Electronic Communications of the EASST
Volume 2 (2006)

Proceedings of the Workshop on Petri Nets and Graph Transformation (PNGT 2006)

Optimization in Graph Transformation Systems Using Petri Net Based Techniques

Szilvia Varró-Gyapay and Dániel Varró

18 pages

Guest Editors: Paolo Baldan, Hartmut Ehrig, Julia Padberg, Grzegorz Rozenberg
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Optimization in Graph Transformation Systems Using Petri Net Based Techniques

Szilvia Varró-Gyapay[1] **and Dániel Varró**[2]

[1] gyapay@mit.bme.hu
[2] varro@mit.bme.hu
Budapest University of Technology and Economics
Department of Measurement and Information Systems

Abstract: The design of business or production systems frequently necessitates to simultaneously fulfill several *logical and numerical constraints* as requirements in order to deliver a functionally correct and optimal system. Such a problem can be typically formulated as a combined optimization and reachability analysis. In the current paper, we show how this problem can be formalized when the evolution of the system is captured by graph transformation systems (GTS) with a cost parameter attached to each graph transformation rule denoting the cost of firing the rule. Furthermore, we discuss how to solve such problems by combining guided state space exploration with algebraic techniques of Petri nets .

Keywords: graph transformation, Petri nets, optimization, verification

1 Introduction

As the quality of service delivered by business systems becomes more and more crucial to information systems, their correct and efficient operation has to be proved already during the design phase. This way, we need to analyze whether the system simultaneously fulfills logical and numerical conditions. Verification and validation techniques based on formal methods are known to assure the correctness of the services. Optimization estimates the quantitative boundaries and characteristics of a system in order to minimize operation time or costs. However, it is a challenging question how to combine the best practices of the two fields.

For instance, in workflows, each activity can be constrained by certain budget restrictions, thus a typical requirement is to find the cost-optimal solution trajectory respecting all budget and temporal correctness constraints along this path that leads to a desirable (target) system configuration. This can be interpreted as a reachability problem with quantitative or qualitative measures.

Graph transformation provides a rule and pattern-based manipulation of graph models to specify the formal semantics of systems in various application fields. Recent successes of the application of graph transformation techniques are related to model-driven development (MDD) and service oriented architecture (SOA).

The current paper proposes (i) a graph transformation based approach that captures the evolution of the system by rules with a cost parameter attached to each graph transformation rule denoting the cost of firing that rule, (ii) and a solution to the joint reachability and optimization problem by combining guided state space exploration with algebraic techniques of Petri nets.

The rest of the paper is organized as follows. Section 2 sketches the outline of our solution. Section 3 provides an overview on graph transformation systems and Place / Transition (P/T) nets and presents a running example where a service system is specified. Section 5 gives a more detailed definition of the problem and our solution. Finally, Section 6 discusses related work, presents our conclusions and proposals for future work.

2 Outline of the solution

Our proposal for solving joint optimization and reachability analysis for graph transformation systems is summarized in Fig. 1.

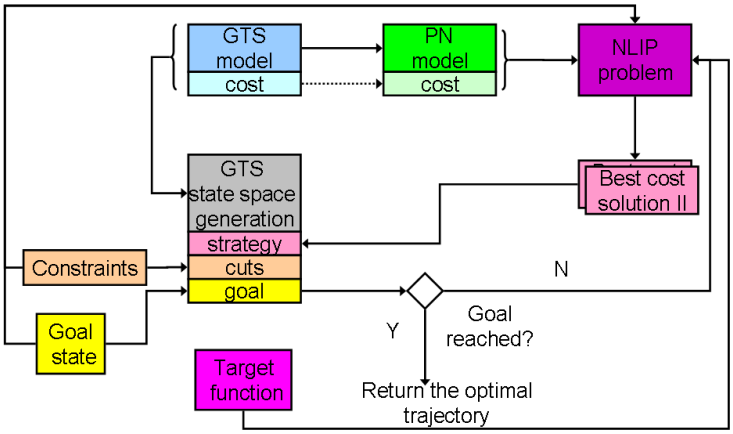

Figure 1: Approach

The base of our solution is to explore the state space (also called a graph transition system) induced by the graph transformation system as in case of model checking. However, this traversal of the state space is restricted by various logical and numerical constraints.

- If a logical constraint is violated, then the corresponding path is cut off permanently (i.e. subsequent states are no longer investigated), as this path cannot provide a valid solution.
- If some numerical constraint is violated, we act as before by terminating the search along this path. In addition to user-defined numerical constraints, we keep track of the best cost solution (so far). If the cost of a solution along a certain path exceeds this best cost, we cut again the corresponding path.
- If a target state is reached (i.e. the constraint specifying the designated target state is satisfied), we terminate our search along the specific path.

While these cuts provide efficient techniques to prune the state space later during the search, it is usually critical to find a good solution as soon as possible. Generic search strategies used in model checking like breadth-first search or depth first search are unable to provide efficient help.

Therefore, we introduce temporal numerical cuts to guide the state space exploration by temporally pruning the search tree to postpone the unpromising paths.

These cuts are based on a Petri net abstraction of graph transformation systems introduced in [VVE+06]. By formulating the target state configuration as submarking of the P/T net, we can solve the integer linear programming problem of the derived P/T net using the incidence matrix to determine a transition occurrence vector which delivers a cost-optimal solution.

Obviously, due to the abstractions, this transition occurrence vector might not be fireable, i.e. there might not exist a corresponding trajectory in the graph transition system. However, this vector provides excellent guidance for the state exploration strategy. Those branches which are not compliant with the optimal transition occurrence vector are temporally cut, i.e. compliant steps are explored first when traversing the state space.

This means that if this minimal cost is exceeded along a path or a graph transformation (GT) rule is applied more than it is prescribed in the transition occurrence vector, then the exploration of the branch is postponed. If no solution is found on the level of graph transformation systems (GTS), then the next optimal transition occurrence vector candidate is derived, and the exploration of the GTS continues towards this direction.

