Incremental Model Synchronization in Model Driven Engineering

Hamid Gholizadeh
Supervisor: Dr. Tom Maibaum
Committee Members: Dr. Wolfram Kahl, Dr. Jacques Carette

March 19, 2013

Abstract

Abstract parts goes here.

1 Introduction(general introduction starting from MDE to reaching Synchronization goes here: Synchronization Definition, demostrate its importance by giving example from Software development, you can refer to Landscape of bidirectional model synch ...

MDE (Model Driven Engineering) is proven to be a demanding approach in software development process [12, 10]. One of the intentions of the MDE approach in Software Development is to bring the intelligence of the human being to the more abstract level of models rather than concrete level of the codes. In Model Driven Software Engineering (MSDE), models are to be considered as active and first-class citizens of the software engineering process artifacts as opposed to the current practice where models –like UML diagrams– are usually used as a supportive and rather passive artifacts, e.g. for software development team inter-communications. The latter happened because of the lack of complete formality and tool support for maintaining models; for example *Model Management* as a core task of MDE, still lacks enough formal foundation and applicable tools, supporting its demanded functionalities like *Model Correspondence Specification* and its maintenance in terms of *Model Synchronization* or *Change Propagation*, and *Conflict resolution.* There are still many research at the moment on some other aspects of *Model Management* like *Model Transformation* which is discussed in the Section 4 in more details.

In a typical practice of software engineering process, models are evolving form more abstract levels to more concrete levels of the code. we call this process *model refinement* and can look at it like vertical movement from more abstraction to less abstract levels. This *Vertical* movement usually needs to be taken in a reverse direction which is also called *reverse engineering*– to fit and accomplish in the iterative method of software development process, where it is called *round-trip engineering.* During round-tripping process, sometimes it is necessary to move between the models at the same levels of abstraction –movement not changing the level of abstraction, Therefore it is called horizontal movement as apposed the latter vertical movement. That is, like the previous case can be in a forward and backward direction (Figure 1.)

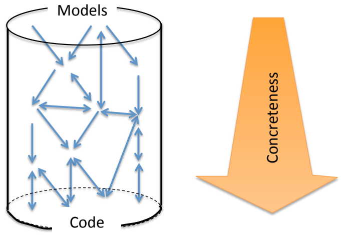


Figure 1: Vertical and horizontal movements in software eng.

The horizontal movement is because each model might demonstrate and focuse on different aspects of the software system, or maybe some models are suitable for different users to work with. For example, in a typical development process of software development using UML diagrams(Models,)[1] each model share some common parts with other models and some have their own specifics feature. E.g. *Class Diagram* and *State Chart* diagram can share some of the class method signature, while attribute names for the classes, and the order of the events happening are consequently considered as a specific(private) part of each.

Because of the inevitable changes which is frequently happing during the software de-

[1] we consider a diagram as a model, but not the other way around, i.e. models not need to be diagrammatic.

velopment process, and also during the maintenance phase [13], it is necessary to handle the *change propagation* which is actually is the semantic of the "movement" in previous paragraph. Hence change propagation can happen *Vertically* or *Horizontally*. By *Vertical* change propagation we mean a Model *refinement* from more abstract level to more concrete one or vice versa. By *Horizontal* refinement we mean change propagation among the models, since they are sharing some concept together as discussed in previous paragraph. Taking software artifacts a models, industrial projects would include hundreds or even thousands of artifacts which are inter-related to each other in some ways, so changing one artifact would require change propagation throughout the whole system. Handling this situation by user without a tool support even in small scale is tedious, complicated, error-prone and very time consuming and for the large cases is nearly impossible. That is why nowadays, much emphasis is put on the code rather than a model, which made Software Engineering deprived of many proved benefit of Model Driven Software Development. Ending up with the inconsistent artifacts of the software development is the typical story of the large scale software projects which most of them are usually looked as a outdated historical record and rather as a decorative pieces than the live and valuable software entities

This thesis proposal examines underlying formalities and existing works in the area of *Model Synchronization* and identifying some gaps in current works and motivates how addressing those gaps is necessary in specification and implementation of formal synchronization framework, since it would provide formal foundation for verifying the implemented synchronization framework. To be more specific we intend to address the formalities underlining the *Shared* and *Private* parts of the related models which matters in *Model Synchronization*. More detailed discussion on the problem is covered in section 5. We will also discuss briefly some other open problems, including *Concurrent Model Synchronization* and *Conflict Resolution* strategies – which might be an extended work of this research, that we observed during the survey on *Model Synchronization* area for future reference.

You can put some examples here as an introduction of the case.

You need to define Model Synchronization somewhere

2 Background :some backgrounds goes here, from the definition of model formally, what is ATG, what is meta-model, What is State-based lenses, what is delta-base, what is Graph Grammer, Synchronization Properties : Correctness, Completeness, Hipocraticnes, Heridatiry and ....

By **Synchronization** we mean reconstructing the *Relation* between two *Models.* These two models already might have been in a consistent state, i.e. are related by R, or might be newly created models which consistency was not established between them yet. Models formally can be represented as an Attributed Types graph [23, 20]. We are going to define some basic definitions regarding the formal foundation of Models in the following. These definition are adapted from [20, 24]

2.1 Model as an Typed Attributed Graph

In general a Model can be represented as a Graph which consist of a nodes and edges connecting these nodes. e.g. a Class diagram in UML is a graph with classes as a node and the associations between the classes as a node. Since we need a richer structure than a simple graph we need to extend the formality of a graph to make it possible to capture further values annotated to the nodes and edges, Hence Attributed Graphs are introduced. Attributed graph are like ordinary graph extended in a way that for every *Node* and *Edge*, it is possible to introduced a Data called *Attribute* and bind that by an edge called *Attribute Edge* to corresponding Nodes and Edges. So, first we need to extend the concept of regular graph to make it possible for the edges to have outgoing edges like nodes. This extended version of graphs is called *E-Graph* and introduced in the following.

