Shuttle: an Instant Model Synchronization Assistant for GMF Editors Based on Concept Synchronization

Chen-yi Kao Cheng-chia Chen
Department of Computer Science, National Cheng-chi University
g9214@cs.nccu.edu.tw chencc@cs.nccu.edu.tw

***Abstract**-Roundtrip engineering and refactoring are killer features of modern Integrated Development Environment (IDE) systems. Most implementations of these features nowadays, however, are tailor-made for scenarios like laying out Win32 GUI or diagramming UML and hence are hard to generalize; moreover, the refactoring is usually restricted to exact string matching and thus unable to synchronize artifacts with different occurrences of the equivalent term. These problems inhibit today's IDEs from supporting developments requiring synchronization across models, languages and documents.*

Shuttle is a novel instant modeling assistant developed by us running on Eclipse Graphical Modeling Framework (GMF) [1] editors. It monitors users' input model elements and link them by related concepts. Later modifications of an element will trigger rules to find the others under the same related concepts and result in various synchronization recommendations which developers may choose to enforce consistency among parts of the developed system.

The link-trigger mechanism of Shuttle is based on what we call concept synchronization (CS), which is inspired by the idea of "concept" in ontology and "concept search" in information retrieval. CS captures the simple idea that model elements with related text descriptions would be very likely modified accordingly if one of them is changed by the developer. To detect all others related to a target model element, we establish a many-to-many mapping ahead of time between elements and WordNet Synsets [2] according to the element text descriptions and then, with WordNet's help [3], all elements related to the target can be found by looking for those mapped to Synsets associated with the target.

Keywords: model synchronization, concept, Eclipse, GMF, WordNet.

1. Introduction

In the history of dealing with data synchronization issues, refactoring [4] provides an exact-keyword-match-based method for program evolving. Data binding [5][6] focuses on general binding and synchronization techniques for runtime object data. Automatic roundtrip engineering [7] goes further to automate model synchronization between different phrases of system development. Visual Paradigm [8] even generates use cases and classes from text description for requirement-to-design model forward engineering.

In general cases, model synchronization is a special case of model, even graph, transformation since every model is essentially a kind of graph. Transformation systems, such as TIGER [9], GROOVE [10], CODEX [11] and VIATRA2 [12], operate the synchronization on the solid basis of graph theory.

However, all methods above are of certainty and don't consider that "usually a model is named and documented with *natural languages*". When it comes to working on natural languages, a method of uncertainty seems inevitable.

In [13] "concept search" of information retrieval is mentioned as the next trend of information searching. Far from "keyword search", concept search uses logical relations instead of precise word occurrences to represent documents like Web pages. Therefore documents having different word occurrences can be found together if they're related semantically. We think that software model elements can be treated in this way since most modern model elements are well documented in natural languages.

We introduce an enhanced, semi-automatic and semantic - textually semantic at first - model synchronization technique of "concept synchronization". It is based on "concept search" from information retrieval. We believe that model elements can be grouped into many *concepts* according to these elements' semantics. If they are textual elements like GMF Labels, then the semantics can be in textual ones like synonym, hypernym or hyponym, etc. Therefore the model synchronization becomes that elements under the same concept reach a new semantic consistency once one of them is modified with that new semantics.

Shuttle is such a semi-automatic and non-interrupting semantically model synchronization assistant for friendly GMF graphical modeling environment. It accepts WordNet lexical database and GMF Labels as two major inputs and tries to group Labels by using WordNet's data. It utilizes CS in three stages: concept linking, concept rule inference and synchronization recommending, which plays as current only output. Tight Eclipse integration via GMF API is going to give users smooth and familiar modeling experiences.

As in this paper, we will discuss more about Eclipse GMF integration in the following section. From section 3 to 5 we will focus on Shuttle's core logic of the three-stage concept synchronization.

2. Monitoring model elements on GMF editors

GMF is a generic framework for graphically manipulating models on Eclipse. Shuttle utilizes WrapLabel, event handler, tooltip and Decoration mechanism of GMF for user-editor interaction. All of them, except the event handler, are prepared for the friendly output generated by Shuttle. The ways of monitoring inputs and showing outputs are introduced here, while more output content is delivered in 5.3.

