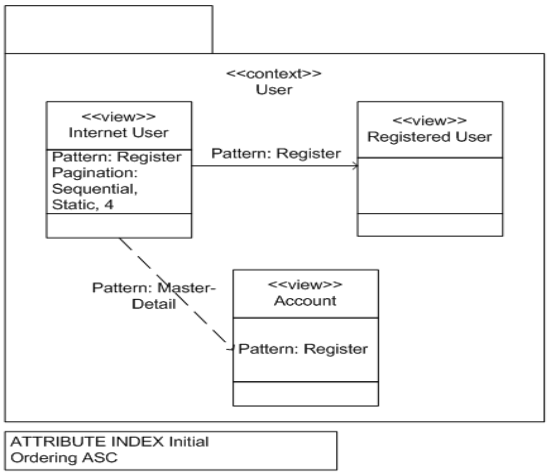

Fig. 5: *User* presentational context for Online Shopping system

As the following section, we describe the traceability relations through the case study. More specifically, the traceability relations are identified regarding the traceability types as described in previous section. The traceability relations are created based on traceability rules as described earlier.

Example of Implement traceability relations.

In this type of relation, an element e1 or a composition of elements implements an element e2 if e1 allows for the achievement of e2. As shown in Figure 6, examples of this relation exists between a composition of concepts *Web User* and *Customer* and their association implements an actor *Web Customer* in the use case diagram. Also, the relation exists between a composition of activities *Add to Shopping Cart, Update Shopping Cart,* and *View Shopping Cart* in an activity diagram and a concept *Shopping Cart* in a domain model.

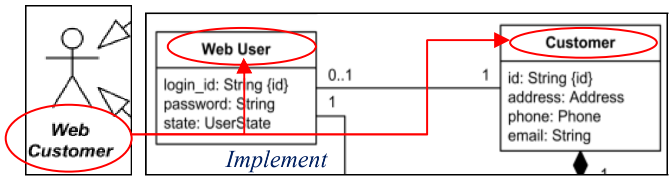

Fig. 6: An example of *Implemente* traceability relation

Example of Refine traceability relations.

This type of relation associates elements in different levels of abstractions. A refinement relation identifies how complex elements can be broken down into components and subsystems, or how elements can be specified in more detail by other elements. Thus, in this type of relation, an element *e1* refines an element *e2* when *e1* specifies more details about *e2*. In Figure 7, an example of this relation exists between the activity *Checkout* in the activity diagram and a use case *Checkout* in the use case diagram.

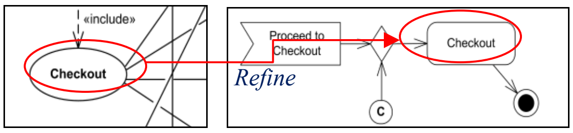

Fig. 7: An example of *Refine* traceability relation

Example of Overlap traceability relations.

In this type of relation, an element *e1* overlaps with an element *e2* (and an element *e2* overlaps with an element e1) if *e1* and *e2* refer to common aspects of a system or its domain. As shown in Figure 8, an example of this relation exists between actor *Registered Customer* in the use case diagram and the concept *Customer* in the domain model. Also, the relation exists between concept *Web User* in the domain model and the context *User* in the navigational map.

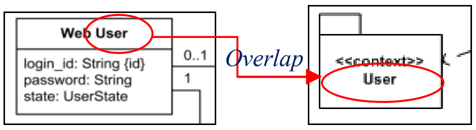

Fig. 8: An example of *Overlap* traceability relation

IV. Implementation and Experiments

A. The Extension of XTraQue

The documents are firstly represented in XML and the rules are represented in an extension of XQuery. We have extended a tool called XTraQue. We envisage the use of our tool as a general platform for automatic generation of traceability relations and support for web applications development. The tool is implemented in Java. It is used to demonstrate and evaluate our work.

