Transformation of Type Graphs with Inheritance for Ensuring Security in E-Government Networks

Frank Hermann, Hartmut Ehrig, and Claudia Ermel

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin
{frank.hermann,hartmut.ehrig,claudia.ermel}@tu-berlin.de

Abstract. E-government services usually process large amounts of confidential data. Therefore, security requirements for the communication between components have to be adhered in a strict way. Hence, it is of main interest that developers can analyze their modularized models of actual systems and that they can detect critical patterns. For this purpose, we present a general and formal framework for critical pattern detection and user-driven correction as well as possibilities for automatic analysis and verification at meta-model level. The technique is based on the formal theory of graph transformation, which we extend to transformations of type graphs with inheritance within a type graph hierarchy. We apply the framework to specify relevant security requirements.
The extended theory is shown to fulfil the conditions of a weak adhesive HLR category allowing us to transfer analysis techniques and results shown for this abstract framework of graph transformation. In particular, we discuss how confluence analysis and parallelization can be used to enable parallel critical pattern detection and elimination.

1 Introduction

Software systems for e-government services have to provide a platform, where internal and external users can input and process large amounts of confidential data. Therefore it is important that considerable efforts are made to secure such data. To improve the security of software systems, recent research has identified that security analysis should be integrated into software engineering techniques and security should be considered from the early stages of the software systems development process [2]. Existing security modelling frameworks such as the UML profile *UMLsec* [3] support the design of security-sensitive systems by offering stereotypes to describe policies of system parts like communication channels or subsystems. Models then can be analyzed to check the satisfaction of security policies, such as access control conditions. Common techniques to elicit security requirements are based on use case modeling and goal-oriented approaches [4]. The problem is that these techniques are better suited for the elicitation of functional requirements. Security requirements being non-functional requirements are closely related to system architecture design and frequently require architectural changes as reactions to detected critical patterns. Moreover, the *UMLsec* profile specifies only core security requirements and has to be refined for more specific application fields like secure e-government services.

In order to be able to specify flexible architectural changes as reactions to detected critical patterns in the design of e-government systems, we propose in this paper a dynamic, general modelling approach based on typed graph transformation for critical pattern detection and elimination.

Public administration is based on a strict hierarchical structure of e-government networks. We reflect this fundamental design paradigm in our modelling approach by supporting hierarchies along a chain of meta-model layers. The common approach of *meta-modelling* uses UML class diagrams equipped with OCL constraints to model a domain-specific language's (DSL's) abstract syntax in a declarative way (see e.g. the MOF approach by the OMG [5]). Graph grammars [6] are a more constructive alternative, based on a formal categorical framework which can also be used for formal analysis and verification. A DSL here is modelled by a type graph capturing the definition of the underlying symbol and relation types. Instances of a DSL are given by graphs typed over (i.e. conforming to) the type graph, and can be further restricted by defining rule-based instance generation operations. A DSL type graph corresponds closely to a meta-model, i.e. also inheritance relations are used[1]. Hence, the main technical contribution of this paper lies in solving the challenge of transformation of graphs with inheritance hierarchies.

As running example, we consider an e-government system application which is based on a standard given by the *E-Government Manual of the Federal Office for Information Security* in Germany. In particular, we here focus on Chapter IV [8]. There are four main zones in the architecture of an e-government system (depicted in Fig. 1), one client zone for the external view and three security zones, which are under control of the corresponding government institution.

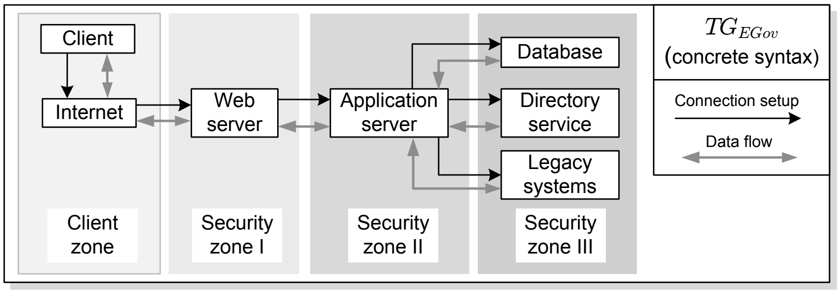


Fig. 1. Scenario: structure of E-Government networks

[1] Note that the type graphs used for network modelling in our previous paper [7] did not yet allow the use of inheritance.

E-government services are installed on web servers in zone one, which can access actual applications of the public agency in zone two, but they are not directly connected to confidential data. Hiding these data in zone three improves the security against external attacks. If the data was stored in zone two already, an intrusion on a web server could directly enable scans of the data file system and further more critical changes.

In the following sections we discuss how this standard structure of an e-government network can be refined, customized and analyzed on the basis of formal type graph transformation with inheritance. Transformations and analysis are performed on the type graph of the e-government network visualized in Fig. 1. The overall model consists of a hierarchy of models with several meta-levels, all formalized by type graphs. Type graphs with inheritance and typed graph transformation have been introduced already in [6, 9] but without transformation of the meta-levels including inheritance. The new formal approach in this paper concerns a generalization of typed graph transformation to the transformation of type graphs with inheritance. The key concepts thus are graphs with inheritance, called I-graphs, and I-graph morphisms based on clan morphisms [9], coming up with a new category **IGraphs**, which is shown to fulfil the requirements of weak adhesive HLR categories [6]. This allows us to make use of formal techniques for confluence and dependency analysis to analyze critical pattern detection and elimination in the e-government network model.

