Incremental Model Synchronization in Model Driven Engineering

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

April 16, 2013

Abstract

Model Driven Engineering (MDE) is proven to be a promising approach in software engineering. software model consistency management and maintenance stands at the core of the MDE procedure while it still needs more theoretical and technical supports for realization of its required functionalities like model transformation, synchronization and change propagation. In this thesis proposal we explore the type of different synchronization scenarios in Model Driven Software Engineering, and will introduce three areas which needs more contribution and research : private/shared parts, least/minimal change application, and concurrent synchronization and conflict resolution.

1 Introduction

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 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 level– from Specification to implementation by code. We call this process *model refinement* and can look at it like vertical movement from more abstract levels to less abstract ones. This *Vertical* –up to down– movement usually happens in reverse direction too where it is called *reverse engineering*– to fit in and accomplish the iterative method of software development process. This up and down process, then 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. However, ike the previous case this movement can be in forward and backward directions (Figure 1.)

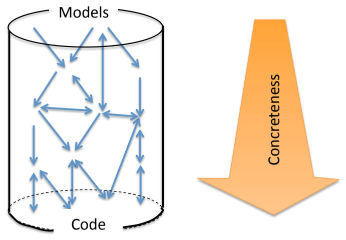


Figure 1: Vertical and Horizontal movements in software eng.

The horizontal movement is necessary because each model might demonstrate and focus 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 [1], each model share some common parts with other models and some have their own specific features. E.g. *Class Diagram* and *State Chart* diagram can share some of the class method signatures, while attribute names for the classes, and the order of the events happening, are correspondingly considered as specific and private parts of each.

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

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

graph. Hence change propagation can happen *Vertically* or *Horizontally.* Taking software artifacts a models, industrial projects would include hundreds or even thousands of models which are inter-related to each other in some ways, so changing one of them would require change propagation throughout the whole system. Handling this situation by hand without a tool support, even in small scale scenarios is tedious, complicated, error-prone and very time-consuming and for the large cases, is nearly impossible. That is why nowadays, most changes are applied directly on codes rather than models, which made software engineering deprived of many proved benefit of Model Driven Software Development. That is ending up with inconsistent artifacts of the software development which is the typical story of the large scale software projects, then most of them are usually looked as outdated historical records and decorative pieces rather than the live, influential and dependable software artifacts.

Change propagation among the models are also called *model synchronization*, in the sense that it makes the involved models consistent and synchronized after changes on one model is propagated to the others. In following section we will discuss by example different aspects of the model synchronization and will identify 3D orthogonal dimensions of the synchronization scenarios. We also would examine the bidirectional model synchronization and synchronizer transformers properties and will introduce the formal representation of the models. Following those, we will identify three connected problems in domain of model synchronization in section 3 which is the subject of this thesis proposal. We will discuss related works in section 4 and will propose the approach to the problem solution and conclude in section 5.

2 Incremental Synchronization

Suppose that we have two model space(sets of Models), A and B, and suppose relation R is defined among these two model spaces. we call every instance of A or B, a Model. i.e. $a \in A$ is called a Model; and suppose that model a and b are related by relation R, i.e $(a, b) \in R$. When model a changes to a' and $(a', b) not \in R$, we require to change b to a b' such that a' and b' are related again. Figure .... shows this scenario.

thawe require to change model b every member of A is an instance of

3 Context of the Synchronization Scenarios

In this section we describe the three dimensions of the synchronization scenarios: *Organizational Symmetry*, *Information Symmetry* and *Incrementality*, which are orthogonal(independent) to each other, in the sense that each is describing an independent aspect of the synchronization scenario. For describing these concepts we will start with an example from a flight reservation system which is modeling a simple flight arrangement and implementation in a typical airline company. later we will introduce the 3D dimension and discuss them in more details. Formal definition of each dimension is presented in our recent work submitted to MODELS13 conference [18]. We will also discuss the concurrency dimension and its relation to later synchronization dimensions. Later we will discuss the concept of Bidirectional Transformation and will introduce properties of the synchronizer transformers. At the end of this section we will introduce typed attributed graphs which is introduced as a formal foundation of the model and meta-modeling concepts.

3.1 Flight Databse Example

Suppose that there are two groups of employee in an airline company. One group is responsible for identifying the business aspect of the flights and identify the necessary flights which should be set between different destinations according to the airline business and marketing policies. Another group in more technical and trying to implement defined flights by group one (Figure 2). For example flight #11 in Model A is implemented by two flights #1 and #2 in Model B. For specifying the relation (implementation) of Model A with Model B, we define an intermediate view over Model B –get(B), which is actually an implementation of the Model A. Intermediate View is pair of (self.fst, self,scd) where :

(Q) $\mathsf{self.fst.to} = \mathsf{self.snd.from}$ and $\mathsf{self.fst.time} \leq \mathsf{self.snd.time}$.