3 Definitions

In this section the basics of graph transformation, P/T nets and ILP problems are shortly discussed. Before the definition, an example is introduced in order to motivate the problem (that is revisited several times through the paper).

3.1 Example

As a motivating example, let us assume a reliable service which is composed of individual services. The state of a service can be *up*, *down*, *active* or *standby*.

- When a service is *active*, clients may issue request to it.
- A *standby* service does not accept requests, but it serves as a *backup* for another active service.
- When a service is *up*, it is operational, but neither *active* nor *standby* role is assigned to it. Typically, a service is in an *up* state right after repairs have completed.
- Finally, when a service is *down*, it is not operating due to some errors until a restart has initiated.

A certain *quality of service (QoS)* is required such as throughput or availability, e.g. at least 3 services have to be up at the same time in order to provide sufficient performance when serving requests. Such a system is shown in Fig. 2.

To satisfy this constraint the service configuration has to be designed in an appropriate way. We assume that regular healthchecks are issued by some middleware service broker. If the current health state of services implies that the required QoS parameters cannot be satisfied by the actual service configuration, reconfiguration operations are to be initiated which lead the system

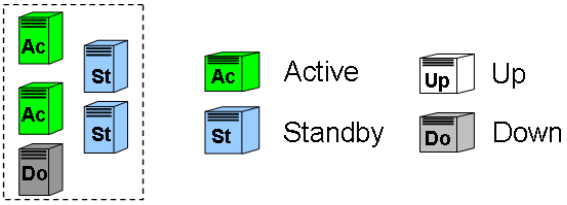

Figure 2: An example system providing reliable service

into a state where all QoS constraints are met. However, these operations have costs that have to be taken into consideration, i.e. we need to find a reconfiguration plan with minimal cost.

The reconfiguration actions of services will be captured by a graph transformation system that is defined subsequently. An overview on using graph transformations for software architecture reconfigurations can be found in [BHTV06].

3.2 Graph transformation

A graph $G = (N, E, src, trg)$ is a 4-tuple with a set N of nodes, a set E of edges, a source and a target function $src, trg : E \to N$. A type graph TG is an ordinary graph. An instance graph G is typed over TG by a typing morphism $type : G \to TG$. Let $card(G, x)$ denote the cardinality (i.e. the number of graph objects) of type $x \in TG$ in graph G. Formally, $card(G, x) = |\{n \mid n \in N \cup E \wedge type(n) = x\}|$.

An example type graph is shown in Fig. 3. The type graph contains only one *Service* node designated graphically as a rectangle. The edges *active, standby, down,* and *up* are used to denote the state of the service such that the source and the target node of this edge is the same node. Edge *backup* connect two different services: when an *active* node goes *down* a *standby* node has to substitute the *active* service.

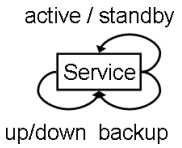

Figure 3: Type graph

Graph transformation. Graph transformation (GT) [CMR+97] provides a rule-based manipulation of graph models. A graph transformation (GT) rule typed over a type graph TG is given by $r = (L \xleftarrow{l} K \xrightarrow{r} R)$ where L (left-hand side), K (context) and R (right-hand side) graphs are typed over TG and graph morphisms l, r are injective. The negative application conditions (NAC) of a GT rule is a (potentially empty) set of pairs (N, n) with N being a graph also typed over TG and $n : L \to N$ an injective graph morphism.

Application of a rule. The *application* of a rule $r = (L \xleftarrow{l} K \xrightarrow{r} R)$ to a *host graph* G alters the model graph by replacing the pattern defined by L with the pattern of the R. This is performed by

1. *finding a match* of the L pattern in model G;
2. *checking the negative application conditions* N which may prohibit rule application, i.e. if there is a match of N in G (as an extension of the match of L in G), then the rule is not applicable;
3. *removing* a part of the model M that can be mapped to the L pattern but not the R pattern yielding an intermediate graph D;
4. *adding* new elements to the intermediate graph D which exist in the R but not in L yielding the derived graph H.

A *GT step* is denoted formally as $G \overset{r,o}{\Longrightarrow} H$, where $o : L \to G$ defines the match of the elements in L to G.

A *graph transformation sequence (GT sequence)* is a sequence of GT steps, i.e., a sequence of rule applications: $G_0 \overset{r_1}{\Longrightarrow} G_1 \overset{r_2}{\Longrightarrow} G_2 \overset{.}{\Longrightarrow} \ldots$ A GT sequence starting from graph G yielding G' is denoted shortly by $G \overset{*}{\Longrightarrow} G'$ where $*$ denotes that more than one GT step may belong to the GT sequence. In the paper, we follow the *Double Pushout Approach* [CMR+97].

Example 1 *The ongoing example is captured by a set of graph transformation rules in Fig. 4. In order to simplify the graphical presentation, we simply write the state* active, standby, up, down *of the service on the service node (which is denoted by a server symbol) instead of self-loop edges. This way, only* backup *edges remain, we also omit the* backup *labels from the edges, i.e. all edges in the graphical representation are* backup *edges.*

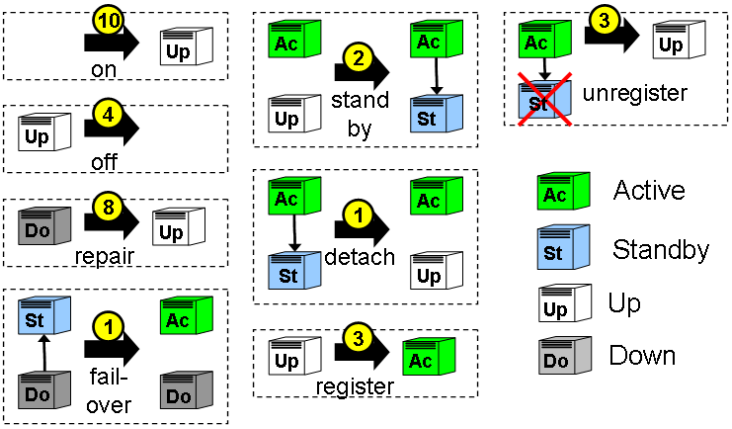

Figure 4: Rules

1. on *adds a new service to the configuration that is initially* up.

