A Visual Control Flow Language and Its Termination Properties

László Lengyel, Tihamér Levendovszky, and Hassan Charaf

***Abstract*—**This paper presents the visual control flow support of Visual Modeling and Transformation System (VMTS), which facilitates composing complex model transformations out of simple transformation steps and executing them. The VMTS Visual Control Flow Language (VCFL) uses stereotyped activity diagrams to specify control flow structures and OCL constraints to choose between different control flow branches. This work discusses the termination properties of VCFL and provides an algorithm to support the termination analysis of VCFL transformations.

***Keywords*—**Control Flow, Metamodel-Based Visual Model Transformation, OCL, Termination Properties, UML.

Open Science Index, Computer and Information Engineering Vol:1, No:8, 2007 publications.waset.org/3337.pdf

I. INTRODUCTION

IN VMTS [1] directed, labeled graphs are used to represent the internal structure of software models, and transformation steps (graph rewriting rules) specify the operational behavior of model processing. The VMTS supports editing metamodels, design models according to their metamodels, transforms models using graph rewriting techniques, and facilitates to check constraints specified in the metamodel during the metamodel instantiation and the transformation step constraints during the model transformation process [1].

VMTS is a UML-based [2] approach for model transformations. The technique is based on graph transformations [3], where UML class diagrams are used to represent the metamodels (graph grammars) of the input and the output of the transformations. The transformations are defined as controlled structure of elementary transformation steps.

Graph rewriting [3] is a powerful tool for graph transformation with a strong mathematical 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. Replacing means removing the elements that are in LHS but not in RHS, and gluing the elements that are in RHS but not in LHS.

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 left-hand-side (LHS) and right-hand-side (RHS) graphs of the rules are built from metamodel elements. It means that an instantiation of LHS must be found in the host graph instead of the isomorphic subgraph of LHS. This metamodel-based approach facilitates to assign OCL constraint to pattern rule nodes (PRNs) – nodes of the rewriting rules.

The Object Constraint Language (OCL) [4] is a formal language for the analysis and design of software systems. It is a subset of the UML standard [2] that allows software developers to write constraints and queries over object models.

The motivation of the work presented in this paper is to support the control flow in visual model transformation systems and to define the conditions exactly which guarantee that if a transformation fulfills them it surely terminates or surely not. An algorithm – VCFL Termination Algorithm (VTA) – is worked out to support the termination analysis of VCFL transformations. The VTA is an offline algorithm, as an input it uses only the control flow model to make the decision. It means that the decision is independent from any host model.

II. 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 rewriting rules. 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 Diagram, which is a technique to describe procedural logic, business process, and work flow. In many ways, it plays a role similar to flowcharts, but the principal difference between it and flowchart notation is that activity diagrams support parallel behavior [5].

In Fig. 1 the control flow model of Prim's algorithm is depicted which implements a greedy-choice strategy for minimum spanning tree. Starting with an empty tree (one

L. Lengyel, T. Levendovszky, H. Charaf, Budapest University of Technology and Economics, Goldmann Gyorgy ter 3, 1111 Budapest, Hungary (e-mails: {lengyel, tihamer, hassan}@aut.bme.hu).

optional vertex with no edges), the algorithm repeatedly adds the lowest-weight edge (u,v) in input graph such that either u or v, but not both, is already connected to the tree. The pseudo code of the algorithm is as follows.

```
PRIMSPANNINGTREEALGORITHM (Graph G)
 1 Select an arbitrary vertex from G to start the tree from
 2 while (there are still non-tree vertices)
 3   Select the edge of minimum weight between a tree and non-tree vertex
 4   Add the selected edge and vertex to the spanningTree
 5 end while
```

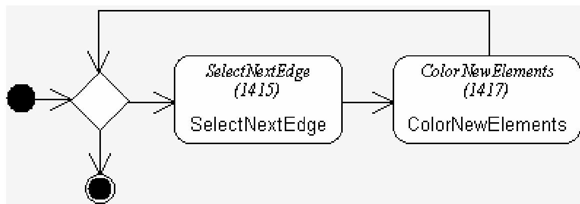

Fig. 1 The VCFL control flow of Prim's spanning tree algorithm

An arbitrary vertex from G to start the tree from is given as a pivot node. A pivot node is an input parameter of the control flow specified by the user. In the graph each vertex and edge has a property (*IsSpanningTreeMember*) which determines if a vertex or an edge has already been added to the spanning tree. In the decision object this property of the vertices is checked. If the graph contains at least one vertex which is not a member of the spanning tree, then the constraints contained by the decision select the path to *SelectNextEdge* rule, otherwise to the rule end.

