Using Formal Concept Analysis to Establish Model Dependencies*

Igor Ivkovic and Kostas Kontogiannis
Dept. of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON N2L3G1 Canada
{iivkovic, kostas}@swen.uwaterloo.ca

Abstract

Software models evolve at different levels of abstraction, from the requirements specification to development of the source code. The models underlying this process are related and their elements are usually mutually dependent. To preserve consistency and enable synchronization when models are altered due to evolution, the underlying model dependencies need to be established and maintained. As there is a potentially large number of such relations, this process should be automated for suitable scenarios. This paper introduces a tractable approach to automating identification and encoding of model dependencies that can be used for model synchronization. The approach first uses association rules to map types between models and different levels of abstraction. It then makes use of formal concept analysis (FCA) on attributes of extracted models to identify clusters of model elements.

Keywords: *software evolution, RUP, model synchronization, model dependencies, formal concept analysis*

1. Introduction

As software evolves, its main constituents, from more abstract ones such as business workflows to very specific ones such as source code, also evolve. The problem with this behavior is in its complexity, that is, concurrently and systematically updating all of the software models as some of them change. This complexity is inherent and twofold. First, models from different stages of development are created by different stakeholders (*e.g.*, source code is created by developers, architectural design documents are created by software architects) with differing rationale in mind. Second, the models that are created can differ greatly; they can be at significantly different levels of abstraction and they can exhibit varied levels of expressiveness and semantics.

We have previously proposed and discussed this problem in a form of a framework for incremental model synchronization titled ***mSynTra*** [6, 14, 5]. Our framework follows the Model-Driven Software Evolution (MDSE) paradigm that is based on the theory of Model Driven Architecture (MDA) [8, 9]. Within this framework, each change in software is made on a model at a particular level of abstraction within an iterative and incremental lifecycle such as the Rational Unified Process (RUP) [4]. To restore consistency between models after changes are made, transformations applied to one model are traced and recorded. Using identified model dependencies, the transformation trace is then translated and applied to all affected models. Model dependencies play a crucial role here, as they indicate what models and what model elements are effected and might need to be changed to reflect a traced transformation. The synchronization process is in the end validated and terminated based on predefined equivalence relations.

In this paper, we aim to further extend the ***mSynTra*** framework and define a basis for systematic identification and establishment of dependencies among related models. We propose the use of formal concept analysis (FCA) [3] as an apparatus for identification of clusters of objects that share common attributes. We view models and model elements as top-level objects that possess certain attributes. As an attribute logic for mapping attributes from different context (*i.e.*, metamodels), we introduce an idea of *attribute association rules*. These rules are defined at two levels: at the metamodel level, where mappings between types are defined; and at the model level, where mappings between attributes and related annotations are defined.

To identify the semantic and abstraction-level differences, we also discuss the process of model extraction, where related metamodels are first refined to enable a more direct mapping between their types. These altered metamodels are then used as a basis to extract intermediate models that are less semantically different and that are more

[1] This work is funded in part by the IBM Canada Ltd. Laboratory, Center for Advanced Studies (CAS) in Toronto.

suited for establishment of model dependencies.

As an example of applicability and validity of our approach, we present a case study of establishing dependencies between Business Process Models (BPMs) (*i.e.*, business workflows that include processes, tasks, decisions, *etc*.) and the underlying source code (*i.e.*, Java and Enterprise Java Beans (EJB)) that is used to enact represented functionality.

The rest of the paper is organized as follows. Section 2 shows how our research relates to previously published results in the area of software reuse and hierarchical data management. Section 3 introduces our framework for identification and establishment of model dependencies using FCA including the discussion of our approach to model extraction. Section 4 presents a case study that demonstrates the use of this framework in practice. Finally, Section 5 presents the conclusions and directions for future research.