2.1. Model elements in GMF

Behind GMF editors model elements (e.g. classifiers, actors, comments, etc.) are classified into three basic graphical parts. They are Shapes, Labels and Connections. At first we focus on Labels, which are the most obvious parts using natural languages and the closest ones to "documents" we called in IR, as shown in Figure 1. Then we can use natural language or IR tools and techniques on these elements without complicated adaption for proof-of-concept.

Every Label has an EditPart (more precisely, `org.eclipse.gef.editparts.AbstractGraphicalEditPart`) as a controller of editing model element under the Model-View-Controller pattern, while the view is WrapLabel and the model its text property. The event listeners built with every Label EditPart sense and handle editing events of the Label. Therefore we design Shuttle's own EditPart event listener classes and register their instances to all Labels for Shuttle's way of detecting Label content changes and linking Labels and concepts described in section 3.

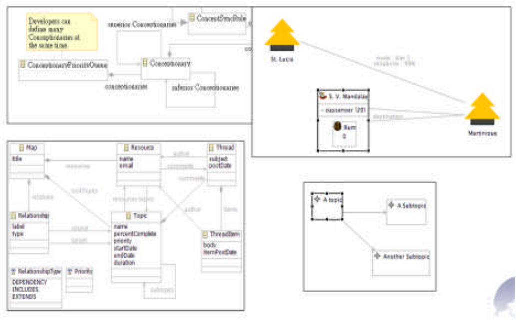


Figure 1. GMF Labels. There're four sub-figures showing various Labels (all text in the figures are 'text' properties of Labels). Some of the figures are depicted from [14].

Each time when a GMF element is monitored by Shuttle its 'figure' property is examined to check if its view is a WrapLabel. When the examination is passed, Shuttle retrieves the WrapLabel's text property and compares it with the previous retrieved one to sense the change of text. Once the change happens Shuttle predicts the elements to be modified correspondingly and brings them to the spotlight using the following GMF Decoration mechanism and WrapLabel tooltips. The detail is also revealed in section 5.3.

2.2. Utilizing GMF Decoration Service

GMF Decoration Service (with `org.eclipse.gmf.runtime.diagram.ui.services.decorator` as its main Java package identifier) [15] is an API group related to GMF editors. The major function of service is to decorate GMF model elements and is right suitable for showing our future recommendations. The lazy decorating of service makes it possible to preprocess the elements (i.e., Labels) in which we're interested behind the scene.

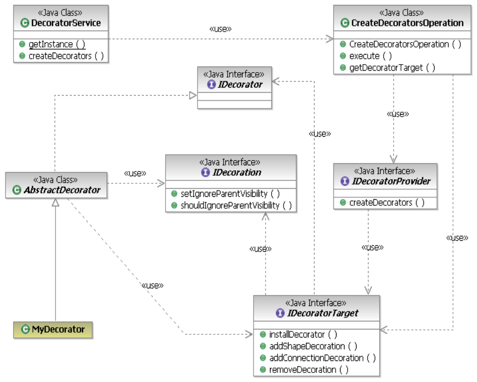
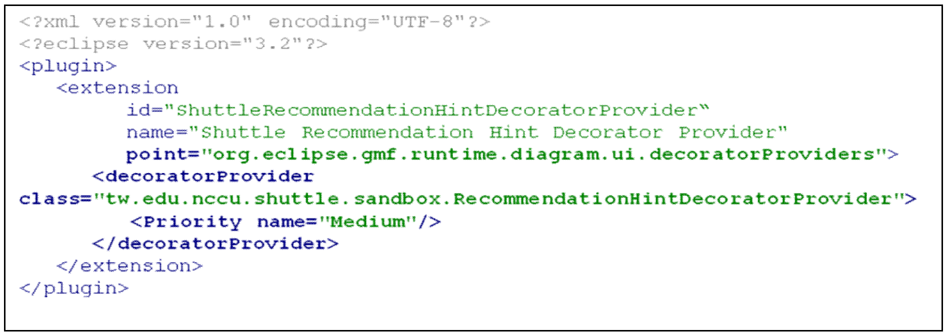