2. off *removes an* up *service from the configuration.*

3. repair *makes a* down *service become* up.

4. failover *assumes that there is a* down *service connected by a* backup *edge to a* standby *service: the effect of the rule is that the* standby *service becomes* active *and the* backup *edge is deleted.*

5. standby *creates a* backup *edge between an* active *and an* up *service such that the* up *service becomes* standby.

6. detach *removes the* backup *edge between an* active *and a* standby *service such that the* standby service *will be* up.

7. register *changes the state of service from* up *to* active.

8. *Finally,* unregister *changes the state of a service from* active *to* up *in case there is no* standby *service connected to the service.*

Graph transformation systems with cost. A *graph transformation system* $GTS = (R, TG)$ consists of a type graph TG and a finite set of graph transformation rules typed over TG.

A *graph transformation system with cost* $GTS_c = (R, TG, c : R \to \mathbb{R}^+ \cup \{0\})$ is a GTS where a cost parameter $c(r_i)$ is added to each GT rule denoting the cost of firing that rule. Graphically, the cost of a rule is denoted by a circled number over the arrow of the rule. A *graph grammar with cost* $GG_c = (GTS_c, G_0)$ consists of a graph transformation system $GTS = (R, TG, c)$ and a so-called *start (model) graph* G_0 typed over TG.

The cost of a GT sequence is equal to the sum of the cost of the contained GT steps, i.e. the sum of the cost of the applied rules.

Example 2 *The cost of the rules are* $10, 4, 8, 1, 2, 1, 3, 3$ *units, respectively, denoted by circled numbers on the GT rule arrow in Fig. 4.*

An example start graph G_0 *is shown in Fig. 5 on the left: the system configuration contains two* active, *two* standby, *and one* down *services.*

State space of a graph grammar. The *graph transition system* (state space) of a graph grammar $GG = (GTS, G_0)$ is defined as a graph where nodes are instance graphs, and edges are graph transformation steps $G \overset{r,c}{\Longrightarrow} H$ such that the source and target nodes of the edge are graphs G and H, respectively. Starting from G_0 the *state space* (i.e. the reachable instance graphs) of the GG is represented taking into account all applicable rules from a given host graph.

A path in the graph transition system of a GG is a GT sequence denoted by $p = (G_0 \overset{r_1,c_1}{\Longrightarrow} G_1 \overset{r_2,c_2}{\Longrightarrow} \ldots \overset{r_{n-1},c_{n-1}}{\Longrightarrow} G_n)$ and it is called also as a trajectory between two graphs. Then we say that a graph G is reachable from G_0 iff there is a path in the GTS. The cost of the path denoted by $c(p)$ is equal to the sum of the cost of the rules applied in the trajectory, i.e. $c(p = (G_0 \overset{r_1,c_1}{\Longrightarrow} G_1 \Longrightarrow G_n)) = \sum c(r_i)$.

Example 3 *In Fig. 5 an extract of the graph transition system of our running example is shown. On the left the root of the graph transition system is the start graph G_0 where the system configuration contains two* active, *two* standby, *and one* down *services. Rules* failover, on, repair, detach, *and* unregister *are applicable to G_0, here we follow only the application of the first three rules. The cost of the individual paths (starting from G_0 to the current graph) is shown on the right of the graph.*

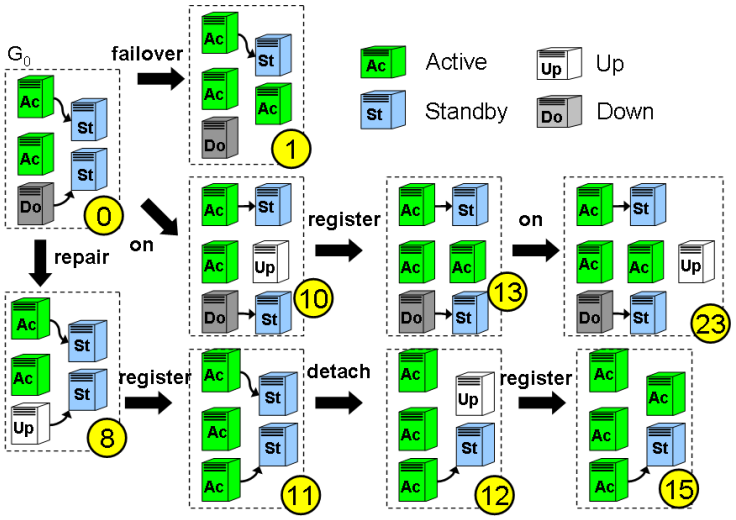

Figure 5: A part of the graph transition system

3.3 Place/Transition nets

In the current section we give a short introduction into the theory of Place/Transition nets based on [Mur89].

A Place/Transition net (or shortly P/T net) is a 5-tuple $PN = (P, T, E, w, M_0)$ where P is a set of places, T is a set of transitions, $E \subseteq (P \times T) \cup (T \times P)$ is the set of arcs (where no arc connects two places or two transitions), $M_0 : P \to \mathbb{N}$ is the initial marking mapping places to nonnegative integers, while $w : E \to \mathbb{N}^+$ maps arcs to positive integers.

Furthermore $\bullet t = \{p \mid (p,t) \in E\}$ denotes the input places, while $t\bullet = \{p \mid (t,p) \in E\}$ denotes the output places of transition t. Finally, $\bullet p = \{t \mid (t,p) \in E\}$ are the incoming transitions while $p\bullet = \{t \mid (p,t) \in E\}$ are the outgoing transitions of place p.

A transition t is enabled if each of its input places contains at least as many tokens as is specified by the weight function, formally, $\forall p \in \bullet t : M(p) \geq w(p,t)$. The firing of an enabled transition t removes a $w(p,t)$ amount of tokens from the input places (p), and $w(t,p)$ tokens are produced for the output places, i.e. $\forall p \in P : M'(p) = M(p) - w(p,t) + w(t,p)$.