Definition 1. *An **E-graph G** is defined as* $G = (V_G, E_G, V_D, E_{NA}, E_{EA}, (source_j\ , target_j)_{j\in\{G,NA,EA\}})$

- $(V_G, E_G, source_G, target_G)$ *constitutes a regular Graph with* V_G *as Nodes,* E_G *as Edges and* $source_G$ *and* $target_G$ *as source and target functions respectively.*

- V_D *is a Data Nodes*

- *E_{NA} and E_{EA} are called Node Attributes and Edges Attributes, which are connecting respectively V_G and E_G to V_D.*
- *source and target functions are defined as below:*
  - *$source_G : E_G \to V_G$, $targetG : E_G \to V_G$;*
  - *$source_{NA} : E_{NA} \to V_G$, $target_{NA} : E_{NA} \to V_D$;*
  - *$source_{EA} : E_{EA} \to E_G$, $target_{EA} : E_{EA} \to V_D$;*

Definition 2. ***morphism between E-graphs*** *G^1 and G^2 with $G^k = (V_G^k, V_D^k, E_G^k, E_{NA}^k , E_{EA}^k, (source_j^k, target_j^k)_{j \in \{G,NA,EA\}})$ for $k = 1, 2$ is defined as $f : G^1 \to G^2$ which is a tuple $(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$ with $f_{V_i} : V_i^1 \to V_i^2$ and $f_{E_j} : E_j^1 \to E_j^2$ for $i \in \{G, D\}$, $j \in \{G, NA, EA\}$ such that f commutes with all source and target functions, for example $f_{V_G} \circ source_G^1 = source_G^2 \circ f_{E_G}$*

Definition 3. *E-graphs and E-graph morphisms form the* ***Category EGraphs****.*

Attributed Graphs are Graphs that is accompanied by a Data Algebra in a sense that the Data Nodes are taken from the subset of algebra *Carrier sets*. To be more precise, we first distinguish a subset of a Carrier set of an algebra and consider the contents of each member in this subset as a *Data Node*. In graphical illustration of the Attributed Typed Graph we ignore those *Data Nodes* that are not reachable from the underlying Graph Nodes and Edges. See an example in [20] for more details.

Definition 4. *Let $D_{SIG} = (S_D, OP_D)$ be a data signature with attribute value sorts S_D and operations OP_D and take $S'_D \subseteq S_D$. An* ***Attributed Graph(AG)*** *$AG = (G, D)$ consists of an E-graph G together with a D_{SIG}-algebra D such that V_D is disjoint union of the selected carrier sets, i.e. $V_D = \dot{\bigcup}_{s \in S'_D} D_s$ where D_s is coresponding carrier set of s.*

Definition 5. *an* ***Attributed Graph Morphism*** *$f : AG^1 \to AG^2$ for two AG graphs $AG^1 = (G^1, D^1)$ and $AG^2 = (G^2, D^2)$ is a pair $f = (f_G, f_D)$ with an E-graph morphism $f_G : G^1 \to G^2$ and an algebra homomorphism $f_D : D^1 \to D^2$ such that diagram (1) commutes for all $s \in S'_D$, where vertical arrows are inclusions:*

$$\begin{tikzcd}
D_s^1 \arrow[r, "f_{D,s}"] \arrow[d, "\cap"'] \arrow[dr, phantom, "(1)"] & D_s^2 \arrow[d, "\cap"] \\
V_D^1 \arrow[r, "f_{G,V_D}"] & V_D^2
\end{tikzcd}$$

For defining the Typed attributed Graph we need to first define the concept of final D_{sig}-algebra for a given algebra. final D_{sig}-algebra for an algebra is mapping somehow the name of the sort as a set to the sort as its interpretation. So cardinality of the interpreted sorts would be 1 for all of them. It maps the constant symbol of sort S_i, to the singleton member of corresponding carrier set Z_{S_i}, and following that, interpret the result of a function application on carrier set to the corresponding singleton member of the result sort.

Definition 6. *Given a signature $SIG = (S, OP)$, the **final** SIG-**algebra Z** is defined by $Z = (S_z, C_z, OP_z)$ where:*

- $S_z = D_s$ *where* $D_s = \{s\}$ *for each sort* $s \in S$*;*
- $C_z = k \in D_s$ *for a constant symbol* $c_s :\to s \in OP$*;*
- $OP_z : \{s_1\} \ldots \{s_n\} \to \{s\} : (s_1 ... s_n) \mapsto s$ *for each operation symbol* $op : s_1 ... s_n \to s \in OP$

Definition 7. *Given a signature $SIG = (S, OP)$, an **Attributed Type Graph(ATG)** is an attributed graph $ATG = (TG, Z)$ where Z is the final SIG-algebra.*

So for having an Attributed Type Graph as a Type of another Attributed Graph, we need to somehow specify the typing of the Attributed Graph Data Structure(Data Algebra) too. That is why we need to define a final Sig-algebra. Typing of the Attributed Graph Data Structure is represented by the final SIG-algebra of the corresponding Data Signature. In the following we define Typed Attributed Graph (TAG).

Definition 8. *A **Typed Attributed Graph(TAG)** over ATG is defined as (AG, t) where AG is an attributed graph and t is an attributed graph morphism from AG to ATG*

Please observe that the data-signature of of the Attributed Type Graph and Typed Attributed Graphs, Typed over that would be the same, as the definitions above implies.

Definition 9. *A* ***TAG morphism*** *between* $TAG^1 = (AG^1, t^1)$ *and* $TAG^2 = (AG^2, t^2)$ *i.e.* $f : TAG^1 \to TAG^2$ *is an Attibuted Graph Morphism(AGT morphism)* $f : AG^1 \to AG^2$ *such that* $t^2 \circ f = t^1$*:*

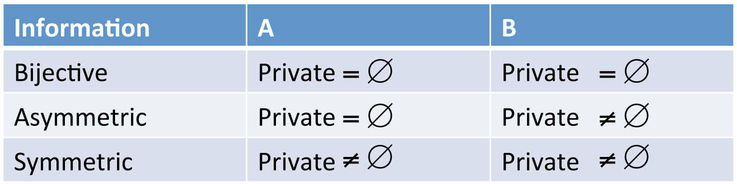
$$\begin{tikzcd}
AG^1 \arrow[rd, "t^1"] \arrow[dd, "f"] & \\
& ATG \\
AG^2 \arrow[ru, "t^2"'] &
\end{tikzcd}$$

In terminology of the Software Engineering, *Models* are equal to *Typed Attributed Graphs* and *Meta-Models* are equal to *Attributed Type Graphs.* This can be extended to more than one level, while meta-meta-model would be Attributed Type Graph for Attributed Type Graph of its beneath level as a Type. We should observe that all the *Entities* (Including Models, Meta-Models and Meta-Meta-Models) are Attributed Graphs, and the Data-signature of the connected *entities* by typing association(Typing Mapping) would be the same. In another word data-signature of the Model, Meta-Model and Meta-Meta-Model, all, would be the same.

2.2 3-Dimension of Synchronization Scenarios

In this section we describe the three dimensions of the synchronization scenarios, including *Information Symmetry*, *Incrementality* and *Concurrency*, and discuss that they are orthogonal(independent) to each other, in the sense that each is describing an independent aspect of the synchronization scenario. The extended version of these dimensions are discussed in our recent submitted paper in [**?**]. we assume two model spaces(set of models) **A** and **B** and they are supposed to be aligned(consistent) w.r.t. a relation **R**. It should be mentioned that we assume the more common cases of synchronization which only two models are involved.