We also define $\mathsf{self.time{=}self.fst.time}$. Computing $\mathsf{self.cost}$ is done by some procedure using airport and airplane data. So we can have a function `get(B)` that computes derived table OsFlight from base table SdFlight. Two 'by'-links relate Ld-flights to their one-stop implementations. Together they form an inter-table (sub)mapping $\mathsf{by}\colon \mathsf{LdFlight} \to \mathsf{OsFlight}$, which constitutes a *model-correspondence mapping* $r : A \to get(B)$ (mapping r could con-

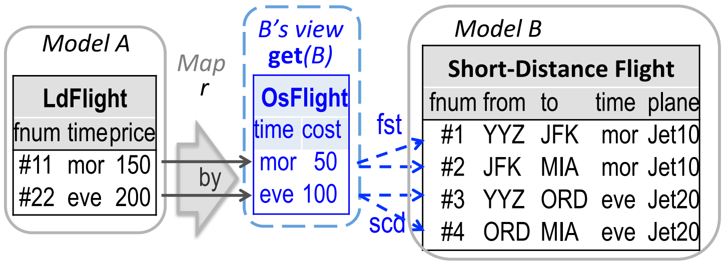


Figure 2: Model B and mapping r implement model A

tain more submappings like by, if model A would have more tables/classes). Note that implementation is a pair (B, r), but we will often say 'implementation' B, leaving the correspondence mapping implicit. We consider implementation (B, r) to be *correct*, if for each ld-flight self we have:

(C) $\mathsf{self.time} = \mathsf{self.by.time}$ and $\mathsf{self.price} \geq \mathsf{self.by.cost} + 100$.

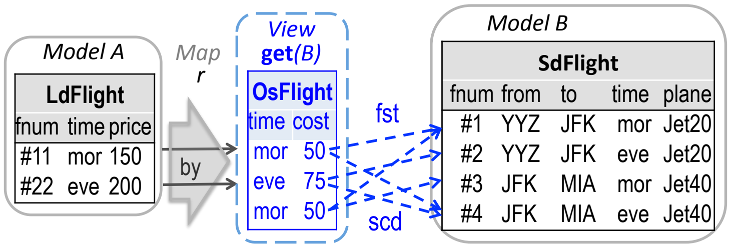


Figure 3: Another Implementation for model A

According to above definition, it is possible to have many implementation (B,r) for the same A. For example Figure 3 shows another implementation for the same LdFlight. This is the typical situation for the implementation of the models in software development where many alternative implementation is possible for the same model. It somehow indicates that there are some *Private* parts in the model B which is hidden from Model A's perspective, e.g. the plane and intermediate stops. From another side, the fnum information of the Model A is not reflected in `get(B)` view. This is the typical case of *Information Symmetric* scenario in model synchronization where each models have their own non-empty *private* parts.

If we make Figure 3 more elaborate, we can indicate the mapping "by", from a projection of A (`get*(A)`) to `get(B)` like Figure 4. The project `get*(A)` would be a view on Model

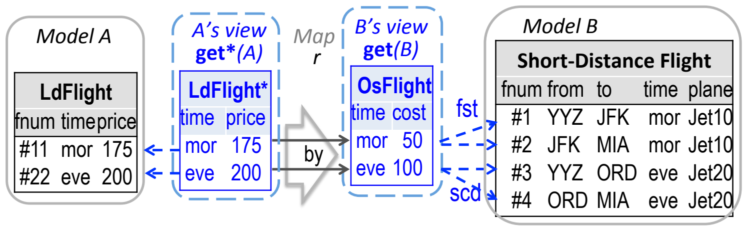


Figure 4: Model B and mapping r implement model get*(A)

A where all its information is completely dependent on Model A. In another word there is not private information in `get*(A)` which can not be obtained from Model A. This is the typical situation of the *Informationally Asymmetric* cases. Taking example above as the basis we continue to discuss different possible scenarios on that, exemplifying some of synchronization types which we will discuss classifying in 3D space later on.

Implementation is derived from the marketing decisions , and the user doesn't care how the flights are implemented by the technical team. Model A has dominacy over Model B in the sense that either there is no independent work on Model B and it is just created using Model A, or if for users working on Model B, it is not important if their changes are discarded. This doesn't seem a good idea in the context of our example, but it might happen in the case of code generation. we will refer to this case as *Organizationally Asymmetric* case and we say that Model A *Organizationally* dominates Model B.

Implementation needs to be preserved, and discarding all changes on implementation when a change is happening on model A is disappointing. This is the most desirable cases in the common cases of the synchronization scenarios. A solution is to implement changes on side A incrementally as shown in Figure 5: the change, or *delta*, on side A, Δ_A, is propagated to a delta on side B, Δ_B, which together with the original implementation B provides an updated implementation. In the figure, solid lines and shaded tables refer to given data, and dashed lines and blank tables denote data produced by the operation of delta propagation. In more detail, Δ_A makes explicit that flight #11 is preserved while flight #22 is added. Correspondingly, delta Δ_B keeps flights #1 and #2, and adds two Sd-flights implementing the required one-stop flight. The range of possible implementations is captured by placing *labeled nulls* $?_i$ into the table: nulls with the same labels must be substituted with the same values and nulls with different labels are independent, but may be also substituted with the same values. In this way, uncertain model B' captures the implementations in Figure 2, and that one in 3, and many others possibilities. Correspondingly, the derived OsFlight table is also uncertain: the cost value is given by applying some known procedure say, F, to unknown argument values $?_i, i = 1, 2, 3$. We term this kind of change propagation *Incremental Synchronization.*