The incidence matrix W of the net describes the net token flow (of the P/T net) when firing of a transition. Mathematically, W is a $|P| \times |T|$–dimensional matrix of nonnegative integers $\mathbb{N}$ such that $W_{ij} = w(t_i, p_j) - w(p_j, t_i)$, where $1 \leq i \leq |P|, 1 \leq j \leq |T|$. After firing a transition t from marking M, the result marking M' can be computed by using the incidence matrix: $M' =$

$M + W \cdot e_t$, where e_t is a $|T|$-dimensional unit vector.

A *transition firing sequence* (or shortly firing sequence) $s = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_k} \rangle$ is a sequence of transition firings between states M_0 and M_k such that $\langle M_0, t_{i_1}, M_1, t_{i_2}, \ldots, t_{i_k}, M_k \rangle$ where for all $1 \leq j \leq k$ t_{i_j} is enabled in M_{j-1} and M_j is yielded by the firing of t_{i_j} in M_{j-1}. The set of the reachable states from a marking M_0 in a Petri net PN is denoted by $Reach(PN, M_0)$ and can be represented by the so-called *reachability graph*. The *transition occurrence vector* or *Parikh–vector* σ of a trajectory $s = t_{i_1}, \ldots, t_{i_k}$ counts the occurrence number of the individual transitions in the firing sequence, i.e. $\sigma(t_j) = |i_l = j, l = 1..k|$.

A marking M is *reachable* from a state M_0 (denoted by $M_0 \stackrel{s}{\Longrightarrow} M$) if there is a transition firing sequence s from M_0 to M. Then the so–called state equation holds: $M = M_0 + W \cdot \sigma$, where σ is the transition occurrence vector of s. A marking M_{part} is *partially reachable* or *coverable* from a state M_0 if there is a transition firing sequence s from M_0 to a marking M such that $M_{part} \leq M$. Then the following inequality holds: $M_{part} \leq M_0 + W \cdot \sigma$, where σ is the *transition occurrence vector* s.

A *Petri net with cost parameters* is a $PN_c = \langle PN, c \rangle$, where the cost function $c : T \to \mathbb{R}^+ \cup \{0\}$ assigns costs to the firing of the individual transitions. Thus, *the cost of the firing sequence* can be interpreted as the sum of the cost values of the transitions in the sequence, formally, $c(s) = \sum_{t \in T} c(t)\sigma_s(t)$, where σ_s is the transition occurrence vector of the firing sequence s, and $c(s)$ denotes the cost of the firing sequence.

3.4 Integer linear programming (ILP) problems

The standard form of a linear programming (LP) problem is given as a matrix form as follows.

maximize $c^T \cdot x$,

subject to $A \cdot x \leq b$, $x \geq 0$,

where the first row gives the linear objective function while linear constraints are formulated as an inequality system. In addition, x is the vector of variables, c is the vector of cost parameters assigned to the variables, and the elements of matrix A are the paramaters of the constraints.

In the following a simple example is given.

- *linear objective function*:

  maximize $c_1 x_1 + c_2 x_2$,

- *linear constraints*:

  subject to

  $a_{11} x_1 + a_{12} x_2 \leq b_1$,

  $a_{21} x_1 + a_{22} x_2 \leq b_2$,

  $a_{31} x_1 + a_{32} x_2 \leq b_3$

- *nonnegative variables*:

  $x_1 \geq 0$, $x_2 \geq 0$.

Other forms, such as minimization problems, problems involving negative variables, etc. can always be rewritten into an equivalent problem in standard form. An *integer linear programming problem* is *integer* if all variables are restricted to be integer.

4 A Petri net abstraction of GTS

Our optimization approach is based on a Petri net abstraction technique recently introduced for GTS in [VVE+06]. Our motivation behind such an abstraction is that solving the optimization problem on the P/T level is of much lower complexity than solving the problem directly on the GTS-level using algebraic optimization techniques.

The essence of this abstraction technique is to derive a cardinality P/T net which simulates the original GTS by abstracting from the structure of instance graphs and only counting the number of elements (nodes or edges) of a certain type by placing tokens to a corresponding place. These tokens are circulated by transitions derived from each GT rule which simulate the effect of the rule on the number of elements of certain types by adding and removing tokens from corresponding places. The original technique is extended now to cope with costs assigned to a GT rule by simply copying them to the corresponding transition.

This mapping $\mathscr{F}()$ is defined as follows.

- *Types into places.* For each node and edge $y \in N_{TG} \cup E_{TG}$ in the type graph TG, a corresponding place $p_y = \mathscr{F}(y)$ is defined in the cardinality P/T net.
- *Instances into tokens.* For each node and edge $x \in N_G \cup E_G$ in an instance graph G with type $y = type(x)$, a token is generated in the corresponding marking $M_G = \mathscr{F}(G)$ of the target P/T net. Formally, for all places $p_y = \mathscr{F}(y)$, the marking of the net is defined as $M_G(p_y) = card(G,y)$.
- *Rules into transitions.* For each rule r in the graph transformation system GTS, a transition $t_r = \mathscr{F}(r)$ is generated in the cardinality P/T graph such that
  - *Left-hand side:* If there is a graph object x of type $y = type(x)$ in the L, then an incoming arc (p_y, t_r) is generated in the P/T net where $p_y = \mathscr{F}(y)$ and the weight of the arc $w(p_y, t_r)$ is equal to the number of graph objects in L of the same type y. Formally, $\forall x,y : x \in L \wedge y = type(x) \wedge \mathscr{F}(y) = p_y \implies (p_y, t_r) \in E \wedge w(p_y, t_r) = card(L,y)$.
  - *Right-hand side:* If there is a graph object x of type $y = type(x)$ in R, then an outgoing arc (t_r, p_y) is generated in the P/T net where $p_y = \mathscr{F}(y)$ and the weight of the arc $w(t_r, p_y)$ is equal to the number of graph objects in R of the same type y. Formally, $\forall x,y : x \in R \wedge y = type(x) \wedge \mathscr{F}(y) = p_y \implies (t_r, p_y) \in E \wedge w(t_r, p_y) = card(R,y)$.
  - *Cost of a rule.* For each rule r the cost of the corresponding transition is equal to the cost of the rule, i.e. $c(\mathscr{F}(r)) = c(r)$.