Graphs with inheritance could also be transformed by encoding the graphs to plain graphs with the help of a special edge type for the inheritance relation and performing standard graph transformation on them. But this leads to several problems. All inheritance *paths* have to be translated to direct edges, and after performing a transformation step the resulting graph would have to be extended by the edges which form the transitive closure of the inheritance relation. Furthermore, extending matching to inheritance hierarchies, as considered in this paper, is not possible if inheritance is encoded by special edges in plain graphs.

The paper is structured as follows: In Sec. 2 we show how type graph rules and transformations including the handling of inheritance can be used to model network configurations for secure client-server architectures for e-government networks [8]. Thereafter, we define the basic formal constructions for transforming type graphs with inheritance and show important properties in Sec. 3, which will then be used in Sec. 4 for analyzing the e-government network model. Sec. 5 discusses related work, and Sec. 6 concludes the paper. Our technical report [22] contains the full proofs for the presented results.

2 Modelling E-Government Networks

In this section we show how type graph transformations including the handling of inheritance can be applied for developing and maintaining meta-models for e-governments networks [8].

Example 1 (Type Graphs for Network Configurations). Graph G_{EGov} in the lower left corner of Fig. 2 is an instance-level graph typed over the type graph

TG_{EGov} for network configurations in the area of e-government. Graph G_{EGov} is shown in concrete syntax in the lower right corner of Fig. 2 and describes a client, which is connected to services of the e-government institution. TG_{EGov} itself is typed over the more abstract type graph TG_{Web} which models domain specific languages of client-server architectures. Type mappings like $TG_{EGov} \rightarrow TG_{Web}$ are denoted by the type name following the respective node or edge name after the colon, e.g. the node "PC:Client" in TG_{EGov} is mapped to the node "Client" in TG_{Web}.

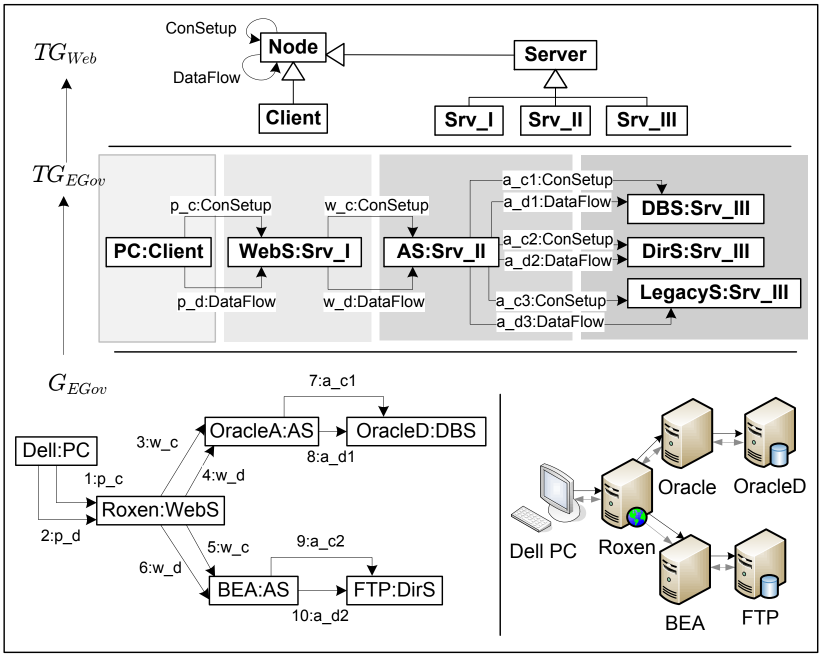


Fig. 2. Instance Graph G_{EGov} and Type Graph Hierarchy $TG_{EGov} \rightarrow TG_{Web}$

The main idea of graph transformation is the rule-based modification of graphs, which represent the abstract syntax of models. While standard graph transformation [6] considers transformations of instances typed over a given type graph only, we present an extension in Sec. 3 to deal with more general transformations including transformations of type graphs with inheritance, which may be typed over a type graph of the next meta level.

The core of a graph transformation rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ as defined in [6] is a triple of graphs (L, K, R), called left-hand side, interface and right-hand side, and two injective graph morphisms $L \overset{l}{\leftarrow} K$ and $K \overset{r}{\rightarrow} R$. Interface K contains the graph objects which are not changed by the rule and hence occur both in L and in R. Applying rule p to a graph G means to find a match m of L in

G and to replace this matched part $m(L)$ in G by the corresponding right-hand side R of the rule, thus leading to a graph transformation step $G \overset{p,m}{\Longrightarrow} H$.