Information-Symmetry captures that if two models are informationally symmetric or asymmetric. In the asymmetric case, Model A information can be completely derived by Model B. it means that having just model A we can get the model B uniquely; e.g. in common table-view scenario [7, 39] view content is completely derivable from the corresponding table content. In Symmetric case neither A, nor B can be informationally derivable from each other. e.g. Class Diagram and Sequence diagram neither can be derivable from each other uniquely. Another perspective of looking at Asymmetry and Symmetry cases can be based on *Private* and *Shared* parts of the A and B. In *Asymmetric* case A has its own *private* and *shared* part and B only consist of a shared part, and the private part would be empty. In *Symmetric* case, neither of private and shared parts for A and B are empty (ref. Figure 2) The latter case is the more general case of the synchronization scenarios in practice. Formalization of this private and public part would be the main contribution of this PhD thesis which is discussed in more details in Section 5. There is a especial case that both private parts of A and B are empty, this is called the *informationally bijective* case. In these cases two models are informationally equal but differently represented. Three cases of information Symmetry is summarized in table bellow:



Information	A	B
Bijective	Private $= \varnothing$	Private $= \varnothing$
Asymmetric	Private $= \varnothing$	Private $\neq \varnothing$
Symmetric	Private $\neq \varnothing$	Private $\neq \varnothing$

Figure 2: Private and Shared Parts in Information Symmetry

Incremetality of the Synchronization can be divided into three cases in terms of its maturity of the method. These three categories are called *Batch-update*, *State-based* synchronization and *Delta-based* synchronization. In the *batch update* model B is completely calculated from scratch whenever changes happen in model A. This naive strategy is not efficient at all in large model spaces −which are usually the case of synchronization application in practice, since for any even small changes on Model A, considerable effort is imposed on the synchronization mechanism to calculate the Model B. That is why it is usually used in small scale scenarios like type hierarchy update of a Java code [4]. Propagation function in this approach accepts A' (Changed A) as the mere input, and generates consistent B out of that from scratch. In comparison, state-based synchronizer takes A' and B as inputs, trying to apply the *Minimum Changes* to B and make it aligned with A'. This is far better than the previous case and commonly is referred to as *Incremental synchronization*. It is shown in [16] that taking A' and B (states of the models) is not always enough in reconstructing the alignment between two models efficiently. So the more mature way of incrementally approach is providing the *Deltas*(Updates) as inputs −rather than states, for the propagation function. we refer to the latter case as *Delta-Based* synchronization.

In both *State-based* and *Delta-based* cases of synchronization it is not yet clearly formalized, what would be the *Minimum Change* as mentioned in previous paragraph. This is another place that this thesis aims to contribute.This idea is closely related to the *private* and *public* part concepts. Since for example we need that the synchronizer doesn't change the *private* part of B at all and changing of the *shared* part of it be minimal when we say *Minimal Changes* on B.

Concurrent synchronization means that if two models are allowed to evolve concurrently or not. when we state that a synchronization is not *Concurrent*, we mean that Models A and B can not be changed simultaneously. It another word, if Model A is evolving to A' we need to first propagate its changes to B and later on, evolution of Model B would be allowed. In implementation it would mean that if even both models evolve simultaneously, we would give more importance on changes of one model and taking it as as *Master* (say A), while ignoring the changes of the other(Say B) and trying to make them aligned again. In comparison, in *Concurrent* scenario, simultaneous changes to the models are allowed and both model changes are taken into consideration in the alignment process. It means that synchronization framework tries to make the *Minimum Changes* on evolved Models A' and B' to produced the new aligned Models A'' and B''. The later case is more mature and the most desirable case of synchronization.

Orthogonality of the three aspects –*Information symmetry*, *Incrementality* and *concurrency*, is somehow clear from the previous paragraphs. For each cases of information symmetric and asymmetric and bijective, and also for every case of *batch-update*, *state-based* update and *delta-based* update, we would have two possibility of concurrent and non-concurrent case. Also being *Information symmetric*, *asymmetric* or *bijective* doesn't depend on the approach of the incrementality. Hence we would have three orthogonal dimension in synchronization scenarios which give rise to 18 different cases in synchronization framework. Figure 3 demonstrate 9 different types. We omit the *concurrency* dimension for simplifying the picture. Considering two cases of concurrency all cases would double.

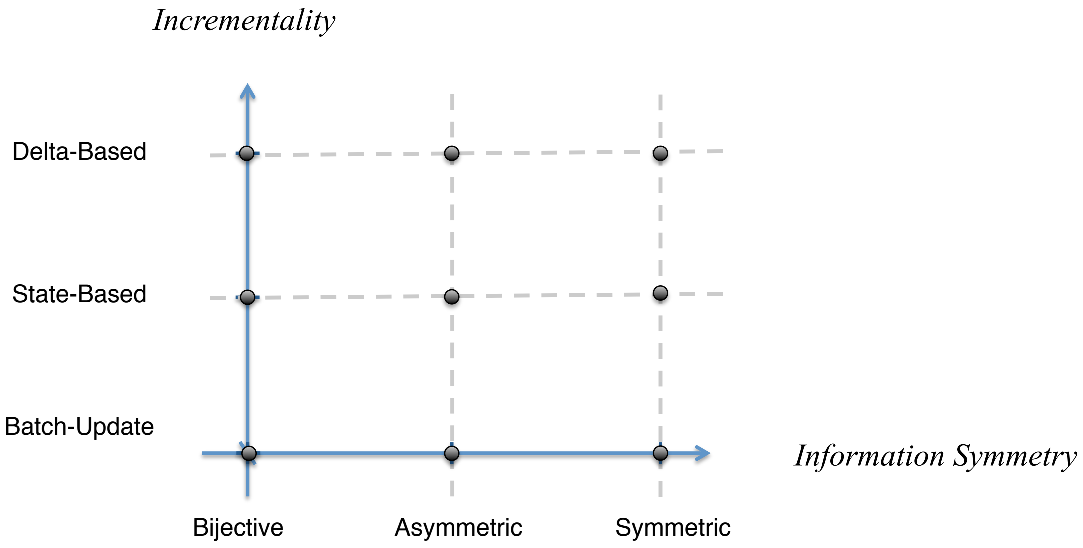


Figure 3: Different Synchronization Scenarios

2.3 Pair of Transformers and Bidirectional Transformation Languages

When we Transfer one model(say A) to another Model(say B) by writing a so-called *forward* transformation, we usually desire to have a *backward* counterpart which transfers B to A. To make our discussion more abstract, suppose that there is a Relation R between A and B as a set of Models, and *Pair of Transformers*(*forward* and *backward*) which are in accordance with this Relation[2]. Often in practice the R is not explicitly defined and just