2. Related Research

In the area of software reuse, Engels *et al.* in [1] have discussed the transformation of Unified Modelling Language (UML) Class Diagrams and UML Collaboration Diagrams [10] into Java code. They have shown how to deal with both the structural and behavioral (*e.g.*, flow) mapping problems between UML and Java using a pattern-based transformation algorithm. The pattern used is an instance of a metamodel from which one can identify parts of the source diagram that is to be transformed. The pattern-based approaches depend on a predefined set of patterns that is not trivial to extract and that has to be updated as new patterns are introduced. Rich and Willis have used subgraphs to recognize program design [11] while Spanoudakis and Constantopoulos have used a distance metric as a similarity measure to evaluate reuse potential of chosen artifacts [13]. In the area of hierarchical data management, Gianolli and Mylopoulos describe a semantic approach where XML data stores are mapped based using a common DTD schema [12] while Faid *et al.* discuss how to use formal concept analysis to discover concepts and rules from structured complex objects [2]. In our approach, we consider semantic mappings of types from complex hierarchical data structures using intermediate models but we also address the problem of inference of relations between individual model elements based on the mappings of related attributes.

3. A Framework for Establishing Model Dependencies using FCA

In this section, we introduce our framework for establishing model dependencies using the theory of formal concept analysis (FCA). As a part of the framework, we describe our approach to model extraction that is used to stem the semantic and abstraction-level differences between related models. We also describe several attribute transformation techniques for establishing associations between attributes that belong to different metamodels

This framework is presented within the context of our ***mSynTra*** framework (Figure 1). Hence, its steps are (1) definition metamodels, (2) extraction of intermediate models, (3) definition of dependency rules and establishment of model dependencies based on those rules, and (4) validation of established dependencies.

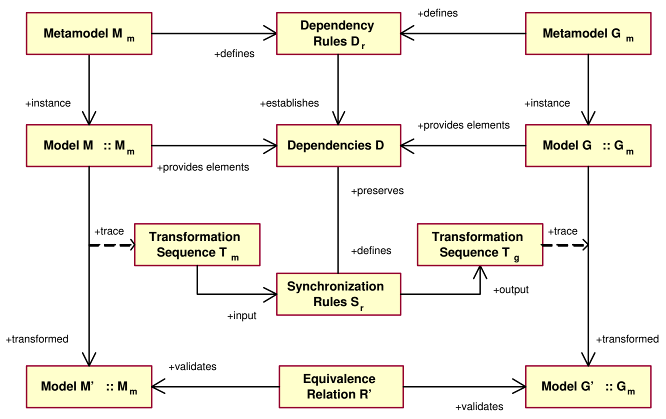


Figure 1. *mSynTra* Framework

We shall demonstrate the usage of this framework in our case study (Section 4) in the process of bridging the semantic and abstraction-level differences between BPMs and Java source code.

3.1. Defining Metamodels

For two models or sets of models that are to be synchronized, it is first necessary to abstract and represent model syntactics and semantics in a format of corresponding metamodels or otherwise referred to as domain models. Generally, such metamodels are already available as part of the design documentations but in some cases they have to be extracted using a suitable domain-analysis technique, such as Feature-Oriented Analysis Technique (FODA) [7].

The resulting domain models should be accurate representations of the type and format of information represented in their respective models (*e.g.*, software design information based on UML syntax), and should be stored in a format that can be easily accessed and manipulated such as XML.

3.2. Extracting Intermediate Models

The relations between models are expressed either at the metamodel or at the model level, where the higher-level relations are used to infer the more specific ones. Establishing relations at the metamodel level includes identification and encoding of properties and features that are mutually dependent. These relations are then used as a basis to establish specific relations among models or among model elements.

For models that are based on different metamodels, establishing meaningful relations between model elements includes identifying differences in model semantics and underlying abstraction levels. We propose to overcome this problem by generating intermediate models, which are more closely related and for which the process of establishing model dependencies is simplified.

We base our approach on a technique for model-driven business process recovery [15], which deals with synchronization of business workflows and the enacting source code, and we extend this technique so that it can be used within our ***mSynTra*** framework. Hence, we interpret the process of model extraction as follows: (1) for two models M and G and their respective metamodels $\mathcal{M}_M$ and $\mathcal{M}_G$, analyze the metamodels and recognize compatible structural properties (*e.g.*, compatible events, metaclasses, data types); (2) refine $\mathcal{M}_M$ and $\mathcal{M}_G$ into $\mathcal{M}_M$' and $\mathcal{G}_M$' by omitting, grouping or breaking up incompatible properties (*e.g.*, metaclasses that cannot be mapped); and (3) use $\mathcal{M}_M$' and $\mathcal{M}_G$' as schemas to extract M' and G' from M and G.

