Vol.2011-IS-117 No.9
2011/9/6

要件－設計－実装間に潜在する要件トレーサビリティ管理方式

宇田川　佳久†

情報システムは社会活動の安心と効率を実現するための基盤を成しており，情報システムの信頼性への要求が高まっている．情報システムの信頼性を向上するための一手法として，統一した開発プロセスによる開発プロセス全体に渡る成果物の管理がある．

本文では，要件トレーサビリティの観点からウォーターフォールモデルを詳細化したモデルを提案する．次に，要求，設計，実装工程にまたがるトレーサビリティについて，Web アプリケーションを例にして論じる．モデリング言語としては，国際的に普及している SysML を採用する．SysML で記述された要件項目と設計項目間のトレーサビリティを検索するためにベクトル空間検索モデルの改良について論じる．続いて，設計とソースプログラム間のトレーサビリティギャップを軽減するため，SysML の記述要素をソースプログラムに変換するルールについて述べる．本実験結果は，これらの技法の組合せが，要件，設計，実装間に潜在するトレーサビリティを管理する上で有効であることを例証している．

Managing latent requirements traceability among requirements, design documents and source codes

Yoshihisa Udagawa†

Demands of high quality software system increase as the software system plays an important role in realizing safety and efficiency of various social activities. An integrated framework to manage artifacts thorough a software system life cycle is recognized as an effective method to keep the quality.

This paper proposes a refinement of the waterfall model in terms of requirements traceability. Next we discuss an approach to manage traceability in requirement, design and implementation phases using a Web application development as an example. SysML (Systems Modeling Language) is used as a modeling language since it provides many useful constructs for describing requirements and design artifacts. We discuss an augmented vector space model featuring a set of closely related terms called "family terms" to improve accuracy of information retrieval. We devise translation rules to reduce gaps between SysML diagrams and source code segments. The initial results indicate that the combination of SysML, the information retrieval and the translation rules leads to a promising solution.

1. Introduction

Software system development consists of different phases such as requirement definition, high-level design, detailed design, implementation and testing etc.[1][8], which result in producing several artifacts including requirements in natural languages, design model elements in a semi-formal/formal models and source codes in programming languages. These artifacts are typically described in different languages at different levels of abstractions, but still they are certainly related each other.

Requirements traceability enables participants in a project to better understand the relationships among various artifacts and contributes to clear and consistent documentation. There has been a growing interest in requirements traceability in the software engineering research community [2][8], industry[1] and academia[17].

In spite of many advances, requirements traceability has many research issues before apply it to industry fields. Main adverse factors for managing requirements traceability can be summarized as follows [14]:

(1) Artifacts in system development are written in different languages (natural languages, graphical modeling languages, programming languages etc.);
(2) They are described at various abstraction levels (requirements, designs, implementations etc.);
(3) Large amount of traceability information and lack of adequate tool support to create, retrieve and maintain the traceability information.

The aim of this paper is to discuss a model framework concerning artifacts in the requirement, design and implementation phases, and traceability among them. There are many ways to document requirements and software designs. In considering globalization of software industries, use of international standards is necessary to promote communication among several companies and even different sites across the globe. Among several global standards for system modeling, we have employed SysML (the System Modeling Language) [11] , a standard of the Object Management Group, since it is designed to provide simple but semi-formal graphical constructs for modeling elements in a wide range of engineering fields. With adequate tool support, this means that SysML has a potential for coping with (1) ~ (3) above. It should be noticed that SysML provides several relationships concerning requirements traceability, i.e. the refine relationship, the verify relationship, the generic trace relationship etc.[13]

Several researchers are working on the traceability model based using SysML. Maeder et

† 東京工芸大学工学部コンピュータ応用学科
Tokyo Polytechnic University, Faculty of Engineering, Department of Applied Computer Science

al. [10] proposes traceability link model for the development processes through requirements and implementation, especially for the case of changing requirements. A set of rules for the verification of the traceability links are developed. But the set of rules is limited to checking the pure existence of traceability links. Soares and Vrancken [16] propose a model-driven approach to requirements engineering based on SysML. Their research covers requirements and use case diagrams of SysML.