[2] By in accordance we mean holding some properties like *Correctness*, *Completeness* and etc. which is defined later. *Correctness* property is necessarily basic one and should hold for all pairs of transformers.

the transformers is defined using some transformation languages. For example languages like *ATL* [40], *Viatra2* [6], *QVT-O* [1] or even Regular programming languages like *Java* are used for writing Transformers. In these situations getting a backward transformation is not trivial and oftentimes requires the user to code the backward transformation in hand. It is often desirable that these *forward* and *backward* transformers be consistent [3] [17, 28, 50] and maintaining the consistency in pair of separate transformers which is both coded by the user is not easy and straight forward. That is why the concept of the *Bidirectional Transformation Languages(BX)* in which *Consistent* transformers can be extracted automatically out of one transformation specification is emerged. This specification in *BX* is somehow more explicitly specifying the Relation R between A and B. For example *QVT-R* [1], *Triple Graph Grammars(TGG)* [46], *Lenses* Framework [28] and *GroundTram* [36] frameworks are providing *BX* facility. From the aforementioned frameworks, only TGG works on TAG[4]. Table below summerises which datastructure they support as a models:

Framework	**Model Representation**
QVT-R	UML Meta-Model
TGG	Typed Attributed Graph
GroundTram	Labeled Graph
Boomerang	String

Although having BX is desirable but in theory and practice there are some obstacles in making them applicable : (1) Semantic of extracting a consistent backward transformation is not trivial, (2) There doesn't exist yet a rule of thumb what is the appropriate collection of properties for the pair of transformers in different application domains (ref. Subsection 3.1)(3) Writing a transformer in BX languages like QVT-R is more complicated and tricky for the user than writing the same transformer in one-directional transformation languages.

Out of aforementioned frameworks for the BX, each one has its pros and cons, but non is yet gained the maturity enough to be widely used in industrial cases. Out of those, since TGG are supporting TAG, we will discuss that in .... in more detail and will examine its specifications in Sec...

[3] refer to footnote 2 at page 10

[4] refere to Definition 8

(You can Write a section on TGG and formally define it and discuss it formally and say its strenghes and weeknesses)

Now write down properties name and define them:

3 Consistency properties of transformers

It is desirable that pair of transformers(forward/backward) satisfy some consistency properties. For Example *Correctness* of the transformers is the rudementary requirement which insures that transformers operate correctly. Other consistency properties like *Completeness*, *Hippocraticness*, *Undoability* and *Invertability* are also discussed in this Section. When we talk about model *Synchronization* we desire that the corresponding transformers become consistent, where the consistency of the transformers is defined as conforming to subset of the following properties depending on the domain of the synchronizer application. First we define the Correctness property.

3.1 Correctness

Let's call *forward* and *backward* Transformers $\overrightarrow{R}$ and $\overleftarrow{R}$ consequently supposing that they are total functions, and suppose that Relation R is a desirable relation between two Set of Models(Meta-Models) A and B. *Correctness* property as it is expected, simply ensures that a *Pair of Transformers* are functioning correctly.

Definition 10. *A Pair of Transformers* $(\overrightarrow{R}, \overleftarrow{R})$ *are* ***Correct*** *w.r.t.* $R \subseteq A \times B$*, if* $(a, \overrightarrow{R}(p_1)) \in R$ *and* $(\overleftarrow{R}(p_2), b) \in R$ *where* p_1 *and* p_2 *are set of input parameters.* p_1 *for* Batch-update*,* State-based *and* Delta-based *cases(ref. 2.2) are* $\{b\}$*,* $\{a, b\}$ *and* $\{\Delta a, b\}$ *consequently where* $a, a' \in A, b \in B$ *and* $\Delta a = (a, \Delta, a')$ *for* Δ *being a mapping between elements of* a *and* a'*.* p_2 *is defined the same way as* p_1 *if we interchange* a *and* b *in* p_1*.*

This property is defined in [50] and [16] for two latter cases of *Information Symmetric* cases. In [11, 37] this property is defined for *Information Asymmetric* and *State-based* Cases.

3.2 Hippocraticness, or check-then-enforce

This property states that the transformers should not cause any change if two models are already consistent w.r.t. R.

Definition 11. *A Pair of Transformers* $(\overrightarrow{R}, \overleftarrow{R})$ *are* ***Hippocratic*** *w.r.t.* $R \subseteq A \times B$*, where* $\forall a \in A$ *and* $\forall b \in B$*: if* $(a, b) \in R$ *then* $\overrightarrow{R}(p_1) = b$ *and* $(\overleftarrow{R}(p_2) = a)$

p_1 and p_2 are defined like the Definition 10. The above property is called *"check-then-enforce"* in QVT-R specification [1]

3.3 Undoability

Undoability is introduced by Steven in [50]. That means if we undo the changes of the A, any changes that happened on B because of synchronization, should be rolled back to exactly its original state. We define this property for *State-Based* scenario as we would see it is trivial for *Batch-Update.*

Definition 12. *A Pair of Transformers* $(\overrightarrow{R}, \overleftarrow{R})$ *are* ***Undoabe*** *w.r.t.* $R \subseteq A \times B$*, where* $\forall a, a' \in A$ *and* $\forall b, b' \in B$*: if* $(a, b) \in R$ *then* $\overrightarrow{R}(a, \overrightarrow{R}(a', b)) = b$ *and* $\overleftarrow{R}(\overleftarrow{R}(b', a), b)) = a$

For the *batch-update* case this property is trivial and holding all the times, since we assume that the transformers are functions, then rolling back a' to a and executing the $\overrightarrow{R}$ on that we are obviously expecting to get b again. However according above definition this might not be the case for the *State-based* or *Delta-basdd* scenarios.

It is arguable that this property is suitable to be imposed on the *Synchronization Implementation Environment* rather than the Synchronization *Transformers.* Since at the moment *undoability* is well supported by modelling environments, while we believe imposing *Transformers* to support that would be more restrictive.

3.4 Invertability

It is informally means that starting from one model and going forward and backward by transformers, we will end up with the same model. In another word it states that the *forward* and *backward* transformers are the inverse of each other functioning in reverse directions.

We define invertability for the *State-based* cases, and for the rest it would be the same. Following definition is adopted from [15].