The first transformation step (*SelectNextEdge*) selects the neighbors of the tree vertices which are not members of the tree yet. It selects the vertex from the neighbor vertices which is connected to the tree with the edge of minimum weight, and sets the *IsSpanningTreeMember* of the selected vertex and edge to true. The *ColorNewElements* step obtains the selected vertex and edge as passed parameter between rules and modifies their color to red (other, more technical, transformation steps can also be defined). As a result of the transformation the minimal spanning tree will be red. The presented transformation does not modify the topology of the model but updates the attribute values.

The VCFL is a visual language for controlled graph rewriting and transformation, which supports the following constructs: sequencing transformation steps, branching with OCL constraints, hierarchical steps, parallel executions of the steps and iteration.

A. Sequencing Transformation Steps

Sequencing transformation steps facilitates a transformation which contains the steps in an ordered sequence ($S_0, S_1 \ldots S_{n-1}$). Assume the case that the input model of the ith step (S_i) is the model M_i and the result of the S_i is the M_{i+1} (where $0 \leq i \leq n-1$). In this case the input model of the $i+1^{th}$ step (S_{i+1}) is the M_{i+1}. It means that during the execution of the step sequence each step works on the result of the previous step. (Obviously, except for the first step, which works on the input model.) The result of the whole transformation is the result of the last step (S_n).

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 as external causality. An external causality creates a linkage between a node contained by the RHS of the ith step and a node contained by the LHS of the $i+1^{th}$ step. This feature accelerates the matching and reduces the complexity, because the ith step provides partial match to the $i+1^{th}$ step. In our example we use external causalities to pass the selected edge and vertex from *SelectNextEdge* rule to *ColorNewElements* rule.

B. Branching with OCL Constraints

There are many scenarios where the transformation is to be applied, it depends on a condition, therefore a branching construct is required. In VCFL the OCL constraints assigned to the decision elements can choose between branch paths of optional numbers, based on the properties of the actual host model and the success of the last transformation step (*SystemLastRuleSucceed*). If the last transformation step fails, then the VCFL could use the values of the *SystemLHSFailure* and *SystemRHSFailure* system variables for the decision. These variables represent whether a failure has occurred, because there was no proper match (LHS failure: topologically not suitable host model or there is at least one constraint not satisfied in the LHS of the transformation step), or the transformation result was not sufficient (RHS failure: there was at least one constraint not satisfied in the RHS of the transformation step).

In VCFL each branch has an exact OCL guard condition which is evaluated by the execution engine during the execution.

When a step is connected to more than one follow-up steps, then maximum one of the branch conditions is allowed to be true. It means that the conditions must not have any common part. This restriction ensures that the control flow execution of the VCFL is deterministic.

We applied VCFL in several case studies (e.g. generate user interface from resource model and user interface handler code from statechart model for mobile platform [6]) which require control flow support, and all of them could be solved without non-determinism. But VCFL provides an interface for nondeterministic control flow as well.

C. 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 rule can contain several simple rules, hiding the details which could be unimportant on a specific abstraction level and represents the contained rules as coherent units.

Often the OCL constraints assigned to a decision object do

Open Science Index, Computer and Information Engineering Vol:1, No:8, 2007 publications.waset.org/3337.pdf

not cover all possible cases. It could result that in certain cases none of the branch paths is selected, in this case the parent step of the actual transformation handles the control flow: breaks the execution of the transformation on the actual level and continues the transformation on the parent level.

D. 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 (Fig. 1).

Recursion could be solved with the combination of the iteration and external causalities. A high level rule can call itself, where external causalities represent the actual parameters of the recursive call.

