International Journal of Advances in Software Engineering & Research Methodology
Volume 3 : Issue 2 [ISSN 2374-1619]
Publication Date : 30 December, 2016

A traceability tool for model-based development dealing with uncertainties

Go Hirakawa, Kenji Hisazumi, Ryoichi Nagatsuji, Tsuneo Nakanishi, Akira Fukuda

***Abstract*—An architecture that considers a system life cycle from the designing stage to the operational stage is needed for solving problems in large-scale, complicated system developments. It is necessary to establish traceability among design assets, development assets, and operational assets through a life cycle of the system in order to introduce this process into an actual development situation. In this paper, we introduce the MetaIndexTools, a traceability tool for model-based development that can deal with uncertainties and consider the life cycle of the system.**

***Keywords*—Life cycle–oriented, uncertainty, model-based development**

I. Introduction

Recently, the development scale of systems has kept increasing and becoming more complicated. The target domain treated by a system is also expanding. Such systems often require a change of specifications by a change in social needs and environmental conditions. Our research group is working on introducing a concept such as uncertainty into system development to address such situations. We believe that applying an architecture that considers a system life cycle from the designing stage to the operational stage can settle this problem. Here, by uncertainty, we mean a specification that cannot be defined uniquely until a system has entered the operational stage. We proved that a description of uncertainty is possible to use for the model description using GSN, which stands for Goal Strategy Notification. We also proposed a method which fixes uncertainties using operational information [1]. However, it is necessary to establish traceability among design assets, development assets, and operational assets through a life cycle of the system to introduce this process into an actual development situation. Conventional model-based development tools do not support traceability among assets, including operational assets. A tool for model-based development that is easy to understand and customize is required. In this paper, we propose MetaIndexTools, which is the traceability tool for the model-based development that can deal with uncertainties and consider the lifecycle of the system. MetaIndexTools is light-weight, customizable for various development methodologies, and available for many platforms. In our research, we will investigate the effectivity of the lifecycle-based development using MetaIndexTools in actual development sites. We introduce related works in Section II, declare a data model for the MetaIndex in Section III, implement MetaIndexTools with the MetaIndex model in Section IV, examine a case study using a sample application using MetaIndexTools in Section V, and conclude in Section VI.

II. Related Works

As a system becomes complicated, traceability in system development is of vital interest. A large number of studies on requirement traceability and traceability in MDD, in particular, have been performed [2]. In other research, traceability links are expressed in metamodels, but there is no standard notification. To correspond to such situation, “traceability metamodeling language” (TML) is proposed by Drivalos et al. [3], but there is no tool to handle it visually.
On the other hand, the MDD tool which has traceability feature has been developed, and it is possible to mention pure::variants and Astah as representative examples. The pure variants [4] have the management feature, which can manage traceability among feature models and development assets, and the automatic code generation feature, which fully supports a variant model. Pure variants are built on the premise of a development process uses a feature model and a variant model. Astah [5] is a UML-based development tool for general purpose; it supports traceability among nodes based on the sysML model. However, these tools need specific development processes. It is hard to secure traceability through the life cycle of a system using conventional tools in actual development environments because every development site adopts different processes that depend on development objects and development scales.

III. A Data Model for the MetaIndex

A. *Requirements*

We need to define a meta-model that can be described following the index before tool development. The index can handle the software development properly. It is possible not to depend on the development property classification and the attribute treated by each tool for index and to treat them equally. The index can be described by the binary relations between the index and group relations, and it can map the paradigm of various development methods, which makes it possible to indicate use cases to actual developers. The MetaIndex model can adapt to various development processes by describing each above-mentioned concept as a meta-model.

B. *The Data Model*

To enable the security of traceability for which we do not depend on a specific development process, we propose a data model for our system. Figure 1 shows a class diagram of our proposed model. The model consists of two parts: methodology and index. In the diagram, the blue classes indicate methodology-related classes, and cream-colored classes illustrate index-related classes.