A number of research efforts have been carried out to automatically recover traceability among software artifacts. Huffman Hayes et al. [7] use a vector space model augmented by a thesaurus. They develop a prototype tool, called RETRO. The results show that the augmented vector space model outperforms the classical vector space model in recall while keeping precision at the same level. Kagdi et al. [9] propose an approach that combines information retrieval technique based on a vector space model for coupling programming entities such as comments and mining software repositories for capturing patterns of change history of the entities. There are many other excellent researches, however, we do not mention about them due to space limitations.

Though SysML provides sufficient modeling facilities covering requirements and design phases in terms of the waterfall development methodology, it fails to support facilities for relating design models to program codes, which is out of the scope of the SysML standard. We devise translation rules to map the SysML activity diagrams to comments in source programs, which contribute for reducing gaps between design models and implementation of the models [6].

The rest of this paper is organized as follows. In Section 2, we discuss a framework for software engineering, which can be seen as a refined waterfall model in the "V" shape in terms of traceability. In Section 3, we present SysML models for a Web application to examine how SysML diagrams are related thorough the <<refine>> traceability relationship. In Section 4, we discuss an augmented vector space model to automatically recover traceability and some results obtained from our case study. In Section 5, we discuss the rules to translate a SysML activity diagram to internal source code documentations. We summarize our approach and future work in Section 6.

2. A Framework for System Engineering

Pohl [12] has defined the process of developing a requirements specification as a movement in the three almost orthogonal dimensional spaces, i.e. representation dimension, specification dimension and agreement dimension. The process begins with different personal views, little system understanding and informal representations, ending up a complete and formally defined specification with sufficient agreements. He discusses the requirements pre-traceability [4] in the three-dimensional space.

The same idea can be applied to the requirements post-traceability of software development processes beginning from the requirements definition down to design, implementation, testing, operation and maintenance [1][8]. The proposed framework is shown in Figure 1 by means of the classic "V" model. It is based on the waterfall development model since it is widely used in industry fields and it provides foundation for other development models such as the spiral model, incremental model etc. Our refined V model includes three kinds of traceability relationships, i.e. decomposition, version and composition relationships.

The decomposition relationships associate artifacts in the left wing of the V model. Needs and benefits are decomposed into the requirements. The requirements are further decomposed into high-level design, and then detailed design. The decomposition continues until all the design entities are implemented into source code statements.

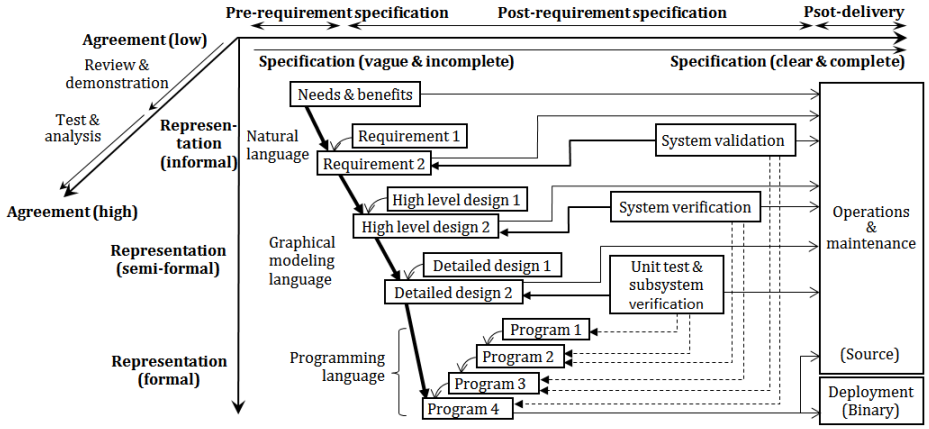

Figure 1. A refined “V” model

Conceptually, the decomposition relationships, i.e. traceability between higher and lower abstraction levels, forms 1:N relationship, because a component in a higher abstraction level is decomposed into a set of components. However, things are much more complicated. Several requirements are usually implemented in some software modules or functions, yielding N:M relationships. The trick is "granularity" of related entities. There are controversial issues in view of data modeling [3]. However, we stop the further discussion

mainly because we need enough evidences for it.

The version relationships deal with traceability over time. They are defined on the same sorts of artifacts enabling us to trace between the different versions of the artifacts. Conceptually, the version relationships form 1:1 relationships. However, we find lots of N:M relationships in documents in industry fields.