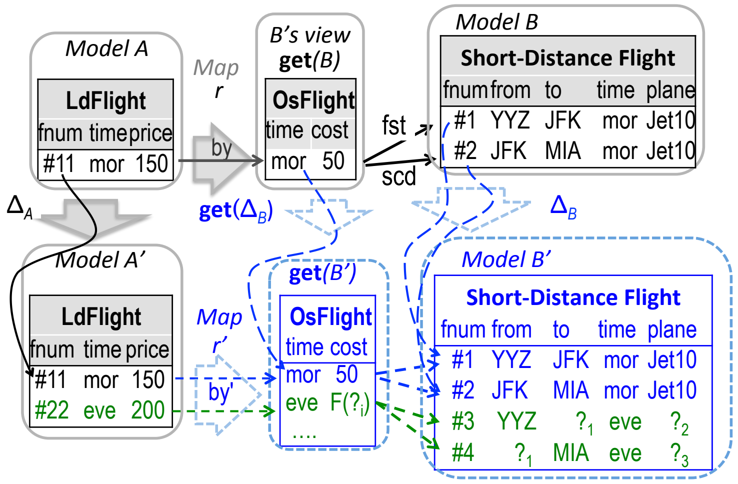


Figure 5: Incremental implementation

suppose that our policy impose that Model B can change to Model B' as far as A and B' are still consistent. Otherwise we will role back the changes to B. If so model B changes to consistent B', in this case its changes would be respected while propagating changes from A to B. Though Model B would have more flexibility than the previous case where all changes are disregarded in update propagation from A to B. The latter policy also implies that we don't have a change propagation from B to A, so this situation is still *Organizational Asymmetry* but with *Incremental* updates.

Round-tripping or Organizational Symmetry. Consider a different business context, in which the technical team gains a greater authority and administrative weight than before. Now, if the technical team, while working with Model B originated from model A, would find a valuable modification B', e.g., new profitable os-flights, but B' would be inconsistent with A, then the B-team may require to modify model A to a state A' consistent with B'. In other words, unlike two previous cases, updates now can be propagated in both directions. We will refer to the case as *Organizational Symmetry.*

Organizational Semi-Symmetry or Partial Symmetry. Suppose the previous case of Round-tripping where update propagation allowed in both direction, and we deleted some information from Model A, for example, ld-flight #11 in Figure 5 is deleted. This deletion can be caused by a real world deletion of sd-flight #11.fst: YYZ–JFK, or #11.scd: JFK–MIA, or both. We cannot arbitrarily choose one of them, because objects of class SdFlight must represent actual flights existing in the schedule. Hence, propagation of deletions in the Model A must be prohibited. But as Figure 5 shows it is possible to propagate new addition from Model A to Model B. In another word some updates on Model A are allowed to be propagated to Model B but some are not. Note that this is more symmetric than the *Organizational Asymmetric* case and less symmetric than the *Organizational Symmetric* case. We call this case *Organizational Semi-Symmetric* case.

3.2 Type Space of Synchronization Scenarios

Following from the previous section we identify 3 dimension of a synchronization types: (1)*Organizational Symmetry*, (2)*Informational Symmetry*, and (3)*Incrementality*, ref. Figure 6. We are going to briefly discuss each dimension; more details and their formalization can be found in submitted paper to MODELS13 conference [18].

Organizational Symmetry. *Organizational Symmetry* represents how far we consider one side active in terms of accepting and propagating its changes during a synchronization scenario. This dimension captures synchronizer domain of application perspective in necessity of change propagation from one side to another side. For example synchronizer doesn't wish to allow any change propagation from A to B even if it is technically possible or not, e.g. in generation of object code from source code. we call this situation *Organizationally Asymmetric* or org-symmetry=0. In another word change propagation in this case is unidirectional and then one side would be always *passive* and the other side would be always

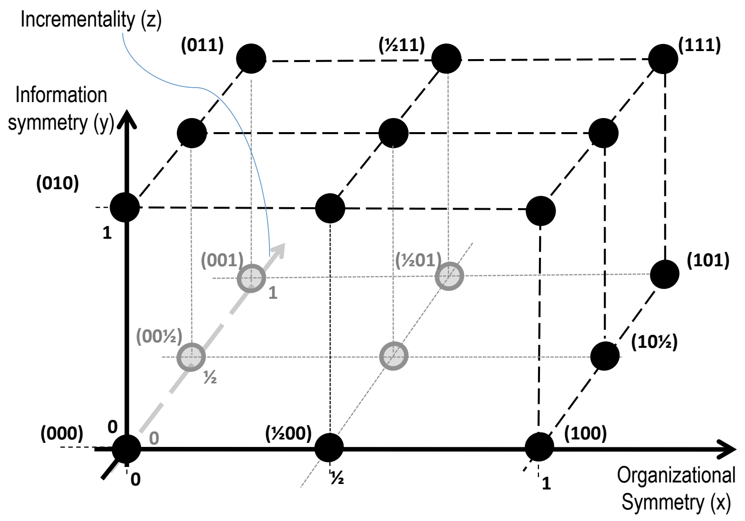
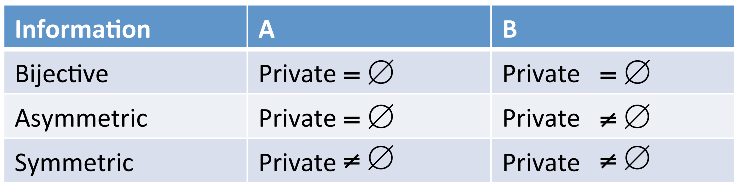


