Modeling Theories and Model Transformation Scenario for Complex System Development

M.R. Dube
Vishwakarma Institute of Technology,
Pune, India.

S.K. Dixit
Walchand Institute of Technology,
Solapur, India.

ABSTRACT
In 2003, the Object Management Group (OMG) officially introduced the Model-Driven Architecture (MDA). The specification explains how different OMG standards could be used together. MDA focuses on the concepts of Platform Independent Models and Platform Specific Models, two viewpoints on software systems, and how mappings between these two can be made in order to streamline software development. Through this approach, the functional specification of the system and the implementation specification are separated, allowing for better reuse and portability. An important technique used in MDE is model transformation. A model transformation is a process of automatic generation of a target model from a source model, according to a transformation definition, which is expressed in a model transformation language.

This paper focuses on model engineering terminology and theory of model generation. The paper describes overall MDA scenario and emphasizes on model transformation. As model transformation is an important area in model driven development, we have compared various techniques available for model transformation. The paper also focuses on model theory and model composition techniques used for model weaving.

General Terms
UML, Metamodel, Model Driven Engineering

Keywords
MDA, PIM, PSM, Model Transformation

1. INTRODUCTION
Models are representation of reality in the form of abstractions which indicates essential aspects of the system while suppressing the others for the time being. The software development process models heavily rely on models for better understanding of the complex systems. Models are the medium to convey system's properties which are desired by its stakeholders as a set of services. Models focus on multiple views of a system and can integrate multiple concepts along with allied notational semantics that can efficiently demonstrate multiple views of the system depending on the context of the problem. These models can be organized in to various levels of abstractions by applying transformations on them. The abstract views of the system can elaborated into concrete views by adding relevant details to it as applicable. The correspondence between structural and behavioral properties of the system elements must be correlated and balanced to visualize systems architecture. In model-centric approach, models of the system drive implementation of the system from the models whereas in model-only approach, developers use models to understand contextual information of business or solution domain or architecture of a solution.

The Object Management Group (OMG) introduced the Model-Driven Architecture (MDA) as an approach for system development based on specification and interoperability expressed with the help of formal models. MDA is based on layering of model concepts and the transformations which are necessary to cause movement form one layer to the other. OMG identifies four types of models: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) described by a Platform Model (PM), and an Implementation Specific Model (ISM) related to technology used for system realization [1].

A platform can be thought as coherent set of functionalities relevant to technology indicating availability of usable services and resources to the stakeholders. The platform independence can be achieved by hiding the details of service profiles at software architecture level from the application level by introducing interfaces which can make the resource available from one platform to the other.

- Computation Independent Viewpoint: The computation independent viewpoint focuses on context of the problem with elicitation of requirements of the system and its structure with environmental needs. It indicates customer, user and stakeholder's perspectives as well as business cases to be addressed by the system.
- Platform Independent Viewpoint: The platform independent viewpoint focuses on analysis and design models of the system with relationships amongst system elements form architecture perspective towards implementation analogies.
- Platform Specific Viewpoint: The platform specific viewpoint indicates implementation level details of the system elements specific to a particular platform. This can be accomplished by using mapping and building transformation rules in case of migrating from PIM to PSM.

As indicated in Figure 1, platform-independent models (PIMs) represents the analysis and design models of the system with the help of general purpose modeling language such as UML. The platform-independent model can be mapped to a platform-specific model (PSM) by mapping the PIM to implementation language using set of well defined transformational rules. Metamodel is the basis for performing the transformations. MDA transforms the source to target models with the help of source to target metamodel mappings. MDA comprises CWM, UML, MOF and XMI as standards for model-driven development. The Common Warehouse Metamodel (CWM) defines a metamodel representing both the business and technical metadata which is essential to

capture the business analysis domains. Hence MDA can be interpreted as layered architecture where information exchange between the cross layers happens [2].

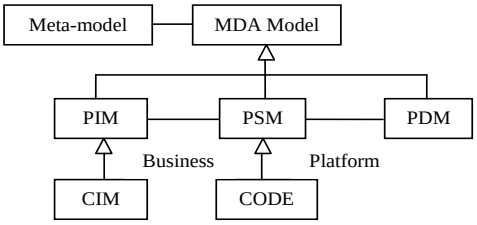

Figure 1: System Representation in MDE

Unified Modeling Language (UML) which is a general purpose modeling language provides support for modeling structural and behavioral properties of the system and an integrated effort of three object-oriented methods (Booch, OMT, and OOSE). The Meta Object Facility (MOF) is an OMG standard defining a common, abstract language for the specification of metamodels. It defines the four-level structure used to represent the details of how the notation repository can be made available to the modeler on model space. MOF semantics defines metadata repository that supports model construction. It is used to perform the transformations using transformation rules. XML Metadata Interchange (XMI) defines XML tags are used to represent objects and their interrelationships [3] [4]. MDA address the issue of portability and reusability in alliance with interoperability in order to achieve cross-platform transfer of system with stable environment. Following are the principles that underlie the OMG's view of MDA:

- Models expressed in a well-defined notation can be used to understand and interpret characteristics of complex systems.
- There exists a set of transformations used to serialize the model elements and their representation in order to express and realize the architectural viewpoints.
- As the emphasis of MDA is on deployment and maintenance the metamodels are used as a means to perform transformations and integration of the system elements which can be asset while performing automation.