Flattening the state machine is an example when we have to apply recursive algorithm that first calls flattening on its children before flattening itself.

The parallel execution of the independent transformation steps is under implementation, it will 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 then 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.

III. Termination Properties

The termination properties of a transformation need to be discussed. The difference between a transformation and a finite sequence of steps is that a finite sequence of steps always terminates, but a transformation, can contain infinite number of steps. Our aim is that VCFL transformations terminate, therefore an algorithm has been worked out to support the early detection of the infinite loop 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*. Recall that applying a graph rewriting rule means finding a match of the LHS in the host graph and replacing this subgraph with the RHS. An *exhaustive* transformation step is executed continuously as long as the LHS of the step could be matched to the host model. The *MultipleMatch* attribute of a rule allows that the matching process finds not only one but all occurrence of the 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 $S_0, S_1 \ldots S_{n-1} \in STEPS \in T$, where S_0 is the start step of the *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 of the VCFL control flows for optional finite input model G_0.

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

A. VCFL Control Flows with Non-Exhaustive Transformation Steps

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

Proof: The transformation *T* contains finite number of transformation steps ($n = \#STEPS \wedge n < \infty$). $\forall i \mid 0 \le i \le n-1$ $S_i \in STEPS$ 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} is searched and the replacing is executed for all found matches, but the 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 $k_i = \sum_{i=0}^{n-1} m_i < \infty$. The number of the steps executed by transformation *T* is finite and *T* terminates.

B. VCFL Control Flows with Exhaustive Transformation Steps

Definition: $(\subseteq)$. $G_m \subseteq G_n$ if and only if G_n has a topologically 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 meta-type, attributes, attribute values and OCL constraints.

An exhaustively applied rule using external causalities gives itself input model and parameters. For an exhaustive rule the algorithm has to take into consideration the attribute modifications and the generated and deleted elements. 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 be also if there is a create type causality and an OCL constraint which holds before the creation and become false afterwards, therefore it prevents to find the same match again on the same place. For example an OCL constraint can

Open Science Index, Computer and Information Engineering Vol:1, No:8, 2007 publications.waset.org/3337.pdf

validate the existence of a neighbor node. In Fig. 2 the presented transformation step connects a married and unemployed man to a company. The unemployed property is checked by the *const_employer* constraint. After the execution of the step in the next iteration the matching process does not match the same pattern again because of the not satisfied constraint.

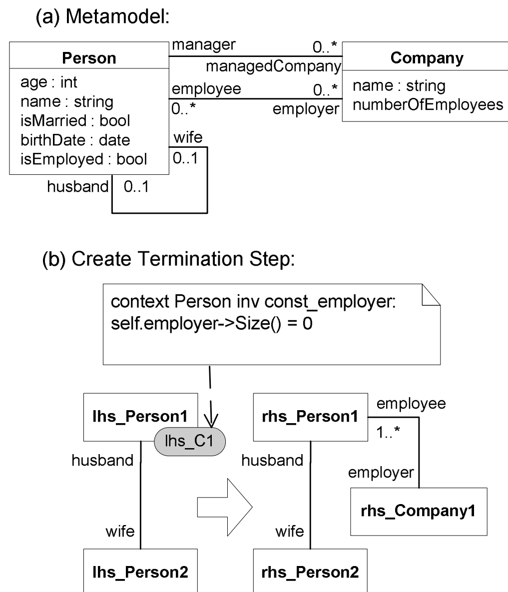

Fig. 2 An example metamodel and a Create Termination Step

Definition (Create Termination Step – CT Step): A *Create Termination Step S* has only create type internal causalities, it contains an optional OCL constraint C_1 in S^{LHS}, which must to 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_1 becomes false.

Definition (Create Termination Step with constraint C_2 – CT Step with C_2): A *Create Termination Step S* has only create type internal causalities, it contains the OCL constraint C_2 in S^{LHS}, which must to 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_2 becomes false.

The difference between a *CT Step* and a *CT Step with C_2* is that in first case the step can have optional number of constraints and an arbitrary one of them has to fulfill the condition, while in the second case the given constraint (C_2) has to comply it.

