Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp30-35)

A Visual Control Flow Language

LÁSZLÓ LENGYEL, TIHAMÉR LEVENDOVSZKY, GERGELY MEZEI, HASSAN CHARAF
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Goldmann György tér 3, Budapest, H-1111
HUNGARY

Abstract: - Graph rewriting-based model processing is a widely used technique for model transformation. Model transformations often need to follow an algorithm that requires a strict control over the execution sequence of the transformation steps. Therefore, in Visual Model Processors (VMPs) the execution order of the transformation steps is crucial. This paper introduces the visual control flow support of Visual Modeling and Transformation System (VMTS). VMTS Visual Control Flow Language (VCFL) uses stereotyped activity diagrams to specify control flow structures and Object Constraint Language (OCL) constraints to choose between different control flow branches. This work discusses the termination properties of VCFL and provides algorithms to combine model transformation steps as well as to support the termination analysis of VCFL transformations.

Key-Words: - Control Flow, Metamodel-Based Model Transformation, OCL, Termination Properties, UML.

1 Introduction

Visual Modeling and Transformation System (VMTS) [1] [2] is an n-layer metamodeling environment which supports editing models according to their metamodels, and allows specifying OCL constraints. Models and transformation steps are formalized as directed, labeled graphs. VMTS uses a simplified class diagram for its root metamodel ("visual vocabulary").

Also, VMTS is an UML-based [3] model transformation system, which transforms models using graph rewriting techniques. Moreover, the tool facilitates the verification of the constraints specified in the transformation step during the model transformation process.

Graph rewriting [4] is a powerful technique for graph transformation with a formal background. The atoms of the graph transformation are rewriting rules, each rewriting rule consists of a left-hand side graph (LHS) and a right-hand side graph (RHS). Applying a graph rewriting rule means finding an isomorphic occurrence (match) of LHS in the graph to which the rule is applied (host graph), and replacing this subgraph with RHS.

Model transformation means converting an input model available at the beginning of the transformation process to an output model. Several widely used approaches to model transformation uses graph rewriting as the underlying transformation technique. Previous work [1] has introduced an approach – metamodel-based rewriting rules –, where the LHS and RHS of the transformation steps are built from metamodel elements. This means that an instantiation of LHS must be found in the host graph instead of the subgraph isomorphic to LHS. This metamodel-based approach facilitates to assign OCL constraints to pattern rule nodes (PRNs) – nodes of the rules.

The Object Constraint Language (OCL) [5] is a formal language for the analysis and design of software systems. It is a subset of the UML standard [3] that allows software developers to write constraints and queries over object models.

The motivation of the work presented in this paper is to support control flow in visual model transformation systems and to define the conditions exactly which guarantee that if a transformation fulfills them it terminates or not. An algorithm is developed to support the termination analysis of VCFL transformations.

2 Related Work

Many approaches have been introduced in the field of graph grammars and transformations to capture graph domains; for instance, the GReAT [6], the PROGRES [7], the FUJABA [8], the VIATRA [9], the Attributed Graph Grammar (AGG) [10] and the AToM^3 [11]. These approaches are specific to the particular system, and each of them has some features that others do not offer.

The GReAT framework is a transformation system for domain specific languages (DSL) built on metamodeling and graph rewriting concepts. The control structure of the GReAT allows specifying an initial context for the matching to reduce the complexity of the general matching case. The attribute transformation is specified by an attribute mapping language, whose syntax is close to C.

PROGRES is a visual programming language in the sense that it has a graph-oriented data model and a graphical syntax. PROGRES provides constructs for rule firing and for sequencing the rules. PROGRES offers refined control structures; both imperative and declarative approaches can be used.

In FUJABA the combination of activity diagrams and collaboration diagrams (story-diagrams) are used to express control structures.

VIATRA (Visual Automated Transformations) is a model transformation framework developed mainly for the formal dependability analysis of UML models. VIATRA uses abstract state machines (ASM) to define the attribute transformation and the control flow of the system.