Figure 2 indicates that MDE relies on two basic facts: (1) the representation of any kind of system by models and (2) the syntactic conformance of any model to a metamodel. Within this context, models are intended to be automatically processed by a set of operators. The system development can concern with Forward engineering, where a system is created from a model or Reverse engineering, where a model is created from a system or Models at runtime, where a model coexists with the system it represents [5][6].

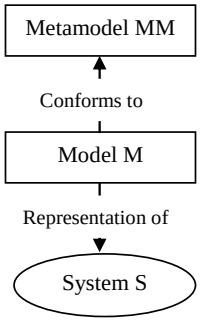

Figure 2: System Representation in MDE

2. RELATED WORK

Model transformation is a core concept in model-driven development that yields the model realization and model evolution by application of well formed transformation rules. In this section we survey existing model transformation technology. Tratt [7] defines model transformation as a program that transforms one model to other on the basis of mutation. OMG defines model transformation as the process of converting a model into another model of the same system based on transformation rules and metamodel correspondence [8]. Kleppe defines model transformation can be realized through a consistent transformation description that can be used to map source model to target model. Mens [9] defines model transformation as the process of generating multiple from multiple source models based on well-formed transformation rules that be used for automation purpose.

- ATL: ATL, the Atlas Transformation Language is a model transformation language specified in terms of a metamodel and a textual concrete syntax. ATL provides developers with a means to specify a technique to produce a number of target models from a set of source models. The ATL language is an integration of declarative and imperative programming. Through the declarative transformation style it expresses mappings between the source and target model elements along with their correspondence. ATL mainly deals with model to model transformation with the help of ATL module. An ATL module is composed of a header section that defines attributes relevant to transformation module; a optional import section to import s existing ATL libraries; a set of helpers that are ATL equivalent to Java methods; and set of rules that defines source to target model mappings. ATL-code is compiled and then executed by the ATL transformation engine. ATL supports only unidirectional transformations.
- Query/View/Transformation (QVT): QVT is a language for model transformation established by OMG in 2005. QVT uses the Object Constraint Language (OCL), Meta Object Facility (MOF). QVT language consists of language dimension and interoperability dimension. Each dimension specifies a formal set of named levels. QVT is structured into two-layer architecture in terms of relations metamodel and core metamodel. Relations metamodel is used to support object pattern matching and object template creation. Core metamodel is defined by extensions to EMOF and OCL. QVT defines three transformation languages:
  i. QVT Relational QVT Relational is a high-level declarative transformation language. It supports the specification of bidirectional transformations. When a bidirectional transformation is executed, the execution direction needs to be specified. The semantics is defined by a mapping to QVT core.
  ii. QVT Core is a simple, low-level declarative model transformation language. It serves as a foundation for QVT Relational and supports pattern matching over a flat set of variables, where the variables of source, target and trace models are treated symmetrically. Trace models must be defined explicitly.
  iii. QVT Operational is an imperative model transformation language, which extends QVT Relational with imperative constructs. The transformations are unidirectional. It uses implicit trace models.

- ModelMorf : ModelMorf is an implementation of the Relational QVT standard issued by OMG. It is a declarative model-to-model transformation. It supports multi-directionality, so the same rule can be used to map in both directions. The main feature of ModelMorf is its general model transformation approach. As implementation of the Relational QVT standard, it is purely declarative. For some transformation problems, a declarative transformation language offers some advantages over other approaches. ModelMorf does not support incremental transformation execution, Transformation extensibility and Graphical Syntax. Because no integrated development environment is provided, the development of transformation code is not as comfortable as in other approaches. Errors in the code are only detected when the transformation is executed. It provides built-in functionality to create traces. It is possible to create in-place transformations.
- Kermeta: Kermeta is a general purpose modeling and imperative programming language, also able to perform transformations. It can be described as a metamodeling language which allows describing both the structure and the behavior of models. It incorporates several features and ideas of other technologies in the model driven architecture domain. It uses operations, which are basically very similar to operations or methods in object-oriented programming languages. Kermeta's imperative approach and advanced language features make it nearly as powerful as conventional programming languages. Kermeta has no built-in tracing support, and loading and saving of model files is somewhat complicated compared to other languages. Rule application control and rule scheduling needs to be specified explicitly by the user.
- ETL: The Epsilon Transformation Language (ETL) is a hybrid model-to-model transformation language. It is part of the Epsilon model management infrastructure. ETL has a hybrid style and features declarative rule specification that consists of approaches as guards, abstract, lazy and primary rules, and automatic resolution of target elements from their source counterparts.ETL indicates specification of transformations which can transform arbitrary source models into arbitrary target models. ETL transformations are organized in modules. A module can contain a number of transformation rules. Each rule has a unique name and also specifies one source and many target parameters.
- VIATRA: VIsual Automated model TRAnsformations is a model transformation framework, which provides support for the life-cycle of engineering model transformations including the specification, design, execution, validation and maintenance of transformations for modeling languages and domains. A transformation definition consists of a set of rules that are organized to match a pattern within a model to which the rule can be applied to and application of the rule in an operational style. VIATRA2 primarily aims at designing model transformations to support the precise model-based systems development with the help of invisible formal methods. Since precise model-based systems development is the primary application area of VIATRA2, it necessitates that (i) the model transformations are specified in a mathematically precise way, and (ii) these transformations are automated so that the target mathematical models can be derived fully automatically. Kermeta is best suited as model transformation language, if the transformation to implement is relatively complex.