The composition relationships in the left wing of the V model are rater complex. One test case covers many design components, thus there is a 1:N relationship between a test case and a set of design components. In addition, one test case is related to a set of source code statements not only source codes before test but also corrected source codes reflecting the result of the test case, i.e. after bug fix.

3. Modeling Login Function in SysML

3.1 Traceability Relationships in SysML

SysML (Systems Modeling Language) is a general-purpose modeling language for systems engineering applications developed within the Object Management Group (OMG) consortium [11]. It is particularly effective in specifying requirements, product architectures as well as their behavior and functionalities. SysML provides modeling constructs to represent text-based requirements and relate them to elements in other SysML diagrams. Several requirements relationships are available. These include relationships for defining a requirements hierarchy, refining requirements, satisfying requirements, verifying requirements, etc. Due to space limitations, we only focus on the requirement containment relationship and the refine relationship in the rest of this paper. Since samples in this paper are general ones, the usage of the two relationships is easily applied to other several relationships specified in the SysML.

In designing complex software systems, it is common to have a hierarchy of requirements. For instance, high-level requirements may be decomposed into more detailed requirements, thus requirements are generally formed in a hierarchy. The SysML requirement containment relationship facilitates to model high-level requirements into more simple ones as a hierarchy. The refine relationship describes how a model element (or a set of model elements) can be used to further refine a given requirement. This relationship is represented by the stereotype <<refine>>.

3.2 Login Function for a Web System

A login function to a business application is inevitable not only to identify a user of the application but also to protect the application from unauthorized access. Concept of the login function is fairly simple, i.e. determining a valid user id and password by looking up a table in a database that contains valid pairs of user ids and passwords. But in practice, a designer have to consider support functions such as password change, password expiration management and password setting rules.

We now develop a Web application that manages project artifacts with focus on requirements traceability (the theme of this paper). Users of the Web application consists of two classes: managers and designers for a project. A user can participate in one or more projects. The user may be a manager in some projects and a designer in other projects. The role of the user in a project is supposed to be stored in a database. Details of database are described in Subsection 3.4 of this paper.

A layout of the login screen is shown in Figure 2. The user who wants to login enters a user id and a password in the corresponding fields, in addition, choices a project name in a list by clicking the arrow icon.

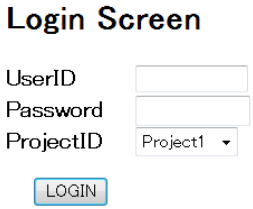

Figure 2. A layout of the login screen

3.3 Requirements on Login Function

As mentioned in the previous subsection, requirements on the login function should be specified in the context of operation and maintenance of user ids and passwords. The essential requirements are summarized in the following.

(1) Data items requirement: User id, password and project id should be entered.
(2) Password management requirement: In the security point of view, a password should be robust in the combination of characters and the period of password changes as well. This requirement is further decomposed into a password setting policy, a password expiration policy and a password change policy, respectively.
(3) Login failure requirement: In case of login failure, a notification must be displayed.

A SysML requirement diagram for the requirements above is shown in Figure 3. Note that each of requirements is tagged by the <<refine>> relationship relating to other SysML design models, which are described in the following subsections. Note that the <<refine>> is defined manually by experts designing the login function as true-links [7]. All of the <<refine>> relationships are retrieved automatically by the proposed retrieval methods. The details are discussed in Section 4 of this paper. A string proceeded by the # character, e.g. #UserID,

means a key term, which play an important role to define a traceability since in most of projects in industry the key terms are used in a consistent manner.