3.3. Applying FCA to Extract Model Dependencies

Depending on the type of information (*e.g.*, structural, behavioral, temporal) that is stored in the two models that have to be synchronized, the amount of data available for and the complexity of the mapping can differ significantly. Moreover, depending on the information type, the level of precision of the established relations also differs. For example, mapping behavioral and structural properties together would make more data available for the mapping but would increase the complexity of the mapping process when compared to mapping only structural properties.

In this paper, we focus on mapping a particular set of structural and behavioral properties based on our case study of mapping business workflows to underlying source code.

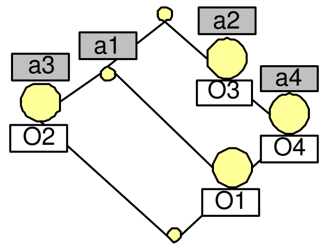
Within ***mSynTra***, our conceptual view of all MOF-compliant models used during the course of software evolution is as directed, labelled, attributed graphs. In this view, all graph properties are expressed in terms of labels and "attribute and value" pairs for individual nodes and edges. All models also comply with their respective metamodels, and their element types conform to corresponding metaclasses. We also presume that models at a particular level of abstraction (*e.g.*, design, architecture) representing a software from a particular domain (*e.g.*, accounting, database management) would be based on stable feature and domain models, and would have convergent vocabularies and ontologies. Hence, within our approach to identification of model dependencies, we would identify individual relations starting at the level of matching metaclasses, then proceed to match features and domain models, to finally match individual elements based on mappings of related vocabularies and ontologies. If a dependency for a particular element cannot be found, we would prompt for user's input and if no input was provided leave the dependency as undetermined.

As an apparatus for identifying and establishing model dependencies, we apply the theory of formal concept analysis as defined in [3].

Context:

	a1	a2	a3	a4
O1	x	x		x
O2	x		x	
O3		x		
O4		x		x

Concept Lattice:



Association Rules:

1 <2> a4 => a2;
2 <1> a3 => a1;
3 <1> a1 a2 => a4;

Figure 2. FCA Example

From this theory, a formal context K := (G, M, I) consists of two sets G and M and a relation I between G and M. The elements of the set G are called the objects and the elements of the set M are called the attributes of the context. To express that object g from G is in a relation I with an attribute m from M, we write this as gIm and read it as "the object g has the attribute m. Hence, the relation I is also called the incidence relation of the context K. [3]

We utilize this definition and interpret metamodels as contexts $\mathcal{M}_M$:= (G, M, I) that consist of a set of metaclasses/types G, a set of related attributes M, and a set of relations I that define associations between the types and the attributes. We also interpret models as contexts M := (H, N, J) that consist of a set of model elements H, a set of attribute values N, and a set of relations J that represent the mapping of types and attributes of model elements instantiated from respective metamodels to corresponding values in N. We also define metamodel $\mathcal{M}_M$ as a metacontext for a model/context M that is instantiated from $\mathcal{M}_M$. As we are limiting our approach to binary contexts only, we map nary-to-binary relations through combinatorial scaling of format (o, $\emptyset$), (o, v_1), (o, ($v_1 \wedge v_2$)), ... (o, ($v_1 \wedge v_2 \wedge ... \wedge v_n$)) where o is an object and v_1, v_2... are n possible values.

To define a concept, we again follow the FCA theory. Hence, it holds that for a set A $\subseteq$ G of objects, let us define A' := {m $\in$ M | gIm for all g $\in$ A}, as the set of attributes common to the objects in A. Also, for a set B $\subseteq$ M of attributes, let us define B' := {g $\in$ G | gIm for all m $\in$ B}, as the set of objects which have all the attributes in B.

A formal concept of the context (G, M, I) is a pair (A, B) with A $\subseteq$ G, B $\subseteq$ M, A' = B and B' = A, where A is the extent and B is the intent of the concept (A, B).

An implication between attributes in M is a pair of subsets of the attribute set M denoted A $\rightarrow$ B, which corresponds to informal statement that every object with the attributes in A also has the attributes in B. The concept lattice can be inferred from implications between attributes as in Figure 2. [3]

We refer to the rules for defining associations between attributes of different contexts as *attribute association rules*.

