Virtual Heterogeneous Model Integration Layer

Muhammad Ali Memon
School of Information Technology,
Shaheed Benazir Bhutto University,
Shaheed Benazirabad, Pakistan

Khizer Hayat
Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology,
Hyderabad, Pakistan

Asadullah Shaikh
College of Computer Science and Information Systems,
Najran University,
Najran, Saudi Arabia

Mutiullah Shaikh
Faculty of Electrical, Electronic and Computer Engineering,
Mehran University of Engineering and Technology,
Jamshoto, Pakistan

***Abstract*—The classic way of building a software today simplistically consists in connecting a piece of code calling a method with the piece of code implementing that method. We consider these piece of code (software systems) not calling anything, behaving in a non deterministic way and providing complex sets of services in different domains. In software engineering reusability is the holly grail, and specially the reusability of code from autonomus tools requires powerful compostion/integration mechanisms. These systems are developed by different developers and being modified inceremently. Integrating these autonomous tools generate various conflicts. To deal with these conflicts, current integration mechanisms defines specific set of rules to resolve these conflicts and accompalish integration. Indeed still there is a big chance that changes made by other developers, or they update their changes in order to make them compliant with other developers cancel the updates done by others. The approach presented here claims three contributions in the field of Hetrogeneous Software Integration. First, this approach eliminate the need of conflicts resolving mechanism. Secondly, it provides the mechanism to work in the presence of conflicts without resolving them. Finally, contribution is that the integration mechanism does not affect if either of the system evolves. We do this by introducing an intermediate virtual layer between two systems that introduce a delta models which consist of three parts; viability that share required elements, hiding that hide conflicting elements and aliasing that aliases same concepts in both systems.**

***Keywords*—Model Driven Engineering; Co-evolution; Co-adaptation; Delta models; Model Integration**

I. Introduction

Scale changes everything. This is one of the main obstacles that software community has to conquer to build systems of the future. Issues that are not significant at smaller scale become significant when scale grows larger. Future systems will move far beyond the size of today's systems by every measure: lines of code; people employing the system for different purposes; data storage, accessibility and manipulation. We call these systems as Large Scale Systems (LSS) or Systems of Systems (SOS).

LSS systems will essentially be distributed and allow only limited possibilities of centralized control over data, development, evolution and operation. The reasons to their distributed nature are their development and usage by wide variety of stakeholders with conflicting needs and continuous evolution. More LSS system, does more conflicts are likely to engender.

LSS system will comprises of different variety of sub systems. These sub systems will have their own domain specific languages (DSLs) designed according to different methodologies and philosophies. Sub systems expand from different places that are created and modified by dispersed teams with different schedules, processes and goals. Some of the sub systems may originate from legacies, that were already designed before they were chosen to be part of LSS system. Moreover all new sub systems will be written in different languages and built on variety of platforms.

One way to integrate sub systems and share as much data between them is by resolving all of their conflicts. Resolving conflicts for these independently developed sub systems looks next to impossible. Second way is to live with these conflicts and still share the features and data. In this paper, we propose an approach that help us to integrate two independently developed subsystems without resolving their conflicts.

Our approach introduces an intermediate mechanism between two subsystems; that work as a virtual layer. We name it as Virtual Heterogeneous Model Integration Layer (VHMIL). VHMIL work independent of both sub systems and do not propose any modification in the structure of both the systems. As the matter of fact, it works as a new central independent layer to facilitate sharing of features and data between both the systems. We already stated that it is a virtual layer, which implies that it does not affect the structure of both sub systems. On the contrary, it takes domain specific language of two systems as input plus necessary requirements from their domain experts and defines a new composed language.

In other words, VHMIL works as a filter of features between both the systems. Domain experts from both systems identify the features that what should be and should not be shared. After identification of features when it comes to the step of sharing stage. Some of the features raise conflicts. Shared features of one system may conflict with the second system's existing structure and vice versa. For the simplification of the process of integration we create two categories; one is non conflicting features and other is conflicting features.