Obviously, this rule property is important only for exhaustive rules or rules which are in loops, because the creation can prevent to find the same match again on the same place and it helps to avoid infinite loops.

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 optional 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 the S_i^{RHS}. $S_i^{LHS} \subseteq S_i^{RHS}$, therefore the 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 deletion and modification type internal causalities and S_i is not a CT step. Assume that T is transformation and $S_i \in T$, the input model of the 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 rule – 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.

C. Combining VCFL Transformation Steps

The goal of the transformation step combination is to create a single step S_C from optional number of transformation steps $S_j, S_{j+1} ... S_k$. The combined step can equivalently replace the original steps, because it produces the same result. 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 it has a decision object.

The combination algorithm takes not only the topology of the steps into consideration but also their internal- and external causalities and the meta-types of the nodes and edges as well. The algorithm works based on the double pushout (DPO) approach [7] [8].

An example for transformation step combination is depicted in Fig. 3.

D. 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.

The main difference between a loop with only non-exhaustive rules and an exhaustive rule is the exit condition. A transformation leaves an exhaustive rule if there is no more match, while in the case of a loop the decision object determines about the exit. If a loop consists of non-exhaustive rules the rule combination algorithm combines them and makes the decision about the termination based on the

Open Science Index, Computer and Information Engineering Vol:1, No:8, 2007 publications.waset.org/3337.pdf

combined rule and the OCL constraints of the decision object.

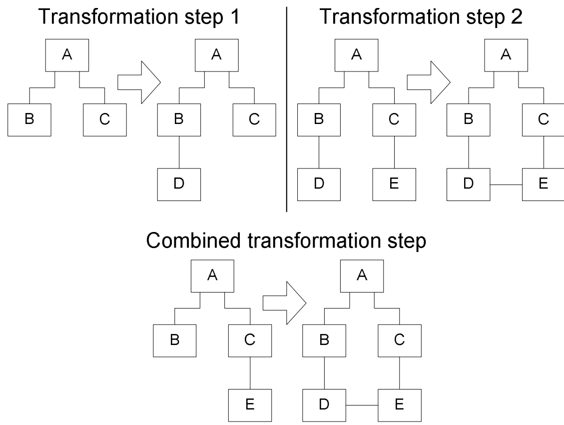

Fig. 3 An example for transformation step combination

An exhaustive rule is itself a specific loop, therefore if a loop contains exhaustive rules then it is a loop of loops. The algorithm examines separately the exhaustive rules and if each of them terminates then analyses the whole loop.

Proposition: Assume that the transformation T contains a loop L, let S_C be the combination of the transformation steps $S_j, S_{j+1} \ldots S_k \in L$. 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.

E. VCFL Termination Algorithm

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 the 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 the LHS of the *combinedStep* **then return** retValue.false
9 **end foreach**
10 **return** retValue.undecided

For an optional 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 $S_h, S_{h+1} \ldots S_k \in L$ and $S_C^{LHS} \subseteq S_C^{RHS}$ the transformation T does not terminate.

If the rule 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 into consideration the topology, node and edge types and, the attributes, the attribute values and also the propagated OCL constraints.

During the combination of steps S_1 and S_2, the S_1^{RHS} and the S_2^{LHS} could have more than one matching variation. The algorithm checks all the possible variations in point of VCFL view (external causalities, meta-types).

In the case of loops the exit conditions (topology, attribute value by modify internal causalities and *SystemLastRuleSucceed*) are also checked by the algorithm.

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 could 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 sure decision based on the VCFL transformation, and there are other cases when not.

IV. RELATED WORK

Many approaches have been introduced in the field of graph grammars and transformations to capture graph domains; for instance, the GReAT [9] [10], the PROGRES [11] [12], the Fujaba [13] [14] and the VIATRA [15]. 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 sequencing of the rewriting rules, parameter passing (external causalities) and the recursion are similar in GReAT and in VCFL.