3. MODELING TERMINOLOGY

- Model: A model is a simplified representation of a system that helps in better understanding of the system [10] [11]. A model is written in the language of its unique metamodel. A model is a constrained directed labeled graph. Models are often expressed in dedicated domain-specific languages or general purpose modeling language such as UML [12].
- Metamodel: The abstract syntax of a model is described by its metamodel. A metamodel can be compared to a grammar in language design. Model transformations cannot be achieved without metamodels. A metamodel is a model such that its reference model is a metametamodel.
- Metametamodel: The abstract syntax of metamodel is described by metametamodel. A metametamodel is a model that is its own reference model means that it conforms to itself.
- Model Transformation Paradigm/Approach: A model transformation paradigm or approach is the design principle on which the model transformation language is built, e.g. imperative, operational, functional, declarative or relational.
- Model Transformation Language: A model transformation language is a vocabulary and a grammar with well-defined semantics and rules for performing model transformations.
- Model Transformation Description: A model transformation description is written in a model transformation language. If the language of a transformation description is rule-based, the transformation description is a set of transformation rules; describing source models are transformation to target models.
- Model Transformation Rule: A model transformation rule describes how a source model can be transformed into a target model. Transformation Rules are subdivided into domains that are the part of the rule responsible for accessing one of the models that are used in the transformation rule. The domains subdivisions define how many domain languages the rule operates, how the domains are declared, if they are implicitly or explicitly declared. Following are the important aspects which should be considered while devising the transformation rule:
  i. Rule application scoping: It allows a transformation to define the target scope of a transformation. A restriction can be made about the parts of a model that participate in the transformation. Both a source and a target scope can be defined.
  ii. Rule application strategy: A rule needs to be applied to a specific aspect within its scope. The strategy can be deterministic, non-deterministic or interactive. The nondeterministic strategies are again subdivided in concurrent, and one-point. An interactive strategy approach is when the user is allowed to interact with the transformation for determining the location where the rule is going to be applied.
  iii. Rule scheduling: A scheduling mechanism can be used to determine the order in which individual rules are applied. In some approaches users have no explicit control on the scheduling algorithm. The rule scheduling is related to the order of rule application. It is subdivided into four other features: form, rule selection, rule interaction, and phasing. The form refers to the way scheduling is being expressed.

iv. Rule organization: Rules can be composed and structured in different ways. Model transformation approaches can vary on the basis of modularity mechanisms, reuse mechanisms and organizational structure.
v. Traceability links: Transformations may record links between their source and target elements. These links can be useful in performing impact analysis, synchronization between models, and determining the target of a transformation.
vi. Directionality: Transformation may be unidirectional or bidirectional. In unidirectional transformations, source model can be transformed into a target model, but not the other way. Bidirectional transformation can be useful for synchronizing models.

- Model Transformation Engine/Tool: A model transformation engine or tool executes or interprets the model transformation description. It applies the model transformation description on the source model to produce the target model. [13]
- Source Model: A model is called as a source model in model transformation, if it is an input to the transformation. It conforms to the source metamodel.
- Target Model: A is called as target model in model transformation, if it is an output of the transformation. It conforms to the target metamodel. A model transformation can have one or more target models.
- Higher-order Transformation (HOT): A model transformation description having a transformation model as source or target model.
- Directed multigraph: A directed labeled multi-graph G = (N_G, E_G,□ Γ_G) consists of a finite set of nodes N_G and a finite set of edges E_G, a mapping function $\Gamma_G : E_G \rightarrow N_G \times N_G$.
- Model: A model M = (G,ω,μ) is a triple where: G = (N_G, E_G,□ Γ_G) is a directed multigraph, ω is itself a model associated to a multigraph $G_\omega = (N_\omega, E_\omega, \omega)$, $\mu : N_G \cup E_G \rightarrow N_\omega$ is a function associating elements (nodes and edges) of G to nodes of G_ω. The function μ associates every node and edge of G ($N_G \cup E_G$) with one element in $\omega(N_\omega)$.
- Reference model: Given a model $M_1 = (G_1, \omega_1, \mu_1)$, and a model $M_2 = (G_2, \omega_2, \mu_2)$,if $\omega_1 = M_2$, M_2 is called the reference model of M_1.