Figure 6: 3D dimension of Synchronization Scenarios

active. As apposed to latter case, *Organizationally Symmetric* or org-symmetry=1 means that change propagation is bidirectional and it is not prohibited from neither side, then both sides would be *active*. There is also some cases where some (and not all) changes from one side are allowed to be propagated. E.g., we prohibit tuple deleting propagation from the view to the source but letting tuple addition in view to be propagated to the source like the case we already discussed in previous section example. We call this type *Organizationally Semi-Symmetry* and show it by org-symmetry=$\frac{1}{2}$. According to the design decision of the synchronization framework implementation, change prohibition or making either side *active* or *passive*, can be implemented either by discarding the prohibited changes or by simply not allowing the user to do changes on models at first place.

Information Symmetry. *Information Symmetry* dimension captures if two models are informationally symmetric or asymmetric. In *Information Asymmetric* or inf-symmetry=0 case, one side existence informationally suffices to create the other side. In another word having one model, we can get the other one uniquely. This is the case in our example in previous section where get*(A) can be completely derived from A or gernerally in all typical cases of table-view scenarios [51, 48], where view contents is completely derivable from the corresponding table contents. As opposed to that, in inf-symmetry=1 or *Information Symmetric* case neither A, nor B can be informationally derivable from each other. E.g.

Class diagram and Sequence diagram, neither can be derived from each other uniquely. Another perspective of looking at this dimension is based on *private* and *shared* parts of each side. In *Asymmetric* one side has its own non empty *private* and *shared* parts where the other side only consist of a *shared* part with its *private* part being empty. In *Symmetric* case, neither of *private* and *shared* parts of involved models are empty (ref. Figure 7). The latter case is the more general case of the synchronization scenarios in practice. It is possible that both *private* parts of each side be empty, this is called the *Informationally Bijective* case which can be considered as especial case of information symmetric case. In this case two models are informationally equal but differently represented. Later on in section 3 we will again discuss this concept of *private* and *shared* parts in more details as one of the intended contribution of PhD thesis.



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

Figure 7: private and shared parts in Information Symmetry dimension

Incrementality. Synchronization scenarios can be classified into three cases in terms of maturity of its *Incrementality* method. These three categories are called *Batch-update*, *State-based* synchronization and *Delta-based* synchronization. In *batch update* or inc=0, one side is completely calculated from scratch whenever changes happen in another side. 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 one side, considerable effort is imposed on the synchronization mechanism to calculate the other side. That is why it is usually used in small scale scenarios like type hierarchy update of Java codes [4]. if you take Models A and B again as both participants of the synchronization scenario, propagation function in this approach accepts A' (Changed A) as the mere input, and generates consistent B' out of that from scratch. In comparison, inc=$\frac{1}{2}$ or *state-based* synchronizer takes A' and B as inputs, trying to calculate B' where it is aligned with A'. This is far better than the previous case and usually is the intended scenarios when*Incremental synchronization* is mentioned in the literature. It is shown in [19] that taking A' and B 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 just states, for the propagation function. we refer to the latter case as inc=1 or *Delta-Based* synchronization.

The way we choose B' out of different alternatives in both *State-based* and *Delta-based* cases, is the subject which is discussed in more details as *minimal/least changes* in section 3 and is intended to be the other contribution in PhD thesis. That is also related to the *private* and *public* part identification too. Since for example it is desirable that the synchronizer doesn't change the *private* part of one side at all if we require the change become *minimal/least* on that side.

Concurrency and its relation to other dimensions. by *concurrent* synchronization we mean that two models are allowed to evolve concurrently. when we state that a synchronization is not *Concurrent*, it means mean that involved Models can not be changed simultaneously. In another word, supposing again Models A and B, if Model A is evolving to A' we first need to 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 which is termed as as *master*, while ignoring the changes of the other, then is termed as *slave* and trying to make them aligned again. As apposed to that, in *concurrent* scenario, simultaneous changes to the models are allowed and both model changes are taken into consideration in the alignment process and non would be *slave*. The latter case is more desirable and the common scenarios in practice while it is more complicated than the former one, since the synchronizer needs to take into account both sides changes while trying to establish the alignment between them again. Though again in either case it is desirable to establish alignment by *minimal/least* changes application, the concurrent case add more complexity to *minimal/least* changes application by adding the necessity to consider both side simultaneously in the procedure. This is the third subject which will also be discussed in section 3 and is intended to be in the scope of the PhD thesis contribution to incremental model synchronization.