Non conflicting features are handled by the visibility

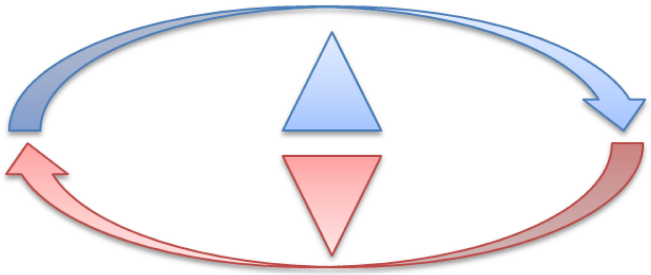

Fig. 1: Path between MM1 and MM2 meta-models

mechanism and conflicting features are handled by invisibility and masking mechanism. Visibility contain those features and data which are needed to be shared and do not generate any conflict. Invisibility contains those features which are not needed to be shared to avoid conflicts. Masking contains those features which are similar in nature and causes conflicts.

Non conflicting : Visibility
Conflicting: Invisibility and Masking

VHMIL creation is an automated activity except the identification of shared features, which domain experts propose manually. Domain experts have to decide which features need to be shared.

II. Model and Metamodel concept

When meta-model state every concept defined in the model and model uses the metamodel concepts according to rules stated by the metamodel then we call that model as instance of that meta-model. Conformance is described by a set of constraints between models and meta-models. When all constraints are validated, the model is said to conform to the meta-model. In this rest of paper we will be using MM for meta-model and M for instance model.

III. VHMIL: Virtual Heterogeneous Model Integration Creation

Consider two meta-models MM1 and MM2, both of them represent language for different systems. MM1 wants to use MM2's some features without modifying its own as well as MM2's language. Similarly MM2 wants to use MM1's some features without modifying its own as well as MM1's language. VHMIL layer consists of two paths. One path is from MM1 to MM2, and other is from MM2 to MM1. Both paths have one an intermediate meta-model as shown in figure 1. Path from MM1 to MM2 is represented as Δ(delta) and path from MM2 to MM1 is represented as $\bigtriangledown$(nebla).

Δ MM12 metamodel share MM1's features for MM2. In the same way MM21 metamodel between path MM1 to MM2 provide MM2 features for MM1.

MM1 # MM2 = MM12 MM2 # MM1 = MM21

Delta and nebla consists of the same structure. Both consists of three sections (Visible, Hiding, Aliasing). Delta makes visible those elements for MM2, which need to be shared, hide those elements which need to hide and cause conflicts and alias those elements which has same syntax and semantics between the MM1 and MM2. Similarly nebla will hold those elements for MM1 which need to be shared for MM2, hide those elements which need to hide from MM2 and alias those elements which has same syntax and semantics between the MM1 and MM2.

Generally, these three sections are explained in the introduction section that what intermediate metamodel consists of and now we will explain in the perspective of metamodel.

We consider that it is more intuitive to represent a delta in operational terms; it is sequence of transformations that make visible , hide or mask elements from a metamodel. We have identified six elementary transformations in a metamodel that will be used as the basis for defining a delta. We assume that it is not possible to change the type of a model element, e.g., a UML Class cannot become a Package, and an element cannot change its UUID.

These operations are:

1) visibleElement: Make visible an element of type of the MM1 which MM2 require.
2) visibleLink: Make visible an MM1link which is required by the MM2.
3) hideElement: Hide an element which is not required by MM2
4) hideLink: Hide an association link which is not required by MM2.
5) aliasElement: Assign an alias(rename) to the element of MM1 required by MM2.
6) aliasLink: Assign an alias to the element of MM1 required by MM2.