In AGG, termination criteria are implemented for Layered Graph Transformation Systems. The criteria they propose are based on assigning a layer to each rule, node and edge type. For termination, they define layered graph grammars with deletion and non-deletion layers. Termination criteria are expressed by deletion and non-deletion layer conditions.

The transformation and simulation tool AToM^3 uses model transformation to simulation traces in order to simulate the operations. Similarly to AGG, the control flow consists of layers; the rules are sequenced by priority numbers within the layers.

3 The VMTS Visual Control Flow Language

One of the most important capabilities of a control flow language is the possibility to express a transformation as an ordered sequence of the transformation steps. Classical graph grammars apply any production that is feasible. This technique is appropriate for generating and matching languages but model-to-model transformations often need to follow an algorithm that requires a more strict control over the execution sequence of the steps, with the additional benefit of making the implementation more efficient.

The VMTS approach is a visual approach and it also uses graphical notation for control flow: stereotyped activity diagrams, which is a technique to describe procedural logic. UML activity diagrams are intended to describe business processes, and work flows.

In VMTS transformation steps the *internal causality* is a relation between LHS and RHS elements, it makes possible to connect an LHS element to an RHS element and to assign an operation to this connection. An internal causality describes what we have to do during applying a transformation step (element creation, element deletion, attribute modification). The *create* and *modify* operations are accomplished by XSL scripts. The XSL scripts can access the attributes of the objects matched to LHS elements, and they produce a set of attributes for RHS element to which the causality point.

Sequencing transformation steps results in a transformation which contains the steps in an ordered sequence ($S_0, S_1 ... S_{n-1}$). Assume the case that the input model of the step i (S_i) is the model M_i and the result of the S_i is the M_{i+1} (where $0 \le i \le n-1$). In this case the input model of the step $i+1$ (S_{i+1}) is the model M_{i+1}. This means that during the execution of the step sequence, each step works on the result of the previous step.

The interface of the transformation steps allows the output of one step to be the input of another step, in a dataflow-like manner. This is used to sequence expression execution. In VCFL this construction is referred to as *external causality*. An external causality creates a linkage between a node contained by RHS of the step i and a node contained by LHS of the step $i+1$. This feature accelerates the matching and reduces the complexity, because the step i provides partial match to the step $i+1$.

Branching with OCL Constraints. Often, the transformation we would like to apply depends on a condition. Therefore, a branching construct is required. In VCFL, OCL constraints assigned to the decision elements can choose between the paths of optional numbers, based on the properties of the actual host model and the success of the last transformation step (*SystemLastRuleSucceed*).

Hierarchical Steps. The VCFL supports hierarchical specification of the transformation steps. High-level steps can be created by composing a sequence of primitive steps and can be viewed as separate transformation modules. A high-level step can contain several simple steps, hiding the details which could be unimportant on a specific abstraction level and represents the contained steps as coherent units.

Iteration (Tail Recursion) and Parallel Executions of the Steps. The iteration is achieved with the help of the decision objects and the OCL constraints contained by them. A decision object evaluates the assigned constraints, and based on the results selects a flow edge which could be a follow-up or a backward edge as well. Recursion can be solved with the combination of the iteration and external causalities. A high-level step can call itself, where external causalities represent the actual parameters of the recursive call. The parallel execution of the independent transformation steps is supported by the *Fork* and *Join* elements.

In VCFL, if a transformation step fails and the next element in the control flow is a decision object, it could provide the next branch based on the OCL statements and the value of the *SystemLastRuleSucceed* variable. If no decisions can be found, the control is transferred to the parent state, if there is no parent state, the transformation terminates with error.

4 Termination Criteria

The termination properties of a transformation are really important for model transformation. We want to investigate under which conditions an arbitrary VCFL transformation can satisfy termination criteria. Our aim is that VCFL transformations terminate, therefore an algorithm (VCFL Termination Algorithm) has been developed to support the early detection of the infinite loops and the validation of the control flow that from each step can reach an end step.