Note that a rule may only be applied if the *gluing condition* is satisfied, i.e. the rule application must not leave *dangling edges*, and for two objects which are identified by m, the rule must not preserve one of them and delete the other one. Furthermore, a rule p may be extended by a set of positive or negative *application conditions* (PACs and NACs) [10, 6]. Intuitively, a NAC forbids the presence of a certain pattern in graph G, while a PAC requires it.

$$\begin{tikzcd}
NAC \arrow[dr, "q"', "/" marking] & L \arrow[l, "n"'] \arrow[d, "m"] \arrow[dr, phantom, "(PO_1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(PO_2)"] & R \arrow[d, "m^*"] \\
& G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

A match $L \xrightarrow{m} G$ satisfies a NAC with the injective NAC morphism $n : L \to NAC$, if there is no injective graph morphism $NAC \xrightarrow{q} G$ with $q \circ n = m$ (where "$\circ$" denotes composition of morphisms), as shown in the diagram to the right. Analogously, a PAC is satisfied if there *exists* such an injective graph morphism $PAC \xrightarrow{q} G$. Our notion of graph transformation is called double-pushout approach (DPO) since both squares in the diagram are pushouts in the category of graphs, where D is the intermediate graph after removing $m(L)$ in G and in (PO_2) H is constructed as gluing of D and R along K.

The following examples show how changes of type graphs *with inheritance*, like TG_{Web} and TG_{EGov} in Fig. 2, can be defined in a formal and concise way.

Example 2 (Rules for Editing Network Meta-Models). Fig. 3 and the top line of Fig. 4 show some typical editing rules, typed over TG_{Web}, where numbers specify the rule morphisms. Interface K contains the numbered elements in L only and is not shown explicitly in Fig. 3. The first two rules insert new nodes and connections. Note that rule "createCS()" can be applied to any pair of nodes, because the node types are specified abstractly. Rule "setUpdateConnection()" contains a NAC and defines the controlled extension of connections, i.e. a pair of links of types "ConSetup" and "DataFlow", starting at a server node in zone 3. A new connection for requesting server updates can be established, but only if there is no incoming connection via the same server, because this would ease an attack from an external Internet connection.

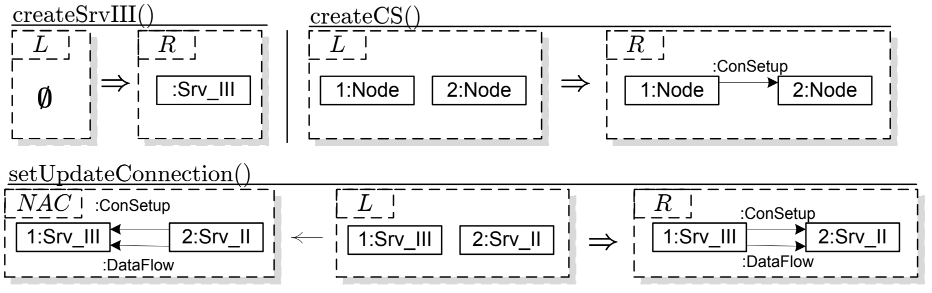


Fig. 3. Rules for Editing Type Graph TG_{E-Gov}

Finally, rule “insertSupertype()” given by the top line in Fig. 4 specifies a sample refactoring operation, where a new super type node is created having three nodes of type “Serv_III” as specializations.

Example 3 (EGov Type Graph Transformation Step). Fig. 4 shows a graph transformation step, where rule “insertSupertype()” is applied to graph G_1, a part of graph TG_{EGov} from Fig. 2, resulting in the transformed graph G_2.

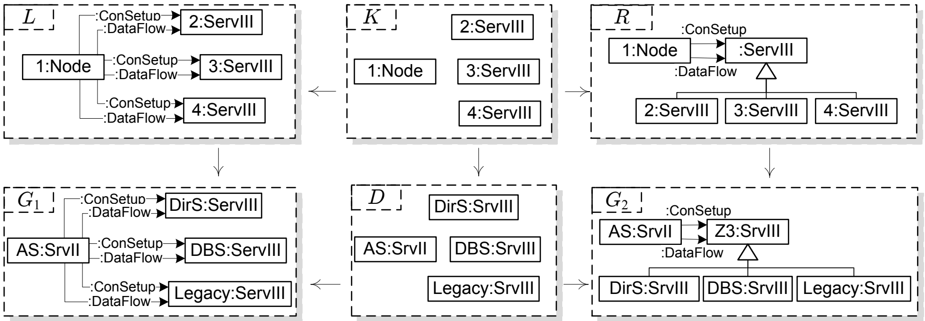


Fig. 4. Type Graph Transformation Step of rule insertSupertype()

The result of applying the rule to the complete type graph TG_{EGov} yields the type graph TG_{EGov2} as shown in Fig. 5.

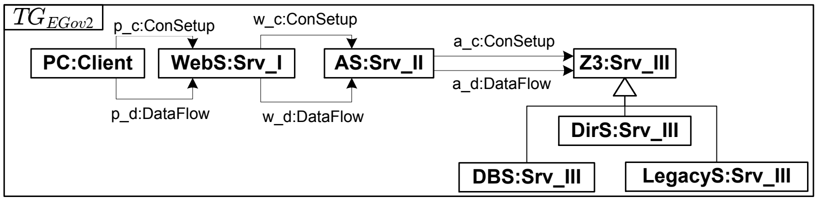


Fig. 5. Resulting Type Graph TG_{EGov2} as update of TG_{EGov}

The examples show how transformations of type graphs with inheritance in e-government networks can be defined in a concise way. After presenting the underlying formalization in the next section we continue the example in Sec. 4 to show the relevant features of the approach for ensuring security in e-government networks.