Some models are their own reference model (ω = M). This allows stopping the recursion introduced in this definition. The relation between a model and its reference model is called conformance. The three levels of model engineering can be identified as metametamodel (M3), metamodel (M2) and terminal model (M1). A metametamodel is a model that is its own reference model. A metamodel is a model such that its reference model is a metametamodel. A terminal model is a model such that its reference model is a metamodel [14].

In any complex software system, mastering complexity means using a variety of semantically and syntactically precise models to describe different aspects and views of the software system. In Model to Model transformation, a model is converted to another model of a system and in Model to Text transformation, a transformation a model is converted directly to an arbitrary fragment of text. In Model to Code Transformation, the text produced by transformation is source code or a fragment of source code from the model. Model to Code transformation is usually used to implement modeling languages in terms of programming languages whereas Model to Model transformation is used for refactoring.

Figure 3 shows the Model Transformation Process. Model composition has impacts on at least three different levels:

- Syntactic level: the compositional way between models can explicitly be expressed as a new model in an appropriate modeling language.
- Semantic level: the meaning of composed models as a unit in terms of semantics of modeling languages involved.
- Methodic level: the integration of model composition techniques in software development processes and tools.

Formally, a modeling language M is a set of well-formed models. So a model m∈M is syntactically well-formed, both by context-free syntax as well as conforming to all context-conditions.

Each of these models gets semantics by mapping it from the language to a well-known semantic domain. This principle is well understood in the field of programming languages, where each syntactic construct has a well defined meaning that describes its effects in terms of operational or denotation semantics [15].

- Given a language M of models, the meaning of each element is usually given by explaining it in a well-known domain D, the semantic domain. This semantic domain describes which artifacts and concepts exist and must be well understood by both the language designer and the language users.
- Given the modeling language M and the semantic domain D each model m∈M must be mapped to D. As explained earlier, it is important to define the meaning (semantics) of models explicitly. So an explicit formal definition of the mapping is a function from M to D: s_m: M → D.

Benefits of a formal mapping function are that we are able to reason about the mapping and thus, about the language and the instances itself. Set-valued semantics allows stating some important properties with respect to the semantic mapping s_m:

- A model m∈M is consistent exactly if $s_m(m) \neq \emptyset$, which means that there is at least one system that obeys the instance's properties. Otherwise, there are some contradicting constraints in the model m itself.
- A model m∈M does not contain information if $s_m(m)=S$. Then any system can serve as an implementation.
- A model m_2 refines another model m1 exactly if $s_m(m_2) \subseteq s_m(m_1)$. So, if we add more data to the model m2, it further constraints the resulting set of systems, which therefore will become smaller.

The MOF is semiformal approach to define modeling languages. It provides four-level hierarchy, with levels M0, M1, M2 and M3. The entities $\tilde{m}$ populating reach level M_i, written $\tilde{m} \in M_i$ are always collections, made up of constituent data elements $\tilde{e}$. Each entity $\tilde{m} \in M_i$ at level i+ 1 meta represents model and is viewed as the metarepresentation of collection of types, i.e., as metadata collection that defines specific collection of types. Each type T is metarepresented as $\tilde{T} \in \tilde{M}$ and characterizes collection of data elements, its value domain. We write that data element $\tilde{e} \in \tilde{m}$ is value of type $\tilde{T} \in \tilde{M}$ as $\tilde{e} : \tilde{T}$. A . A metarepresentation at level i +1 of collection $\tilde{M} \in M_{i+1}$ types characterizes collections of data elements $\tilde{m} \in M_i$ at level i. A specific data collection $\tilde{m} \in M_i$ said to conform to model , which is metarepresented by its collection of $\tilde{M} \in M_{i+1}$, iff for each data element $\tilde{e} \in \tilde{m}$ there exists type $\tilde{T} \in \tilde{M}$ such that $\tilde{e} : \tilde{T}$. A [16].

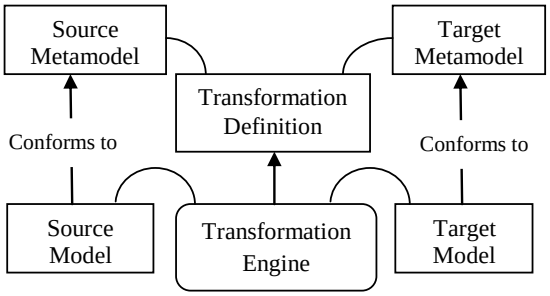

Figure 3: Model Transformation Process

Following are the critical factors influencing Model Driven Development scenarios:

- Model-to-model transformations. A PSM is created by transforming a PIM and a PDM. Reverse engineering roughly corresponds to the opposite operation of transforming a PSM into a PIM.
- Heterogeneity. MDE homogenizes all software artifacts as uniform models; concrete systems can make use of a number of different technologies. Hence the problem exists in terms of ability to handle transformations between MDE and other technologies.
- Complexity. Complexity is partitioned into Problem-domain and Solution-domain. The Problem-domain complexity comes from the issues related to requirements and stakeholder agreements. The Solution-domain complexity usually describes in terms of interrelated system elements or components [17].