Example 4 *In Fig. 6 rule* failover *of our example in Section 3 is shown on the left with the corresponding type graph. The P/T net abstraction is shown on the right. According to the type*

graph of the example, the corresponding cardinality P/T net has a place for all node types, namely for type Service, *and edge types, namely* backup, standby, down, active, *and* up.

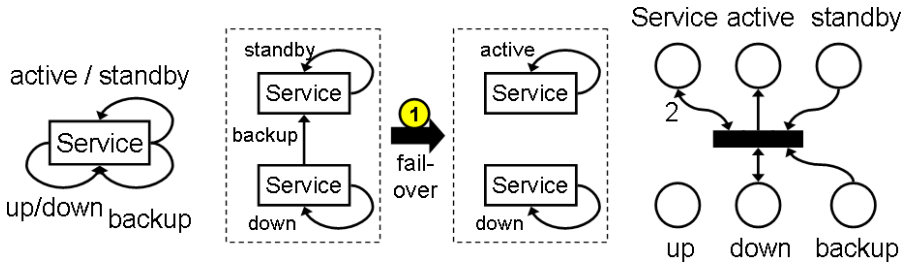

Figure 6: Rule *failover* and the corresponding cardinality P/T net

For instance, the left–hand side L of rule failover *contains two services and edges* backup, standby *and* down. *Thus the corresponding transition with the same name has four incoming arcs starting from the corresponding places, where arc* $(Service, failover)$ *has weight* 2 *since* 2 *services are present in L. Similarly, the right–hand side of the rule consists of two services and edges* active, *and* down *thus there are three outgoing arcs to places* Service, active, *and* down *with weights* $2, 1, 1$, *respectively.*

In this way whenever rule failover *is applied the number of the tokens at the involved places changes according to the cardinality of the graph types.*

The incidence matrix of the P/T net abstraction of the example GTS is in Fig. 7. The places (columns) refer to the type places corresponding to the type graph of Fig. 6, while transitions (rows) refer to corresponding rules of Fig. 4.

Note that it was proved that the mapping $\mathscr{F}()$ is a proper abstraction in the sense that the derived P/T net simulates the original GTS as shown in [VVE+06]. In other terms, whenever a rewriting step is executed in the GTS on an instance graph, then the corresponding transition can always be fired in the corresponding marking in the P/T net, furthermore, the result marking is an abstraction of the result graph. In this respect, for all firing sequence in the GTS there is a firing sequence in the cardinality P/T net but not the other way around. Moreover, the cost of the two sequences are equal.

A possible abstraction of NACs into cardinality P/T nets is also discussed in [VVE+06]. However, that abstraction would deliver an integer non-linear programming problem for the optimal trajectory problem for which solution techniqes have greater complexity than solution techniqes for an (I)LP problem. Thus we ignore the abstraction of NACs in the current paper.

Since calculating a cost optimal solution on the P/T net level is relatively cheap, this solution can then be used as a hint when exploring the state space of the original GTS.

5 Optimization of GTS by Guided State Space Traversal

Now we provide a detailed description how to carry out optimization of graph transformation systems by using a state space traversal strategy guided by optimal solutions of the P/T net abstraction of the GTS.

W =

	Se	Ac	St	Up	Do	Ba
On	1	0	0	1	0	0
Off	-1	0	0	-1	0	0
Repair	0	1	0	0	-1	0
Failover	0	1	-1	0	0	-1
Standby	0	0	1	-1	0	1
Detach	0	0	-1	1	0	-1
Register	0	1	0	-1	0	0
Unregister	0	-1	0	1	0	0

Figure 7: Incidence matrix of the P/T net abstraction

5.1 Optimal Trajectory Problem for GTS

Combined reachability and optimization problems can be defined as follows: (i) decide, whether a particular state of the system is reachable from the initial state using the available resources, and (ii) if the state is reachable, an optimal trajectory has to be computed. Hence, the problem of finding an optimal trajectory between the initial and some desired target states can be translated into an *optimal trajectory (OT) problem*.

Frequently, in engineering problems only a subset of nodes and edges is relevant from practical point of view. Therefore, in case of graph transformation systems 'partial reachability' means that we should reach a graph G that *covers* the desired (partial) graph $G_{partial}$, i.e. there is a subgraph of graph G that is isomorphic to $G_{partial}$ (denoted by $G \supseteq G_{partial}$). Then we also say that $G_{partial}$ is partially reachable from G_0, which is denoted by $G_0 \stackrel{*,partial}{\Longrightarrow} G_{partial}$.

If additional numerical constraints can be given, e.g., a limited cost budget, the optimal trajectory problem can be described as follows. Given a graph grammar $GG = (GTS, G_0)$, a graph $G_{partial}$, and a set of constraints $Const$, find a trajectory (path) tr from graph G_0 to graph G ($G_0 \stackrel{p}{\Longrightarrow} G$) such that $G \supseteq G_{partial}$, and it is optimal, i.e. $\forall tr' : G_0 \stackrel{tr',partial}{\Longrightarrow} G_{partial} : c(tr) \leq c(tr')$, and it satisfies all the requirements in $Const$. We denote this problem as $OT = ((GTS, G_0), G_{partial}, Const)$.

Existing solutions to the OT problem. An obvious way to solve the optimal trajectory problem for GTS is to traverse the entire state space of the graph grammar, e.g. using DFS or BFS search strategies. However, there is no guarantee when the best cost solution is found, thus the size of the explored state space may explode easily. In order to be able to handle the size of the explored state space, we define additional cuts to drive the state space traversal. These cuts can be either predefined constraints like limited cost budget or an approximate optimal cost that can prune the search space during the exploration.