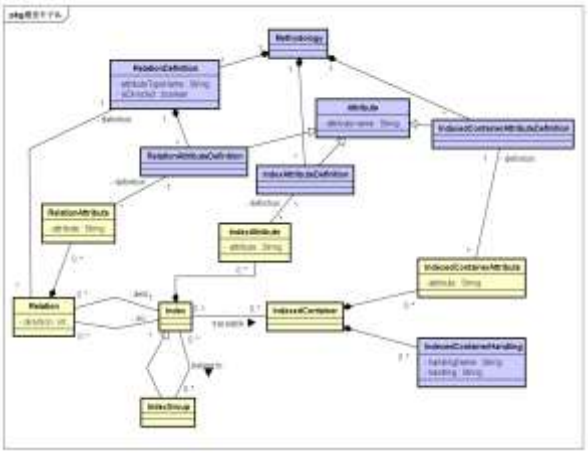

Figure 1 class diagram of MetaIndex model

To define the methodology, we should make instances of methodology classes. It is enough to make a set of instances of the classes for each methodology. For example, when we want to introduce a feature diagram for the software product line methodology [6], we should generate instances of them to define the feature diagram. If we need to define another methodology, such as use-case driven development [7], we should make another set of instances of them. The methodology consists of definitions of indexes, relations, and indexed containers. The index is a data structure that can help us trace parts of software artifacts. The index has certain attributes. The relation is an association between indexes. The indexed container is a part or the entirety of software artifacts associated with one or more indexes, and it is traceable from an index to one or more indexed containers. The indexed container has some handlers that are invocated by tools such as highlighting. The class diagram presents the definition of attributes of indexes (IndexAttributeDefinition), relation (RelationDefinition) and its attributes (RelationAttributeDefinition), attributes of index containers (IndexContainerAttributeDefinition), and handlers (IndexedContainerHandling). To make a real index model, such as a feature model, we instantiate cream-colored index classes. Instances of the index classes are generated for each diagram. The class diagram shows indexes (Index), attributes of indexes (IndexAttribute), relations (Relation), attributes of relation (RelationAttribute), groups of indexes (IndexGroup), indexed containers (IndexedContainer), and attributes of indexed containers (IndexedContainerAttribute). All attributes and handlers are defined by the methodology definition.

IV. Development of MetaIndexTools

The model-base development tool and MetaIndexTools, which treat a MetaIndex model, are needed to manage development property based on such model. MetaIndexTools requires the use of the model editor feature, which treats indexes as meta models. The traceability editor function achieves traceability among indexes and other assets.

In this research, we investigate whether the MetaIndex method is effective in actual development scenes that adapt various development methods. Therefore, MetaIndexTools has a number of non-function requirements; it needs to be:

1. Applicable to various development methods;
2. Adaptable to the development of the various scales, from the small to the large scale;
3. Able to collect metrics at the different stages of development;
4. Simple to understand and of a small learning cost;
5. Lightweight and easy to use;
6. Possible to use in multiple platforms.

There is no model-based development tool that meets requirement 1, and it is hard to acquire arbitrary metrics. Therefore, we plan to develop a prototype of MetaIndexTools and apply it to various development projects at and outside our university. By getting feedback from actual development, we improve MetaIndexTools to meet requirements 2, 4, and 5. We have developed MetaIndexTools, which has index operation, traceability making, and traceability confirmation functions. MetaIndexTools is applicable to various development methods, and it is necessary to customize it for these development methods. We achieved customization by correspondence with a data model based on the MetaIndex model defined in Section II. MetaIndexTools should be multi-platform because the present development environment, represented by OS, programming language, and development tools, is varied. We achieved multi-platform by developing it as an electron application using web-based technology, html5, and node.js. We also achieved cooperation with third-party development tools by defining REST API that handles indexes based on the MetaIndex data model. The outline of MetaIndexTools is shown in figure 2.

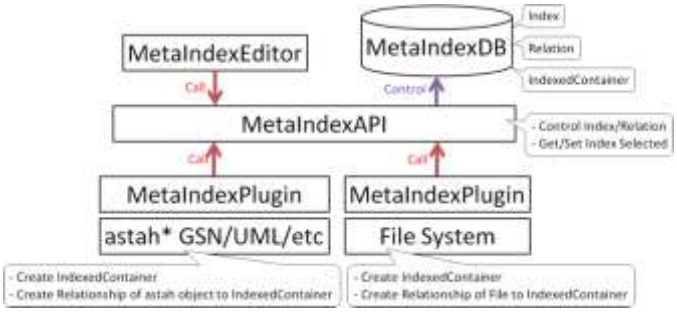

Figure 2 Block diagram of MetaIndexTools

MetaIndexTools consists of MetaIndexEditor, which performs the operation and visualization of the index and relative operations among the indexes; MetaIndexPlugin, which performs operations of IndexedContainer and relative operations with the index and IndexedContainer on the third-party tools; MetaIndexDataBase, which stocks the index based on the MetaIndexModel; and MetaIndexAPI, which controls the index operations from MetaIndexEditor and MetaIndexPlugin.