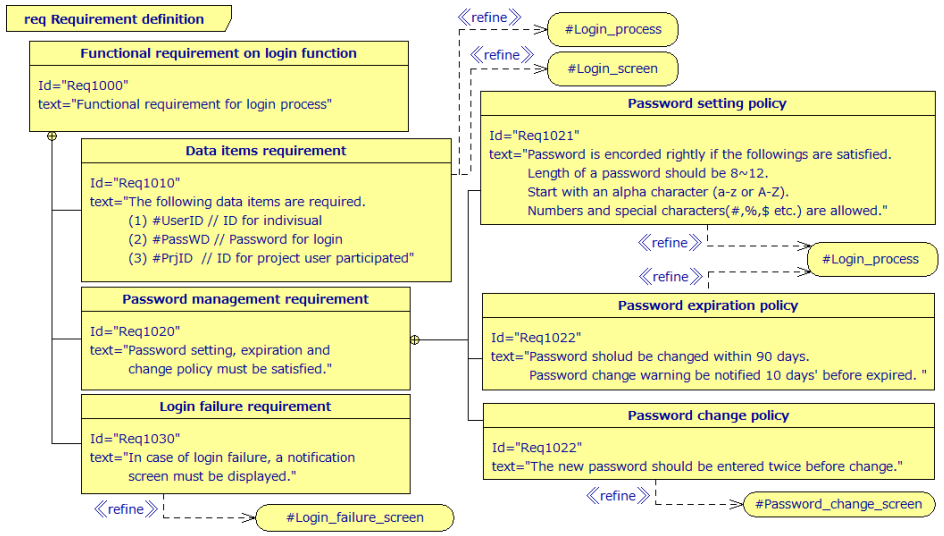

Figure 3. A SysML requirement diagram

3.4 Database Design Model

The aims of database design are to describe the data items, relationships and constraints on the data items for the application, e.g. user id should be 8-12 characters in length. Since there are established data modeling techniques and excellent articles lecturing on them [3], omitting the details of data modeling, we only present the result of the database design.

The entity "user" and "project" are identified. The entity "user" has attributes on user id, password, user name, and a date last updated, etc. The entity "project" has attributes on project id, a name of the project, members of the project and his/her role. The entity "user" and "project" are translated into the relation R_user and R_project, respectively. Since a user can participate in one or more projects, there are one-to-may relationship between the entity "user" and "project". The one-to-may relationship between the entity "user" and "project" is realized by the UID (user id) as a foreign key in the table R_project as shown in Figure 4 which is depicted by the UML class profile for data modeling [15].

Note that the identifiers or names of the data items defined this design phase should be referenced consistently by the subsequent phases as key terms, which makes us possible to define traceability implicitly or automatically among different design models.

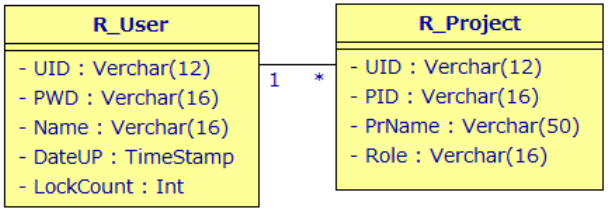

Figure 4. A table structure

3.5 Screen Transition Model

The state machine diagram represents behavior as the state history of an object in terms of its states and transitions. The activities that are invoked during the state transition are specified by an associated event. Since a screen in the Web application transits by an associated event, e.g., clicking button and the result of a process, screen transitions can be represented by a SysML state machine diagram where screens corresponds to states.

Figure 5 shows the overall screen transitions depicting screens, events and relationships with the stereotype <<refine>> that are defined by experts as true-links which are retrieved automatically by the proposed retrieval methods. Again a string proceeded by the # character, e.g. #Login_screen, means a key term.

3.6 Process Model

The SysML activity diagram is a graph-based diagram showing flow of control, thus we use it to define processes that are associated with the screen transitions. Figure 6 shows the login process that is fired when the login button on the login screen is clicked. This fact is traceable automatically by the identifier "#Login_process" premising that the consistent technical names shall be defined and used.

The body of the process is described in the central lane, while the login screen is described in the left lane and the next screens in the right lane. Again the <<refine>> relationships between the requirements and the login process are defined manually by the experts as true-links.