The tool consists of *Traceability Relation Generator* which generates traceability relations by executing the traceability rules created by the *Rule Instantiator.* According to our approach, we have added initial rules to support traceability on web applications. The Rule Instantiator is responsible for creating a set of instantiated rules by instantiating the placeholders for the documents in the initial rules. Then, the Traceability Relation Generator is responsible for (i) executing the instantiated rules created by the Rule Instantiator; and (ii) generating traceability relations between the identified documents.

B. Experiments

We created experiments according to various scenarios in order to discover all possible traceability relations between different artefact types. There are two experiments based on different scenarios during web application development. Those are: i) There is a change on the requirements as a web application is being developed and completely done its design; and ii) There are new requirements on existing a web application.

We consider the number of documents, traceability rule templates, and instantiated rules expected and the number actually used in the experiments. It is found that E1 refers the experiment 1 and E2 refers the experiment 2. In the experiments we deployed a total of 23 traceability rule templates that have been instantiated depending on the documents used in each experiment and the traceability relations to be identified. The traceability rule templates are activated depending on the types of documents and traceability relations. Each traceability rule template is applied and instantiated to be traceabilty rules. As shown in Table 3, according to the experiment 1 (E1), the number of traceability rule templates that are expected to be applied is 17 and the number of traceability rule templates that are actually applied is 17. The traceability rules are actually instantiated as 100 rules as expected. These are applied to identify the traceability relations between 15 documents. According to the experiment 2 (E2), the number of traceability rule templates that are expected to be applied is 15 and the number of traceability rule templates that are actually applied is 15. The traceability rules are actually instantiated as 192 rules as expected. These are applied to identify the traceability relations between 20 documents.

Additionally, Table 3 shows a summary of the number of traceability relations, for each respective relation type, manually detected by the user (UT) and automatically detected by the tool (ST). The number of traceability relations being generated depends on the number of relevant documents, and traceability relation types. For example, the number of *implement* traceability relations that are manually identified in the experiment 1 (E1) is 66 as the number of *implement* traceability relations that are automatically detected by the tool in the experiment 1 (E1) is 72. The number of *overlap* traceability relations that are manually identified in the experiment 1 (E1) is 23 as the number of *overlap* traceability relations that are automatically detected by the tool in the experiment 1 (E1) is 19. Also, the number of *refine* traceability relations that are manually identified in the experiment 1 (E1) is 76 as the number of *refine* traceability relations that are automatically detected by the tool in the experiment 1 (E1) is 80.

For the experiment 2 (E2), the number of *implement* traceability relations that are manually identified is 87 as the number of *implement* traceability relations that are automatically detected by the tool is 96. The number of *overlap* traceability relations that are manually identified is 33 as the number of *overlap* traceability relations that are automatically detected by the tool is 15. Also, the number of *refine* traceability relations that are manually identified is 112 as the number of *refine* traceability relations that are automatically detected by the tool is 108.

In the experiment 1 (E1), the numbers of traceability relations that are identified by the tool and manually are 165 and 171 respectively. The number of traceability relations that are identified by both methods is 152. Also, in the experiment 2 (E2), the numbers of traceability relations that are identified by the tool and manually are 232 and 219 respectively. The number of traceability relations that are identified by both methods is 201.

TABLE 3: SUMMARY OF TRACEABILITY RELATIONS DETECTED IN EXPERIMENTS

Types of traceability relations	**UT1**	**ST1**	**UT2**	**ST2**
No. of *implement* traceability relations	66	72	87	96
No. of *overlap* traceability relations	23	19	33	15
No. of *refine* traceability relations	76	80	112	108

Additionally, we analyse the results of our experiments in terms of recall and precision rates concerning the number of traceability relations identified by the users and by the tool. The traceability relations generated by the tool in each different experiment were compared against traceability relations manually identified by users as shown in Table 3. The results give us positive evidence regarding our approach to generate traceability relations. More specifically, the results turn out with a high level of precision values for two experiments (88.888 % and 91.780%) and recall values for two experiments (86.637% and 92.121%). The average of precision is 90.334% and of recall is 89.379%.