Figure 2. GMF Decoration Service [15].



```xml
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         id="ShuttleRecommendationHintDecoratorProvider"
         name="Shuttle Recommendation Hint Decorator Provider"
         point="org.eclipse.gmf.runtime.diagram.ui.decoratorProviders">
      <decoratorProvider
class="tw.edu.nccu.shuttle.sandbox.RecommendationHintDecoratorProvider">
         <Priority name="Medium"/>
      </decoratorProvider>
   </extension>
</plugin>
```

Figure 3. Shuttle's "plugin.xml", where bold text means necessary attributes. Therefore our customized decorator provider, `RecommendationHintDecoratorProvider`, is discoverable by a GMF editor.

The preprocessing is done via GMF extension point "Presentation Decorator Providers" [16], which lets GMF plug-ins declare their implementations of interface `IDecoratorProvider`, as shown in Figure 3. Via the interface developers directly retrieve GMF editors and even Labels as the targets (called decorator targets) of decorating operations through method "`provides(IOperation operation)`". Shuttle retrieves all monitored Labels in this way. At first the Decoration Service utilization is prepared for Shuttle's output, at last its side effect helps us complete monitoring the input.

When a monitored element is modified and the corresponding modifications on other elements are predicted, first the decorations on those elements are shown up to inform Shuttle users, and second clues for the modifications are written to the tooltips of those elements. Hence only modification clues for elements that the users may have interests in are shown on hovering.

By utilizing GMF Decoration Service we bypass the canonical GMF editor generation steps such as developing graphical/tooling/mapping definitions with operating EMF.Codegen and genmodels while we directly interact with general Labels. This makes Shuttle a ready-for-use plug-in, rather than a ready-for-building one following classical GMF scheme.

3. Concept linking

Automatically and semantically concept linking plays the key part in Shuttle's logic. To achieve such automatic semantic, especially in natural languages, analysis, we use WordNet technology. WordNet provides massive databases of lexical semantics in natural languages. Size does matter and it's why automatic linking becomes possible.

3.1. Text boundary detection

In refactoring keyword matching is used directly in model synchronization. However following the naming convention of modern modeling, many model elements are named in compounds. And compounds mean compound semantics. So extracting those compounds is the first step toward further semantic analysis and is done by detecting their boundaries.

To handle both general sentences and compound words, we apply at first Sentence BreakIterator, then Word BreakIterator, all instantiated from `java.text.BreakIterator`, to each Label text. Currently Word BreakIterator only detects boundaries of camel-case or underscore-separated compounds. We plan to replace it by some more powerful BreakIterator from International Components for Unicode (ICU) [17] when it's available in the future.

3.2. WordNet Synset concept

In the prototype of Shuttle we have only one kind of concept: WordNet Synset (Set of cognitive synonyms [2]), which comes from the resolution of WordNet lexical database navigation. WordNet is basically a database and to access it we use MIT Java Wordnet Interface (JWI) by Massachusetts Institute of Technology.

3.2.1. Linking WordNet Synset concepts and model elements. For Label text Shuttle uses two steps to take the advantage of WordNet. They are WordNet Dictionary lookup, for stemming, and (both basic and advanced) concept search, as depicted in Figure 4. In our demonstration scenario a developer builds a model of Rule System containing a class called "Rule", which contains two operations named "apply" and "recommendApplication" in each own Label, as illustrated in the figure's upper part. In the lower left part some model editor objects represent our Rule System model.

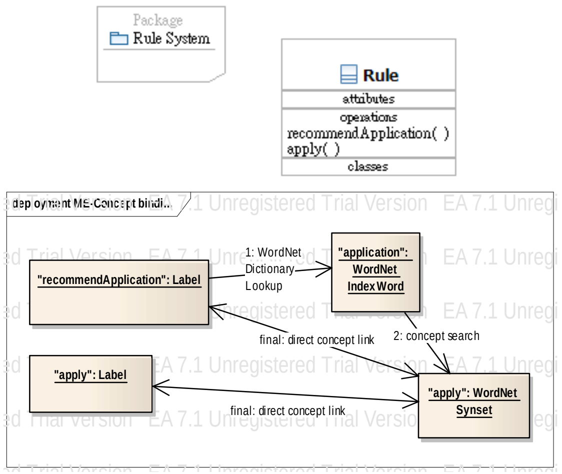

Package
Rule System
Rule
attributes
operations
recommendApplication( )
apply( )
classes
deployment ME-Concept bindi...
"recommendApplication": Label
1: WordNet Dictionary Lookup
"application": WordNet IndexWord
final: direct concept link
2: concept search
"apply": Label
"apply": WordNet Synset
final: direct concept link

Figure 4. The upper part shows a modeling scene from a GMF-based Eclipse UML2 Class Diagram editor; lower part is a UML deployment diagram depicting the "concept linking" behind the scene.