4. MODEL PROPERTIES

Model transformation languages have been classified by Czarnecki and by Mens [20]. Following are the properties of model transformation problems:

- Abstraction Change: Source and target model must preserve the level of abstraction during model transformations. The level of abstraction is indicates essential aspects in a model. Model transformations introduce new aspects, reduce the aspects or leave it unchanged. This property is independent of the change in the metamodel; source and target metamodel can be the same or different. A horizontal model transformation is a transformation, where the source model and the target model belong to the same abstraction level. In case of vertical, different levels of abstraction are used in the source and target model. It is also called refinement, because additional information is added, resulting in decreasing the abstraction level of the target model. On the other hand, information could be removed from the source model to create a more abstract target model. The transformation of platform independent model to platform specific model is a case of a vertical transformation, because models are on different levels of abstraction. Transformations which affect models expressed in the same language are called endogenous. Both source and target models conform to the same metamodel. Opposite to endogenous transformations, exogenous transformations are expressed between models conforming to different metamodels. Exogenous transformations can also be called translations, because it is focused on the mapping of a model in different metamodels. Abstraction transformations produce the target model by reducing the amount of detail.
- Metamodel Change: It indicates the difference between source and target metamodels [20]. There are three types of changes handled during metamodel change handling. An unaffecting change does not require any adaptation of existing models, which is mostly true for additive changes to the metamodel. In resolvable changes, an algorithm can be defined for migration of existing instances to the new metamodel version. In irresolvable changes, manual interaction is required to make existing models conform to the new metamodel, if possible at all. Existence Change describes the addition or deletion of an element in the metamodel which an instance of a MOF class. This can cover classes, attributes, associations. Property Change describes a property of a metamodel element represents an attribute in the MOF class of which the element is an instance that is changed. Link Change represents all changes in features that are associations in the MOF model. This includes containment, inheritance and typing [18].

Transformations can be built in such a way that source model and target model have a common property, which is not changed by the transformation [19].

- Semantics-preserving: If the source and target metamodels are similar, a mapping can be found that is semantics preserving. It means that the transformation rules are able to retain the meaning of the two models same, even though they are represented in a different abstract syntax. Refactoring is the process of changing the internal structure of a model without simultaneously changing the externally observable behavior or functionality of the corresponding program, in order to improve quality attributes of the model. Refactoring transformations are semantics-preserving, since they do not change the behavior of the model, but improve structure and quality of the model.
- Behavior-preserving: A transformation is behavior-preserving if the explicit or implicit constraints of the behavior in the source model correspond to the target model after the transformation occurs. Model to code transformation is not semantics-preserving because of levels of details, it is behavior-preserving.
- Syntax-preserving: A syntax preserving transformation is usually an endogenous horizontal transformation that does not change the abstract syntax of the model.
- Number of Models: A model transformation can have several source models and several target models [20]. The minimum number of models involved is one, where source and target model are the same.

Model transformation languages follow different language paradigms as indicated below:

- Imperative languages specify a sequential control flow and provide means to describe how the transformation language is supposed to be executed. From a model transformation point of view, the imperative approach is also similar to direct-manipulation. A well-defined control flow exists, which means that all statements in the transformation code are executed in order. Imperative model transformation techniques focus on how the transformation has to be executed. Because of the simplicity, imperative model transformation languages are easier to learn. The transformation is described as a sequence of actions, which is especially useful if the order of a set of transformation rules needs to be controlled explicitly [21].

- Declarative languages do not offer explicit control flow. Declarative approaches focus on what the transformation has to accomplish. Procedural details of the transformation are hidden, which results in a smaller amount of code. These transformations describe relationship between the source and the target metamodels which can be interpreted as bidirectional.
- Hybrid transformation languages offer both imperative language constructs and declarative language constructs. Imperative approaches are powerful, but based on verbose therefore harder to read and understand. Declarative languages may not be suitable for complex transformation tasks.
- Graph transformation languages are based on algebraic graph grammars and are a subcategory of declarative languages. Model elements and their relationships are seen as graph vertices and edges. Graph transformation rules have a left hand side, and a right hand side, which correspond to the source and target model of the transformation. Models are interpreted as graphs, and graph transformations manipulate sub-graphs. Graph-based does not necessarily mean that the transformation is specified in a visual way.
- Template-based languages are used for model-to-text transformations. Templates contain fragments of the target text and a metaprogram that can access the source model. These languages are combined with the visitor pattern to traverse the internal structure of a model.