3 Transformation of Graphs with Inheritance

Graph transformation with node type inheritance [6, 9] provides main aspects of inheritance, in particular inheritance of attributes and edge types from parent node types to children node types. In this section we lift transformations from the instance level to the meta levels in order to support a formal basis for editing and analyzing meta-models, i.e. type graphs with inheritance within the framework of graph transformation. Recall further that meta-modelling is captured by graph transformation using the concept of type graph hierarchy [7, 11].

Note that we use the algebraic notion of graphs, where a graph $G = (V, E, s, t)$ is given by a set of nodes V, a set of edges E and functions $s, t : E \to V$ specifying source and target nodes for each edge. A graph morphism $f : G_1 \to G_2$ is a pair of mappings $(f_V : V_1 \to V_2, f_E : E_1 \to E_2)$ compatible with source and target functions, i.e. $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$. In order to improve readability of the paper we present our inheritance concepts first for graphs without attribution, but in [22] we show how all concepts and results can be extended to attributed graphs. Note that the following notion of I-graphs slightly differs from [6] by using a relation for capturing the inheritance information (instead of a separate graph with distinguished abstract nodes) in order to simplify further constructions.

Definition 1 (I-Graph). *Graph with Inheritance*, short I-Graph, is given by $GI = (G, I)$. It consists of graph G and inheritance relation $I \subseteq G_V \times G_V$, where for $v \in G_V$ $clan_I(v) = \{v' \in G_V \mid (v', v) \in I^*\}$ with I^* being the reflexive and transitive closure of I.

Remark 1. According to [6, 9] as well as MOF [5] and UML [12] we do not require that the inheritance relation is cycle free.

I-graph morphisms - not considered in [6] - are based on clan-morphisms [6] taking into account inheritance.

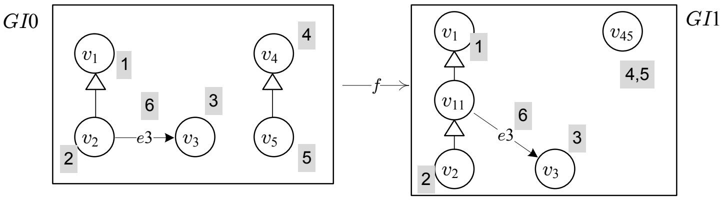
Definition 2 (Clan-Morphism). Given graph $G1$ and I-graph $GI2 = (G2, I2)$ a pair of mappings $f = (f_V, f_E) : G1 \to G2$ is called *clan-morphism*, written $f : G1 \to GI2$, if $\forall\ e1 \in G1_E$:

$f_V \circ s_{G1}(e1) \in clan_{I2}(s_{G2} \circ f_E(e1)) \quad \wedge \quad f_V \circ t_{G1}(e1) \in clan_{I2}(t_{G2} \circ f_E(e1))$.

I-graphs and I-graph morphisms define the category **IGraphs**.

Definition 3 (Category IGraphs). Given I-graphs $GI1 = (G1, I1)$ and $GI2 = (G2, I2)$, an I-graph morphism $f : GI1 \to GI2$ is given by a clan-morphism $f : G1 \to GI2$, which is I-compatible, i.e. $(v, w) \in I1$ implies $(f(v), f(w)) \in I2^*$. The composition of I-graph morphisms $f : GI1 \to GI2$ and $g : GI2 \to GI3$ is defined by $g \circ f : GI1 \to GI3$ with $(g \circ f)_V = g_V \circ f_V : G1_V \to G3_V$ and $(g \circ f)_E = g_E \circ f_E : G1_E \to G3_E$. The category of I-graphs and I-graph morphisms is denoted by **IGraphs**.

Example 4 (I-graph Morphism). The following example shows I-graph morphism $f : GI0 \to GI1$ where grey numbers indicate the mappings. According to I-compatibility the identification of nodes v_4 and v_5 contained in $GI0$ is possible, because $(v_{45}, v_{45}) \in I1^*$. Furthermore, inheritance between $v1$ and $v2$ of $GI0$ can be refined into several steps as shown by node v_{11} in $GI1$. The clan morphism f can additionally map edges to edges between nodes of super types as shown by $e3$.



Remark 2. 1. I-compatibility is equivalent to

$$(v, w) \in I1^* \text{ implies } (f_V(v), f_V(w)) \in I2^*.$$

2. Given I-graph morphisms f and g then: $g \circ f : GI1 \to GI3$ is an I-graph morphism, because I-compatibility of f and g implies that of $g \circ f$ and we can show for all $e_1 \in G1_E$: $(g \circ f)_V \circ s_{G1}(e_1) = g_V \circ f_V \circ s_{G1}(e_1) \in clan_{I3}(s_{G3} \circ (g \circ f)_E(e_1))$.
3. Each clan-morphism $f : G1 \to GI2$ is also an I-graph morphism $f : GI1 \to GI2$ with $GI1 = (G1, I1)$ and $I1 = \emptyset$, because in this case I-compatibility is trivial. This implies also that the composition of a clan-morphism $f : G1 \to GI2$ with an I-graph morphism $g : GI2 \to GI3$ is a clan morphism $g \circ f : G1 \to GI3$.