To define our dependencies between models more formally, let M and G be two models/contexts at different levels of abstraction with corresponding metamodels/metacontexts $\mathcal{M}_M$ and $\mathcal{M}_G$. Let O_M and O_G be the objects stemming from these models and let A_M and A_G be the attributes contained within those models, let A_{MG} := {a_M, a_G | $a_M \subseteq A_M \wedge a_G \subseteq A_G$ where $a_M \Rightarrow a_G$} be a set of related attributes from A_M and A_G. The attribute logic for defining attribute associations $a_M \Rightarrow a_G$ of attributes from different contexts is defined as a set $\mathcal{D}_r$ of attribute association rules (*i.e.*, dependency rules in the ***mSynTra*** framework).

For nonempty sets O_M' $\subseteq O_M$ and O_G' $\subseteq O_G$ of objects, let A_{MG}' := {m $\in A_{MG}$ | g_1Im $\wedge$ g_2Im for all $g_1 \in O_M$' $\wedge$ all $g_2 \in O_G$'}, as the set of attributes common to the objects in O_M' and O_G'. Also, for a nonempty set A_{MG}' $\subseteq A_{MG}$, let B_{MG}' := {$g_1 \in O_M \wedge g_2 \in O_G$ | g_1Im $\wedge$ g_2Im for all m $\in A_{MG}$'}, as the set of objects which have all the attributes in A_{MG}'.

An inter-context concept of two contexts $\mathcal{M}_M$ and $\mathcal{M}_G$ is a set (O_M', O_G', A_{MG}') with O_M' $\subseteq O_M$, O_G' $\subseteq O_G$, A_{MG}' $\subseteq A_{MG}$, and B_{MG} = (O_M' $\cup$ O_G').

Finally, a model dependency is an element d of a set $\mathcal{D}$ (*i.e.*, dependencies in the ***mSynTra*** framework) of inter-context concepts for models M and G that have associated attributes.

Establishment of model dependencies involves: (1) identifying a set of related A_{MG} attributes from A_M and A_G; (2) selecting and encoding corresponding attribute association rules $\mathcal{D}_r$; (3) extracting inter-context concepts using attribute association rules as relations between attributes; and (4) refining the extracted concepts by excluding the ones that are irrelevant (*e.g.*, include only the objects from one context) or that are redundant (*e.g.*, equivalent association results based on different sets of attributes).

Attribute Association Rules

Using an assumption that individual contexts based on particular domains would exhibit stable properties (*e.g.*, feature maps, ontologies), the attribute association rules can be viewed as functions that map recognized properties between attribute with the goal of determining they are related.

The rules can be classified as follows:

Hierarchical Association Models are viewed as parts of broader model hierarchies and corresponding feature maps are extracted. Based on the feature mapping, the two types of associations are recognized: direct (*e.g.*, a model implements a particular feature) or implied (*e.g.*, a model implements a subfeature of a higher-level feature).

Type-Based Association Metaclasses defined in metamodels are matched based on structural compatibility, and the association rules are created following these mapping. Additional rules are added for recognized compatible data types.

Spatial Association Parts of the flow in behavioral models are represented as attributes (*e.g.*, element A precedes element B) and the associations are established between the attributes using type-based associations for resolving element or data type mappings.

Text-Based Association Attributes are viewed as strings of text. The potential syntactic differences are augmented through techniques such as thesaurus replacements — recognizing different synonyms, stemming — reducing each word to its root (*e.g.*, allocation, allocated, allocating to allocat), abbreviation expansion — expanding recognized abbreviations,stop-word elimination — eliminating words with no semantic meaning, word-matrix matching — recognizing clusters of attributes that share semantically-relevant words, and nGram matching — substring matching irrelevant of position based on substrings of n size.

The given rules are not domain specific nor do they exhibit properties that are related to a particular domain. Creating rules that are domain specific is implausible with regards to their updating and expansion as it requires extensive domain analysis that might have to be repeated every time the underlying contexts are changed.

Attribute Association Rules in Practice

The conflict resolution for objects that belong to more than one cluster is achieved through scoring, where the number of cluster matches for any two objects represents the score and the match is selected through the process of maximizing that score. We also note that matches are tuples and not only pairs, so that the relations between objects can be one-to-one, one-to-many, and many-to-many. Hence, where there is more than instance of a match with a maximum score, established dependency tuple would contain all of the related objects from all of the instances that exhibit the maximum score.