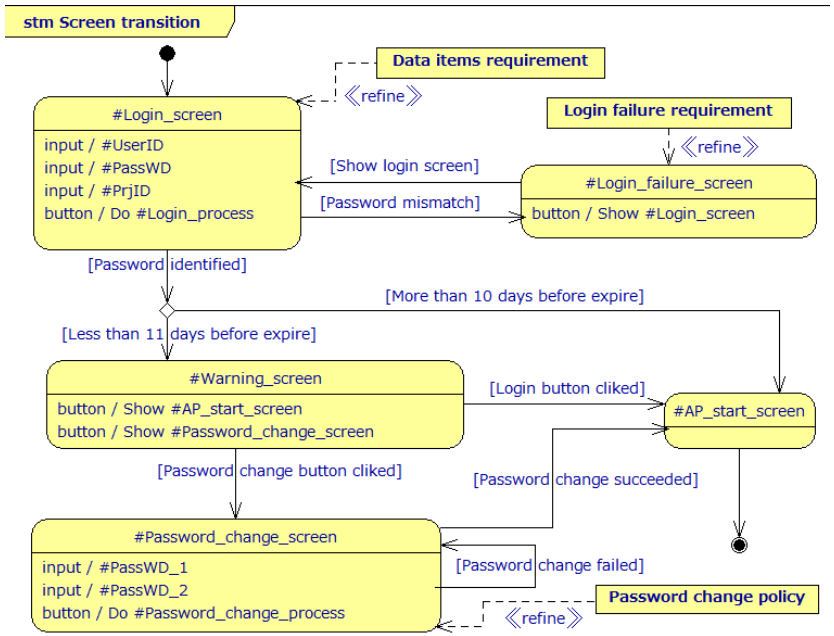

Figure 5. A screen transitions diagram

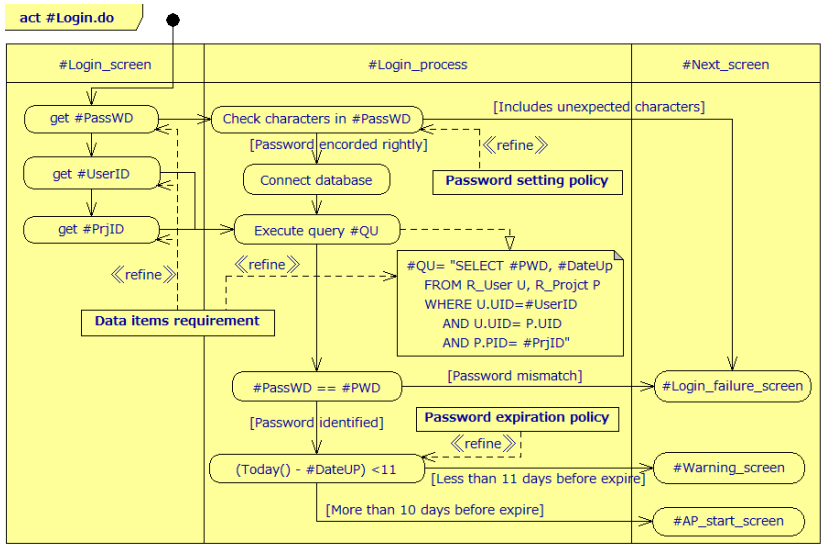

Figure 6. A process diagram

4. Recovering Requirement Traceability

4.1 The Vector Space Model

Our approach to recover requirements traceability is to augment the vector space model for improving the results of information retrieval. Briefly, the process consists of two steps, i.e. translating a SysML diagram into vector representations and computing similarities.

4.2 Analyzing SysML Diagrams

At first, a document analyzer prepares the document for retrieval. Documents are indexed based on a term that is extracted from the documents themselves. Note that each term has "document identifiers" indicating the document in which the term has been extracted. Extracting terms and indexing documents are done in the following steps:

(1) SysML diagrams are divided into documents for the information retrieval. The process works in accordance with the given mapping rules between SysML diagrams and documents for retrieval.
(2) Each document is tokenized. All capital letters are transformed into lower case letters.
(3) Terms preceded by the # character are processed as key terms. In addition, terms connected by the character "_" consisting a key term are divided into each of the main terms.
(4) Stop-words (i.e., words that are not useful for the purposes of retrieval such as articles, preposition, numbers, etc.) are removed. Note that our document analyzer is sensitive to a SQL statement. For example, the preposition "from" in a SQL statement is extracted as a main term in this study.
(5) The remaining words are stemmed to ensure that different forms of the same term are treated as the same one, i.e. converting plurals into singulars, transforming conjugated forms of verbs into infinitives.

(6) The key terms and the main terms are stored in the repository.
(7) A vector representation of the document is created and stored in the repository.