Definition 13. *A Pair of Transformers* $(\overrightarrow{R}, \overleftarrow{R})$ *are **Invertabe** w.r.t.* $R \subseteq A \times B$, *if* $\forall a, a' \in A$ *and* $\forall b, b' \in B$: $\overrightarrow{R}(a, \overleftarrow{R}(a, b', b), b) = b'$ *and* $\overleftarrow{R}(b, \overrightarrow{R}(a, a', b), a) = b'$ *where* $\{(a, b), (a', b')\} \subset R$

3.5 Completeness

Completeness says if there exists at least one model, say b, which is in Relation with Model a, then transformer should be able to transform a to it. In another word *Completeness* states that the transformers should be total functions and return nonempty in the case there is a counterpart for its input(s) regarding relation R

Definition 14. *A Pair of Transformers* $(\overrightarrow{R}, \overleftarrow{R})$ *are **Complete** w.r.t.* $R \subseteq A \times B$, *if* $\forall a \in A, \overrightarrow{R}(a) \neq \emptyset$ *if* $\exists b \in B$ *where* $(a, b) \in R$ *and if* $\forall b \in B, \overleftarrow{R}(b) \neq \emptyset$ *if* $\exists a \in A$ *where* $(a, b) \in R$

4 Literature Review

Talking about Model Synchronization, we already tied with all other aspects of MDE concepts including : *Model Transformation*, *Meta-Modeling*, *Model Specification Language*, *Constraint* and *Bidirectional Model Transformations(BX)*. At the moment there are many research activities progressing in each area independently and inter-relatively.

Considering *Models* as *Graphs*, *Graph Transformation* Techniques are considered as a *Model Transformation* foundation in that sense. The book in [45] discusses Graph Grammar and DPO(Double Push out) and SPO(Single Push out) approaches as two well-known specification for *rule application* definition in Graph Transformation . Another book [20] discusses some algebraic foundations of graph transformation using Category Theory [8, 5]. These two books are providing informative materials in domain of Graph Grammars and Graph Transformation which are closely related to model specification and model transformation basics respectively. Attributed Typed Graph(ATG) are introduced in [18] as a formal representation of Models. There are some alternatives for ATG specification which are also discussed in [?].

There are many research at the moment progressing on *Graph Transformation* and also some academic tools have been implemented for them. [?, ?, ?]. Among them *Triple Graph Grammars*(TGG) as an extension of regular Graph Grammars and their tools holds more than 15 year research background [47] and some successful application [?]. TGG rules are specifying source and target model evolution in a constructive manner. It is introduced by Andy Shüre in [46] and formally defined in [43], [19] and [32] recently. TGG approach is taking Graph Grammars as a Triple of *Source*, *Correspondence*, and *Target* Models and defining the relationship between them as a Triple Graph Grammar Rules [41]. TGG is considered as a Graphical Bidirectional Transformation Languages (BX) too, which forward and backward rules can be extracted from the *Triple Rules* in a straight forward manner [?]. Some properties of of TGG like *Functional Behaviour*, *Correctness*, *Completeness*, and *Termination* are formally studied here [35, 22, 21]. Besides the Theory of the TGG there are some tools developed for that in academia and some of them applied successfully for industrial projects [?, ?]. Among these we can mention MoTE [29], eMofelon [2], TGG Interpreter [33] and Emorf [42]. Although Bi-directionality of the TGG provides some advantages over its usage in term of providing the consistent forward and backward rules for granted, at the same time its usage is limited in defining complex transformation scenarios which are demanded in most industrial cases [?]. Besides there are still some space left for research to formalize the application of *Constraints* in rule definition which is necessary to use in most of its applications [?].

Giese and Wagner discuss incremental model synchronization implementation in TGG in [30]; But no formal foundation is provided to verify how far their incremental implementation is correct in the sense of being *Incremental.* This is actually one of the objectives of this PhD research to establish a mathematical specification which incrementality of the synchronization can be verifiable. Regarding Synchronization Techniques in TGG, [34] studies Conflict Resolution strategies in TGG approach. (Maybe you need to read that paper and put some comments here).

Lenses Framework is discussed in [37]. It is trying to provide a consistent forward and backward transformation out of single specification language. Boomerang framework [11] is implemented based on that idea. But the language application is limited to the String Type of Data at the moment and covers the Asymmetric cases in Synchronization Scenario. There is no discussion on the incremental update integration to the framework and the emphasize in on defining BX language. GroundTram [36] is another thrend trying to define a BX programming languages based on edge-labeled Graphs. They are using $UnQL^+$ language(an Extension of UnQL [14]) for defining the Bidirectional Transformation, but again the emphasize is on the Bidirectionality rather than the incrementality.

(Should read Zinovy Tile Algebra paper and also Algebraic approach to comment on that) (I also need to read xiong paper on incremental approach and comment on that) (I also need to read TGG sychronization of Ehrig and Zinovy and comment on that)

Shynchronization scenarios is categorized in eighteen different categories in [4]

ofGrapEhrig et. al discusses the fundamentals of algebraic Graph Transformation is a seriese of books, and articles. defining Attributed Typed Graph as Model, Graph Grammars and loot[Some Ref here to TGG, all ref to TGG, GroundTram, , Constraints: donloaded constraint papers, BX: Crystoph paper] . Among aforementioned Subjects the first one, Model Transformation and BX is closely coupled with Model Synchronization. At the moment there are couple of approaches [TGG, Lenses, GroundTram] , languages [Boomrang, ATL, QVT, Viatra2, and many academic tools [AGG, Emofelon, MoTE, TGG-Interpreter, Emorf, MediniQVT] developed based on those thories and there is active research groups working on them.

mention the tools related to TGG, TGG-interpreter, Mofelon, Gies tool, emorf, emf, japan tool, mediniQVT, smartQVT, SDM,

5 Problem Definition

I think it is possible to move some parts from the background to here like the space of synchronization

Almost all Synchronization formalization is based on the assumption that we have R clearly defined by some specification constraints [**?**, **?**, **?**]. But in real application usually