5. COMPOSITION AND WEAVING

Model composition in its simplest form refers to the mechanism of combining two models into a new one. Without further information or requirements the definition of model composition is quite abstract. Denoting the universe of models with *M* we get the following definition of model composition operators:

- Model composition operator: A model composition operator $\otimes$ is a function with two models as input, which produces a composed model as output: $\otimes: M \times M \rightarrow M$. Given the semantics of models, we can infer properties of the semantics of a composition operator $\otimes$ by relating the semantics on its source and resulting model.
- Property preserving (PP) operator: A composition operator $\otimes: M \times M \rightarrow M$ is property preserving on the left argument, if for any $m_1, m_2 \in M$ it holds: $s_m(m_1 \otimes m_2) \subseteq s_m(m_1)$. Analogously, it is property preserving on the right argument, iff $s_m(m_1 \otimes m_2) \subseteq s_m(m_2)$ and property preserving (PP) if both properties hold. Property preservation is important for a composition operator, as it ensures that no information and thus, no design decisions that were present in a source model are lost in the composition. We can infer that property preservation is equivalent to: $\forall m_1, m_2 \in M: s_m(m_1 \otimes m_2) \subseteq s_m(m_1) \cap s_m(m_2)$.
- Fully property preserving (FPP) operator: A composition operator $\otimes: M \times M \rightarrow M$ is fully property preserving, iff $\forall m_1, m_2 \in M: s_m(m_1 \otimes m_2) = s_m(m_1) \cap s_m(m_2)$.
- Consistency preserving (CP) operator: A composition operator $\otimes: M \times M \rightarrow M$ is consistency preserving (CP), iff $\forall m_1, m_2 \in M: s_m(m_1) \cap s_m(m_2) \neq \phi \Rightarrow s_m(m_1 \otimes m_2) \neq \phi$
- General Semantic Composition Operator: The semantic composition operator $\otimes$ is a function with two sets of systems as input which produces a set of systems as output: $\otimes: D \times D \rightarrow D$. We say the diagram commutes iff $\forall m_1, m_2 \in M: s_m(m_1 \otimes sm_2) = s_m(m_1) \otimes s_m(m_2)$. A commuting diagram corresponds to a fully property preserving composition as defined above and exhibits the same advantages as discussed above. We therefore impose the requirement that the model element should always commute. If not, at least the relaxed version must be considered: $\forall$ m1, m2 $\in$ M: s_m (m1 $\otimes$ m2) $\subseteq$ s_m (m1) $\otimes$ s_m (m2). Therefore, the syntactic operator $\otimes$ reflects the semantic composition $\oplus$ and an additional refinement. However, in the following we use intersection as semantic composition only.
- The semantic mapping s_m defines an equivalence relation on models as $m_1 \cong m_2 \Leftrightarrow s_m(m_1) = s_m(m_2)$. The set of semantically equivalent models is denoted by $[m_1] = \{ m_2 \mid m_1 \cong m_2 \}$.
- A weaving model must conform to the weaving metamodel. A weaving model contains abstract and declarative links that are used to generate integrated model transformations for the target model. A model integration transformation addresses the interoperability issue. The weaving process begins with the designer provides the input metamodels MM_A and MM_B and a sequence of matching transformations is executed. They produce a weaving model with a set of links between MM_A and MM_B. The designer verifies the links (and corrects them, if necessary) and a transformation model is generated based on the set of links. The transformation produced is used to transform a model conforming to MM_A into a model conforming to MM_B.
- A weaving metamodel is a model $MM_W = (G_M, \omega_M, \mu_M)$, that defines link types, such that:
  - $G_M = (N_M, E_M, \Gamma_M)$,
  - $N_M = N_L \cup N_{LE} \cup N_O$, N_L denotes the link types, N_{LE} denotes the link endpoint types and N_O denote other auxiliary nodes,
  - $\Gamma_M: E_M \rightarrow (N_L \times N_{LE}) \cup (N_O \times N_M)$, i.e., a link type refers to multiple link endpoint types and the auxiliary nodes refer to any kind of node.
- A weaving model is a model $M_W = (G_W, \omega_W, \mu_W)$, a graph $G_W = (N_W, E_W, W)$, such that its reference model is a weaving metamodel ($\omega_W = MM_W$).

Depending on the goals to be achieved by model merging, following properties are required to be addressed:

- Completeness: If a concept appears in one of the source models, it is represented in the merged model as well. This is to ensure that no information is lost in the merge process.
- Non-redundancy: If a concept appears in more than one source model; only one copy of it is included in the merged model.
- Minimality: Merge does not introduce new information, which is neither present nor implied by the source models.
- Totality: Merge is computable for an arbitrary set of models. This property is of particular importance if one wants to tolerate inconsistency between the source models.
- Soundness: Merge supports the expression and preservation of semantic properties.