4.3 Vector Space Model and its Augmentation

Given a set of documents *D*, a document d_j in *D* is represented as vectors of term weights,

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{N,j})$$

where *N* is the total number of terms in the document and $w_{i,j}$ is the weight of the *i*-th term. We use a well-known metric called *tf-idf* to compute $w_{i,j}$ [7].

A user query is also converted into a vector.

$$q = (w_{1,q}, w_{2,q}, \ldots, w_{N,q})$$

The similarity between document d_j and query q can be computed as the cosine of the angle between vectors d_j and q in the *N*-dimensional space:

$$\text{Similarity}(d_j, q) = \cos(d_j, q) = \frac{\sum_{i=1}^{N} w_{i,j} * w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} * \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

We have augmented the vector space model by using the document identifiers and a set of closely related terms for a similarity computation as follows.

(1) Drop terms in accordance with the context of traceability recovery based on the document identifiers. For example, when a user intended to recover traceability between the requirements and the designs, we can drop terms as follows:
- terms having only the document identifier "requirement" should be dropped from vectors concerning the requirement diagrams,
- terms having only the document identifier "design" should be dropped from vectors concerning the design diagrams.

Theoretically, terms only found in the requirements doesn't match those terms only used in designs and vice versa.

(2) Closely related terms, called family terms, are treated as a set. For example, if main terms {A, B, C, D} are found in a query then they are treated as family terms. The family terms are applied to documents in turn. That is if a document contains {A, B, C} then the term D is inferred to be contained in the document with some possibility. As the initial study, the possibility of an inferred term is set to 1, meaning the inferred them is included definitely. We also assume that a set of family terms is triggered when more than half of the terms are included in a document vector.

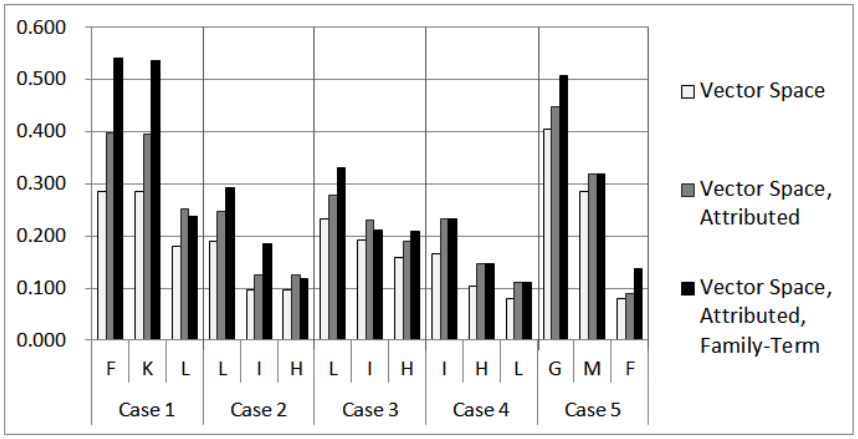

Figure 7. Similarities of the 1st, 2nd and 3rd candidates

4.4 Tracing Requirements to Designs

We have applied the augmented vector space model to recover traceability relationship in the three diagrams shown in Section 3. 82 main terms including 16 key terms attributed by the document identifiers have been extracted from the SysML diagrams. The three diagrams are divided into 13 documents for the vector-space-based information retrievals. Our tools generated 82 x 13 term-by-document matrix.

Each of five documents translated from the requirement diagram in Figure 3 is used as query vectors. Other eight documents are translated from Figure 5 and 6. We have carried out five kinds of retrievals from "requirements to designs" in the three modes, i.e. the simple vector space mode, the mode using the document identifiers, the mode using the family terms.

Similarities of the 1st, 2nd and 3rd candidates are depicted in Figure 7. There are several strategies for selecting true-links. The constant threshold is the straightforward one. Our vector model shows the most preferable result when the threshold is around 0.23. Traditionally the accuracy of results by information retrieval is assessed in recall and precision [7]. Roughly, the result of our information retrieval achieves recall of 100% with 87.5% (=7/8) precision. Figure 8 summarizes the results of the information retrieval from requirements to design diagrams with the similarity values computed in our augmented vector space model.