In the VCFL a transformation step has two specific attributes: *Exhaustive* and *MultipleMatch* [12]. Recall that applying a model transformation step means finding a match of LHS in the host model and replacing this subgraph with RHS. An *exhaustive* transformation step is executed continuously as long as LHS of the step can be matched to the host model. The *MultipleMatch* attribute of a step allows that the matching process finds not only one but all occurrence of LHS in the host model, and the replacing is executed on all the found places.

Definition (VCFL Transformation). A *VCFL transformation* is a stereotyped UML activity diagram. A *VCFL transformation T* defines a strict order of the contained transformation steps $S_0, S_1 ... S_{n-1} \in STEPS \in T$, where S_0 is the start step of *T*. Transformation *T* contains OCL constraints assigned to decision objects to choose between different control flow branches, and external causalities between transformation steps to support parameter passing.

Definition (Termination of VCFL transformations). A VCFL transformation *T* for a finite input model G_0 *terminates*, if there is no infinite derivation sequence from G_0 via transformation steps $STEPS \in T$, where starting from S_0 (start step of the *T*) steps $STEPS$ are applied as it is defined by the transformation *T*.

For non-exhaustive and also for exhaustive transformation steps, the *MultipleMatch* attribute of the steps does not modify the termination property of the VCFL control flows for arbitrary finite input model G_0.

The termination checker algorithm differentiates between certain cases. It needs to take into account whether the VCFL transformation contains loops with decision object or exhaustive transformation steps.

4.1 VCFL Control Flows with Non-Exhaustive Transformation Steps

Proposition. A VCFL transformation *T*, which contains only non-exhaustive transformation steps $(S_0, S_1 ... S_{n-1})$ and does not contain loops for an arbitrary finite input model G_0 always terminates.

Proof. The transformation *T* contains finite number of transformation steps $(n = \#STEPS \wedge n < \infty)$. $\forall i \mid 0 \le i \le n-1 \quad S_i \in STEPS$ is executed at the most once because it is a non-exhaustive step.

If the multiple match attribute of a step $S_i \in STEPS$ is true, all occurrence of the S_i^{LHS} (LHS of the step S_i) is searched for, and the replacement is executed for all the found matches, but step S_i is executed only once. The number of the found matches (m_i) is also finite because of the finite input model G_0. $n < \infty \wedge m_i < \infty \mid 0 \le i \le n-1$, therefore $k = \sum_{i=0}^{n-1} m_i < \infty$. The number of the steps executed by transformation *T* is finite and *T* terminates.

4.2 VCFL Control Flows with Exhaustive Transformation Steps

Definition ($\subseteq$*).* $G_m \subseteq G_n$ if and only if G_n has a structurally isomorphic subgraph G_l to G_m, and in the G_l and in the G_m the corresponding nodes and edges have the same metatype, attributes, attribute values and OCL constraints.

An exhaustive transformation step must contain either attribute modification or element deletion to prevent that the same match be found again and again by the matching process. A solution can also be found if there is a causality of type *create*, and an OCL constraint that holds before the transformation step becomes false afterwards, therefore it prevents to find the same match again on the same place.

Definition (Create Termination Step – CT step). A *create termination step S* has only create type internal causalities, it contains an arbitrary OCL constraint C_I in S^{LHS}, which must stand for the host models matched to the S^{LHS}, and as a result of the step execution, the condition required by the constraint C_I becomes false.

Obviously, this transformation step property is important only for exhaustive steps or steps which are in loops.

Following propositions contain statements about termination properties of the transformations with exhaustive transformation steps.

Proposition. Let the transformation step S_i be an exhaustive step. If $S_i^{LHS} \subseteq S_i^{RHS}$ and the step S_i has a match M on an arbitrary input model G_i the step S_i never terminates for the input model G_i.