Once Shuttle starts the linking operation some intermediate objects are generated as shown in the lower right part. For example, during the first step Dictionary lookup, after boundary detection of "recommendApplication" we use `edu.mit.jwi.dict.IDictionary`'s `getIndexWord(java.lang.String lemma, PartOfSpeech pos)` method to collect the Dictionary IndexWords of subtext "Application" for all part-of-speeches.

In the second step of concept search we get Synsets that the IndexWords are belonged to as the basic concepts of the element text. In the next here comes Shuttle's intelligent play to retrieve *advanced concepts* by advancing to related Words (for lexical relations in various POS's) and Synsets (for semantic relations under the same POS) via methods `edu.mit.jwi.item.ISynset`'s `getRelatedWords()` and `edu.mit.jwi.item.IWord`'s `getRelatedSynsets()`. Current advancing strategy includes navigating all kinds of WordNet search pointers [3] with 1 hop advance. In Rule System scenario we find "apply" as a *derivationally related form* of "Application" [18].

Since we have all these basic and advanced concepts, finally we can directly link the concepts to the elements for faster traversal among concepts and elements in the future. Additionally each linkage is bidirectional for easy propagation of element modification events under the same concept. For Rule System example, modification to either Label "recommendApplication" or "apply" is going to be notified on the other one via the common concepts "application" and "apply". For future development of Shuttle the links are especially generalized and wrapped in *rules* and we are going to introduce this in the next section.

4. Concept synchronization rules

Our concepts are synchronization relations thought good to be enforced. To reflect this intentional enforcement rule architecture with recommendation mechanism is chosen for Shuttle. Therefore we have to repack our concept links as rules, i.e., concept synchronization rules, with setting triggers to collect Label text modification events and recommenders to recommend correspondent modification synchronizations. All of these are done via traversing the links.

We design a pair of trigger and recommender for each model element (Label). And the couples are shared by rules. In the example of Figure 5 Rule X and Y share trigger1, trigger3, recommender3 and recommender6, which correspondingly hook to model element me1, me3 and me6. This time me1 is modified and triggers concept rule X, causing linked me3 to get notified with some recommendations. The propagation of modification event is painted in red in the figure.

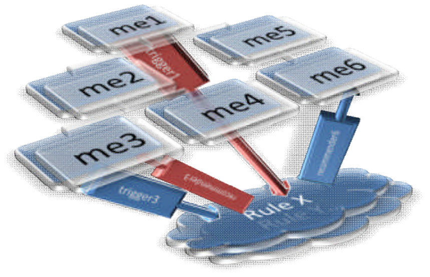


Figure 5. The interaction between model elements, triggers, recommenders and rules.

5. Recommending the change of concept linking

The modification of Label text leads to the change of concept linking behind. Recommendations and hints are visual and taking the advantage of GMF Decoration Service to reveal such change. Furthermore our non-interrupting recommendation keep users focusing on their urgent modeling jobs while it provide worthy modeling tips to help them reach higher productivity.

5.1. Finding the change when it happens

Our goal is to recommend *subtext change only*.

For example the original operation "recommendApplication" in our Rule System links to at least two concepts, "recommend" and "application", and we want only recommendations about the change of "Application" but to avoid ones of "recommend" if there's just "recommendApplication" changed to "recommendExecution".

Levenshtein Algorithm is an ideal way for detecting such partial changes. First we align original and modified Label text as well as the algorithm does and retrieve a series of Levenshtein Editings. Secondly we collect subtexts with the Editings costing larger than 0, for now, as *Likely Changing Boundaries*. These Boundaries are just the text unit for concept linking. Hence the difference between original and modified text-concept linking is found.