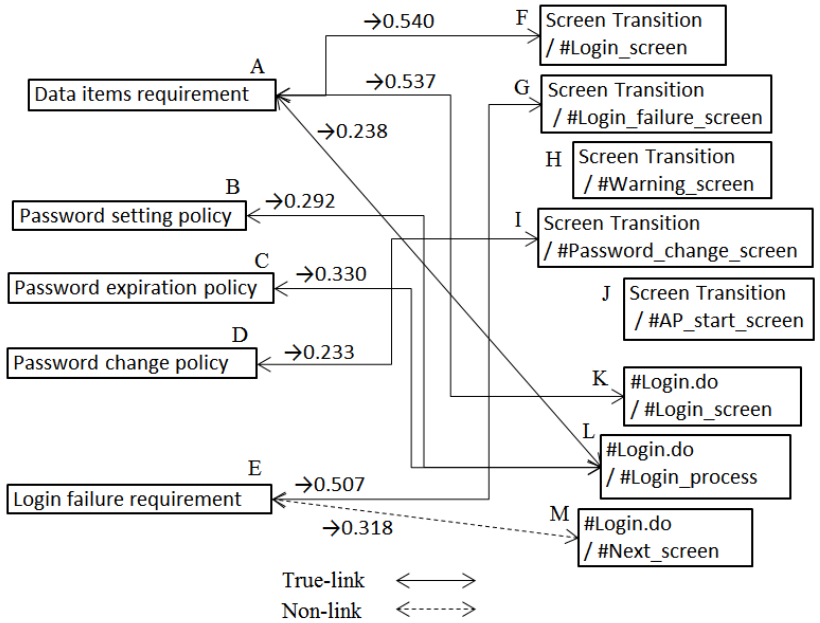

Figure 8. A graph summarizing the results

5. Bridging Design and Implementation

In the MDE (Model-Driven Engineering), requirements are created, refined into design models and eventually translated into executable programming languages [5]. However, the last transformation, i.e. from design models to programming languages, is often done in a weak systematic manner [6].

We now describe shortly the rules to translate a SysML activity diagram to internal source code documentation where to begin implementing source programs.

Rule 1: Translation proceeds from upper left to lower right.

Rule 2: Translation ends if all the model elements processed at least once.

Rule 3: A sequence of actions, each of which is connected only one flow of control, is translated into a sequence of statements.

Rule 4: An action followed by more than one control flow is translated into an If-statement with branches for each flow of control. A statement described in the action corresponds to comparison expression in the If-statement. The guard of a flow of control is translated into //+ GuardName +//, which corresponds to the result of the comparison expression.

```
1: #QU= "SELECT #PWD, #DateUp
2:         FROM R_User U, R_Projct P
3:         WHERE U.UID=#UserID
4:           AND U.UID= P.UID
5:           AND P.PID= #PrjID"
6: //+ #Login_screen +//
7: { get #PassWD;}
8: { get #UserID;}
9: { get #PrjID;}
10: //+ #Login_process +//
11: If (Check characters in #PassWD) {
12:   //+ Includes unexpected characters +//
13:   { #Login_failure_screen;}
14: } else {
15:   //+ Password encorded rightly +//
16:   { Connect database;}
17:   { Execute query #QU;}
18:   If (#PassWD== #PWD) {
19:     //+ Password mismatch +//
20:     { #Login_failure_screen;}
21:   } else {
22:     //+ Password identified +//
23:     If (Today() - #DateUp<11) {
24:       //+ Less than 11 days before expire +//
25:       { #Warning_screen;}
26:     } else {
27:       //+ More than 10 days before expire +//
28:       { #AP_start_screen;}
29:     }
30:   }
31: }
```

Figure 9. A document translated from Figure 6

The //+ GuardName +// links a SysML model and a target program, so it should be managed by a syntax check tool to make sure the //+ GuardName +// is kept unchanged during the programming stage. A SysML activity diagram in accordance with the above rules can be successfully translated into an internal source code documents to bridge the SysML diagram with the corresponding target program. Figure 9 shows the result of the translation of the SysML diagram in Figure 6.

Once traceability between design diagrams and the source code statements is established, developers can take advantage of IDEs (Integrated Development Environment) for consistent management of artifacts leading to a high quality software development.