Concurrency is not orthogonal to aforementioned three dimensions. For example in org-symmetry=0, one side being *passive* would make the other side being always treated as *master* in non-concurrent cases, so making concurrent and non-concurrent cases identical. Another observation is that it seems being *passive* in terms of *Organizational Symmetry* differs from being *slave* in concurrent updates, since being *master* or *slave*, not necessarily fixed in non-concurrent case while being *passive* is a fixed parameter in *Organizational Symmetry* dimension .

3.3 Pair of Transformers and Bidirectional Transformation Languages

In model synchronization, changes propagation is usually applied by execution of a pair of transformers which we call *forward* and *backward* transformers or propagation functions. These transformers as we will discuss in next section are necessary to be treated as a pair of functions which satisfy some properties. When we transfer one model(say Model A) to another model(say Model B) by *forward* transformation function, we usually desire to have a *backward* counterpart which transfers B to A.

Suppose that A and B are consistent if they satisfy the constraints defined by a relation R . Often in practice R is not explicitly defined and just the transformers is defined using some transformation languages. For example languages like *ATL* [50], *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 separately by hand. It is often desirable that these *forward* and *backward* transformers be consistent [2] [20, 31, 64], but maintaining the consistency in pair of transformers which is separatly coded by 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 (forward) 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)* [59], *Lenses* Framework [31] and *GroundTram* [45] frameworks are providing *BX* facility. From

[2]holding some properties like *Hippocraticness*, *Undoability* and etc., which is discussed later

the aforementioned frameworks, only TGG works on TAG[3]. Table below summarizes the datastructure each of these languages support for model representation:

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. 2.4) (3)Specifying transformers as one specification in BX languages like QVT-R is more complicated and trickier than writing the same transformer in one-directional transformation languages. Whatever approach we take for specifying the synchronizer transformers, they are required to satisfy some properties which we will discuss in the following section.

3.4 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.

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

[3]refere to Definition 13

between two set of models or meta-models, A and B. *Correctness* property as it is expected, simply ensures that a pair of transformers are functioning correctly.

Definition 1. *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 [64] and [19] for *Information Symmetric* and *Delta-based* case. In [11, 46] this property is defined for *Information Asymmetric* and *State-based* Cases.

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 2. *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 Definition 1. This property is one of the requirements of the QVT-R specification in OMG standard [1]

Undoability. Undoability is introduced by Steven in [64]. 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 3. *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-basd* 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 this property on transformers sometimes might be more restrictive.

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 and functioning in reverse directions. We define invertability for the *State-based* cases. the definition can be easily extended for the other synchronization scenarios. Following definition is adopted from [17].

Definition 4. *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* $\{(a, b), (a', b')\} \subseteq R$

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 5. *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* $(a, b) \in R$

3.5 Models as Typed Attributed Graphs

As we mentioned in previous section, different languages in synchronization and transformation environments, support various data structure for model representation. Out of those, Typed Attributed Graph(TAG) is known to subsume other data structure in terms of being more expressive in models formal representation. for example the meta-model concept can not be well expressed in labeled Graphs while we would see it has a clear definition in terms of TAG.

In general a model can be represented as a graph consisting of a nodes and edges. E.g. a Class diagram in UML is a graph with classes as nodes and associations between them asedges. 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 graphs are like ordinary graphs 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 edges to have outgoing edges like nodes. This extended version of graphs is called *E-Graph* and introduced in the following.

Definition 6. *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 7. ***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 8. *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 [23] for more details.

Definition 9. *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 10. *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:*

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

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 11. *Given a signature $SIG = (S, OP)$, the **final** SIG**-algebra** $\boldsymbol{Z}$ 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 12. *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 13. *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 14. *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 $t^2 \circ f = t^1$:*

$$\begin{tikzcd}
AG^1 \arrow[d, "f"] \arrow[dr, "t^1"] & \\
AG^2 \arrow[r, "t^2"] & ATG
\end{tikzcd}$$

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.

4 Problem Definition

We are going to discuss three issues regarding the synchronization scenarios: *Private* and *Shared* part identification, *Minimal/Least* change application and *Concurrent* synchronization. These three issues are somehow related to each other as we will discuss it. We are going to explain each issue in more details in the following.

Private and Share Parts. When some changes happens in one of the two related models, there are three ways to react: (1) changes are discarded (2) changes doesn't need to be propagated (3) changes impose changes to model counterpart and should be propagated.

choosing each of above options depends on the value of the synchronization type on *Organizational* axis: if it is *Asymmetric*, *Partial Symmetric* or *Symmetric.* Suppose that we define the case as organizationally symmetric and incremental like in Figure 5. The main question is how we can distinguish which changes on Model B will impose necessary changes on Model A and Which don't? The general definition of the *private* and *shared* part is based on the change propagation necessity. By *private parts* of the Model B, we mean the entities that any change to them would not impose changes to Model A to restore consistency, as apposed to *shared parts* which any changes to them requires some changes on Model A to satisfy the constraints and establish the consistency relation again (ref. Figure 8), So our later question would be "what are the *private* and *shared* parts of the Model B?"