In order to enable automatic critical pattern detection and user driven transformation for meta-models we lift graph transformation from the instance level to all meta levels within the abstract framework of weak adhesive HLR categories [6]. This way we can apply the well-known results for the abstract framework, e.g. analysis and correction can be parallelized and distributed to meta-model parts in case of several e-government networks.

For defining a weak adhesive HLR category we need to distinguish a suitable class $\mathcal{M}$ fulfilling certain properties. We propose the class $\mathcal{M}_{S-refl}$ of subtype-reflecting morphisms, because on the one hand DPO-rules based on these morphisms are powerful enough to generate all kinds of cycle-free inheritance graphs on the meta-model level and on the other hand $(\mathbf{IGraphs}, \mathcal{M}_{S-refl})$ can be shown to be a weak adhesive HLR category with componentwise construction of pushouts and pullbacks. Note that this fails to be true for the class $\mathcal{M}$ of all injective I-graph morphisms.

The notion of *subtype reflection, short S-reflection*, defines the condition that for each node n in the image of a morphism f it holds that all subtypes of n are in the image of f as well. We will need this condition for the proof of Thm. 1.

Definition 4 (S-reflecting Morphism). An S-reflecting morphism $f1 : GI0 \to GI1$ is an I-graph morphism $f1 : GI0 \to GI1$, where $f1$ is an injective graph morphism and has the S-reflection property: $\forall\ (v_{11}, v_1) \in I1^*, v0 \in GI0_V : v_1 = f1_V(v_0) \Rightarrow\ \exists\ v_{01} \in GI0_V : f1_V(v_{01}) = v_{11} \land (v_{01}, v_0) \in I0^*$.

All rules in Figures 3 and 4 are S-reflecting, i.e. their rule morphisms are S-reflecting. Note that standard graph transformation rules, i.e. rules without inheritance, can be interpreted as S-reflecting rules by adding empty inheritance relations to their graphs. According to Thm. 1 and Thm. 2 in [22] pushouts and pullbacks along S-reflecting I-graph morphisms can be constructed componentwise and the class $\mathcal{M}_{S-refl}$ is closed under pushouts and pullbacks. Therefore, DPO transformations of S-reflecting rules are well defined and can be constructed componentwise in **IGraphs**. Furthermore theses properties are part of the conditions for weak adhesive HLR categories and in fact, the category $(\mathbf{IGraphs}, \mathcal{M}_{S-refl})$ is a weak adhesive HLR category (see Remark 3 below).

Theorem 1 ($(\mathbf{IGraphs}, \mathcal{M}_{S-refl})$ is Weak Adhesive HLR Category). The category **IGraphs** of graphs with inheritance together with the class $\mathcal{M}_{S-refl}$ of S-reflecting morphisms is a weak adhesive HLR category.

Remark 3 (Weak adhesive HLR category). According to the definition of weak adhesive HLR categories (see Definition 4.13 in [6]) $(\mathbf{IGraphs}, \mathcal{M}_{S-refl})$ has this property if

1. $\mathcal{M}_{S-refl}$ is a class of monomorphisms closed under isomorphisms, composition and decomposition
2. **IGraphs** has pushouts and pullbacks along $\mathcal{M}_{S-refl}$-morphisms and $\mathcal{M}_{S-refl}$ is closed under pushouts and pullbacks
3. $(\mathbf{IGraphs}, \mathcal{M}_{S-refl})$ has the weak VK-property, i. e. given a cube as below, where the bottom face is a pushout with $f1 \in \mathcal{M}_{S-refl}$ and the back faces are pullbacks and one of the following two cases is satisfied, then we have: top square is pushout ⇔ front squares are pullbacks.
   case 1 Also $f2 \in \mathcal{M}_{S-refl}$.
   case 2 Also $l1, l2, l3 \in \mathcal{M}_{S-refl}$.

   We can conclude for each direction of the equivalence by item 2 in case 1: also $g1, g2, h1, h2, k1, k2 \in \mathcal{M}_{S-refl}$ and in case 2: also $g2, h1, k2, l0 \in \mathcal{M}_{S-refl}$.

$$\begin{tikzcd}
 & GI4 \arrow[rr, "h1"] \arrow[dl, "h2"'] \arrow[dd, "l0"] & & GI5 \arrow[dl, "k1"] \arrow[dd, "l1"] \\
GI6 \arrow[rr, "k2"] \arrow[dd, "l2"'] & & GI7 \arrow[dd, "l3"] & \\
 & GI0 \arrow[rr, "f1"] \arrow[dl, "f2"'] & & GI1 \arrow[dl, "g1"] \\
GI2 \arrow[rr, "g2"] & & GI3 &
\end{tikzcd}$$

Proof (see [22]).

Corollary 1 (Results for $(\mathbf{IGraphs}, \mathcal{M}_{S-refl})$). The following results for graph transformation based on $(\mathbf{IGraphs}, \mathcal{M}_{S-refl})$ are valid:

- Local Church Rosser Theorem for pairwise analysis of sequential and parallel independence (Thm. 5.12 in [6])
- Parallelism Theorem for applying independent rules and transformations in parallel (Thm. 5.18 in [6])
- Concurrency Theorem for applying E-related dependent rules simultaneously (Thm. 5.23 in [6])
- Embedding and Extension Theorem for transferring transformations and analysis results to more complex scenarios (Thms. 6.14 and 6.16 in [6])
- Local Confluence Theorem and Completeness of critical pairs for analyzing conflicts and for showing local Confluence (Thm. 6.28 and Lemma 6.22 in [6])

Proof (Idea). These results are shown in [6] for weak adhesive HLR categories with some additional properties (see Remark 6 in [22]) and are valid for (**IGraphs**, $\mathcal{M}_{S-refl}$) by Thm. 1.