Proof. The step S_i has a match M on the input model G_i it generates its output (G_i^1) with S_i^{RHS}. $S_i^{LHS} \subseteq S_i^{RHS}$, therefore S_i^{LHS} has match in G_i^1. The step S_i is an exhaustive step and it always has match on the result model of the previous iteration, therefore the S_i never terminates for the input model G_i.

Proposition. Let the transformation step S_i be an exhaustive step which does not contain internal causalities of deletion and modification type, and S_i is not a CT step. Assume that T is a transformation and $S_i \in T$, the input model of transformation T is the model G_0, and the input model of the step S_i is the model G_i. If the S_i^{LHS} has a match M on model G_i, the transformation T never terminates for the input model G_0.

Proof. The step S_i is an exhaustive transformation step, it is executed as long as the S_i^{LHS} has match on model G_i. The S_i has a match M, which is not modified by the step – there is no deletion, attribute modification, and S_i is not a CT step –, therefore the matching process finds the match M in each iteration. The step S_i never terminates for the input model G_i, and T never terminates for the input model G_0.

4.3 Transformation Step Combination in VMTS

The intention of the transformation step combination is to create a single step S_C from an optional number of transformation steps $S_j, S_{j+1} \ldots S_k$. The combined step can equivalently replace the original steps, because it produces the same result and imposes the same input conditions as the sequence composed of individual rules. In the termination analysis, we can use the combined step instead of the original transformation steps. It facilitates to replace the steps contained by a VCFL loop with their combined transformation step. The result of the replacement is similar to an exhaustive transformation step, with the difference that a combined step may have a decision object.

The combination algorithm takes not only the structure of the steps into consideration but also the external- and internal causalities, the metatypes of the PRNs and edges and the constraints contained by PRNs.

The external causalities defined between the steps simplify the complexity of the step combination. They define exactly the mapping between the RHS elements of the step i and the LHS elements of the step $i+1$. There cannot be any contradiction between the constraints contained by the PRNs mapped to each other based on the parameter passing specifications, because they are checked during the control flow design.

Internal causalities connect the LHS and RHS elements within a transformation step. Therefore, taking both the internal- and external causalities into account, we can follow the node mapping between the transformation steps and also within them. This means that the PRNs can be unambiguously identified and followed through a loop or a whole transformation.

The metatypes of the PRNs and edges, compared to the general case, also simplify the computation complexity of the algorithm. The metatypes narrows the problem area. The worst case is, when all the PRNs have the same metatype. It is equal with the general case of the rule combination. External causalities provide an initial mapping, which should be extended based on the metatype-based mapping. All possible cases and combinations should be examined and considered as a possible mapping. If there is no external causality defined, the algorithm starts form a node with metatype which has the fewest occurrences in the LHS of the transformation step.

The constraints propagated to the PRNs should be checked whether there is any contradiction between the constraints contained by the PRNs mapped to each other during the combination.

The algorithm works based on the double pushout (DPO) approach concurrency theorem [13].

The VMTS Transformation Steps Combinator (VTSC) algorithm is as follows.

```
VTSCOMBINATOR(TransformationStep[] TSs, bool exhaustive):
TransformationStep[]
  1 combinationList = GETFIRSTSTEP(TSs)
  2 if exhaustive
  3 return COMBINERECURSIVELY(GETFIRSTSTEP(TSs)
 4 else
 5   foreach Transformation Step stepNext in TSs (except first step)
 6     foreach Transformation Step S in combinationList
 7       initialMapping = CREATEMAPPINGBYEXTCAUS(S, stepNext)
 8       newCombinations = EXTENDMAPPINGBYMETATYPES(
                                 initialMapping, S, stepNext)
 9       ADDTOLIST(newCombinationList, newCombinations)
10     end foreach
11     combinationList = newCombinationList
12   end foreach
13 end if
14 return combinationList
```

Proposition. If the transformation steps $S_j, S_{j+1} ... S_k$ are applicable successfully for an input model G_0, then a transformation step S_C, created from transformation steps $S_j, S_{j+1} ... S_k$ using VTSC algorithm, has the same effect on the input model G_0 as the transformation steps $S_j, S_{j+1} ... S_k$.