3.4. Validation of Established Dependencies

Validation of established dependencies can be automated, if there exists a set of previously encoded dependencies in a suitable format such as XML, or semi-automated, if no encoded dependencies exist and if developers and domain experts need to be queried for feedback. If the precision and recall levels obtained through validation are not satisfactory, several categories of changes can be applied to improve the overall results such as (1) including or excluding particular attribute association rules, (2) changing implied attribute mappings and object granularity levels, and (3) modifying threshold levels and other input parameters.

4. Case Study: Establishing Dependencies between BPMs and Java Source Code

In our case study we are dealing with an industrial size system that consists of a hierarchy of business process models (*i.e.*, business workflows)and is enacted through a corresponding set of Java and EJB source code. Through domain analysis and discussion with developers, we found out that business workflows were in some cases created based on the information flow from the source code but were in most cases built independently. We also found that design documentation specifying the mapping between the two was incomplete and out of date, and that the precise dependencies between the two sets were implied and only known by developers and architects who worked on the system.

Our goal in this case study was to assist in the process of systematically propagating change in both directions between business workflows and the source code. However, before that could be accomplished, we first needed to identify and establish dependencies among those models that were related.

4.1. Recovering Intermediate Models

This part of the process had two parts: simplifying and annotating the BPMs, and abstracting and annotating the source code models. We implicitly include the creation of corresponding metamodels, as the preface to creation of suitable intermediate models.

The simplification of the BPMs included the following: (1) analysis of the business workflows represented in XML, with identification of a simplified metamodel of objects and attributes that have meaningful representation in the source code; (2) augmentation of the metamodel with the annotation attributes for those elements for which additional information is available (*e.g.*, notes explaining the meaning and the usage context of particular workflow elements); and (3) automatic extraction of the XML files based on the annotated metamodel from the original BPM XML files.

The abstraction of the source code included the following: (1) analysis of the source code files and extraction of a suitable metamodel of source code elements that have meaningful representation in the business workflows; (2) augmentation of the metamodel with the annotation attributes for those elements for which additional information is available (*e.g.*, top-level comments, comments proceeding methods or individual statements); and (3) automatic extraction of the XML files based on the annotated metamodel from the source code files. More details about the source code abstraction process are available in [15].

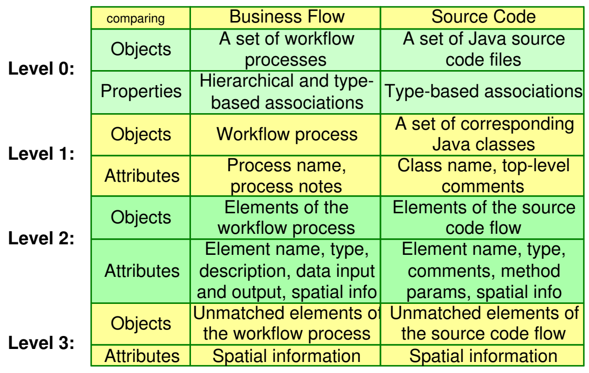
4.2. Extracting Dependencies using FCA

Based on the refined metamodels, we then proceeded to apply the FCA to identify and establish model dependencies. The first step in this process was creation of attribute association rules and definition of corresponding FCA clusters.

On the BPM side at the metamodel level, in addition to type-based association rules, we have also made use of the fact that business workflows are part of an overall workflow hierarchy to create hierarchical association rules. On the source code side, we could not establish hierarchical relations and at the metamodel level we have only established type-based associations. At the model level, we have identified compatible attributes and annotations, and have defined a suitable set of spatial and text-based associations through experimentation on a selected (through domain analysis) a set of workflows and related source code. Figure 3 shows our mapping of the BPM and the source code attributes and properties.

4.3. Validating Extracted Dependencies

The extracted dependencies could not be validated automatically, as the previously established relations were not accurate nor complete. Therefore, we have included feedback from system developers and architects as part of our validation, and have performed several iterations of feed-



	comparing	Business Flow	Source Code
Level 0:	Objects	A set of workflow processes	A set of Java source code files
	Properties	Hierarchical and type-based associations	Type-based associations
Level 1:	Objects	Workflow process	A set of corresponding Java classes
	Attributes	Process name, process notes	Class name, top-level comments
Level 2:	Objects	Elements of the workflow process	Elements of the source code flow
	Attributes	Element name, type, description, data input and output, spatial info	Element name, type, comments, method params, spatial info
Level 3:	Objects	Unmatched elements of the workflow process	Unmatched elements of the source code flow
	Attributes	Spatial information	Spatial information