A. *MetaIndexDatabase*

We proposed the MetaIndexModel in Section III, which could deal with development assets in different development

methods. Because the development assets we treat are various, we need MetaIndexDB, which can manage these assets universally, so we designed and implemented a DB scheme belonging to the DataModel defined in Section III. MetaIndexDataBase is a database that stocks the elements of the MetaIndexModel. We have designed this DataBase to be independent from any other data and modeling tools, including MetaIndexEditor.

B. *MetaIndexAPI*

The MetaIndexModel has the ability to add traceability between the index and the assets. However, MetaIndexTools does not have the ability to manage entire assets, which depend on the development technique. Therefore, we have achieved an operating MetaIndexModel by using various tools, including MetaIndexEditor, and by determining MetaIndexAPI. MetaIndexAPI is the external API, which operates the index on MetaIndexDataBase corresponding to the MetaIndexEditor and MetaIndexPlugin.

API operates Index and Relation add Index delete Index modify Index add Relation delete Relation modify Relation.

To visualize traceability with a different tool, API also has Index selection features. Index selection is selected.

We implemented MetaIndexAPI as RESTAPI and enabled these MetaIndexModel operations from various external tools by HTTP communication. We used Ruby and Sinatra for implementation.

C. *MetaIndexEditor*

MetaIndex can treat suitable data models for various development methods, but the tool that can treat such a meta-model does not exist. Therefore, we developed MetaModelEditor at first, which was a general-purpose model editor. Then, we built the data model based on FeatureModel and expanded that to treat traceability. Finally, we applied it to MetaModelEditor and produced MetaIndexEditor experimentally. MetaModelEditor is a general-purpose model editor and can treat a model according to various meta-models described in eCore [8]. MetaModelEditor consists of an editing meta-model feature based on eCore description and editing, a model feature according to the meta-model. MetaIndexTools should be multi-platform because the present development environment, represented by OS, programming language, and development tools, is varied. We achieved multi-platform by developing it as an electron application using web-based technology, html5, and node.js. MetaIndexEditor is the modeling tool, which performs the operation and visualization of Index and relative operation among the Indexes. We developed MetaIndexEditor by improving the general-purpose meta-model editor developed by Kyushu University [9]. MetaIndexEditor is the domain-specific model editor that provides editing specific model features, using the meta-model described in eCore. As a trial, we designed a data model indicated in figure 3, which can treat Index as a feature model and make a traceability description among Index and other development assets.

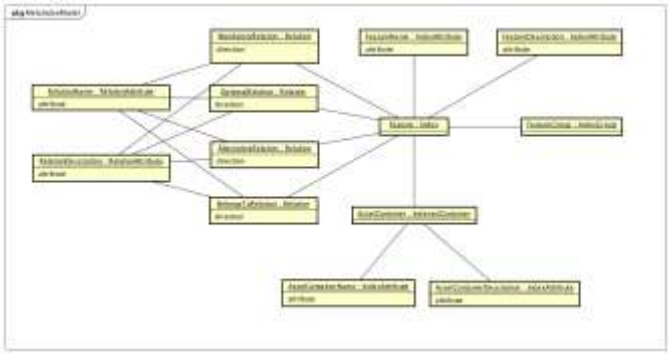

Figure 3 a data model based on FeatureModel

We defined Feature as an instance of Index, which has the name, description, and definition of Group. Also, we defined AssetContainer as an instance of IndexedContainer in conjunction with Feature for traceability with development assets. In general, FeatureModel can indicate a choice among the types of associations between features, which can be mandatory, optional, and alternative. So, we defined these choices, each as instances of Relation and also defined Description as the name in Relation. We produced an experimental MetaIndexEditor to describe this MetaIndexModel in Ecore and input it into MetaModelEditor. Figure 4 is an actual screen of MetaModelEditor.

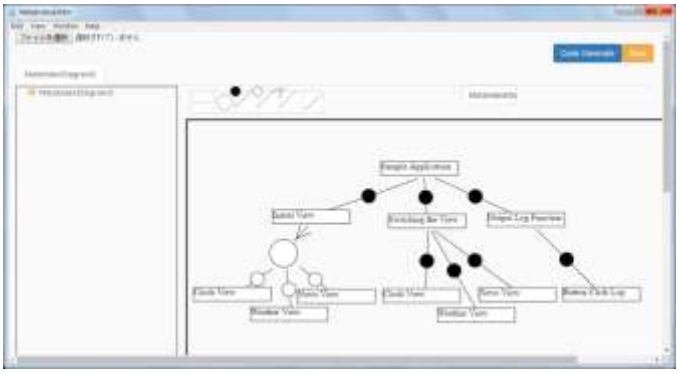

Figure 4 MetaIndexEditor as a feature modeling editor

D. *MetaIndexPlugin*

MetaIndexModel can associate external assets with Index, but there are many types of assets. Therefore, it is difficult to handle assets universally. We enabled that we operated MetaIndexModel from the external tool which dealt with the various assets by developing a Plugin using MetaIndexAPI. We developed MetaIndexPlugin for Astah and Astah GSN as an example of the third-party tool at this moment.