R is not clearly defined especially in the complex cases of Model Transformation and Synchronization. For example in TGG Approach [19, 43], Relation between source and target is defined implicitly by TGG rules in constructive and operational way [**?**], or in ATL [40], Viatra2 [6] R is defined in one direction as a function. Defining a Relation as a function is useful if the nature of the Relation be one-to-many or info-Asymmetric like the Table-View Relation in database domain [**?**]. Apparently the backward transformation of this case would not be a function and would require an update policy to make it a function [**?**]. There are two points regarding our latter discussion that we need to mention out here: (1) there are many cases of model synchronization that are not info-Asymmetric, so the relation definition can not be specified by a function in neither direction (both forward/backward) (2) even in the case of info-Asymmetry, we need to avoid the naive, inefficient and non-intelligence method of batch update of the target for every change that happens on the source. In both above cases (info-asymmetric and info-symmetric) we need to know two important facts: (1) which change in the source imposes changes on the target according to the defined Transformation Function or Relation. we define *Private parts* of the source, the entities that any change to them would not impose a change to the target as apposed to *Shared parts* which any changes to them requires a change to the target to satisfy the constraints and establish the relationship again. As we will show by example latter, distinction of *Private* and *Shared parts* is not trivial and straight forward result gained out of Relation or Function definitions. (2) if it is determined that the target should be changed (because of *Private part* change in source,) how to preserve the relationship(make the target aligned again with the source) by *minimum possible change* on the target. In the general case of the info-symmetry(many-to-many Relation) many possible alignments would be possible, however we are interested in one(or maybe a set) of them which are the most desirable(minimum possible change on the target). we will discuss this case more by an example later. For addressing this problem it seems that we might need to define some measurements for the updates and calculate the most desirable updates according to these measurement. Hence we would need to identify suitable measurements according to some factors like usability and practical application of synchronization cases. if we can address the previous problem in terms of defining a *Private* and Shared parts formally for the models based on the synchronization scenario, and also identify some criteria for the *desirable updates* on the target, then we can give a formal semantic for the concept of incremental synchronization. Therefore synchronization frameworks can be implemented and verified according to that semantics.

it seems that choosing these *desirable* changes can not be mechanized?! And they might be just user choices, but in the case that the alternatives are too much for the user to choose there should be some filters which makes them a well-acceptable set of option in some sense out of which user can select. In latter situation those filter parameters can be defined by Transformer Executer (user).

The extended case of the incremental synchronization formalization would be the *concurrent* cases of synchronization which the changes are allowed simultaneously at the source and target in one step. should explain this more

QVT-R [?] and TGG are two approaches that can be used for Relation Definition among two models in industry, so try to demonstrate the problem by some example based on these two approaches. (What are the other Relational Languages for Relation definition among two models.)

Steven Criticises on the QVT-R semantics!

Solution: some speculation defining a transformation as a function is ATL and Viatra2 is easier than QVT-R, so maybe better idea to have two functions which is practical than one function and let the tool keep the properties among them in according to the constraint that user desire to be maintained by the tool. I mean give the option to the user to choose that criteria getput, hipocratic, invertability, undoable, and .... in maintained for these two functions in pair (forward/backward). since it makes the transformation writting for users easy and practical.

the following part should be commented or moved to the methodology and approach section (one look is that looking a the transformation functions defined in ATL and Viatra, and see how they define that function. according to that function the source is sliced into some pieces which each piece is mapped to one target . Reverse looking at this function is a kind of defining half of the full relation in a way that $t \in T$ there exist many $s \in S$ where sRt . so out of forward transformation function (lets take it as total just by mapping non-mappable cases (not transferable cases) to null) we can get the half of the relation like $\{s_1Rt_1, s_2Rt_1, s_3Rt_1, s_4Rt_2, s_5Rt_3, ...\}$, we can look at that like $\{\{s_1, s_2, s_3\}Rt_1, \{s_4, s_5\}Rt_3\}$ too. Forward transformation function is not always surjective. By the reverse case (backward function) we would have the other half of R $\{t_1Rs_1, t_2Rs_1, t_3Rs_1, t_4Rs_2, t_5Rs_2...\}$, we can look at that like $\{\{t_1, t_2, t_3\}Rs_1, \{t_4, t_5\}Rs_2...\}$ or reverse of it $\{s_1R\{t_1, t_2, t_3\}, s_2R\{t_4, t_5\}...\}$. so the question is that if these two functions together define the desired Relation R?! I think this is closely related to the constraints that forward and backward transformation should preserve some constraints like hipocraticness(check and force), inevitability, GetPut and ....) to be a BX. They should be studied more!!!! Another thing that come to my mind is that it can be related to meta-models too! so meta-models for sure are involved in definition of this private and shared parts.

you need to mention AGG tool [51] in your proposal! , also SDM [27], also EMF [48]

5.1 Example Demonstrating the Problem

I should add example to demonstrate the problem with some examples. Here we are going to show two examples demonstrating the issues related to the *Private* and *Shared* Parts. Look at the Figure 4. *People* Table(Model Space A) representing the records(Models) of People while *Employee* Table(Model Space B) representing Employee records. The Relation R between *People* and *Employee* is that each person is related to the employee with same `Name` and the `age` :

(one tricky thing is that how to represent the example 1 Model spaces with TAG)

$\forall a \in A$ and $\forall b \in B, (a, b) \in R$ if $a.Name = b.Name$ and $a.Age = b.Age$

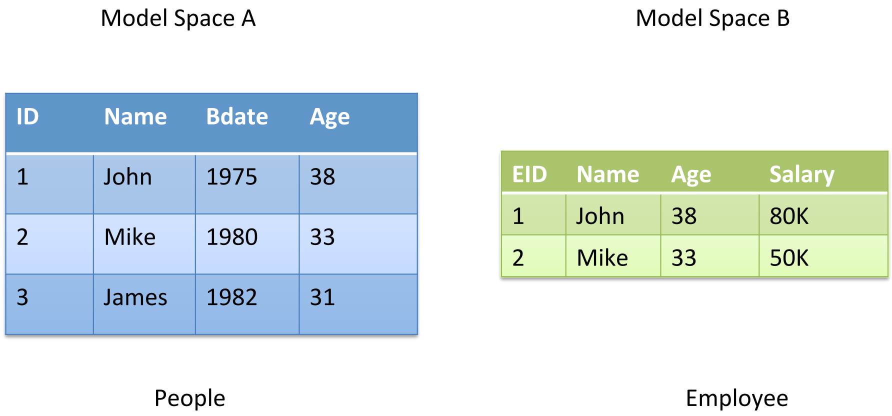
According to above relation `John` and `Mike` record are related to each other. In first look it seems that the `Name`and `Age`are the *Shared* part of *People* and *Employee* in the sense that their changes in each table will cause a change in another table. This result is also can be seen by looking at the Relation Definition above. We can say since the names of the elements(`Name` and `Age`) are literally exists in the Relation Definition, so those elements are part of the *Shared* entities. But when we change the `Bdate` content in the *People*, it would also cause a change in the Employee `Age` in Employee table. So Bdate Changes should also propogate to the Employee too. That is because of internal functional dependancy [**?**]

between the Age and Bdate in People table.

