Domain-specific modeling of Cooking Recipes - Project Report

Xiaoxi Dong

McGill University, Montreal Qc Hxx xxx, Canada

Abstract. Visual Modeling and Transformation System (VMTS) is a general purpose meta-modeling and transforming environment. It combines the two important area of modeling together. VMTS functionalities include n-layered meta-modeling which is handled as a labeled directed graph by Attributed Graph Architecture Supporting Inheritance and visual transformation which is controlled using internal and external causalities exhaustively located the instantiation of LHS nodes and transform to the corresponding RHS. VMTS focuses on the separation concern that divide the storage, presentation and validation modules as clear as possible. This report has a brief introduction to the design of VMTS and present the visual approach used for modeling and transforming.

1 Introduction

Visual modeling and model transformation is a popular research area in developing the scope of Modeling and Simulation. It faces challenges from the great variety of the models. This report introduces a software package for model editing and storage and model transformation called the Visual Modeling and Transformation System (VMTS). The VMTS facilitates an approach that treats the model storage and model transformation uniformly by supporting the notion of meta-model. The flexible visually modeling environment is built on the basis of meta-modeling which is able to be configured and specify the constraints. The model transformation makes it possible to analyze an model in particular forms and to create the output for the model such as database files and C++ code. The VMTS system includes several plug-ins for OMG standards of modeling and it finds a way to incorporate complex model transformation rules into less expressive standard descriptions like UML. This approach allows VMTS to create a general transformation system. The basis of VMTS formalisms is directed graph model with labels.

In the design of VMTS it focuses on the separation concern: the presentation, storage and validation modules has been separated as clearly as possible([1]). The client ends and server ends are separated so as to obtain a potential of developing the two ends without influence the other. As we can see in the (see Figure 1), VMTS is an visual development environment for modeling and model transformation. It is comprises of 5 components. The VMTS architecture defines

the following structure: Database, (ii) Server side - AGSI Core, Rewriting engine, and (iii) Client side - Rule Editor, Modelers and other applications. The control core called ASGI. The database which applies SQL server to store the models. The visual interface of Adaptive Modeler Control flow language which instructs the step of model transformation the input/output can read/write database and generate c++ code from model. VMTS has been implemented in .Net C#, but it focuses on the separating the kernel of class and relationship from the developing environment. As a result, only XML database support and Object Oriented language are assumed([1]). The rest of the report is organized as, Section 2 is

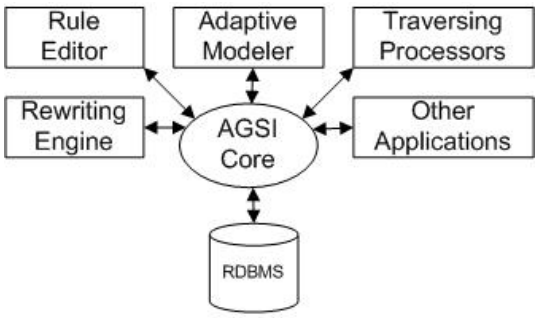

Fig. 1. VMTS Structure

about the model storage system and meta-modeling; section 3 introduces the issue about model transformation, including the visual transformation process and the control flow language as well as the way VMTS handles the constraints. Section 4 displays VMTS' under interface and the way to use it; and Section 5 is the discussion.

2 N-layer meta-modeling Environment

Basically, model is an instantiation of relations. The VMTS uses a simplified UML class diagram as a basic meta-model language to define models. VMTS have a five layered structure(see Figure 2). Different from MOF 4 layer structure, the VMTS also have a layer for model storage called Attributed Graph Architecture Supporting Inheritance(AGSI) as the root of labeled directed graph. Three of layers(meta-model, model and information) are involved in instantiation of the model objects and two are read only meta meta-model which specifies the meta-modeling language and the one in the internal structure ie. the labeled directed graph.([1]) AGSI is defined a recursively so that every model can be a meta model for other models. The designer of VMTS believes that the five layer structure is enough for practical model application, more layers is redundant. AGSI handles graph with node, edges and labels that assigned to nodes