Atlas Model Weaver (AMW) is used for inter-model correspondence on the basis of use of operators. The semantics of these relationships can be defined as a transformation expressing how related elements have to figure in the composed model. It offers abstraction mechanisms by the definition of simple correspondences (weaving operators) between two metamodels. The operational semantics of the weaving operators is determined by a higher-order transformation that takes a weaving model as input and generates model transformation code. The weaving models are compiled into low-level transformation code in terms of ATL which is a mixture of declarative and imperative language constructs. Thus, it is difficult to debug a weaving model in terms of weaving operators, because they do not explicitly remain in the model transformation code. Finally, a weaving operator always connects source metamodel elements to target metamodel elements, so it is not possible to realize complex transformation logic by the composition of operators [22].

Kompose is a composition tool built on top of Kermeta, a metamodeling environment allowing adding behavior to metamodels. Kompose is a generic framework to support model composition. The core composition mechanism is implemented in Kermeta as a separate metamodel that can be specialized for a specific domain metamodel in order to easily define composition operators for that domain. The framework is made of a generic model element merge algorithms and a directive language. The specialization for a specific metamodel is done by defining appropriate signatures for the classes of this metamodel. Kompose uses signatures to infer model relationships. A signature is a set of values extracted from model elements properties values, such as its name and its type. iMAP is an approach for creation of links. The links are then used for matching source model transformation to target model [23].

Modeling Aspects using a Transformation Approach (MATA) proposes a radically different approach than the two previous ones. Indeed, the composition becomes asymmetric, as one model plays the role of base model while the other is seen as an aspect. In MATA, composition of a base and aspect model is specified by a graph rule. Given a base model, M_B, crosscut by an aspect model, M_A, a MATA composition rule merges M_A and M_B to produce a composed model M_{AB}. MATA graph rules are defined over the concrete syntax of the modeling language. This is in contrast to almost all known approaches to model transformation which typically define transformations at the meta-level, that is, over the abstract syntax of the modeling language. MATA considers aspect composition as a special case of graph transformation. In general, a graph consists of a set of nodes and a set of edges. Each metaclass becomes a node in the type graph and each meta-association becomes an edge in the type graph.

Similarity Flooding can be used for model transformation in three phases. First transformation phase calculates similarity values between model elements and creates a propagation graph, second transformation phase propagates the similarity values through nodes that are connected, and third transformation phase selects the best matching results and creates a weaving model. The output weaving model contains links generated by the matching transformations. The accuracy of the weaving model depends basically on similarities calculations and selection of similarity values.

Clio produces transformations based on a set of relationships. Clio has supports the production of complex mappings with nested structures. The definition of the relationships cannot be extended, which hardens the task of creating complex kinds of mappings [24] [25].

6. COMPARATIVE ANALYSIS

Model Transformation is a central issue in Model Driven Development which deals with conformance of source model to target model on the basis of metamodel mappings. The comparative analysis is based on the transformation rule support, rule scheduling, and rule application control. It can be interpreted from the comparison that imperative model transformation languages offer powerful features and flexibility than other approaches. Relational approaches offer multidirectionality and it is up to the user to select the basis of model transformation. Table 1 indicates transformation rules while Table 2 indicates transformation language feature. Table 3 summarizes the rule application control whereas Table 4 addresses the rule application control.

Table 1. Transformation rules

Feature	ATL	QVT	ModelMorf	Kermata
Multidirectionality	N	N	Y	N
Syntactic Preservation	Y	N	Y	N
Application Conditions	Y	Y	N	Y
Aspects	N	N	N	Y

Table 2. Transformation Language Features

Feature	ATL	QVT	ModelMorf	Kermata
Abstract Syntax	Y	Y	N	N
Text Syntax	Y	Y	Y	Y
Graphical Syntax	N	N	N	N
Language Paradigm	Hybrid	Imperative	Imperative	Imperative
Element Creation	Implicit	Implicit	Explicit	Explicit

Table 3. Rule Application Control

Feature	ATL	QVT	ModelMorf	Kermata
Deterministic	Y	N	Y	N
Non Deterministic	N	N	N	N
Interactive	N	N	N	N

Table 4. Rule Scheduling

Feature	ATL	QVT	ModelMorf	Kermata
Implicit Form	Y	N	Y	N
Explicit Form	Y	Y	Y	Y
Explicit Condition Selection	N	N	N	N
Iterative Selection	N	N	N	N
Conflict Resolution	N	N	N	N
Rule integration	Recursive	Recursive	----------	Recursive
Phasing	Y	Y	Y	N

7. CONCLUSION

In this paper, we have discussed the principal components of MDA and the interrelated components. We have discussed the model theory and terminologies that can facilitate the model designer for better understanding of the model. We showed that model generation process consists of model composition, transformation and weaving. We have elaborated on the semantic aspects of model element composition through use of composition operators. We observe that a transformation describes design decisions with relation to the mapping between metamodel elements.

We have also discussed the four primary elements of the MDE process: languages, models, programs, and model transformations. This knowledge is used in the following chapters to obtain a better understanding of the performance of model transformations. Transformation languages are designed to generate an output model, given an input metamodel, an input model, and an output metamodel. Throughout the years, numerous projects have implemented their ideas on how a transformation language ought to look like. The popular transformation languages are the ATL and QVT, created by INRIA and the OMG respectively. ATL is a hybrid language, which compiles its transformations to byte code, QVT is a declarative language that allows bidirectional transformations to be created.