IV. Methodology

VHMIL consists of several sub tasks. Figure 2 provides an overview of the proposed approach. We propose an approach to share common concepts between two metamodel variants with the impact of application of those common concepts on their instance models with emphasis on the minimization of manual effort. The envisioned steps are

1) **Requirements collection and writing pre directives:** Input gathering from domain experts of both metamodel variants and writing directives for these requirements manually .
2) **D2. Raw delta concepts:** Taking two meta-models variants plus directives as input and with transformation generate a list of direct (visible, hidden and alias) concepts. Etract information from paper to represent it.
3) **Generation of delta meta-model:** In this step is to generate an intermediary metamodel from list of direct concepts.
4) **Intermediate Directive Transformation (IDT) or Adapter**: This step have two options depending on the requirements of the system. One option is IDT, that take delta meta-model as input and generate a generic rule based transformation. Second option

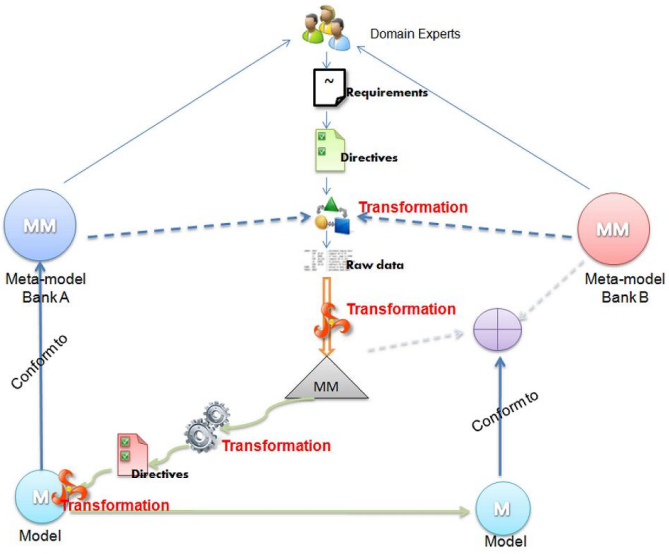

Fig. 2: VHMIL methodolgy diagram

is adapters to provide interface to existing instance model.

5) **Execution of IDT :**This step simply executes transformation on instance model generated from IDT. After applying these directives on the instance models of MM1 will yield new instance models that will work with the MM2 and delta meta-model.

V. Running Example

We present here a small scenario to show that how VHMIL is created on a real time case study. We consider two systems of different banks. Both banks have different systems designed by different teams and built with different programming languages. Both the banks have different type of accounts and services and customers. Management of both banks decided to integrate and share some of the features between these banks in order to facilitate customers or we can call it as partial merger of banks.

Figure 3 shows a metamodel of bank A and figure 4 shows a metamodel of bank B. The ultimate goal is to facilitate customers of both banks to access their accounts and services from both the banks. Simply customer of bank A is able to access his account and associated services from bank B and vice versa.

Δ help customers of bank A to access

To avoid the repetition, I explain only calculation of Δ here, because calculation of $\bigtriangledown$ also follow the same process .

A. Requirements Collection and Writing Pre-directives

In this step, domain experts provide their view that what features have the same similar meaning between both meta-models. In our example, two accounts and one service have same semantics in both banks. Bank B will treat customer of Bank A ”SuperCurrent” account as their basic account and provide ”InternetBanking” service who requests for ”WebInfo” service. Similarly in Foreign Currency account