Models in general sense are not as simple as records. as we showed in Section **??** they can be formally represented by Typed Attributed Graph. The Notion of the dependancy between the data items in the record can be generalized to the cross cutting constraints between the model elements inside the Graph. So in that sens it is not trivial to distinguish the *Private* and *Shared parts* out of looking at the Relation Definition between two Models. maybe refer to another example with graphs in the following sections. In practice, it is also not the case that we always have a clear Relation Defintion literally between two Model spaces in terms of their elements. That is because specifying Relational dependancy between large Model is so complicated as apposed to specifying the Transformers by functions, so latter is preferred – That is one of the reason why languages like ATL and Viatra2 gained more interest over the relational counterparts like QVT-R and TGG in industry. To this end, further investigation of specification of the *Private* and *Shared Parts* even specifically in each approach of Model Transformation and Synchronization is necessary. It is worth recalling that the this concept is demonstrating its importance while the changes on one model are so small in comparison to the whole model and we want to make the least effort in maintaining the consistency after these changes. [**?**]

Another aspect of the problem is identifying the least possible changes in the target for maintaining the consistency relation. Suppose in above example we change the Age Group in Employee table from 1 to 2. The most reasonable option for aleviating the Age and Bdate value in People Table is that reounding them to the nearest lowest and largest value to maintain the Relationship. We call this the *Minimum/Least Change* concept. This change is not always in terms of the values as it is in the database tables, and it needs to be defined in terms of Graph structures as of Model representatives. So this needs to be more investigated and studied in terms of Models and Model Incremental Synchronization. It is clear that probably we'd better not to limit the options of changes to exactly one cases (i.e. change Age to 60), but provide the user with a range of most probable options (i.e. 60- 70) rather than (60 - ...).

As another example we are considering the transformation between the rail projects and the petrinet discussed in [41] using Tripple Graph Grammars [**?**]


Model Space A
Model Space B
People
Employee

ID	Name	Bdate	Age
1	John	1975	38
2	Mike	1980	33
3	James	1982	31

EID	Name	Age	Salary
1	John	38	80K
2	Mike	33	50K

Figure 4: People and Employee model spaces

you can take it in a unidirectional domain and study that in that area, I think there should be some text in the literature which covers this topic. for example you wrote a transfo. on ATL from A to B, then A changes, so how to propagate its change to B, not touching unrelated data. That arise the fact that how to extract unrelated data from just unidirectional transformation. In another world How we can get the relation R from just unidirectional case, in the sense that we almost consider Transfo as a function, so it means that the R is a function. This is not realistic!! in practice , we define Transfo, as a function, but taking the shared part of the target as domain. in that sense yes that is function. but that doesn't mean that whole design space of the Target make that transfo a function. let say an example: we define transfo from class diagram to code : this is function, Does the relation from class diagram to code is function? No, Why? because for one class diagram there are many implementations which differ in body of the class methods. so the relation from model to code in not a function. In practice we usually specify this relation as function. So we can just examine the matter of model synchronization in unidirectionally cases as a special case of bidirectional case. this gives the opportunity to study this area independent of the BX which its result can be later extended to BX.

6 Methodology and Approach

- just looking at different example of model transformation and investigating them according to the abstract classification of them done in the paper. read next paragraph
- need to investigate more examples and their formal representation in AGG.
- need to work with AGG tool as an Algebraic graph transformation Tool.
- it seems that we need to study the relation between two models deeply.
- It seems that the meta-models can be involved in *Private* and *Shared* part definition. But I don't know how?
- I think since this is gonna be practical work my approach would be examining the functional way of defining the relationship which is currently practiced in industry and is more popular.

One prospective approach to the problem would be trying to abstractly capturing the real worl transformation by ATL and Viatra2 and study how a transformation function on them define the Relation R. then trying to identify the private parts and shared parts in those instances. so trying for different concrete examples and trying to capture the abstract idea of private and shared part and formalising it. this can be done for even Relational languages.

Since the formalization of the Models and Typed Models are very tied with the CAT theory, it seems that maybe some categorical patterns and concepts would help there abstracting out the concept. therefore increasing the knowlage of category theory and studing the patterns related to the Modeling and MetaModeling seems to be useful.

For the rest of 3 Month examing two Languages ATL and Viatra 2 and reading about the CAT theory pattern. seeing examples in them, matching them with the 3D space concept in the paper.

for the 2 month formalizing the idea.

the rest of 5 month publishing the thesis.

5 month gap for paper submission and tackling new obstacles identified during research which is not clear at the moment.

so within 15 month from now, thesis is gonna be submitted.

7 Conclusion

You are gonna put the conclusion here

- What is this paper is about?!
- What is the problem identified
- How you gonna approach it.
- if you solve this what will happen (what is the contribution)

References

[1] Meta object facility (mof) 2.0 query/view/ transformation specification, omg document, January 2011.

[2] Anthony Anjorin, Marius Lauder, Sven Patzina, and Andy Schürr. emoflon: Leveraging emf and professional case tools. *Proc. of MEMWe*, 2011.

[3] Anthony Anjorin, Sebastian Oster, Ivan Zorcic, and Andy Schürr. Optimizing model-based software product line testing with graph transformations. *ECEASST*, 47, 2012.

[4] Michał Antkiewicz and Krzysztof Czarnecki. Design space of heterogeneous synchronization. In Ralf Lämmel, Joost Visser, and João Saraiva, editors, *Generative and Transformational Techniques in Software Engineering II*, volume 5235 of *Lecture Notes in Computer Science*, pages 3–46. Springer Berlin Heidelberg, 2008.

[5] Steve Awodey. *Category Theory.* Oxford University Press, 2009.

[6] András Balogh and Dániel Varró. Advanced model transformation language constructs in the viatra2 framework. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1280–1287, New York, NY, USA, 2006. ACM.

[7] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, December 1981.

[8] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Centre de Recherches Mathématiques, 1999.

[9] Michael R Berthold, Ingrid Fischer, and Manuel Koch. Attributed graph transformation with partial attribution. In *Proceedings Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems*, pages 171–178, 2000.

[10] Jean Bézivin. In Search of a Basic Principle for Model Driven Engineering. *UPGRADE – The European Journal for the Informatics Professional*, 5(2):21–24, 2004.

[11] Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang: resourceful lenses for string data. *SIGPLAN Not.*, 43(1):407–419, January 2008.

[12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice.* Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.

[13] L.C. Briand, Y. Labiche, and L. O"Sullivan. Impact analysis and change management of uml models. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 256–265, Sept.

[14] Peter Buneman, Mary Fernandez, and Dan Suciu. Unql: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal—The International Journal on Very Large Data Bases*, 9(1):76–110, 2000.

[15] Zinovy Diskin. Algebraic models for bidirectional model synchronization. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 21–36, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. From state- to delta-based bidirectional model transformations: the asymmetric case. *Journal of Object Technology*, 10:6: 1–25, 2011.