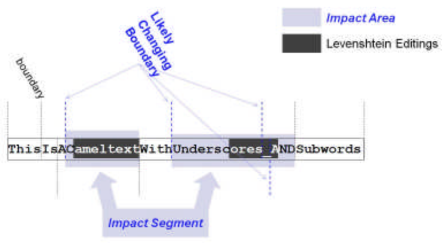


Figure 6. Finding the modified subtexts (i.e., Likely Changing Boundaries) in an original text.

5.2. Recommendations and hints

Recommendation hints notify the existence of recommendations without interfering users' current modeling operations.

In a concept change to the original text, for the trigger Labels their concept links are eliminated due to the change and so are their original attached recommendations; for the recommender Labels recommendations are directed to show off via the links. For the present, recommendations reveal only the causes and no objects of model synchronization.

5.3. Showing hints/recommendations on GMF editors

Shuttle displays hints and recommendations right on Labels for a more comprehensive, friendly and productive modeling experience. In GMF Decoration Service a Decorator Target (i.e., our Label) is responsible for printing decorations on itself. We treat hints as GMF Decorations, which can be hooked into Labels and displayed later when any recommendation is generated and attached to these Labels.

We use WrapLabel tooltips to carry recommendations for Labels. Cooperating with hints, tooltips that show on mouse hovering make Shuttle not interrupt user's current work and idea. And later on users can totally determine which recommendation to apply.

6. Conclusion

6.1. Shuttle in a nutshell

Our system finally looks like Figure 7. The system operates in three stages, where stage I is right described in section 2, stage II in section 3 and stage III in both section 4 and 5. After initializing, Shuttle collects GMF Labels of interest from editors and binds listeners to those Labels during stage I to sense future text changes. In stage II we detect text boundaries, search WordNet with sub-text for Synset concepts and hence link Labels and concepts. Once any linked Label is changed, the linkage, or rule, is triggered to open stage III. And some synchronization recommendations are prepared for the other Labels through the triggered rule.

Note that in Figure 7 editor A and B are typical GMF editors. They are not components of Shuttle. Neither Shuttle designs any proprietary GMF editor. Shuttle is intentionally pure functionality enhancement for every GMF editor. And this policy makes Shuttle work on possibly all GMF editors with the least efforts.

6.2. Future works

On the road to ideal model synchronization Shuttle is just at the beginning. More specifically, what we describe in section 4 are 'half' rules. Comparing to rules written in general rule languages, a Shuttle rule functions no enforcement. We want to extend such half rules to full ones to enforce recommendations. That may make our recommendations truly "recommend" the change of target Label text, or the suggested objects of synchronization, in the future.

Since we have chosen such uncertain semantic analysis to detect synchronization relations, recall and precision measurement from information retrieval becomes necessary for proving its effectiveness. It's expectable to see high recall but low precision after using our approach since natural language terms used in modeling are highly regulated. In the future we want to provide some relative experimental results for reference. In the next we may use machine learning skills to improve Shuttle's precision. Allowing custom even *structural rules* may improve it too.