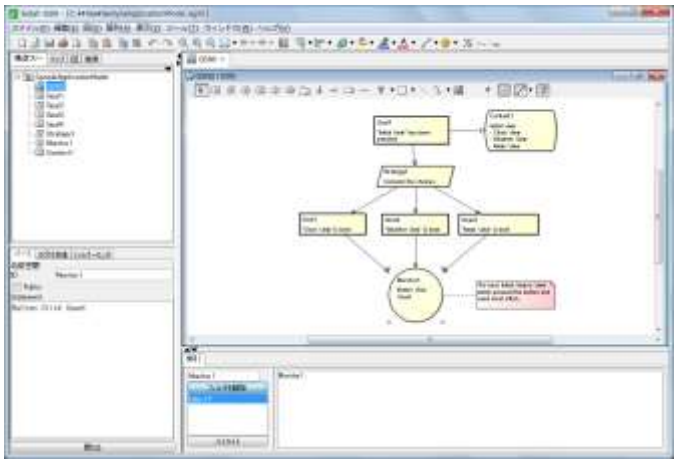

Figure 5 Screen example of MetaIndexPlugin for astah

We developed another MetaIndexPlugin for the file system to make the log folders into IndexedContainer as an example of traceability in practical use.

Figure 6 Screen example of MetaIndexPlugin for the file system

Also, we implemented a synchronized display function on the plugin which automatically indicates an asset in association with the index when a user chooses index in MetaIndexEditor.

E. *MetaIndexTools*

We are enabled to associate and display traceabilities between MetaIndexModel and assets using a third-party modeling tool by implementing MetaIndexDB, MetaIndexAPI, MetaIndexEditor, and MetaIndexPlugin.

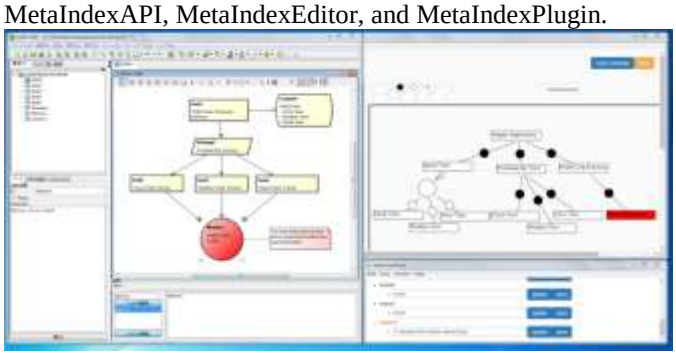

Figure 7 Traceability indication examples

v. Case studies using a sample model

We believe that it is effective to secure traceability using MetaIndex model in various system development including uncertainty. Therefore we inspected whether MetaIndexTools was available for system development having uncertainty. To confirm the utility of MetaIndexTools, we made a simple example model and a simple example application using MetaIndexTools. This application provides a choice from three kinds of information to display for the user and includes the uncertainty.