In the domain of Petri nets, the optimal trajectory problem was described in [GP06]. In that paper linear algebraic algorithms (process network synthesis (PNS) algorithms) are used to generate a candidate transition occurrence vector. Since these techniques do not guarantee the fireability of the solution the feasibility of the candidate is analyzed by subsequent fireability checks.

5.2 Overview of our solution

In the current paper we propose the following solution (see Figure 1) to the optimal trajectory problem of GTS.

1. At first, the GTS is abstracted into the cardinality P/T net with the cost parameters as described in Section 4.

2. Then subsequent integer programming problems are formulated based on the state inequality (see Section 3) of the P/T net satisfying the following conditions.

   - Numerical and logical constraints are encoded into the ILP problem as inequalities. The violation of these constraints result in the pruning of the state space along a certain path.
   - The goal state or graph $G_{partial}$ is translated into a marking $M_{partial}$ that is encoded into the ILP problem.
   - The objective function of the ILP is given by the minimization of the overall cost of the trajectory.

3. Since the P/T net abstraction does not guarantee that there is an executable rule application sequence on the GT level for this *candidate transition occurrence vector*, we will investigate its fireability directly on the GT level. In case of an optimal but not fireable solution, we iteratively derive the next best cost solution to the ILP problem. This way the cost of this next solution could be higher than the previous one.

   A candidate transition occurrence vector counts the number of rule applications, and thus provide a guaranteed minimal cost (i.e. an underapproximation) for the OT problem by solving the ILP programming problem derived according to Section 5.2.1.

4. The best cost candidate, and the candidate solution vector is used as a hint to guide the search strategy during the exploration of the GTS state space (for further details, see Section 5.2.2).

   - If the current best cost is exceeded or an execution path is incompatible with the current solution vector, then this path is considered unpromising, and thus its exploration is postponed.
   - If the traversal of the state space fails on the remaining paths, the next best cost solution is derived and the state space exploration is continued.

5.2.1 Deriving an ILP problem for the optimal trajectory problem of GTS

Let an optimal trajectory problem of GTS $OT = ((GTS, G_0), G_{partial}, Const)$ be given and let the abstraction cardinality P/T net be $\mathscr{F}(GTS)$ together with $M_0 = \mathscr{F}(G_0)$, and $M_{partial} = \mathscr{F}(G_{partial})$.

Since the abstracted P/T net simulates the underlying GTS, for each path (trajectory) p in the GTS starting from G_0 and leading to an instance graph G which covers $G_{partial}$, there is a

transition firing sequence s in the P/T net such that $M_{partial}$ is partially reachable from M_0 by s. In addition, each component of the transition occurrence vector σ of s is equal to the number of corresponding GT rule applications in trajectory p.

Thus such a transition occurrence vector σ has to satisfy the state inequality $M_{part} \leq M_0 + W \cdot \sigma$, where W is the incidence matrix of the P/T net. In other words, the search for an appropriate GT sequence can be carried out by finding a transition occurrence vector which search is followed by a firing check, i.e. the existence of an executable GT trajectory which is compatible with the transition occurrence vector.

Since the cost of a trajectory is equal to the cost of the corresponding transition occurrence vector σ, the minimization of the cost of the GTS trajectory is subsumed by the minimization of $c(\sigma)$. Since each rule may be applied finitely many times (i.e. $\sigma \in \mathbb{N}^{|T|}$, where T is the set of transitions of the cardinality P/T net), we can generate candidate solutions for the optimal trajectory problem of GTS by the following *integer* linear programming problem.

minimize
$c^T \cdot \sigma$,
subject to
$W \cdot \sigma \geq M_{part} - M_0$,
$\sigma \in \mathbb{N}^{|T|}$,

where vector c is composed of the cost of the corresponding transitions. If there are numerical constraints, they are also added as inequalities.

5.2.2 Guiding exploration of the GT state space

The idea to use the solution vector of the ILP problem is to guess the best cost solution in order to drive the state space traversal. The exploration of the graph grammar state space is driven by heuristics like constraints and by the solution transition occurrence vector in the form of logical and numerical cuts.

If there is a candidate solution vector we can perform the following checks on the state space in order to reduce its size.

1. The traversal of a branch of the graph transition graph is suspended, i.e. the discovery of the unpromising paths is postponed
   (a) if the cost of the path from the initial graph to the current graph exceeds the cost of the candidate solution transition occurrence vector, or
   (b) if a GT rule was applied more times then the corresponding transition was fired in the candidate transition occurrence vector.
2. If a constraint is violated along a path, then the graph transition system is permanently pruned along that path.

During the exploration of the state space the newly generated instance graphs are analyzed whether (i) they satisfy all the constraints, or (ii) the goal state is reached, i.e. the required partial graph is covered by the current state (graph) in the graph transition system. If the cost of a path

equals to the algebraic optimum, and it also leeds to a goal state, then the optimal trajectory is found.

If there are no branches to continue the state space traversal (according to the defined cuts based on the current candidate solution vector), the next best solution to the ILP problem is generated by excluding the first solution retrieved by the solver with additional constraints. The optimal solution of the new ILP problem is taken into account in the search strategy during the exploration of the previously unpromising paths.

The algorithm terminates when the first feasible solution is found i.e. (i) by enumerating potential costs in an increasing order using the ILP solver, and (ii) checking the feasibility of the current solution by the model checker. This way all possible solutions are encountered.

5.3 Example: Solving an optimal trajectory problem

The GT optimal trajectory problem is the following. Let given the GG be given as described in Section 3. Then our aim is to find an optimal trajectory from the given initial graph to a goal state where at least 4 services are active in order to serve 4 requests simultaneously, i.e. there are at least 4 services in the instance graph with label *active*. An additional constraint is that our budget is limited: maximum 20 units of cost is available for the reconfiguration of services. For instance, the path *on*, *register*, *on* in Fig. 5 is invalid because its operation cost (23) exceeds our budget.