Before we show how the results in Corollary 1 can be applied in our scenario of e-government networks let us discuss other approaches which may avoid to work in the category **IGraphs**. The intuitive semantics of an I-graph GI is the graph $\overline{GI}$ defined by closure or flattening of the inheritance relation I (see Def. 6 in [22]) as considered already for type graphs with inheritance in [6]. The inheritance closure is a cofree construction (see Thm. 7 in [22]) leading to a cofree functor from **IGraphs** to **Graphs**. This implies that pullbacks are preserved, but as shown in Example 5 in [22] - pushouts are not preserved in general. For this reason, transformations with inheritance cannot easily be reduced to standard graph transformation by flattening (see more details in [22]).

4 Analysis of E-Government Network Meta-Models

During each phase of system design critical patterns may occur, which can imply unwanted behaviour and possibilities for a loss of security. The earlier they can be detected and the earlier they can be corrected the lower is the risk of a system containing critical parts in its implementation. This motivates to apply analysis techniques as early and as abstract as during the meta-model development. This section shows how critical patterns can be specified and automatically eliminated. In order to explain our approach we first describe a specific attack to an e-government system. Even though the cause of this attack is hard to detect on the implementation level the elimination of a suitable critical pattern in the meta-model ensures that this attack cannot occur. For the attack we assume that an intruder got access to the web server already.

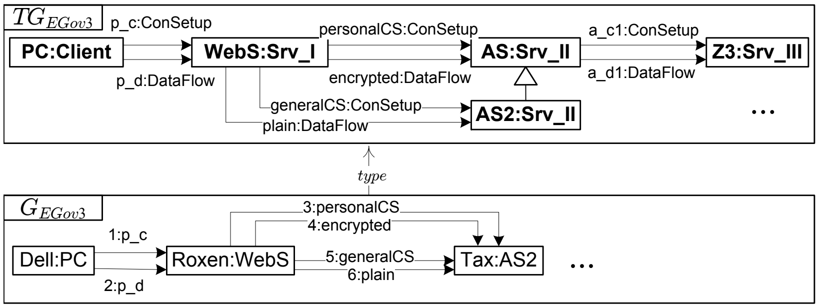


Fig. 6. Configuration for possible attack

Example 5 (Intrusion Attack). Fig. 6 shows a meta-model TG_{EGov3} and an instance G_{EGov3} with clan morphism *type*. There are two types for possible connection setups from server "Roxen" to server "Tax", because of the inheritance relation between “AS2” and “AS” in TG_{EGov3}. Assume that the application server “Tax” in G_{EGov3} processes both confidential requests for receiving and updating personal information for tax declaration via secure encrypted data channel “4:encrypted” and requests for general information regarding dates, laws and submission address for preparing a tax declaration via unencrypted channel “6:plain”. The following sequence describes the intrusion:

- A user requests general information, stays connected and performs a log-in to request in addition also personal information.
- Because of high load of channel 4 a scheduling algorithm on web server “Roxen” decides to transfer some personal data via channel 6.
- The user receives the data, which is not encrypted during the communication.
- The intruder with access to the web server may now observe the insecure communication and intercept some confidential data.

A successful interception of the response is hidden. Even if misuse of confidential data for another service is detected at a later stage, locating the error is hard. Even though the channels were initially assigned correctly according to the kind of data the intrusion happened, because of a side affect of the scheduling algorithm, which is hidden to the model. Hence, possibilities for side effects on the implementation basis should be minimized.

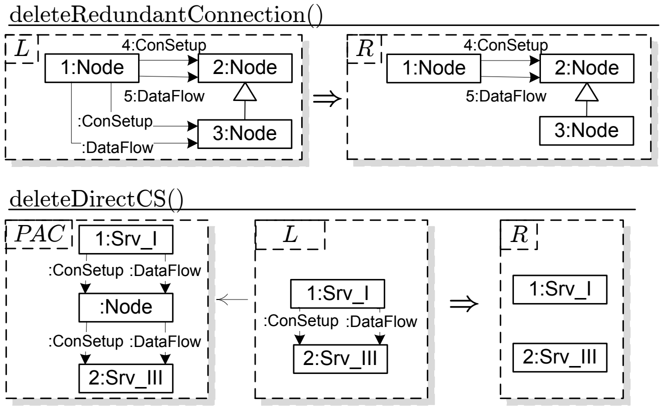


Fig. 7. Checking rules for analysis

Rule “deleteRedundantConnection()” in Fig. 7 can detect the critical pattern of web servers that can communicate via different types of connections simultaneously. A valid match of the rule states a detection and the developer of the model may apply the rule for automatic correction causing the deletion of the more specific connection type. This deletion of edges “generalCS” and “plain” in TG_{EGov3} implies in particular that instance G_{EGov3} is not typed correctly any more, because the edges 5 and 6 cannot be mapped type consistently.