6. Conclusion and Future Work

An integrated framework to manage artifacts thorough a software system life cycle is even more required as demands of high quality software system increase. There are on the order of thousands pages of documents and hundred thousand lines of source codes in a large-scale software development project. Not to mention, we need a computer support to deal with this size of artifacts.

At first we propose a refined V model in view of the requirements traceability as a "straw-man proposal" with some points of issues. SysML, a standard of OMG, is used as a modeling language. Several experiments are carried out to examine the requirements traceability among requirements, designs and implementations using the login function for a Web application as an example. Our augmented information retrieval techniques yield promising results for recovering traceability among SysML diagrams. We also discuss a set of rules for translating a SysML activity diagram into internal source code documentations which provides a base of implementing source codes in a target programming language while keeping links to SysML diagrams.

Our research issues in the near future include, (1) improving and examining the accuracy of the augmented information retrieval model, (2) refining translation rules between SysML diagrams and internal documentations, (3) developing tools to support traceability and to maintain consistency among artifacts throughout a software development life cycle.

References

[1] California Department of Transportation, "Systems Engineering Guidebook for ITS, Ver. 3.0", 2009. http://www.fhwa.dot.gov/cadiv/segb/

[2] CMMI Product Team, "CMMI for Development, Ver. 1.2 --- Improving processes for better products," CMU/SEI -2006-TR-008, Software Engineering Institute, Carnegie Mellon University, http://www.sei.cmu.edu/, 2006.

[3] Connolly T. M. and Begg C.E., "Database Systems: A Practical Approach to Design, Implementation and Management", Addison-Wesley, 5rd Edition, 2010.

[4] Gotel O. and Finkelstein A., "An Analysis of the Requirements Traceability Problem", Proc. of the 1st Inter. Conference on Requirements Engineering, 1994, pp. 94-101.

[5] Hailpern B., Tarr P., "Model-driven development: the good, the bad, and the ugly", IBM Systems Journal, Vol.45, No.3 2006, pp.451-461.

[6] Heidenreich F., Johannes J., Seifert M. and Wende C., "Closing the Gap between Modelling and Java", Software Language Engineering, Lecture Notes in Computer Science, Vol.5969, 2010, pp.374-383.

[7] Huffman H. J., Dekhtyar A., Sundaram S.K., and Howard S., "Helping Analysts Trace Requirements: An Objective Look," Proc. of the 12th IEEE International Requirements Engineering Conference, Kyoto, Japan, 2004, pp.249-261.

[8] INCOSE, "Systems Engineering Handbook Ver. 3 - A guide for system life cycle processes and activities", June 2006, pp.1-185.

[9] Kagdi H., Gethers M., Poshyvanyk D, and Collard M. L., "Blending conceptual and evolutionary couplings to support change impact analysis in source code," Proc. 17th IEEE Working Conference on Reverse Engineering, October 2010, Boston, MA, USA, pp.119-128.

[10] Maeder P., Philippow I., Riebisch M., "A Traceability Link Model for the Unified Process", Proc. of the 8th ACIS Inter. Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007, pp.700 - 705.

[11] Object Management Group, "OMG Systems Modeling Language Version 1.2", June 2010, http://www.omgsysml.org/.

[12] Pohl K., "PRO-ART: Enabling Requirements Pre-Traceability", Proc. of the IEEE Intl. Conference on Requirements Engineering (ICRE), 1996, pp.76-84.

[13] Ramesh B. and Jarke M., "Toward reference models for requirements traceability," IEEE Transactions on Software Engineering, 2001, Vol.27, No1, pp58-93.

[14] Rilling J., Charland P. and Witte R., "Traceability in Software Engineering - Past, Present and Future", CASCON 2007 Workshop Report, IBM Technical Report, Oct. 2007.

[15] Scott W.A., "A UML Profile for Data Modeling", 2009, http://www.agiledata.org/essays/umlData ModelingProfile.html

[16] Soares M., Vrancken J., "Model-Driven User Requirements Specification using SysML", Journal of Software, Vol 3, No 6 2008, pp.57-68.

[17] Spanoudakis, G. and Zisman A., "Software Traceability: A Roadmap", Handbook of Software Engineering and Knowledge Engineering. World Scientific Publishing, 2005, pp.395-428.