Proof. If the transformation steps $S_j, S_{j+1} ... S_k$ are applicable successfully for an input model G_0, then it means that the step S_j can be executed on the model G_0 and produces the model G_j, the step S_{j+1} can be executed on the model G_j and produces the model G_{j+1}, and so on. Each step can be applied on the result of the previous step, and finally, the steps $S_j, S_{j+1} ... S_k$ produce the model G_{j+k}. The step S_C produces the same result on the input model G_0 as the transformation steps $S_j, S_{j+1} ... S_k$, because step S_C is created considering all the modifications committed by steps $S_j, S_{j+1} ... S_k$. Therefore step S_C executes all the modifications of steps $S_j, S_{j+1} ... S_k$ in a single step.

4.4 Termination Properties of VCFL Loops

A loop contains n transformation steps (where $n>0$) and a decision object. A decision object evaluates the assigned constraints on the actual host model and based on the results selects a flow edge which could be a follow-up or a backward edge as well.

Proposition. Assume that the transformation T contains a loop L, let S_C be the only possible combination of the non-exhaustive transformation steps $S_j, S_{j+1} ... S_k \in L$. The input model of the transformation T is the model G_0, and the input model of the step S_C is the model G_C. If $S_C^{LHS} \subseteq S_C^{RHS}$ and the step S_C has a match M on input model G_C the transformation T never terminates for the input model G_0.

Proof. The transformation step S_C has a match M on input model G_C it generates its output model $G_C^1 \mid S_C^{RHS} \subseteq G_C^1$. $S_C^{LHS} \subseteq S_C^{RHS}$, therefore the $S_C{}^{LHS}$ has match on model G_C^1. The step S_C represents a loop and it always has match on the result model of the previous iteration, therefore the S_C never terminates for the input model G_C and the transformation T never terminates for the input model G_0.

4.5 VCFL Termination Algorithm

For an arbitrary VCFL transformation T the termination algorithm validates the following.

1) If transformation T does not contain loop or exhaustive transformation step then T terminates.
2) If $S \in T$ is an exhaustive transformation step and $S^{LHS} \subseteq S^{RHS}$ the transformation T does not terminate.
3) If $S \in T$ is an exhaustive transformation step, S does not contain delete and modify type internal causalities and S is not a CT step then the transformation T does not terminate.
4) If $L \in T$ is a loop and S_C is the combination of the transformation steps $S_h, S_{h+1} ... S_k \in L$ and $S_C^{LHS} \subseteq S_C^{RHS}$ the transformation T does not terminate.

The pseudo code of the VCFL termination algorithm is the following.

```
VCFLTERMINATIONALGORITHM(Transformation T): retValue
 1 if T does not contain loop or exhaustive step then return retValue.true
 2 foreach Transformation Step S in T
 3   if S is exhaustive and RHS of the S contains LHS of the S then return retValue.false
 4   if S is exhaustive and S does not contain modify or deletion and S is not an ST step then return retValue.false
 5 end foreach
 6 foreach Loop L in T
 7   combinedStep = COMBINETRANSFORMATIONSTEPS(transformation steps of the L)
 8   if RHS of the combinedStep contains LHS of the combinedStep then return retValue.false
 9 end foreach
10 return retValue.undecided
```

If the transformation step contains *create* type internal causality, the algorithm checks whether the host model with the newly added elements contains new possible match places. The algorithm takes the structure of the pattern, the metatypes of the nodes and edges, their attributes along with attribute values as well as the propagated OCL constraints into consideration.

VTA is an offline algorithm; the termination in many cases depends not only on the VCFL transformation model but also on the actual host model. A simple constraint can be itself a significant difference between two steps or an attribute value between two models. The problem is not trivial. There are certain cases when the algorithm can make a decision based on the VCFL transformation, and there are other cases when not.