AGSI Root (Labeled Directed Graph)	M3 (Hard-wired meta-meta model)	M2 (metamodel)	M1 (model)	M0 (information)
MetaNode	Node (abstract)	-	-	-
MetaNode	Atom	MetaClass	Class	Object
MetaNode	Atom	MetaNote	Note	-
MetaNode	Atom	MetaUseCase	UseCase	-
MetaNode	Container	MetaPackage	Package	-
MetaRelationship	Relationship	MetaAssociation	Association	
MetaRelationship	Relationship <<MetaInheritance>>	Inheritance	-	-
MetaRelationship	-	Relationship <<MetaInheritance>>	Inheritance	-
MetaRelationship	-	Relationship <<MetaContainment>>	Containment	-
MetaRelationship	-	Relationship <<MetaAsoociationClassRelationship>>	AssociationClassRelationship	
MetaRelationship	Relationship	MetaNoteAnchor	NoteAnchor	
MetaRelationship	Relationship	Communicate	-	-

Fig. 2. Layer structure for VMTS core: first 2 columns are read only and is hard-wired in VMTS. the other 3 is instantiation of model objects.

and edges. AGSI also support another 3 elements: 1.traversing model in containment hierarchy, 2.support inheritance,and 3.store the association classes in regular graph structure. These features are hard wired in AGSI. The inheritance is supported so that the descendant types inherit the relationship types from the ancestor types and the relations between ancestors can be passed down the inheritance hierarchy in the transformtion. As mentioned before that the model in VMTS is a labeled directed graph, the attributes of objects are stored in labels. ASGI stores the labels as XML documents and the attributes are represented in XMI like format. There are two types of attributes, one is meta-attributes and the other is attributes in meta-model elements which is equivalent to the operation in UML.(see Figure 3) The abstract class can not have attributes instantiated. As the attributes are converted to XSD file to store. It is easy for VMTS to operate on these attributes. It is worthy to note that when it involves inheritance, the priority of attributes in inheriting class is higher than parent when there is same attribute names in inheritance hierarchy.

3 Model Transformations

The other part of VMTS functionalities is the model transformation. Model transformation converts an input model to an output model. VMTS maintains two kinds of transformation include Traversing Model Processor and Visual Model Processor. The two processors serves for different purpose respectively. The TMP is specially useful in generating code in VMTS framework which is regarded as a UML class diagram. The process of operations to the model by objects in regular object oriented programming language. To generate code in object oriented programming languages, two model levels must be considered. That is model layer and the meta-model layer. The processors regard the elements as UML classes. VMTS define the transformation of components using graphical pattern matching of LHS and RHS. To control the transformation It develop a control flow language. is basically a activity diagram for user to de-

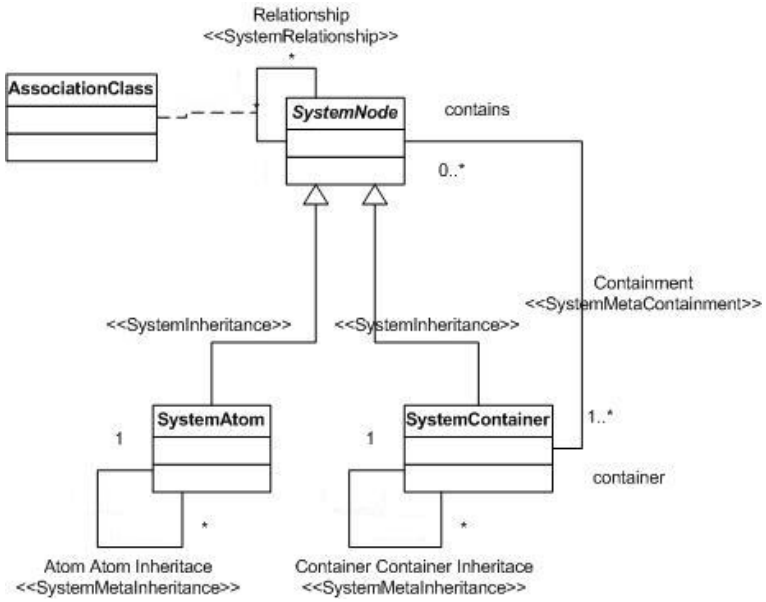

Fig. 3. Meta Meta-model

fine the sequence of executing transformations. The explanation of this part will come later in this section.