The abstracted cardinality P/T net and the corresponding (initial and partial) markings are generated as defined in Section 4. The incidence matrix of the example in Section 3 is shown in Fig. 7. The one constraint can be given in a numerical form added to the core ILP problem as follows.

minimize

$(10,4,8,1,2,1,3,3)\cdot\sigma$

subject to

$(4,4,0,0,0,0) \leq (5,2,2,0,1,2) + W\cdot\sigma,$

$(10,4,8,1,2,1,3,3)\cdot\sigma \leq 20, \sigma \in \mathbb{N}^{|8|}.$

The first optimal algebraic solution is $(0,0,0,2,0,0,0,0)$, and its cost is 2 cost units. Now let us start to construct the graph transition system of the GG. Since only one transition is fired in the solution transition occurrence vector, in the GG only rule *failover* is applied (see Fig. 8). However, rule *failover* cannot be applied subsequently twice, because the left–hand side of the rule cannot be matched after the first rule application. Therefore the next best solution is generated (such that the inequality $(10,4,8,1,2,1,3,3)\cdot\sigma \gneq 2$ is added to the ILP).

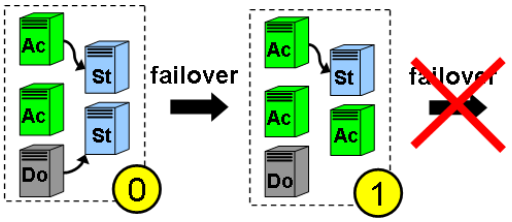

Figure 8: State space of GG

The next best solution is $(0,0,0,1,0,1,1,0)$, and its cost is 5 cost units. Thus the search strategy is that only these three rules *failover*, *detach*, and *register* are applied exactly once (if they are applicable). In this way, we omit the traversal those paths where other rules are applied. Since there is an appropriate GT sequence $\langle failover, detach, register \rangle$, we terminate the traversal of the state space as the optimal trajectory is found, see Figure 9.

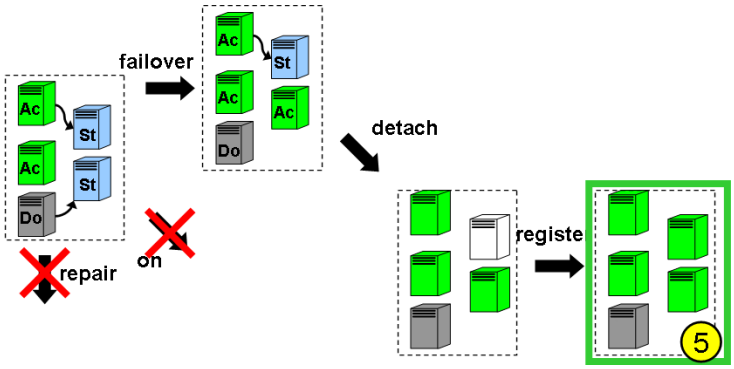

Figure 9: State space of GG

6 Related work

While we believe that the modeling of optimization problems by graph transformation systems is quite novel idea in the field, it is not unprecedented in a broader research scope, as described below.

Analysis of Petri nets. The literature of Petri nets is quite rich in papers that solve reachability, or scheduling problems using linear algebraic techniques.

The use of integer programming methods in the (reachability) analysis of Petri nets is not a novel idea. In [MR97] *deadlock detection* was reduced to a mixed integer linear programming problem. In [KK00] the authors presented a further development of this approach to prove *deadlock detection, mutual exclusion, and marking reachability and coverability*. This solution was based on the *unfolding* of the Petri net. Another unfolding-based solution is discussed for *safe* Petri nets in [ES01].

Linear algebraic algorithms were used to solve the Petri net reachability problem without state space explosion in [Des98, STC98]. These techniques are powerful, and they provide semi–decision techniques to decide the reachability of a given marking (similarly as our method).

A Petri net based technique was presented to solve reachability problems of GTS recently in [VL06]. A commonality in this approach and our technique in [VVE+06] is that both abstract from the graph structure, thus node and edge types are considered instead of the nodes and edges themselves.

Optimal trajectory problem. The first author studied the problem of simultaneous optimization and reachability analysis in the field of Petri nets in various papers [GP03, GP06]. The essence of the approach is to use Process Network Synthesis (PNS) algorithms (well-known techniques in the field of chemical engineering) to derive the optimal solution, and then check whether the target state is reachable along the optimal path. In case of failure, PNS algorithms derive the next optimal solution, and the reachability of this trace is checked again. Several heuristics are used to filter unreachable solutions quickly without the traversal of the state space.

We studied a joint optimization and reachability problem in case of graph transformation with time in [GSV04] using the model checker SPIN. However, this method required an a priori upper bound on the number of the graph elements of a certain type.

The use of various model checking techniques for optimization problems were reported in [Ruy03] where scheduling problems were solved using a direct SPIN encoding. Our technique presented in the paper can also be regarded as a directed model checking approach as categorized by [EJL06]. They use (an ad hoc) SPIN encoding for the analysis of graph transition systems.

Model checking tools for GTS. There are several model checking approaches to analyze graph transformation systems. One can categorize them as *interpreted approaches* like [BK02, Ren04, KK06], which store system states as graphs and directly apply transformation rules to explore the state space, and *compiled approaches* such as [SDR04, SV03, EJL06] which translate graphs and graph transformation rules into off-the-shelf model checkers to carry out verification.

We believe that most of these approaches can potentially be extended in the future to handle cost attributes as well for optimization and verification (e.g. exposed by temporal constraints) purposes. In particular, GROOVE [Ren04] can directly be used to implement our approach (by adapting its search strategies appropriately), and we believe that unfolding based cuts and optimization is also an interesting direction for the future based on AUGUR [KK06].

7 Conclusions and Future Work

In the paper, we studied the optimal trajectory problem for GTS, a combined optimization and reachability problem. We proposed to use graph transformation extended with rule costs for modeling the evolution of systems where the cost of the operation to be performed is crucial. In this way, we are able to find the optimal transformation sequence leading to a desired system configuration defined by a reachable (partial) state.