8. REFERENCES

[1] Seidewitz, E., What models means, IEEE Soft. 20(5), 26–32 (2003)

[2] Selic, B, The pragmatics of model-driven development, IEEE Software 20(5), 19–25 (2003)

[3] Barbero M, Jouault F, Bézivin J (2008) Model driven management of complex systems: implementing the macroscope's vision. In: Proceedings of the 15th annual IEEE international conference and workshop on engineering of computer based systems (ECBS 2008), 31 March–4 April 2008. IEEE Computer Society, Belfast, pp 277–286

[4] OMG, UML 2.0 Superstructure Specification, August 2005. Document formal/05-07-04. Available at http://www.omg.org/

[5] OMG, Meta Object Facility (MOF) Core Specification, version 2.0, January 2006, Document formal/06-01-01, Available at http://www.omg.org/

[6] OMG, UML 2.0 Infrastructure Specification, March 2006, Document formal/05-07-05, Available at http://www.omg.org/

[7] L. Tratt, Model transformations and tool integration," Software and Systems Modeling, vol. 4, no. 2, pp. 112

[8] L. Tratt. Model transformations and tool integration. Journal of Software and Systems Modeling, 4(2):112–122, May 2005.

[9] T. Mens and P. Van Gorp, \A taxonomy of model transformation," Electr. Notes Theor. Comput. Sci, vol. 152, pp. 125-142, 2006.

[10] OMG Architecture Board, Model driven architecture—A technical perspective, OMG Document ormsc:01-07-01. Available at www.omg.org

[11] OMG, MOF 2.0 query/views/transformations RFP, OMG Document ad/02-04-10, Available at www.omg.org

[12] OMG, OMG meta-object facility (MOF), OMG Document formal/01-11-02, Available at www.omg.org

[13] Mellor, S.J., Scott, K., Uhl, A., Weise, D., MDA Distilled: Principle of Model Driven Architecture. Addison-Wesley, Reading (2004)

[14] Kent, S., Model driven engineering. In Proceedings of IFM International Formal Methods 2002, vol. 2335, Lecture Notes in Computer Science. Springer, Berlin Heidelberg New York (2002)

[15] Czarnecki, K., Helsen, S., Classification of model transformation approaches. In: Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, Anaheim (2003)

[16] Mens, T., Demeyer, S., Janssens, D., Formalizing behavior preserving program transformations, In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) Proceedings Graph Transformation—First International Conference, ICGT 2002, Barcelona, Spain, vol. 2505, Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York (2002)

[17] Van Der Straeten, R., Jonckers, V., Mens, T., Supporting model refactoring through behavior inheritance consistencies, In: BaarT., et al (eds.) UML 2004—The Unified Modeling Language, Model Languages and Applications, 7th International Conference, Lisbon, Portugal, vol. 3273, Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York (2004)

[18] Bézivin J (2005), On the unification power of models. Software System Model (SoSym) 4(2): 171–188

[19] Jouault F, Kurtev I (2006), On the architectural alignment of ATL and QVT. In: Proceedings of the 2006ACMsymposium on applied computing (SAC 06), ACM, Dijon, pp 1188–1195

[20] Balogh, A., Németh, A., Schmidt, A., Ráth, I., Vágó, D., Varró, D., Pataricza, A.: The VIATRA2 model transformation framework, In: Proceedings of ECMDA 2005—Tools Track, 2005

[21] B´ezivin J, Lemesle R (2000), Some Initial Considerations on the Layered Organization of Metamodels, In: SCI 2000/ISAS 2000, International Conference on Information Systems, Analysis and Synthesis, vol IX, Orlando, August 2000

[22] L. Tratt. The MT model transformation language. In Proc. ACM Symposium on Applied Computing, pages 1296–1303, April 2006.

[23] Amelunxen C, Königs A, Rötschke T, Schürr A (2006), MOFLON: A standard-compliant metamodeling framework with graph transformations, In: Rensink A, Warmer J (eds) Model driven architecture—foundations and applications: second European conference, ECMDA-FA 2006. Lecture notes in computer science, Vol. 4066. Springer, Bilbao, pp 361–375

[24] Bézivin J, Jouault F, Rosenthal P, Valduriez P (2005), Modeling in the large and modeling in the small. In: Model driven architecture, European MDA workshops: foundations and applications, MDAFA 2003 and MDAFA 2004. Lecture notes in computer science, Vol. 3599. Twente, The Netherlands, pp 33–46

[25] M. D. Del Fabro and P. Valduriez, Semi-automatic model integration using matching transformations and weaving models," in SAC '07:Proceedings of the 2007 ACM symposium on Applied computing. New York, NY, USA: ACM Press, 2007, pp. 963-970.