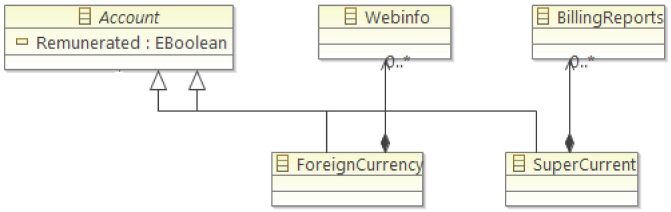

Fig. 3: Meta-model of bank A

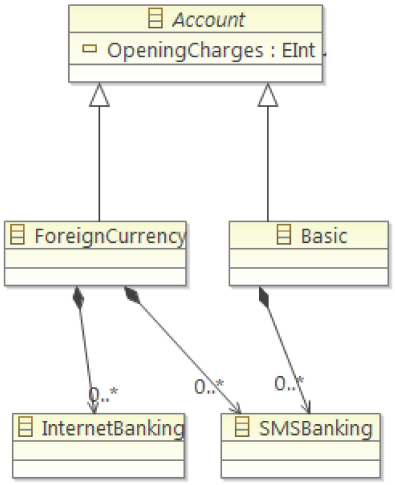

Fig. 4: Meta-model of Bank B

is treated as ForeignCurr as shown in the table I. All other related features will be imported of Bank A meta-model that Bank B does not have. Delta model will import the service of BillingReports from Bank A. Similarly the concept that are not required from Bank A accounts will be make hidden for example ”SMSBanking”.

After identifying the similar concepts, developer writes pre directives for those concepts in directive language as shown below:

1) ABC::ForeignCurrency.name=ForeignCurrency
2) ABC::SuperCurrent.name=Basic
3) ABC::WebInfo.name=InternetBanking

B. Raw Delta Concepts

In this step an automated transformation takes both Bank A, Bank B meta-models and pre directives as an input and produce the raw input of all the concepts containing three parts: visible elements, alias elements and hidden elements for both delt and nebla.

Bank A	Bank B
Account : ForeignCurr	Account : ForeignCurrency
Account : SuperCurrent	Account : Basic
Service : WebInfo	Service :InternetBanking