In practical implementation of the synchronization types (Figure 6), the later definition of the private/shared part is not so useful and we need a kind of mechanism to distinguish the private/shared part before execution of the transformation function or alignment procedure to decide weather the type of the changes are private or shared, and based on that trigger the alignment procedure or not. This is important to know, since otherwise we will require that with any changes to the model we transform the whole model – which is very costly, not efficient and time consuming in real application.

In the case of the example in section 2.1, it seems that distinguishing the *private/shared* part can be based on the relation definition. E.g. we can say that `from, to` and `time` are shared parts of the Model B and the `plane` and `fnum` are the private parts. This seems to be known out of the relation definition syntax where if we see the names of entities in the relation definition, it would be considered as the *shared part* and otherwise it would be *private*. Even in the case of the flight price in model B, we can include it in the shared part.

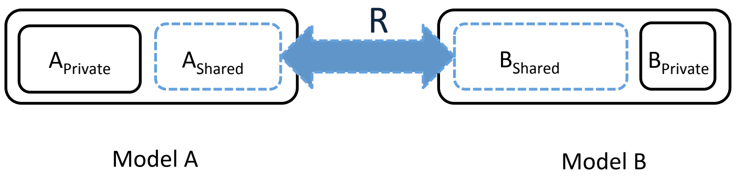


Figure 8: Each Model is consisting of Private/Shared parts

That is though seems somehow working in above example, there are two points it is necessary to mention out regarding that method: (1) Models in general sense are Typed Attributed Graphs(ref. section 2.5) and the way the relation is defined among them in practice are not always like declarative mathematical relations. i.e. in some more popular transformation languages like ATL [50] and Viatra2 [6] (non-BX), the relation is defined as functions and in some approaches like Triple Graph Grammars(TGG) [22, 54], the relation definition is operational rather than being declarative. Furthermore the notion of the functional dependancy, which is necessary in distinguishing the shared parts in simple models like database tables– those entities functionally dependent to shared parts are also belong to shared parts (transitive closure) – need to be generalized to the cross cutting constraints between the model elements inside the Graph of Model, which needs more investigation and study. It is also important to clarify how these private/shared parts granularity is defined based on the TAG entities: nodes, edges, or attributes or combination of them (ref. TAG definition in 2.5)(2) The more we can precisely distinguish the boundaries of the private part, the more efficient and less costly would be our synchronization alignment procedure. The worse extreme safe case is to take the whole model B being shared, so triggering the alignment procedure in any changes which occurs on Model B which is far away from being optimal technique.

Minimal/Least change application in model alignment. suppose we have the case of Inc.$=\frac{1}{2}$ or 1 and org-symm.=1, and we want to propagate the synchronization from Model

B to Model A. Again you can take the example in Figure 5 into account and suppose the update is changing the price of the flight in Model B from 50 to 200. Remember that the flight price in Model B should be at least \$100 less than the model A. we assume that we already identified the price in model B as a private part– So we already need to know about private and public parts. The question would be "what would be the changes on the corresponding flight in Model A when we change the price from 50 to 200 in Model B?" The answer here seems to be simple. According to price constraint, we increase the corresponding flight price in Model A up to the extend that make it 150 plus the flight price in Model B. But why not we increase the flight price to $300 + 10$ or $300 + 20$. Either of these changes would result in a consistent Model with Model B. The thing is that from synchronizer perspective it doesn't matter if we make the flight price in model A \$300, or 10, or 20 dollar or X dollar more! (ref. Figure 9) But it seems that some range of options are more desirable than the others in the sense that they change model A to the nearest model A' where they can establish again the alignment. In Our example we can say that this is an increase of price up to the price of the Flight plus 100, is the least change in Model A.

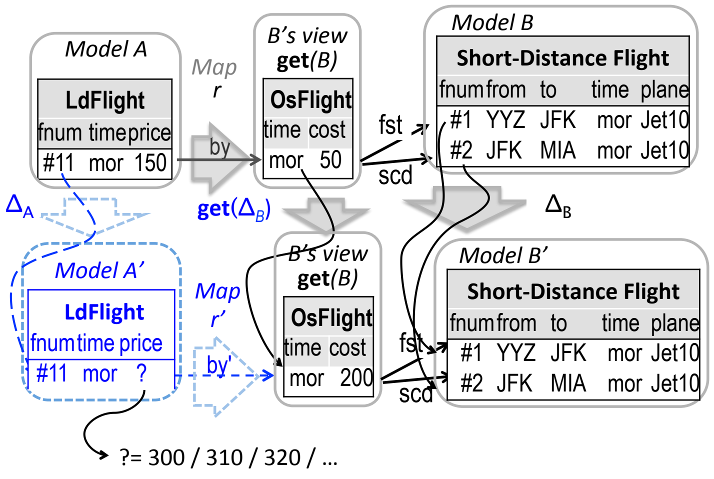


Figure 9: What is the Least Change to make Model A and B aligned again

Like the previous part this is the general issue that come up in the Incremental synchronization scenarios. This *least/minimal* change application on the target model to establish the alignment again between models is getting more complicated when the relationship among them getting complicated and also the models expressed by TAG. There are some works in the literature which claims doing an incremental synchronization [37], while without precise definition of the *private/shared* parts on the models and also the concept of the *minimal/least* change, the claim could not be neither verified nor compared with other approaches.