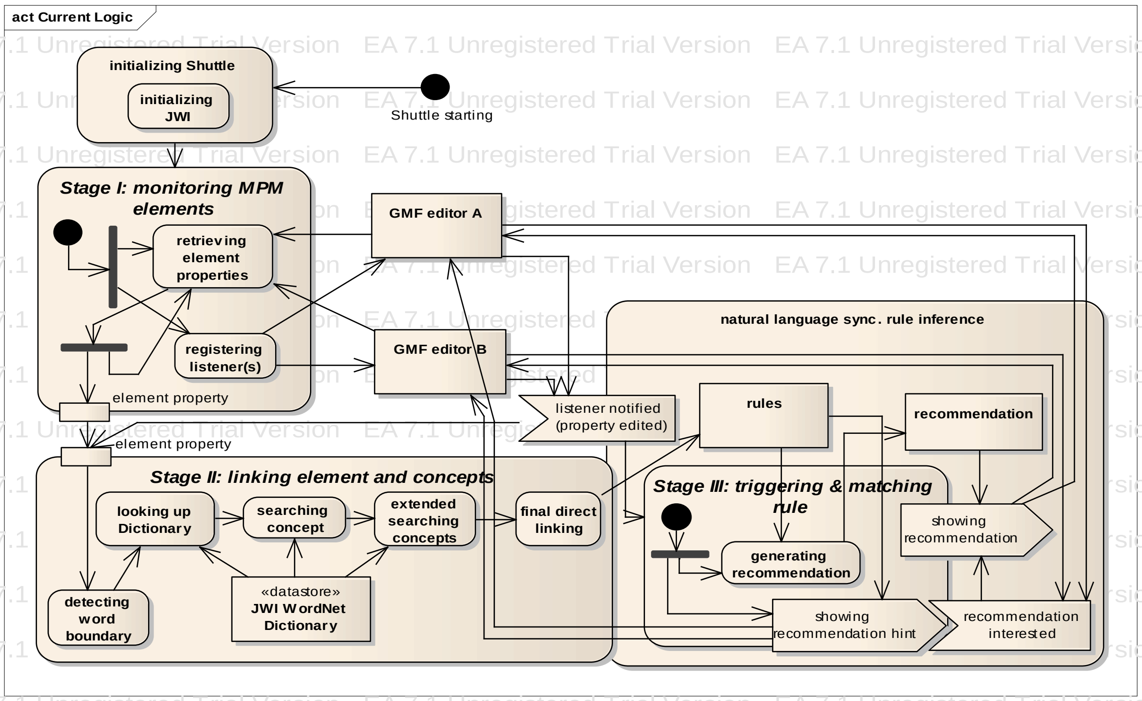
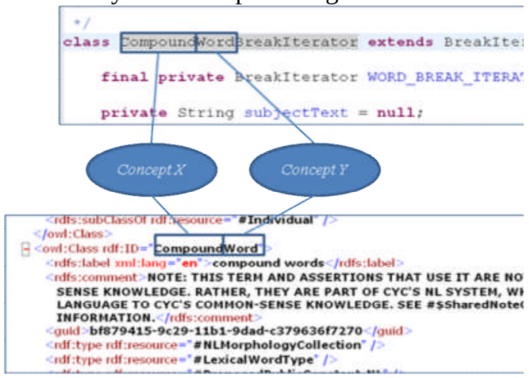

act Current Logic
initializing Shuttle
initializing JWI
Shuttle starting
Stage I: monitoring MPM elements
retrieving element properties
registering listener(s)
element property
GMF editor A
GMF editor B
natural language sync. rule inference
listener notified (property edited)
rules
recommendation
element property
Stage II: linking element and concepts
looking up Dictionary
searching concept
extended searching concepts
final direct linking
«datastore» JWI WordNet Dictionary
detecting word boundary
Stage III: triggering & matching rule
generating recommendation
showing recommendation
showing recommendation hint
recommendation interested

Figure 7. The UML Activity Diagram of Shuttle

Also as an instant tool efficiency improvement is one of our continuous works. Detailed performance issues will be addressed in the future.

6.2.1. Structural concepts/rules. This is our ultimate goal of adopting rule architecture – to use knowledge reasoners with general rules to achieve more intelligent model synchronization. We will monitor not only Labels, but Shapes and Connections; not only text, but all kinds of model element properties. The way is to link models and ontologies on the basis of our achievement of WordNet Synset concept linking.


```
*/
class CompoundWordBreakIterator extends BreakIte

    final private BreakIterator WORD_BREAK_ITERA

    private String subjectText = null;
```

```
    <rdfs:subClassOf rdf:resource="#Individual" />
  </owl:Class>
- <owl:Class rdf:ID="CompoundWord">
    <rdfs:label xml:lang="en">compound words</rdfs:label>
    <rdfs:comment>NOTE: THIS TERM AND ASSERTIONS THAT USE IT ARE NO
     SENSE KNOWLEDGE. RATHER, THEY ARE PART OF CYC'S NL SYSTEM, WH
     LANGUAGE TO CYC'S COMMON-SENSE KNOWLEDGE. SEE #$SharedNote
     INFORMATION.</rdfs:comment>
    <guid>bf879415-9c29-11b1-9dad-c379636f7270</guid>
    <rdf:type rdf:resource="#NLMorphologyCollection" />
    <rdf:type rdf:resource="#LexicalWordType" />
```

Figure 8. Linking model and ontology. The ontology (lower one) is from [19]錯誤! 找不到參照來源。 and assuming the model is about an object class in textual view.