[17] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann, and Fernando Orejas. From state- to delta-based bidirectional model transformations: the symmetric case. In *Proceedings of the 14th international conference on Model driven engineering languages and systems*, MODELS'11, pages 304–318, Berlin, Heidelberg, 2011. Springer-Verlag.

[18] Hartmut Ehrig and Karsten Ehrig. Overview of formal concepts for model transformations based on typed attributed graph transformation. *Electron. Notes Theor. Comput. Sci.*, 152:3–22, March 2006.

[19] Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. Information preserving bidirectional model transformations. In *Proceedings of the 10th international conference on Fundamental approaches to software engineering*, FASE'07, pages 72–86, Berlin, Heidelberg, 2007. Springer-Verlag.

[20] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation.* Springer Publishing Company, Incorporated, 1st edition, 2010.

[21] Hartmut Ehrig, Claudia Ermel, and Frank Hermann. On the relationship of model transformations based on triple and plain graph grammars. In *Proceedings of the third international workshop on Graph and model transformations*, GRaMoT '08, pages 9–16, New York, NY, USA, 2008. ACM.

[22] Hartmut Ehrig, Claudia Ermel, Frank Hermann, and Ulrike Prange. On-the-fly construction, correctness and completeness of model transformations based on triple graph grammars. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, MODELS '09, pages 241–255, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graph transformation. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 161–177. Springer Berlin Heidelberg, 2004.

[24] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graph transformation. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 161–177. Springer Berlin Heidelberg, 2004.

[25] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Fundamental theory for typed attributed graph transformation. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 161–177. Springer Berlin Heidelberg, 2004.

[26] Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel, editors. *Graph Transformations and Model-Driven Engineering - Essays Ded-*

icated to Manfred Nagl on the Occasion of his 65th Birthday, volume 5765 of *Lecture Notes in Computer Science*. Springer, 2010.

[27] Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. Story diagrams: A new graph rewrite language based on the unified modeling language and java. In *Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, TAGT'98, pages 296–309, London, UK, UK, 2000. Springer-Verlag.

[28] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. *SIGPLAN Not.*, 40(1):233–246, January 2005.

[29] Holger Giese, Stephan Hildebrandt, and Leen Lambers. Bridging the gap between formal semantics and implementation of triple graph grammars. *Software and Systems Modeling*, pages 1–27, 2012.

[30] Holger Giese and Robert Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 8:21–43, 2009.

[31] Ulrike Golas, Leen Lambers, Hartmut Ehrig, and Holger Giese. Toward bridging the gap between formal foundations and current practice for triple graph grammars - flexible relations between source and target elements. In *ICGT*, pages 141–155, 2012.

[32] Ulrike Golas, Leen Lambers, Hartmut Ehrig, and Holger Giese. Toward bridging the gap between formal foundations and current practice for triple graph grammars: flexible relations between source and target elements. In *Proceedings of the 6th international conference on Graph Transformations*, ICGT'12, pages 141–155, Berlin, Heidelberg, 2012. Springer-Verlag.

[33] Joel Greenyer, Ekkart Kindler, Jan Rieke, and Oleg Travkin. Tggs for transforming uml to csp: Contribution to the agtive 2007 graph transformation tools contest. Technical report, Department of Computer Science, University of Paderborn Paderborn, Germany, 2008.

[34] Frank Hermann, Hartmut Ehrig, Claudia Ermel, and Fernando Orejas. Concurrent model synchronization with conflict resolution based on triple graph grammars. In *Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering*, FASE'12, pages 178–193, Berlin, Heidelberg, 2012. Springer-Verlag.

[35] Frank Hermann, Hartmut Ehrig, Fernando Orejas, and Ulrike Golas. Formal analysis of functional behaviour for model transformations based on triple graph grammars. In *Proceedings of the 5th international conference on Graph transformations*, ICGT'10, pages 155–170, Berlin, Heidelberg, 2010. Springer-Verlag.

[36] Soichiro Hidaka, Zhenjiang Hu, Kazuhiro Inaba, Hiroyuki Kato, and Keisuke Nakano. Groundtram: An integrated framework for developing well-behaved bidirectional model transformations. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 480–483. IEEE, 2011.

[37] Martin Hofmann, Benjamin Pierce, and Daniel Wagner. Symmetric lenses. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '11, pages 371–384, New York, NY, USA, 2011. ACM.

[38] ikv++ technologies. medini QVT homepage. http://projects.ikv.de/qvt.

[39] Michael Johnson and Robert Rosebrugh. Constant complements, reversibility and universal view updates. In José Meseguer and Grigore Roşu, editors, *Algebraic Methodology and Software Technology*, volume 5140 of *Lecture Notes in Computer Science*, pages 238–252. Springer Berlin Heidelberg, 2008.

[40] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008.

[41] Ekkart Kindler and Robert Wagner. Triple graph grammars: Concepts, extensions, implementations, and application scenarios. Technical Report tr-ri-07-284, Software Engineering Group, Department of Computer Science, University of Paderborn, June 2007.

[42] Lilija Klassen and Robert Wagner. Emorf-a tool for model transformations. *Electronic Communications of the EASST*, 54, 2012.

[43] Alexander Königs and Andy Schürr. Tool integration with triple graph grammars - a survey. *Electron. Notes Theor. Comput. Sci.*, 148(1):113–150, February 2006.

[44] Jens Lechtenbörger. The impact of the constant complement approach towards view updating. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '03, pages 49–55, New York, NY, USA, 2003. ACM.

[45] Grzegorz Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation: volume I. foundations.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

[46] Andy Schürr. Specification of graph translators with triple graph grammars. In ErnstW. Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163. Springer Berlin Heidelberg, 1995.

[47] Andy Schürr and Felix Klar. 15 years of triple graph grammars. In *Proceedings of the 4th international conference on Graph Transformations*, ICGT '08, pages 411–425, Berlin, Heidelberg, 2008. Springer-Verlag.

[48] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework.* Addison-Wesley Professional, second edition, December 2008.

[49] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2nd edition, 2009.

[50] Perdita Stevens. Bidirectional model transformations in qvt: semantic issues and open questions. *Software and Systems Modeling*, 9:7–20, 2010. 10.1007/s10270-008-0109-9.

[51] Gabriele Taentzer. Agg: A tool environment for algebraic graph transformation. In Manfred Nagl, Andreas Schürr, and Manfred Münch, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 1779 of *Lecture Notes in Computer Science*, pages 481–488. Springer Berlin Heidelberg, 2000.

[52] Eclipse modeling framework. Eclipse project Website.