V. CONCLUSION AND DISCUSSION

The traceability rules used in our work are considered as direct rules, which support the creation of traceability relations that do not depend on the existence of other relations. The documents are represented in XML and the rules are represented in an extension of XQuery. A prototype tool tool, together with the case study of web application development, has been used to demonstrate and evaluate our work in the experiments. The results of the experiments are encouraging with other approaches that support automatic generation of traceability relations, particularly for the domain of web applications. The results provide positive evidence about our approach to automatic generate traceability relations at a high level of recall and precision.

Additionally, the time spent during the generation of the traceability relations in the experiments used in our evaluation varies depending on the size of the documents and the number of various relation types. For example, the experiments took longer to be executed than the others that are significantly smaller. Moreover, the textual characteristics of some of the documents and the number of rules that may exist for a certain relation type also contributes to an increase of the processing time. However, the tool allows a user to select specific documents and traceability relation types to be processed in an interaction. This feature of the tool can be used to assist and control the amount of time during traceability generation.

Also, we proposed the rule-based approach for generating the traceability relations. Particularly, we proposed a rule-based approach for enabling traceability activities in the domain of web applications. A set of traceability rules is developed and used to justify the identification of traceability relations automatically. The rules are expressed in an extension of XQuery that includes extra functions implemented in XQuery and Java languages. When the conditions in a rule are verified, traceability relations are generated and represented in XML documents.

As the result, the research has demonstrated the possible situations of the use of traceability relations during the development of web applications as described below:

- Reuse: The research has found that the degree of reusing artefacts affects the cost of the development of web applications. The cost of the web application development depends on the proportion of reuse of artefacts of existing web applications. However, the poor reuse would have caused higher cost to the web application development. Traceability relations are used to assist the development by reducing the cost i.e. effort and time. Since it is a common situation that stakeholders such as software engineers need to relate the existing software artefacts to new requirements in order to assist the development of the new requirements. The research has demonstrated that traceability relations can be used to facilitate such activities in the situation.
- Understanding: The research has shown that different stakeholders, who have different experiences in the product family system development process, have different perspectives regarding to software artefacts. Traceability relations enable stakeholders to comprehend the associations between the artefacts in an easier way. Coarse-grained associations such as common and variable aspects between a web application and fine-grained associations such as ones between elements in different artefacts are illustrated though traceability relations and can facilitate the understanding of the generated documents.

REFERENCES

[1] Pastor, O., J. Fons, V. Pelechano. (2003). "OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models". IWWOST 03, Proceedinds 3rd Int. Workshop on Web-Oriented Software Technology, Oviedo, Spain.

[2] Pastor, O., J. Gomez, E. Insfran, and V. Pelechano. (2001). "The OO-Method Approach for Informatin Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming". Information Systems. Elsevier Science. Vol. 26, pp. 507-534, No. 7.

[3] Deshpande, Y, et.cl. (2002). "Web Engineering". Journal of Web Engineering. Vol. 1, No.1. 003-017. Rinton Press.

[4] Online Shopping, [online] http://www.uml-diagrams.org/examples/online-shopping-example.html. Retrieved 28 January 2014

[5] Pressman, R.S. (2001). "Web Engineering: An Adult's Guide to Developing Internet-Based Applications". Cutter IT Journal, vol 14, no. 7, pp 2-5

[6] Deshpande, Y., Hansen, S. (2001). "Web Engineering: Creating a Discipline among Disciplines". IEEE Multimedia, Special issues on Web Engineering, vol 8, no 2, pp 82-87

[7] Gotel, O., and A. Finkelstein. (1994). "An Analysis of the Requirements Traceability Problem". Pages 94 -101. the First International Conference on Requirements, England.

[8] Jarke, M. (1998). "Requirement Tracing, Association for Computing Machinery". Association for Computing Machinery. Communications of the ACM 41.

[9] The Standish Group. (1994). The Chaos Report. www.ibv.liu.se/content/1/c6/04/12/28/The%20CHAOS%20Report.pdf