3.1 Visual Transformation Process

The visual transformation process uses graph rewriting techniques as an alternative way of representing model transformation. It has set of rules consisting of rules that have LHS graph and RHS graph, which are defined in meta model. The processors will find occurrence of match of class instances in LHS instead of the direct occurrence of graph nodes. By doing this, the VMTS transformations support inheritance multiplicity. That is when derived class can be passed where the ancestor is expected, which enables generalization in the rules. This is how VMTS describe the semantic of LHS. Then it comes to the RHS for attribute transformation. The LHS and RHS have configuration of multiplicity, which put constraints on the attributes of both sides. LHS and RHS has causality relationship defined as internal causalities.

3.2 VMTS Control Flow Language

The VMTS Control Flow Language is a stereotyped UML activity diagram, where the activities are specified by transformation rules. Control Flow Language allows user to specify the external causality, which are the conditions and constraints to pass between rules, that is the causality between RHS of rule1

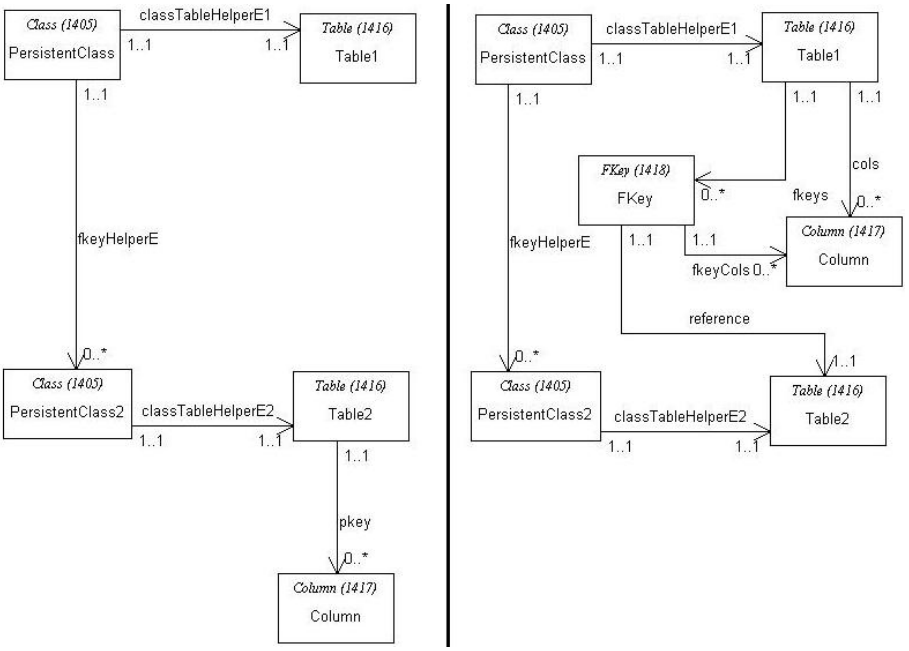

Fig. 4. Example of LHS and RHS visual transformation rules

and LHS of rule2 when there are a transformation between the two rules. Once the external causality is satisfied, the next rule will be automatically invoked. As can be seen in the figure below. the users can specify the sequence of rules to be executed and it also allows circles and branches. By using CFL, VMTS allows users to get greater control over the transformation process. ([2])

In VMTS, the control flow graph is as shown in the figure below. The nodes are transformation rules. We can add constrains to the relation as external constraints which specifies in what condition the rules be executed.