TABLE I: Similar concepts between Bank A and Bank B

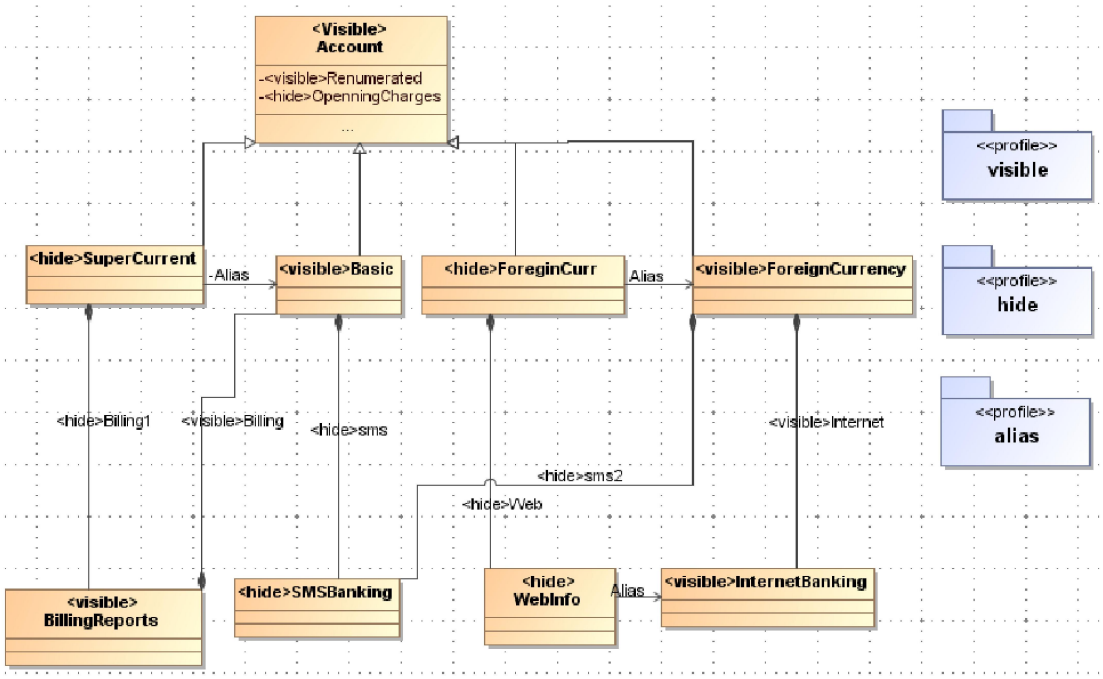

Fig. 5: Representation of delta meta-model for current example

△ Dela meta-model
VisibleElement(Class, BillingReports)
VisibleElement(Parameter, Renumerated)
VisibleLink(Link,SuperCurrent,BillingReports)
VisibleLink(Link,ForeignCurrency,WebInfo)
HideElement(Class, SMSBanking)
HideAttribute(Class,Account, OpeningCharges)
HideElement(Containment, Basic, SMSBanking)
HideElement(Cotainment, ForeignCurrency, SMSBanking)
HideElement(Cotainment, ForeignCurrency, InternetBanking)
Alias(Class, SuperCurrent, Basic)
Alias(Class, Webinfo, InternetBanking)

▽ Nebla meta-model
VisibleElement(Class, SMSBanking)
VisibleElement(Parameter, OpeningCharges)
VisibleLink(Link,Basic,SMSBanking)
VisibleLink(Link,ForeignCurrency, SMSBanking)
VisibleLink(Link,ForeignCurrency, InternetBanking)
HideElement(Class, BillingReports)
HideAttribute(Class,Account, Renumerated)
HideElement(Containment, SuperCurrent, BillingReports)
HideElement(Cotainment, ForeignCurrency, WebInfo)
Alias(Class, Basic, SuperCurrent)
Alias(Class, InternetBanking, WebInfo)

C. Generation of Delta Meta-Model

In this step an intermediary delta meta-model is generated showing the visible, hidden and aliases concepts from list of direct concepts as shown in figure 5.

D. IDT Transformation or Adapter

In this step, the intermediary delta meta-model is translated into an executable generic transformation. The Intermediate Directive Transformation (IDT) takes as an input the final composed delta meta-model, and generate as output a model transformation written in a particular transformation language (e.g., ATL, XSLT, SQL-like, kermeta) as shown in figure 6.

```
import 'model.xmi';
Element : Webinfo
{
name ::=InternetBanking
}
Element : ForeignCurr
{
name ::=ForeignCurrency
}
Element : SuperCurrent
{
name ::=Basic
}
```

E. Execution of IDT

This step simply executes transformation on instance model. After applying transformation generated from IDT directives on the instance models of MM1 will yield new instance models that will work with the MM2 and delta meta-model.