4.6 Summary of the Termination Criteria

Termination of transformations is not always guaranteed. If a control flow model contains an exhaustive step that can be applied infinitely to the result models, the transformation does not terminate.

All derivation sequences over transformation steps $STEPS \in T$ are terminating if each transformation step $S \in STEPS$ terminate. Since the non-exhaustive termination steps terminate, we can state the following proposition.

Proposition. A VCFL transformation T terminates if all exhaustive transformation step $S_E \in STEPS$ and loop $L \in T$ terminate.

5 Conclusion

This paper has provided a control flow technique for model transformations based on graph rewriting. The transformations are represented in the form of explicitly sequenced transformation steps. We have shown the fundamental concepts of the VCFL approach.

Termination is an important issue for model transformations. Since model transformations can become very complex, we consider not only the application of single transformation steps, but also transformations where step applications are restricted according to a strict control flow.

In this work, we discussed the properties of the VMTS Visual Control Flow Language. We stated and proved several termination criteria for transformation steps, loops and transformations.

The introduced approach can be generalized to other control flow languages, which facilitate to assign constraints to transformation steps and supports constraint evaluation.

VCFL has successfully been applied in industrial projects, like generating user interface from resource model and user interface handler code from statechart model for Symbian [12] and .NET Compact Framework mobile platform [14].

6 Acknowledgement

The fund of “Mobile Innovation Centre” has supported in part, the activities described in this paper.

References:

[1] T. Levendovszky, L. Lengyel, G. Mezei, H. Charaf, A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS, *ENTCS, International Workshop on Graph-Based Tools, GraBaTs,* Rome, 2004.

[2] The VMTS Homepage. http://avalon.aut.bme.hu/~tihamer/research/vmt

[3] OMG UML 2.0 Spec., http://www.omg.org/uml/

[4] G. Rozenberg (ed.), *Handbook on Graph Grammars and Computing by Graph Transformation: Foundations*, Vol.1 World Scientific, Singapore, 1997.

[5] OMG Object Constraint Language Specification (OCL), www.omg.org

[6] G. Karsai, A. Agrawal, F. Shi, J. Sprinkle, On the Use of Graph Transformation in the Formal Specification of Model Interpreters, *Journal of Universal Computer Science*, 2003.

[7] A. Schürr, A. Zündorf, Nondeterministic Control Structures for Graph Rewriting Systems, *in Proc. WG'91 Workshop in Graph- Theoretic Concepts in Computer Science, LNCS 570,* 1992.

[8] FUJABA, http://wwwcs.upb.de/cs/fujaba/

[9] D. Varró and A. Pataricza, VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML, *Journal of Software and Systems Modeling*, 2003.

[10] G. Taentzer, AGG: A Graph Transformation Environment for Modeling and Validation of Software*, Application of Graph Transformations with Industrial Relevance (AGTIVE’03)*, vol. 3062. Springer LNCS, 2004.

[11] J. Lara, H. Vangheluwe , M. Alfonseca, Meta-modelling and graph grammars for multi-paradigm modelling in AToM, *Software and Systems Modeling (SoSyM)*, 3(3):194-209, 2004.

[12] L. Lengyel, T. Levendovszky, G. Mezei, B. Forstner, H. Charaf, Metamodel-Based Model Transformation with Aspect-Oriented Constraints, *International Workshop on Graph and Model Transformation, GraMoT*, Tallinn, Estonia, September 28, 2005.

[13] H. Ehrig, Introduction to the Algebraic Theory of Graph Grammars, *In:Graph Grammars and Their Applications to Computer Science and Biology, Springer*, Ed. V. Claus, H. Ehrig, G. Rozemberg, Berlin, 1979.

[14] L. Lengyel, T. Levendovszky, H. Charaf, Implementing an OCL Compiler for .NET, *In Proceedings of the 3rd International Conference on .NET Technologies*, Pilsen, Czech Republic, May-June 2005, pp. 121-130.