3.3 Manage Constraints

In the process of building a model and transformation of the models, we need to define constraints. VMTS define the model constraints in terms of OCL constraints placed in meta model. The transformation constraints are also expressed in OCL as the rules are specified in the meta-model elements of input and output models. VMTS automatically support preserving, guaranteeing and validating constraints. We can set constraints to both control flow and transformation rules. It is worthy to note that the constrains are difficult to define before the first draft of transformations and it is hardly to be verified intuitively. Moreover, in the transformation process we will need to weave the constraints to transformation steps using some weaving methods. Inside the rules, we also have constraints associated with each components in both sides. In detail, it is what we are going to

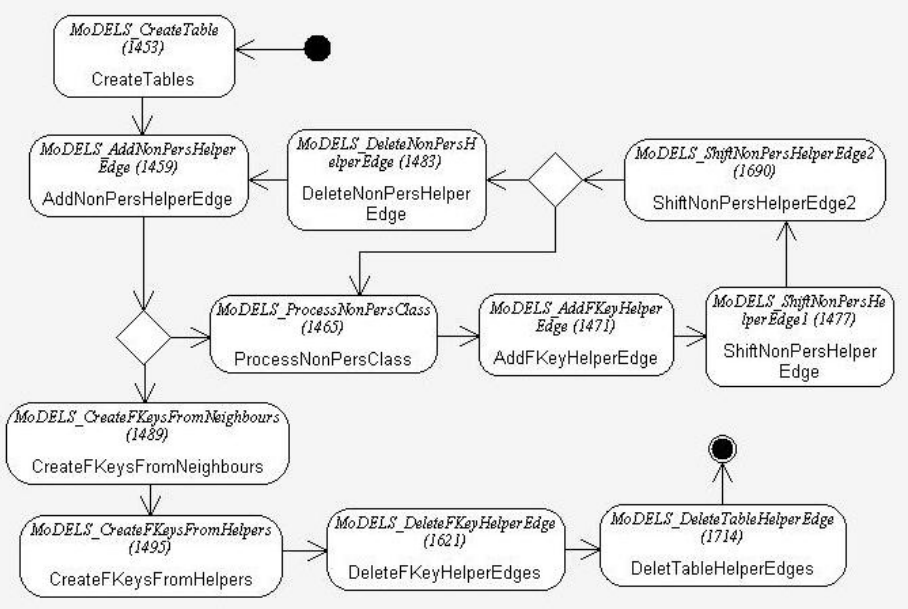

Fig. 5. Example of Control Flow

do with each elements, such as deleting LHS elements, modifying the attributes, and creating new elements in RHS.

4 VMTS Presentation Framework

VMTS Presentation Framework(VPF) facilitate a means of rapid development for plug-ins as a customized presentation of the concrete syntax of the models.

we can bulid meta-model in VMTS and make it a plug-in. It takes five steps,

1. VPD meta model
2. the concrete syntax using model transformation
3. VPD to CodeDOM model transformation
4. CodeDOM model to code
5. source code should be compiled using .NET and add the generated .dll file to VMTS directory.

After these steps we adds a new formalism to VMTS Adaptive modeler.

To manage constraints, VMTS uses Constraint Weavers to weave the constraints to its corresponding processes. Here are example forms for constraints weavers.

When we have had the meta model, the transformation rule and transformation control flow. We can start to transform our model to the formalism we