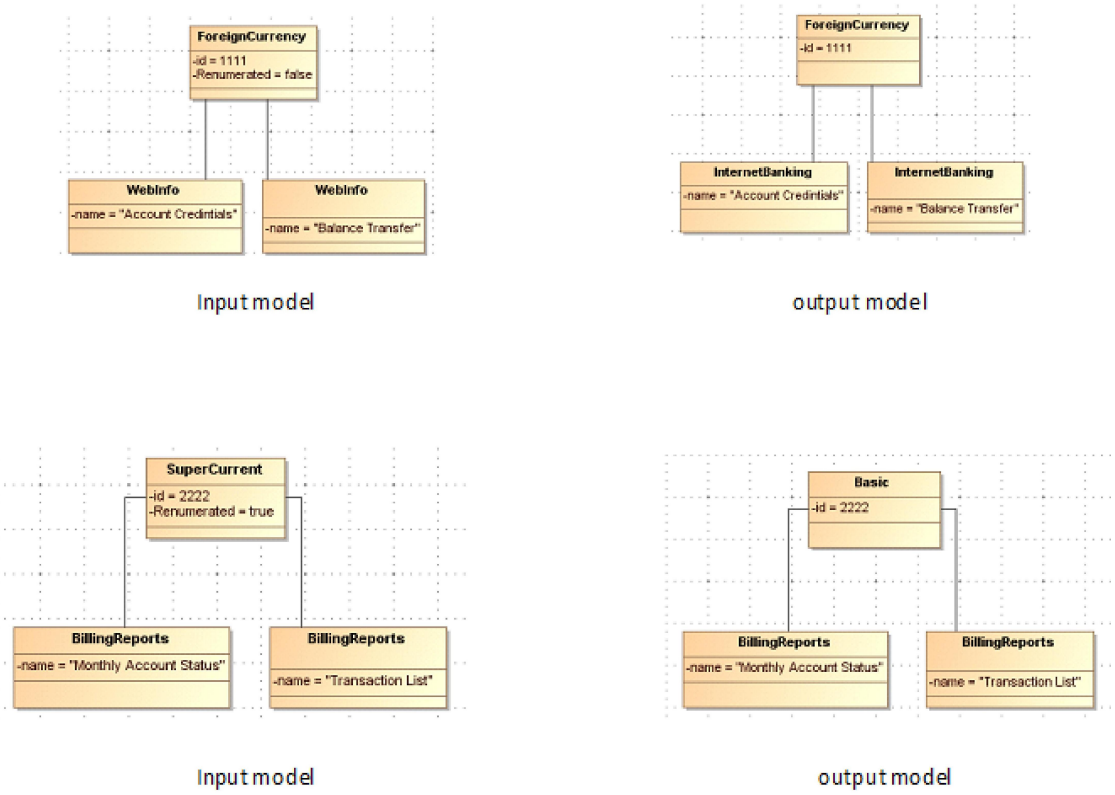

Fig. 6: Transformation of input models of one bank to another using detal and nebla meta-models

VI. Related work

We may divide the related approaches work on the basis of Software Integration and Software Evloution.

Xavier lanc et al. [2] presented a software integration mechanism called model bus. Model Bus is an approach to support data integration and control integration. ModelBus is based on the copy-modify merge mechanism. Models are split into several XML fils that are stored in a central storage, called the model base. Control integration facilitate heterogenous tools to invoke modeling services of each other. Each developer has his own workspace, called the model workspace, that stores only the XMI files modified by the developer. Developers can then modify their models locally and save their modifications to the model base. To manage the integration of developers' updates into the model base, ModelBus proposes dedicated delta extraction and delta integration mechanisms The section.When several developers modify a shared model and then generate different deltas on a same model, a delta integration mechanism has to combine together all the changes they do. Such a mechanism should automatically combine all possible changes but should also raise conflicts when changes cannot be automatically combined.to resolve these conflicts it also provides the conflicts resolving mechanism. In contrast to our approach we donot resolve these conflicts but try to live with these conflicts.[2]

Software Federations [13], [6], [5] approach provides the mechanism to build a large software application by composing Commercial Off The Shelf tools (COTS). This approach is based on two level programming paradigm.

Marcus et al. describes a generic approach for calculation of difference between two versions of models. When two developers change the same subset of the model, this lead to conflicts and it may not be possible to apply all changes in the model. On the contrary, In our approach, we calculate delta model between two model variants but not model versions and we do not compose this difference model to acquire new model. Instead delta model work with the base model in synchronization rather [1].EMF compare is a tool provided by the Eclipse Project. EMF Compare generate delta model, which reports simple changes between terminal models pairs or metamodel pairs [4]

Software Evolution is the most important factor to consider because if software evolves, it affects the validity of other software systems integrated with it. Hoessler et al. [10] identify transformation model for transition of M to M' conforming to the new Metamodel. This model consist of three types of patterns. First pattern is metmamodel extension that describes addition of class or property. Second pattern is metamodel projection for removal of a class or property. Third pattern is factoring for refinement of metamodel(Property Movement,Property Amalgamation, Class Splitting and Class Amalgamation). They do not develop any implementation for their proposed models.