Concurrent synchronization and conflict resolution. As we already discussed in section 2.2, concurrency dimension seems not to be orthogonal to the org-symm dimension and identifying the role of the concurrency dimension in relationship with all the other dimension would need more study and investigation. Yet in the case of concurrent model evolution, there is the concept of conflict resolution which however is not well studied. Suppose that in our example in Section 2.1, we have the informational and organizational symmetric scenario. We increase the price of the flight in Model B from 50 to 200 because of the increase in the fuel cost, while at the same time the marketing side decrease the price of the corresponding flight in Model A from 150 to 100 because of the marketing and business decision. At first look this seems to be the distinctive issue of the application domain, but the abstract scenario is a typical situation which arises in synchronization of the info-symmetric/asymmetric and org-semi/full symmetric cases, while simultaneous changes are in conflict. Yet we see we already need to distinguish if these changes belong to the *private* part of each models or not. Later we need to decide how to make the alignment procedure with the *minimal* change ($\Delta_{B'_1}$ or $\Delta_{B'_2}$ in Figure 10), with the difference that we need to apply the procedure of minimal changes simultaneously to both sides which is a kind of inter-related procedure.

The conflict in our example is that the two changes are not compatible (one is increase and one is decrease). The conflict resolution decision is to decide how to resolve this conflict and make the models consistent again. As we already discussed one solution would be giving more priority on one side, say Marketing Model A, calling it *master*, and discarding the changes on Model B, then applying the solution of the previous section (least/minimal change on roll backed Model B). But this way the changes and concerns of the technical side in flight example would not be taken into account which might make them upset. The

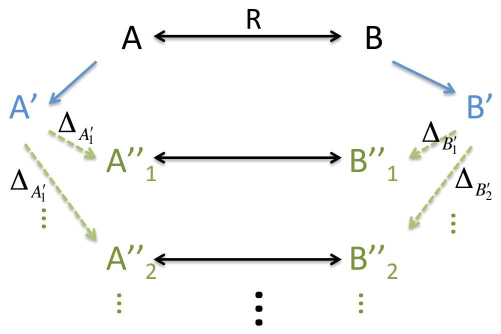


Figure 10: least change and conflict resolution in concurrent scenario

latter approach is not usually a desirable solution in the practical software engineering environment while two teams are working on a set of inter related models while non of them would be happy that their evolutions since their last alignment be disregarded completely. So we need to study and investigate this issue which is termed *conflict resolution* in synchronization domain. The desired solution would be the case we change both models to the new state with the *least/minimal* changes possible to make both models consistent again. The formulation of this approach is another spot that this thesis is following to address.

5 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.

Looking at *models* as *Graphs*, *Graph Transformation* techniques are considered as a *Model Transformation* foundation in that sense. The book in [58] 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 [23] 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 [21] as a formal representation of Models. The Attributed Graph Grammar System(AGG) is a tool which provides some feature for implementation of the ATG [67, 65]. Ecore is an implementation of the OMG meta-modeling standard which exist inside Eclipse Modelling Framework (EMF) [61] and many implemented tools are using it to specify meta-modeles in their environment specification. KM3 [32] is another meta-modeling specification language which is closer to objecte oriented style of class definition.

There are many research at the moment progressing on *Graph Transformation* and also some academic tools have been implemented for them. [16, 67, 33, 2]. Among them *Triple Graph Grammars*(TGG) as an extension of regular graph grammars and their tools holds more than 15 year research background [60] and some successful application. TGG rules are specifying source and target model evolution in a constructive manner. It is introduced by Andy Shüre in [59] and formally defined in [54], [22] and [40] 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 [52]. 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[52]. Some properties of of TGG like *Functional Behaviour*, *Correctness*, *Completeness*, and *Termination* are formally studied here [44, 25, 24, 40]. Inheritance, concurrency and application condition in TGG approach are discussed here [41, 43, 38]

Besides the Theory of the TGG there are some tools developed for that in academia and some of them applied successfully for industrial projects [35]. Among these we can mention MoTE [34], eMofelon [2], TGG Interpreter [42] and Emorf [53]. Although Bidirectionality 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 [57]. 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 [37]; But no formal foundation is provided to verify how far their incremental implementation is correct in the sense of being *Incremental*.

Apart from TGG tools there are some other tools like MediniQVT [47] based on OMG QVT specification [1]. Stevens criticise QVT semantics here [64] which is a hinder in QVT proper implementation. Story Diagrams is another approach in combining graph grammars and the Activity diagrams of the UML, for realization of the class operations [30].