A further rule for analysis and correction is given by "deleteDirectCS()" in Fig. 7. The positive application condition PAC requires a possible connection setup via a proxy node, while the left hand side L already matches a direct connection setup link between a server of zone I and a server of zone III. This situation may easily occur, if verbal requirements for the model are realized directly. Since communication shall only be possible between neighbouring zones this pattern is critical and has to be corrected by applying rule "deleteDirectCS()". Note especially that the pattern is very flexible, because the proxy node is of the general type "Node".

In the following we show how we can apply the well-known results for adhesive HLR systems (see Cor. 1).

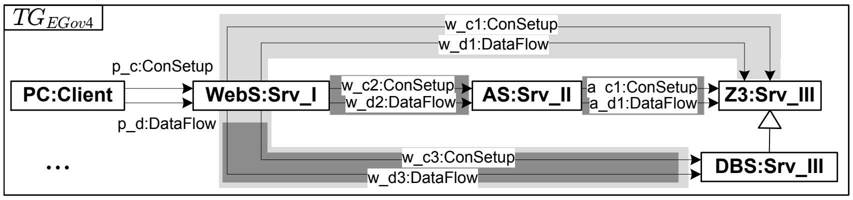


Fig. 8. Conflict situation for rules deleteDirectCS() and deleteRedundantConnection()

Example 6 (Critical Pair). Fig. 8 shows graph TG_{EGov4}, which demonstrates a conflict situation for the rules "deleteDirectCS()" and "deleteRedundantConnection()" (see Fig. 7). Both rules can be applied to this graph and the matches are indicated by dark respectively light grey marked regions, where the first match is a proper clan-morphism. Both matches overlap on edges "w_c3" and "w_d3" that will be deleted by the rule applications. Thus, these rule applications are parallel dependent and there is a conflict of deciding which one to apply. This leads to a critical pair.

If in other situations the rule applications overlap only in their interfaces they are parallel independent and according to the Local Church Rosser and Parallelism Theorem (see Corollary 1) we can apply the rules in any order or in parallel.

According to the general result on completeness of critical pairs (see Corollary 1) there is a critical pair for each possible conflict. Hence, it suffices to calculate all critical pairs using tool support, which is available for standard graph transformation already [1]. If all critical pairs are strictly confluent we can apply the Local Confluence Theorem (see Corollary 1) in order to show that different applications of the analysis rules lead to the same result. Otherwise the aim is to group the analysis rules, such that there is no critical pair between two of the same group. In this way the analysis in each group can be applied in parallel using one parallel rule according to the Parallelism Theorem.

In practical situations meta-models are more complex, which results in a higher amount of node and edge types. Since critical patterns do normally contain only few nodes and edges it is quite usual that several rules are independent from each other and can be put in the same independence group. Therefore, our approach scales up for complex systems, where an automatic critical pattern detection and elimination is highly desirable. Note in particular that pure critical pattern detection without correction will never involve conflicts, since there is no deletion. For this reason it can be parallelized and distributed without calculating critical pairs.

Altogether we can use the results in Corollary 1 for parallel critical pattern detection and analyze how far different orders of the elimination of these patterns lead to the same result.

5 Related Work

In this paper we consider rule-based meta-model transformations in order to change meta-models in a way that makes them adhere to security requirements. This includes refactoring steps, such as inserting supertype nodes. Usually, model refactorings are performed at instance model level. Various approaches exist using graph transformation to provide a formal specification of model refactorings [13–16]. It has the advantage of defining refactorings in a generic way, while still being able to provide tool support in commonly accepted modeling environments such as EMF [17]. In addition, the theory of graph transformation allows the modeller to formally reason about dependencies between different types of refactorings. Synchronized rules are applied in parallel to keep coherence between models. Considering the special case where exactly two parts (one model diagram and the program or two model diagrams) are related, the triple graph grammar (TGG) approach by Schürr et al. [18] is used frequently.

Our transformation approach at meta-model level is most useful during meta-model development to ensure security requirements before instance graphs are created. An interesting line of research is the co-evolution of meta-models of higher levels and the corresponding meta-models at lower levels, down to instance models. Changing one meta level may cause implications for model updates of lower levels to keep them consistent (*migration problem*). A promising approach for automatic migration of instances is described in [19], where meta-model changes are transferred to lower levels by pullback constructions using non-injective morphisms. In this case, the rule morphisms $K \xrightarrow{l} L$ for the meta level transformations have to be non-injective. This leads to non-functional behaviour of DPO rewriting. In [20], SqPO rewriting is introduced, which is an extension of DPO rewriting taking into account this problem.

6 Conclusion

The formal basis for type graphs with inheritance was presented already in [21, 6, 9] and the semantics given by the closure construction coincides with the one of the inheritance concept of the meta-modelling language MOF [5]. For this reason, the presented extension of the theory to transformations of type graphs with inheritance enables DSL modellers to define modifications of meta-models which contain inheritance information. Apart from the presented case study of e-government network security, a wide range of meta-model based application domains are conceivable, in particular hierarchical and integrated systems of meta-models.