Kelly et al. [7]have presented an approach for adapting model to evolving metamodels. This approach consists of three steps. In first step it computes similarities and differences between base metamodel and evolved metamodel by executing set of heuristics . In the second step HOT (Higher Order Transformation)translate takes all equivalences and differences and yield an executable transformation in ATL. Afterwards the transformation transform the model confirming to the base model to the newly evolved model while preserving the unchanged model elements.

Prawee Sriplakich et al. [18] shows a work of collaborating development and follows the copy-modify-merge approach to MDA models, where copy-modify-merge is consist of two steps delta calculation and delta integration. Further Author also propose a framework for automated to build programs for automated conflicts resolution resulted of deltas modifying same set of elements.

Gracia et al. [9] presents semi-automatic process for metamodel evolution that supports the co-evolution of transformation. The work is devided in two steps: (1) detection stage, where the changes in the metamodel are identified and classified (2) the action for the changes is being taken in co-evolution stage.

Schonbock et al. [17] presented an extensive survey for several co-evolution approaches from different areas such as software engineering like data, ontology, and grammar engineering by considering a certain roadmap for development in the field of co-evolution for agile MDE. Furthermore, the authors presented a conceptual co-evolution framework with help of running example for decrease in co-evolution effort and increase in co-evolution performance.

Garcés et al. [8] presented an approach for transformation that covers external transformation of metamodel. The approach works by identifying the transformation which deal with either refactoring/destruction changes or construction changes. It semi-automatically generates transformation with the help of AtlanMod matching language.

Yu et al. [21] proposed a framework for metamodel which is based on formal analysis for composition and adaptation. The framework is able to explore and evolve metamodel based on certain strategies. It also re-establish uniformity between existing models and the metamodels. The approach proves that model is able to accumulate information from the analysis.

Demuth et al. [3] presented a vision of co-evolution between metamodels and models that has an ability to adopt change propagation. The approach handles co-evolution issues without being dependent on specific metamodels or evolution scenarios. The approach can also detect failures that can occur during co-evolution process and generate appropriate suggestions for corrections.

Paige et al. [16] discussed key problems that occur in evolution process in the Model Driven Engineering (MDE) field. The authors discussed about up to date work of models and metamodels evolution and summarized the challenges it in this paper.

Taentzer et al. [19] proposed an approach which is related to graph transformation for model co-evolution. The models represent graphs with model relation and type-instance relations with respect to their metamodels.

Meyers et al. [15] presented a technique that help the user to solve the issues related to migration from one model to another. This approach simplifies migration specification and reduces the work of evolver. The formal framework of migration of the model which is independent of specific modeling approaches is presented by Mantz et al. [14]. While Wagelaar et al. [20] proposed a model transformation language for coupled evolution in metamodel. It shows executable semantics for EMFMigrate. Iovinoet al. [11] discussed the metamodel change impact of existing artifacts.

OCL plays an important role with respect to models and metamodels. Kusel [12] expressed the solution of actions for all metamodel that violate the changes of syntactic correctness of OCL expressions.

VII. Conclusions

It is a permanent effort in computer sciences, to find ways to integrate existing applications with new ones. This paper presents the current state of work in attempting to overcome the difficulties involved in application integration and reuse each other's services. While services in each application have different signature and structures, which may cause conflicts when trying to be integrate with each other. We propose an intermediatory layer to facilitate this integration of applications by eliminating the need of resolving these conflicts. In contrast conflicts are being handsled in a way that there conflicting behaviours effect the integration. Future work involves rigerous testing, verification and validation of our approach with large models.

References

[1] Marcus Alanen and Ivan Porres. Difference and union of models. pages 2–17, 2003.

[2] Xavier Blanc, Marie-Pierre Gervais, and Prawee Sriplakich. Model bus: Towards the interoperability of modelling tools. In *MDAFA*, pages 17–32, 2004.

[3] Andreas Demuth, Markus Riedl-Ehrenleitner, Roberto E Lopez-Herrejon, and Alexander Egyed. Co-evolution of metamodels and models through consistent change propagation. *Journal of Systems and Software*, 111:281–297, 2016.