GReAT distinguishes primitive and test rules. The primitive rules of the language are to express the steps of the transformations. A test rule is a special expression and it is used to change the control flow during execution. A test rule has only LHS. If a test rule is successful (the matching was successful), the rule after the test node is executable.

Open Science Index, Computer and Information Engineering Vol:1, No:8, 2007 publications.waset.org/3337.pdf

Open Science Index, Computer and Information Engineering Vol:1, No:8, 2007 publications.waset.org/3337.pdf

PROGRES is a visual programming language in the sense that it has a graph-oriented data model and a graphical syntax for its most important language constructs. The control structure of the PROGRES has the atomic, the boolean and the non-deterministic character.

Similarly to GReAT, PROGRES also has test rules, which have only LHS graphs and test the result of the step but do not modify the host graph.

In FUJABA the combination of activity diagrams and collaboration diagrams (story-diagrams) are used to express control structures. Story-diagrams are a visual programming language that facilitates the specification of complex application-specific object structures. Moreover, FUJABA extended story-diagrams by statecharts to so-called story-charts. Story-charts use statecharts and activity diagrams to define complex control flows and collaboration diagrams to specify the entry, exit, do, and transition actions that deal with complex object-structures [14].

VIATRA uses abstract state machines to define the control flow of the system.

In [16] a termination criteria for model transformation is presented. The criteria is based on dividing the grammar in deleting or non-deleting layers. The introduced principles offer visual and formal techniques based on rules, in such a way that model transformations can be subject to analysis.

In [17] a contribution towards solving the termination problem for rewriting systems with external control mechanisms is given. It extends the concept of transformation unit to high-level replacement systems. For high-level replacement units, several abstract properties based on termination criteria are stated and proved.

V. Conclusion

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

As it was presented, a control structure language needs a sequence as well as a conditional branch mechanism, hierarchy, parallel executions and iteration constructs. VCFL has all these control structures in a deterministic implementation.

Termination is an important issue for model transformations. In this work we discussed the termination properties of the VMTS Visual Control Flow Language. We stated and proved several termination criteria for transformation steps, loops and transformations. An algorithm to validate the termination is also provided.

Acknowledgment

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] UML 2.0 Specifications, http://www.omg.org/uml/

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

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

[5] M. Fowler, UML Distilled, “A Brief Guide to the Standard Object Modeling Language”, 3rd edition, Addison-Wesley, ISBN: 0321193687, 2003.

[6] 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, to be published.

[7] 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.

[8] H. Ehrig, M. Korff, M. Löwe, “Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts”. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science, volume 532 of Lecture Notes in Computer Science, pages 24-37. Springer Verlag, 1991.

[9] 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, Special issue on Formal Specification of CBS, 2003.

[10] A. Agrawal, “A Formal Graph-Transformation Based Language for Model-to-Model Transformations”, PhD Dissertation, Vanderbilt University, Dept of EECS, August, 2004.

[11] A. Schürr, “PROGRES for Beginners”, Technical Report, Lehrstuhl für Informatik III, RWTH Aachen, Germany

[12] 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, Springer Verlag (1992), pp. 48-62, also: Technical Report AIB 91-17, RWTH Germany, 1991.

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

[14] Hans J. Köhler, Ulrich A. Nickel, Jörg Niere, Albert Zündorf, “Integrating UML Diagrams for Production Control Systems”, Proc. of the 22nd International Conf. on Software Engineering (ICSE) Limerick Ireland, ACM Press, 2000, pp. 241-251.

[15] 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.

[16] Hartmut Ehrig, Karsten Ehrig, Juan de Lara, Gabriele Taentzer, Dániel Varró and Szilvia Varró-Gyapay, “Termination Criteria for Model Transformation”, LNCS, Vol. 3442: Fundamental Approaches to Software Engineering: 8th International Conference, FASE 2005, Edinburgh, UK, April 4-8, 2005, pages 49-63. Springer-Verlag, 2005.

[17] Paolo Bottoni, Manuel Koch, Francesco Parisi-Presicce, Gabriele Taentzer, “Termination of High-Level Replacement Units with Application to Model Transformation”, Electr. Notes Theor. Comput. Sci. 127(4): 71-86, 2005.