The paper showed that graphs with inheritance together with the introduced class of S-reflecting morphisms forms a weak adhesive category. Hence, the introduced formalization of transformations of meta-models allows modellers to apply various techniques for analysis of the meta-modelling process, due to the fact that well-known results for confluence analysis and conflict detection exist for weak adhesive HLR systems [6]. For instance, in the case of the sample scenario, when the necessary meta-model changes of several modellers conflict each other, the formal techniques for merging and conflict detection support a consistent synchronization. And in the case of local changes of parts or views of the model, the changes can be embedded into the overall model if the consistency condition of the Embedding Theorem is fulfilled. Note that the presented approach is suited also in other application domains for checking formally the fulfillment of security requirements during design phase.

Future work on the theoretical formalization will include an analysis of the gluing condition and characterization of critical pairs for transformations of graphs with inheritance. Moreover, the migration problem discussed in Sec. 5 is an important problem when meta-models have to be modified where instance models exist which have to be kept consistent. The SqPO rewriting approach [20] seems to be a good candidate for future extensions of the presented theory in the context of model migration. Finally the critical pair analysis of the tool AGG shall be extended to the case of graphs with inheritance.

References

1. AGG Homepage. http://tfs.cs.tu-berlin.de/agg.
2. Mouratidis, H., Giorgini, P., eds.: Integrating Security and Software Engineering: Advances and Future Vision. Idea Group, IGI Publishing Group (2006)
3. Jürjens, J.: Secure Systems Development with UML. Springer (2005)
4. Haley, C., Moffett, J., Nuseibeh, B.: Security Requirements Engineering: A Framework for Representation and Analysis. IEEE Trans. on Software Engineering **34**(1) (2008) 133–153
5. Object Management Group: Meta-Object Facility (MOF), Version 2.0. (2006) http://www.omg.org/technology/documents/formal/mof.htm.
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science. Springer (2006)
7. Braatz, B., Brandt, C., Engel, T., Hermann, F., Ehrig, H.: An approach using formally well-founded domain languages for secure coarse-grained IT system modelling in a real-world banking scenario. In: Proc. 18th Australasian Conference on Information Systems (ACIS'07) (2007)

8. Federal Office for Information Security (BSI), ed.: Chapter IV: Secure Client-Server Architectures for E-Government. In: E-Government Manual. INTESIO (2006) 1–179 http://www.bsi.bund.de/english/topics/egov/6˙en.htm.
9. Lara, J., Bardohl, R., Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Attributed Graph Transformation with Node Type Inheritance. Theoretical Computer Science **376**(3) (2007) 139–163
10. Habel, A., Heckel, R., Taentzer, G.: Graph Grammars with Negative Application Conditions. Special issue of Fundamenta Informaticae **26**(3,4) (1996) 287–313
11. Ehrig, H., Ehrig, K., Ermel, C., Prange, U.: Consistent Integration of Models Based on Views of Visual Languages. In: Proc. Fundamental Approaches to Software Engineering (FASE'08). Volume 4961 of LNCS, Springer Verlag (2008) 62–76
12. Object Management Group: Unified Modeling Language: Superstructure – Version 2.1.1. (2007) formal/07-02-05,
http://www.omg.org/technology/documents/formal/uml.htm.
13. Mens, T., Taentzer, G., Müller, D.: Model-driven software refactoring. In Rech, J., Bunse, C., eds.: Model-Driven Software Development: Integrating Quality Assurance. Idea Group Inc. (2008) 170–203
14. Mens, T., Taentzer, G., Runge, O.: Analysing refactoring dependencies using graph transformation. Software and System Modeling **6**(3). Springer (2007) 269–285
15. Grunske, L., Geiger, L., Zündorf, A., Van Eetvelde, N., Van Gorp, P., Varro, D.: Using Graph Transformation for Practical Model Driven Software Engineering. In Beydeda, S., Book, M., Gruhn, V., eds.: Model-driven Software Development. Springer Verlag (2005) 91–118
16. Bottoni, P., Parisi-Presicce, P., Mason, G., Taentzer, G.: Specifying Coherent Refactoring of Software Artefacts with Distributed Graph Transformations. In v. Bommel, P., ed.: Handbook on Transformation of Knowledge, Information, and Data: Theory and Applications, Idea Group Publishing (2005) 95–125
17. Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In: Proc. Model Driven Engineering Languages and Systems (MoDELS'06) (2006)
18. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Proc. WG94 Workshop on Graph-Theoretic Concepts in Computer Science. Volume 903 of LNCS, Springer Verlag (1994) 151–163
19. Löwe, M., König, H., Peters, M., Schulz, C.: Refactoring Information Systems. In: Proc. Software Evolution through Transformations: Embracing the Chance (SeTra'06). Volume 3 of Electronic Communications of the EASST (2006)
20. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-Pushout Rewriting. In: Proc. Graph Transformation (ICGT'06). Volume 4178 of LNCS, Springer Verlag (2006) 30–45
21. Bardohl, R., Ehrig, H., de Lara, J., Taentzer, G.: Integrating Meta-Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation. In: Proc. Fundamental Aspects of Software Engineering (FASE'04). Volume 2984 of LNCS, Springer Verlag (2004)
22. Hermann, F., Ehrig, H., Ermel, C.: Transformation of Type Graphs with Inheritance for Ensuring Security in E-Government Networks (Long Version). Technical Report 2008/07, TU Berlin, Fak. IV (2008).
http://iv.tu-berlin.de/TechnBerichte/2008/2008-07.pdf.