[4] Eclipse.org. Emf compare, (2008). http://wiki.eclipse.org/index.php/EMF.

[5] J. Estublier, H. Verjus, and P. Y. Cunin. Modelling and managing software federations. volume 26, pages 299–300, New York, NY, USA, 2001. ACM.

[6] Jacky Estublier, Herv Verjus, and Pierre yves Cunin. Designing and building software federations. In *1st Conference on Component Based Software Engineering. (CBSE*, 2001.

[7] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. Managing model adaptation by precise detection of metamodel changes. In *ECMDA-FA '09: Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, pages 34–49, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] Kelly Garcés, Juan M Vara, Frédéric Jouault, and Esperanza Marcos. Adapting transformations to metamodel changes via external transformation composition. *Software & Systems Modeling*, 13(2):789–806, 2014.

[9] Jokin García, Oscar Diaz, and Maider Azanza. Model transformation co-evolution: A semi-automatic approach. *Software Language Engineering*, 7745:144–163, 2013.

[10] Joachim Höler, Hajo Eichler, and Michael Soden. Coevolution of models, metamodels and transformations. Wissenschaft & Technik Verlag, June 2005.

[11] Ludovico Iovino, Alfonso Pierantonio, and Ivano Malavolta. On the impact significance of metamodel evolution in mde. *Journal of Object Technology*, 11(3):3–1, 2012.

[12] Angelika Kusel, Juergen Etzlstorfer, Elisabeth Kapsammer, Werner Retschitzegger, Johannes Schoenboeck, Wieland Schwinger, and Manuel Wimmer. Systematic co-evolution of ocl expressions. *11th APCCM*, 27:30, 2015.

[13] Tuyet Le-anh, Jorge Villalobos, and Jacky Estublier. Multi-level composition for software federations. 2003.

[14] Florian Mantz, Gabriele Taentzer, Yngve Lamo, and Uwe Wolter. Co-evolving meta-models and their instance models: A formal approach based on graph transformation. *Science of Computer Programming*, 104:2–43, 2015.

[15] Bart Meyers, Manuel Wimmer, Antonio Cicchetti, and Jonathan Sprinkle. A generic in-place transformation-based approach to structured model co-evolution. *Electronic Communications of the EASST*, 42, 2012.

[16] Richard F Paige, Nicholas Matragkas, and Louis M Rose. Evolving models in model-driven engineering: State-of-the-art and future challenges. *Journal of Systems and Software*, 111:272–280, 2016.

[17] J Schonbock, Juergen Etzlstorfer, Elisabeth Kapsammer, Angelika Kusel, Werner Retschitzegger, and Wieland Schwinger. Model-driven co-evolution for agile development. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pages 5094–5103. IEEE, 2015.

[18] Prawee Sriplakich, Xavier Blanc, and Marie-Pierre Gervais. Supporting collaborative development in an open mda environment. In *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 244–253, Washington, DC, USA, 2006. IEEE Computer Society.

[19] Gabriele Taentzer, Florian Mantz, and Yngve Lamo. *Graph Transformations: 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29, 2012. Proceedings*, chapter Co-transformation of Graphs and Type Graphs with Application to Model Co-evolution, pages 326–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[20] Dennis Wagelaar, Ludovico Iovino, Davide Di Ruscio, and Alfonso Pierantonio. *Theory and Practice of Model Transformations: 5th International Conference, ICMT 2012, Prague, Czech Republic, May 28-29, 2012. Proceedings*, chapter Translational Semantics of a Co-evolution Specific Language with the EMF Transformation Virtual Machine, pages 192–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[21] Ingrid Chieh Yu and Henning Berg. A framework for metamodel composition and adaptation with conformance-preserving model migration. In *Model-Driven Engineering and Software Development*, pages 133–154. Springer, 2015.