Figure 3. Attribute and Property Mappings

back solicitation and refinement. The initial approximated recall levels that were obtained show a gradation of results that stem from almost perfect scores for our training data set, which represents 0.15 of the complete data set, 0.8 for the average case, which represents approximately 0.60 of the complete data set, to approximately .60 in the worst case, which represents 0.25 of the complete data set. The differences in scores depend on several factors such as the experienced attrition of information when clustering less BPM-specific source code elements, the lack of conformance of workflows to source code flows in certain subsets, the fact that certain workflow and source code subsets were not synchronized *etc*. We have measured recall based on the top results of our matching process and have not enforced the conflict resolution rules based on the feedback received from the developers.

5. Conclusions and Future Research

In this paper, we have presented a framework for establishing model dependencies using formal concept analysis. We have described the steps in this approach including creation of metamodels, extraction of intermediate models, application of the formal concept analysis to extract model dependencies, and validation of the established dependencies. We have also discussed in some detail attribute association rules that can be used to infer relations between attributes from different contexts. Finally, we have demonstrated the usage of our approach by applying it to an industrial case study of synchronization of business process models with enacting Java source code.

In future research, we intend to further investigate the suitability of this approach by applying it to different case studies that relate to different stages of the software development lifecycle. For example, we intend to specifically investigate the synchronization of UML design diagrams with as-designed or as-implemented software architectures in one direction or with the underlying source code in another. This work is performed in collaboration with the IBM Canada Ltd. Laboratory, Center for Advanced Studies (CAS) in Toronto.

References

[1] G. Engels, R. Huecking, S. Sauer, and A. Wagner. Uml collaboration diagrams and their transformation to java. In *Proceedings of the Second International Conference on The Unified Modeling Language (UML)*, Fort Collins, CO, Oct 1999.

[2] M. Faid, R. Missaoui, and R. Godin. Knowledge discovery in complex objects. *Computational Intelligence*, 15(1), Jan 1999.

[3] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1999.

[4] IBM. Rational unified process (rup). Online by IBM Corporation, 2004. http://www.ibm.com/software/awdtools/rup/.

[5] I. Ivkovic and K. Kontogiannis. Model synchronization as a problem of maximizing model dependencies. In *Proceedings of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004)*, Vancouver, BC, Oct 2004.

[6] I. Ivkovic and K. Kontogiannis. Tracing evolution changes through model synchronization. In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, Chicago, IL, Sep 2004.

[7] L. Kean. Feature-oriented domain analysis. Software technology review, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.

[8] A. Kleppe, J. Warmer, and W. Bast. *The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.

[9] OMG. Model driven architecture - a technical perspective. Object Management Group's (OMG's) Architecture Board ORMSC Document ORMSC/01-07-01, Object Management Group, Jul 2001.

[10] OMG. Unified modelling language (uml) specification. Technical report, Object Management Group, Mar 2003. http://www.omg.org/docs/formal/03-03-01.pdf.

[11] C. Rich and L. M. Willis. Recognizing a program's design: A graph-parsing approach. *IEEE Software*, Jan 1990.

[12] P. Rodriguez-Gianolli and J. Mylopoulos. A semantic approach to xml-based data integration. In *Proceedings of the 20th International Conference on Conceptual Modeling*, 2001.

[13] G. Spanoudakis and P. Constantopoulos. Measuring similarity between software artifacts. In *Proceedings of 6th International Conference on Software Engineering and Knowledge Engineering (SEKE 94)*, Jurmala, Latvia, Jun 1994.

[14] T. Tong, R. McKegney, T. Lau, K. Kontogiannis, I. Ivkovic, P. Liew, Y. Zou, Q. Zhang, and M. Hung. Model synchronization for efficient software application maintenance. In *Proceedings of the 12th International Workshop on Program Comprehension (IWPC 2004)*, Bari, Italy, Jun 2004.

[15] Y. Zou, T. Lau, K. Kontogiannis, T. Tong, and R. McKegney. Model driven business process recovery. In *Proceedings of the 11th IEEE Working Conference on Reverse Engineering (WCRE 2004)*, Amsterdam, The Netherlands, Nov 2004.