As depicted in Figure 8, if we can link both model element "CompoundWordBreakIterator" and ontology element "CompoundWord" to Synset concept X and Y, then via common concepts X and Y the host model and ontology get linked. Once ontologies are able to bind to models, rules, like ones written in Semantic Web Rule Language, operating on ontologies shall more easily reflect their inference results on models through the common bounded ontologies.

References

[1] "gmf - GRAPHICAL MODELING FRAMEWORK", available: http://www.eclipse.org/gmf/, 2006, accessed on 2006/7/2.

[2] "About WordNet", available: http://wordnet.princeton.edu/, accessed on 2007/1/16.

[3] "WNSEARCH(3WN) manual page", available: http://wordnet.princeton.edu/man/wnsearch.3WN, accessed on 2007/7/27.

[4] Wikipedia contributors, "Code refactoring", available: http://en.wikipedia.org/wiki/Refactoring, 2007/7/4, accessed on 2007/7/13.

[5] S. Violet, "Data binding", Sun Microsystems, Inc., available: http://developers.sun.com/learning/javaoneonline/2006/desktop/TS-1594.pdf.

[6] D. Orme, M. Ward, B. Wellhöfer , B. Reynolds and others, "JFace data binding", available: http://wiki.eclipse.org/index.php/JFace_Data_Binding, 2007/4/7, accessed on 2007/7/13.

[7] A. Henriksson and H. Larsson, "A definition of

round-trip engineering", 2003, University of Linköping, Sweden.

[8] "Visual Paradigm", available: http://www.visual-paradigm.com/, 2006/12/4, accessed on 2007/1/16.

[9] Tiger Development Team, "Tiger project", available: http://tfs.cs.tu-berlin.de/~tigerprj/, accessed on 2006/ 8/11.

[10] "GRaphs for Object-Oriented VErification (GROOVE)", available: http://groove.sourceforge.net/groove-index.html, accessed on 2006/8/10.

[11] H. Larsson and K. Burbeck, "CODEX - an automatic model view controller engineering system", Presented at The Workshop on Model Driven Architecture: Foundations and Applications 2003, *CTIT Technical Report TR–CTIT–03–27*, pp. 37-48, available: http://trese.cs.utwente.nl/mdafa2003/proceedings.pdf.

[12] Eclipse Foundation Inc., "VIATRA2 subproject", available: http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/VIATRA2/index.html, 2006/7/28, accessed on 2006/9/5.

[13] "Concept search: Cutting large keyword searches down to size", *ASBMB News*, HighWire Press, available: http://highwire.stanford.edu/inthepress/asbmb/asbmb_2003mar.dtl, 2003/3.

[14] Y. P., D. Roy and R. Gronback, "GMF tutorial", available: http://wiki.eclipse.org/GMF_Tutorial, 2007/4/24, accessed on 2007/7/28.

[15] "Developer Guide to the GMF Runtime Framework", available: http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.gmf.doc/prog-guide/runtime/Developer%20Guide%20to%20Diagram%20Runtime.html, 2005, accessed on 2007/4/16.

[16] "Presentation Decorator Providers", available: http://help.eclipse.org/help32/topic/org.eclipse.gmf.doc/reference/extension-points/org_eclipse_gmf_runtime_diagram_ui_decoratorProviders.html, 2004, accessed on 2007/4/16.

[17] "ICU Home Page", available: http://www.icu-project.org/, accessed on 2007/7/12.

[18] "WordNet Search - 3.0", available: http://wordnet.princeton.edu/perl/webwn?o2=&o0=1&o7=&o5=&o1=1&o6=&o4=&o3=&s=Application&i=5&h=0010000000#c, accessed on 2008/7/30.

[19] S. Reed, "Open cyc ontology", available: http://www.cyc.com/2003/04/01/cyc, 2003/3/31, accessed on 2007/7/ 28.