The proposed solution for the optimal trajectory problem for GTS is based on the use of the P/T net abstraction and algebraic optimization techniques to reduce the traversed state space of the GTS. The main advantages of the approach are that (i) the GTS based modeling provides a highly expressive technique for the modeling the evolution of the system, (ii) using P/T net abstraction and optimization techniques provides good initial candidates to drive state space exploration (iii) the system structure does not have to be a priori bound (which is an extension to our own previous work [GSV04]).

Future work. In the future work, we plan to implement this approach by altering search strategies in GROOVE, which provides extensive support for identifying isomorphic graphs and its

ability the proper handling of symmetries, etc. for state space exploration. Optimization is foreseen to be carried out using off–the–shelf optimization tools like GAMS or CPLEX.

A possible performance measure to evaluate our approach could be the comparison of the number of the traversed graphs in the graph transition system using different search strategies. In addition, we can also measure the total execution time of optimization runs with the chosen tool set. This latter can be further accelerated by precomputing the first n algebraic optimum before starting the state space exploration itself.

Acknowledgements: This work was partially supported by the DECOS and SENSORIA European integrated IST projects. Furthermore, the second author was also supported by the J. Bolyai Scholarship.

Bibliography

[BHTV06] L. Baresi, R. Heckel, S. Thöne, D. Varró. Style-Based Modeling and Refinement of Service-Oriented Architectures. *Journal of Software and Systems Modelling* 5(2):187–207, 2006.

[BK02] P. Baldan, B. König. Approximating the Behaviour of Graph Transformation Systems. In Corradini et al. (eds.), *Proc. ICGT 2002: First International Conference on Graph Transformation*. LNCS 2505, pp. 14–29. Springer, Barcelona, Spain, October 7–12 2002.

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe. *In [Roz97]*. Chapter Algebraic Approaches to Graph Transformation — Part I: Basic Concepts and Double Pushout Approach, pp. 163–245. World Scientific, 1997.

[Des98] J. Desel. *Petrinetze, Lineare Algebra und lineare Programmierung*. Teubner-Texte zur Informatik 26. B. G. Teubner Stuttgart-Leipzig, 1998.

[EJL06] S. Edelkamp, S. Jabbar, A. Lluch-Lafuente. Heuristic Search for the Analysis of Graph Transition Systems. In *Proc. Third International Conference on Graph Transformation*. LNCS 4178, pp. 414–429. Springer, Natal, Brazil, 2006.

[ES01] J. Esparza, C. Schröter. Unfolding Based Algorithms for the Reachability Problem. *Fundamenta Informaticae* 46:1–17, 2001.

[GP03] S. Gyapay, A. Pataricza. Optimization methods for reachability analysis of Petri net models. In Tarnai and Schnieder (eds.), *Formal Methods for Railway Operation and Control Systems (Proceedings of Symposium FORMS-2003, Budapest, Hungary, May 15-16)*. Pp. 53–60. L' Harmattan, Budapest, May 17–23 2003.

[GP06] S. Gyapay, A. Pataricza. Optimal Trajectory Generation for Petri nets. *Acta Cybernetica* 17(2), 2006.

[GSV04] S. Gyapay, Á. Schmidt, D. Varró. Joint Optimization and Reachability Analysis in Graph Transformation Systems with Time. In *Proc. GT-VMT 2004, International Workshop on Graph Transformation and Visual Modelling Techniques*. ENTCS 109, pp. 137–147. Elsevier, 2004.

[KK00] V. Khomenko, M. Koutny. Verification of Bounded Petri Nets Using Integer Programming. Technical report, Department of Computing Science, University of Newcastle upon Tyne, 2000. Technical Report CS-TR-711.

[KK06] B. König, V. Kozioura. Counterexample-Guided Abstraction Refinement for the Analysis of Graph Transformation Systems. In *TACAS*. Pp. 197–211. 2006.

[MR97] S. Melzer, S. Römer. Deadlock Checking Using Net Unfoldings. In *CAV '97: Proceedings of the 9th International Conference on Computer Aided Verification*. Pp. 352–363. Springer-Verlag, 1997.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. In *Proc. IEEE*. Volume 77(4), pp. 541–580. 1989.

[Ren04] A. Rensink. The GROOVE Simulator: A Tool for State Space Generation. In Nagl et al. (eds.), *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*. LNCS 3063. Springer-Verlag, 2004.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformations: Foundations*. World Scientific, 1997.

[Ruy03] T. C. Ruys. Optimal Scheduling Using Branch and Bound with SPIN 4.0. In *Proc. 10th International SPIN Workshop*. LNCS 2648, pp. 1–17. Springer, Portland, Oregon, USA, May 9–10 2003.

[SDR04] O. M. dos Santos, F. L. Dotti, L. Ribeiro. Verifying Object-Based Graph Grammars. *Electr. Notes Theor. Comput. Sci.* 109:125–136, 2004.

[STC98] M. Silva, E. Teurel, J. M. Colom. Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems. In W. Reisig (ed.), *Lectures on Petri Nets I:Basic Models*. LNCS 1791, pp. 309–373. Springer, 1998.

[SV03] Á. Schmidt, D. Varró. CheckVML: A Tool for Model Checking Visual Modeling Languages. In Stevens et al. (eds.), *Proc. UML 2003: 6th International Conference on the Unified Modeling Language*. LNCS 2863, pp. 92–95. Springer, San Francisco, CA, USA, October 20-24 2003.

[VL06] P. P. P. Velasco, J. de Lara. Petri Nets and Matrix Approach to Graph Transformation: Reachability. 2006. To appear.

[VVE+06] D. Varró, S. Varró-Gyapay, H. Ehrig, U. Prange, G. Taentzer. Termination Analysis of Model Transformations by Petri Nets. In *Proc. Third International Conference on Graph Transformation (ICGT 2006)*. LNCS 4178, pp. 260–274. Springer, Natal, Brazil, 2006.