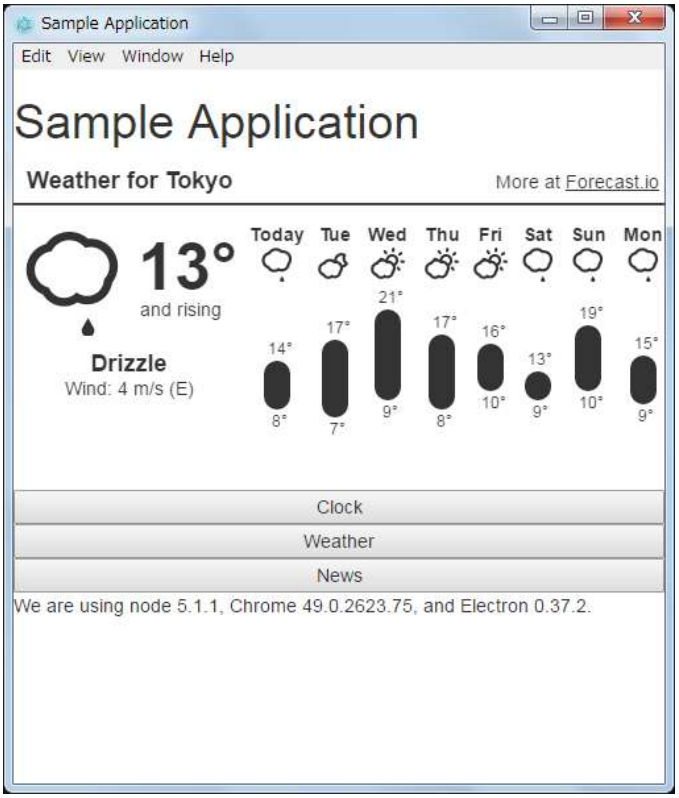

Figure 8 Screen image of sample application

We chose an initially indicated screen due to the uncertainty of this application. By using GSN analysis, we proved that starting the application with the most viewed screen was desired. To record the view count, we developed the log output function that records the count of the button push-downs in this application and also performed traceability between a log output folder and log output feature.

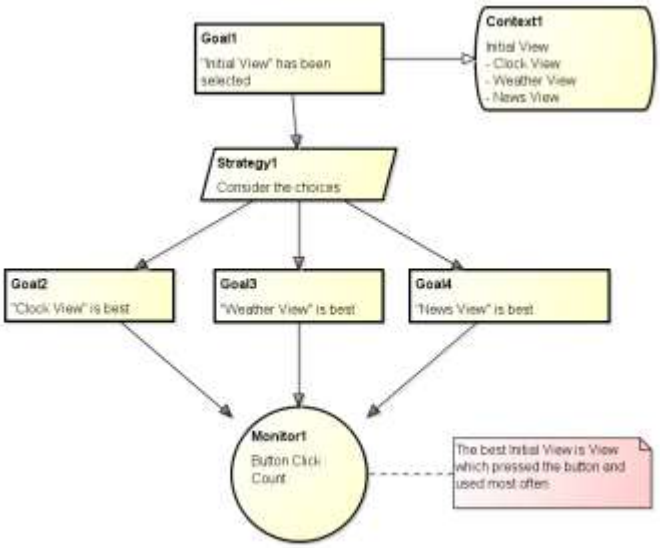

Figure 9 Sample application model

We completed an evaluation experiment, in which three subjects tried our sample application. From the log analysis, we found that News View was the highest in the number of viewing times by the user during the trial. We made enhancements, choosing News View as an initial screen to our sample application, following the experiment result.

Using MetaIndexTools, we could make traceability between the feature of an initially indicated screen as an uncertainty and the operational log files as operational assets.

VI. Concusion

We proposed and developed traceability tools for the model-base development to achieve life cycle oriented system development. Using a simple case study, it was confirmed that MetaIndexTools could be effective for the life cycle oriented system development. In future work, we will apply this tool to larger system development and also develop MetaIndexPlugin for various tools for model-based development. We also have planned the speeding of MetaIndexEditor so that it may be possible to treat a great deal of Indexes without stress.

Acknowledgment

This work is partially supported by JSPS KAKENHI Grant Number 15H05708.

References

[1] T.Nakanishi, K.Hisazumi, and A.Fukuda: A Framework to Manage Uncertainty in System Development, IPSJ SIG Technical Report, Vol.2015-SE-187, No.38, 2015(in Japanese)

[2] Winkler, S., & Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development. Software and Systems Modeling (SoSyM), 9(4), 529-565.

[3] Drivalos, N., Kolovos, D. S., Paige, R. F., & Fernandes, K. J. (2008, September). Engineering a DSL for software traceability. In *International Conference on Software Language Engineering* (pp. 151-167). Springer Berlin Heidelberg.

[4] Beuche, D. (2012, September). Modeling and building software product lines with pure:: variants. In Proceedings of the 16th International Software Product Line Conference-Volume 2 (pp. 255-255). ACM.

[5] http://astah.net

[6] Pohl, K., Böckle, G., & van Der Linden, F. J. (2005). Software product line engineering: foundations, principles and techniques. Springer Science & Business Media.

[7] Rosenberg, D., & Stephens, M. (2007). Use case driven object modeling with UML. APress, Berkeley, USA.

[8] https://eclipse.org/modeling/emf/

[9] S.Hiya, K.Hisazumi, A.Fukuda, and Tsuneo Nakanishi: clooca: Web-based tool for Domain-Specific Modeling, Proc. ACM/IEEE the 16th Int. Conf. on Model Driven Engineering Languages and Systems(MODELS 2013), 5pages,2013.