Lenses Framework is discussed in [46]. 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 String data at the moment and covers info-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 [45] is another thrend trying to define BX programming languages based on edge-labeled Graphs. They are using $UnQL^{+}$ language(an Extension of UnQL [14]) for defining Bidirectional Transformation, but again the emphasize is on the Bidirectionality rather than the incrementality.

6 Conclusion and Future work

conclusion goes here...

Model are formalized as Typed attributed Graph(ref. section 2.5), and Deltas (Updates) among them can be considered as morphism between the graphs. So the models and the morphism are making a category of the TAGraph [23]. Further study of the connection between category theory and the Model synchronization with specific focus on the aforementioned problems in section 3 could shed more lights these issues and would help to better understanding and formalizing the problems discussed there.

Beside that looking at different example of model transformation in large scales in practice and investigating them, while keeping in mind the space of the synchronization scenario and also probing the problems and their probable domain specific solution to those problems would help in generalizing the solution to other domain. This is important because at the moment there are different approaches and languages used for model synchronization and transformation(ref. section 4) and this also happens in different domain of applications [63, 4].

So our next steps are divided into two main branches of theoretical investigation of the Category theory and model formalization and the practical and industrial investigation of the synchronization scenarios with the focus of the identified aforementioned problems, where we tend to join these two path together, However we are not taking them separate from the beginning.

We expect that the resolution of the issues in this thesis proposal will contribute to making the synchronization procedure and application more efficient and practical which will be an advancement in providing a technical and theoretical support in Domain of MDE in its specific and also in its broader sense [15]

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] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*, ICMT '09, pages 260–283, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] J. de Lara and H. Vangheluwe. AToM^3: A tool for multi-formalism and meta-modelling. In Ralf-Detlef Kutsche and Herbert Weber, editors, *FASE*, volume 2306, pages 174–188. Springer, 2002. http://link.springer.de/link/service/series/0558/bibs/2306/23060174.htm.

[17] 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.

[18] Zinovy Diskin, Arif Wider, Hamid Gholizadeh, and Krzysztof Czarnecki. Symmetrization of model transformations: Towards a rational taxonomy of model synchronization types. In *Submitted to the 16th International Conference on Model Driven Engineering Languages and Systems, Miami, Florida, USA*, MODELS '13, 2013.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel, editors. *Graph Transformations and Model-Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, volume 5765 of *Lecture Notes in Computer Science*. Springer, 2010.

[30] 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.

[31] 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.

[32] Fréderic Jouault, Jean Bezívin. KM3: a DSL for Metamodel Specification. In *Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*, LNCS, pages 171–185, Bologna, Italy, 2006.

[33] Fujaba Developer Team. Fujaba-Homepage. http://www.fujaba.de/, 2007.

[34] 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.

[35] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. Towards integrating sysml and autosar modeling via bidirectional model synchronization. In Giese et al. [36], pages 155–164.

[36] Holger Giese, Michaela Huhn, Ulrich Nickel, and Bernhard Schätz, editors. *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme V, Schloss Dagstuhl, Germany, 2009, Tagungsband Modellbasierte Entwicklung eingebetteter Systeme*, volume 2009-01 of *Informatik-Bericht*. TU Braunschweig, Institut für Software Systems Engineering, 2009.

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

[38] Ulrike Golas, Hartmut Ehrig, and Frank Hermann. Formal specification of model transformations by triple graph grammars with application conditions. *ECEASST*, 39, 2011.

[39] 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.

[40] 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.

[41] Ulrike Golas, Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Attributed graph transformation with inheritance: Efficient conflict detection and local confluence analysis using abstract critical pairs. *Theor. Comput. Sci.*, 424:46–68, 2012.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] M. Johnson and R. Rosebrugh. Constant complements, reversibility and universal view updates. In Meseguer and Rosu [56], pages 238–252.

[49] M. Johnson and R. Rosebrugh. Constant complements, reversibility and universal view updates. In Meseguer and Rosu [56], pages 238–252.

[50] 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.

[51] A. Keller. Comments on Bancilhon and Spyratos' "Update Semantics and Relational Views". *ACM TODS*, 12(3):521–523, 1987.

[52] 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.

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

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

[55] 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.

[56] José Meseguer and Grigore Rosu, editors. *Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008, Urbana, IL, USA, July 28-31, 2008, Proceedings*, volume 5140 of *Lecture Notes in Computer Science*. Springer, 2008.

[57] Sven Patzina and Lars Patzina. A case study based comparison of atl and sdm. In Andy Schürr, Dániel Varró, and Gergely Varró, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 7233 of *Lecture Notes in Computer Science*, pages 210–221. Springer Berlin Heidelberg, 2012.

[58] 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.

[59] 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.

[60] 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.

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

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

[63] Perdita Stevens. A landscape of bidirectional model transformations. In *Generative and Transformational Techniques in Software Engineering II*, pages 408–424. Springer, 2008.

[64] 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.

[65] 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.

[66] Eclipse modeling framework. Eclipse project Website.

[67] AGG — the attributed graph grammar system. http://user.cs.tu-berlin.de/ gragra/ag-g/index.html.