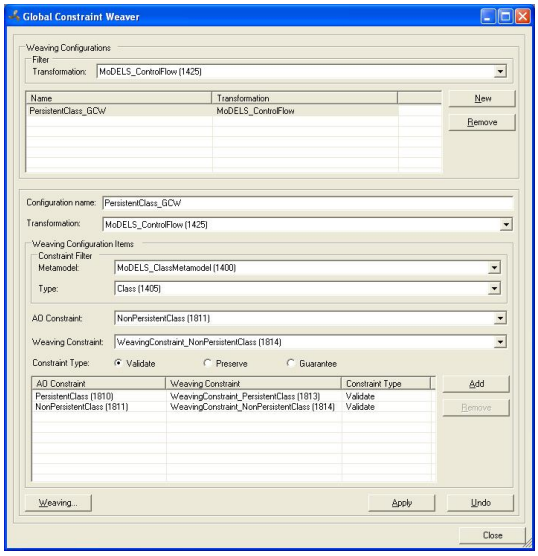

Fig. 6. Gloabal Constrains Weaver

prefer. In VMTS, the tool to execute transformation is called Visual Model Processor(VMP) plug-in. To transform from one model to another, the user needs to maintain at least three different elements in VMTS. One is the input meta-model, the second is output meta-model and the transformation control flow. These elements need to resident in the same project. When transforming, the user will choose the corresponding plug-ins in Visual Model Processor window, where the textual result of matching and transforming is also displayed.

5 Conclusion

VMTS is one of the visual development environment for meta-modeling and model transformation. It combines the two important aspects which are multi-layered meta-model and visual model transformation in modeling and simulation in one tool.

VMTS provides many interfaces to standards published by the Object Management Group, which make it easier to use the production of VMTS in other environment. VMTS support inheritance which make it possible for the derived class inherit the relationship from the ancestors. The constraints defined to the attributes conform to OMG's standards such as Object Constraints Language. The transformation controller language CFL allow users to have strict control over the transformation process. This feature is different from non-deterministic process in many other model transformation techniques. The output of VMTS model can be transformed to .xml files which makes it possible to be transferred to other. VMTS also provide user with an friendly user interface for developing.

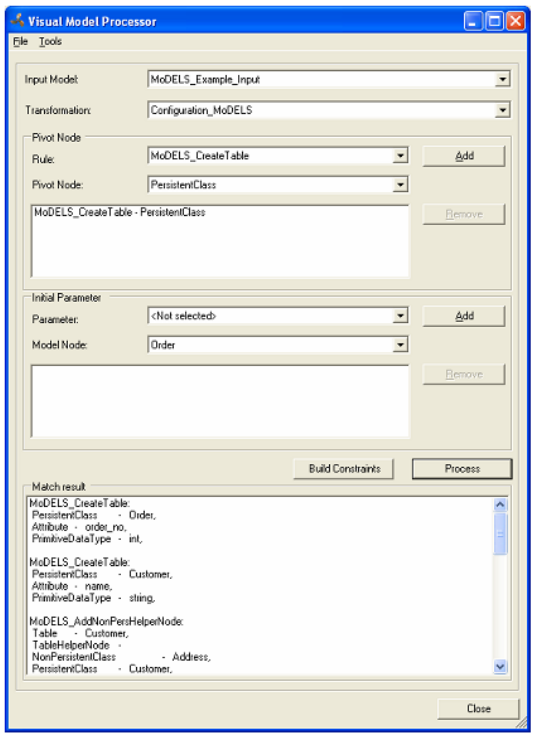

Fig. 7. Visual Model Processor Form

The constraints of VMTS is firstly it must be installed in a machine supporting .NET and SQL Server, which make it difficult for researchers. The other problem is although providing a general tool helps, it do not have plug-ins for existing formalisms.

References

1. Levendovszky, T., Lengyel, L., Mezei, G., Charaf, H.: A systematic approach to metamodeling environments and model transformation systems in vmts. In: International Workshop on Graph-Based Tools (GraBaTs) Electronic Notes in Theoretical Computer Science (to be published). (2005)
2. Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varró, D., , Varró-Gyapay, S.: Model transformation by graph transformation: A comparative study. In: ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica (October 2005)
3. Visual Modeling and Tranformation System Tutorial: Validated Model Transformation In:`http://avalon.aut.bme.hu/ tihamer/research/vmts/`
4. Visual Modeling and Tranformation System Tutorial: Automated Concreted Syntax Definition In:`http://avalon.aut.bme.hu/ tihamer/research/vmts/`