Consistent Integration of Models Based on Views of Visual Languages

Hartmut Ehrig[1], Karsten Ehrig[2], Claudia Ermel[1], and Ulrike Prange[1]

[1] Technische Universität Berlin, Germany
ehrig,lieske,uprange@cs.tu-berlin.de
[2] University of Leicester, United Kingdom
karsten@mcs.le.ac.uk

Abstract. The complexity of large system models in software engineering nowadays is mastered by using different views. View-based modeling aims at creating small, partial models, each one of them describing some aspect of the system. Existing formal techniques supporting view-based visual modeling are based on typed attributed graphs, where views are related by typed attributed graph morphisms. Such morphisms up to now require a fixed type graph, as well as a fixed data signature and domain. This is in general not adequate for view-oriented modeling where only parts of the complete type graph and signature are known and necessary when modeling a partial view of the system.

The aim of this paper is to extend the framework of typed attributed graph morphisms to *generalized* typed attributed graph morphisms, short GAG-morphisms, which involve changes of the type graph, data signature, and domain. This allows the modeler to formulate type hierarchies and views of visual languages defined by GAG-morphisms between type graphs, short GATG-morphisms. In this paper we study the interaction and integration of views, and the restriction of views along type hierarchies. In the main result we present suitable conditions for the integration and decomposition of consistent view models. As a running example we use a visual domain-specific modeling language to model coarse-grained IT components and their connectors in decentralized IT infrastructures.

1 Introduction

In recent years, the complexity of large system models in software engineering is mastered by using different views or viewpoints. View-based modeling rather aims at creating small, partial models, each one of them describing some aspect of the system instead of building complex monolithic specifications. Visual techniques nowadays form an important part of the overall software development methodology. Usually, visual notations like the UML [1], Petri nets or other kinds of graphs are used in order to specify static or dynamic system aspects. Hence, the syntax definition of visual modeling languages is an important basis for the implementation of tools supporting visual modeling (e.g. visual editor generation) and for model-based system verification.

Two main approaches to visual language (VL) definition can be distinguished: grammar-based approaches or meta-modeling. Using graph grammars and graph transformation [2], multidimensional representations are described by graphs. Graph rules are used to manipulate the graph representation of a language element. Meta-modeling (see e.g. [3]) is also graph-based, but uses constraints instead of a grammar to define a visual language. The advantage of meta-modeling is that UML users, who probably have basic UML knowledge, do not need to learn a new external notation to be able to deal with syntax definitions. Graph grammars are more constructive, i.e. closer to the implementation, and provide a formal basis for visualizing, validating and verifying system properties.

For the application of graph transformation techniques to VL modeling, typed attributed graph transformation systems and grammars [2] have proven to be an adequate formalism. A VL is modeled by a type graph capturing the definition of the underlying visual alphabet, i.e. the symbols and relations which are available. Sentences or models of the VL are given by graphs typed over (i.e. conforming to) the type graph. Such a VL type graph corresponds closely to a meta model. In order to restrict the set of valid visual models, a syntax graph grammar may be defined, consisting of a set of language-generating graph transformation rules, typed over the abstract syntax part of the VL type graph.

In this paper we extend the graph transformation framework in order to allow an adequate specification of different views and their relations. In the literature, approaches already exist to model views as morphisms between typed attributed graphs [4]. Up to now such morphisms require a fixed type graph, as well as a fixed data signature and domain. This is in general not adequate for view-oriented modeling where only parts of the complete type graph and signature are known and necessary when modeling a partial view of the system. Hence, in this paper we develop the notion of *generalized attributed graph morphisms* (GAG-morphisms) which allows the modeler to change the type graph, data signature and domain. GAG-morphisms are the basis for more flexible, view-oriented modeling since views are independent of each other, now also with respect to the data type definition.

For view-oriented modeling, mechanisms are needed to integrate different views to a full system model. In order to integrate two or more views, their intended correspondences have to be specified. Here, typed graphs and the underlying categorical constructions support an integration concept which goes much further than an integration merely based on the use of common names. In this paper, we define type hierarchies and views based on GAG-morphisms, and study the interaction and integration of views, as well as the restriction of views along type hierarchies, the notion of view consistency, and the integration and decomposition of models based on consistent views.

As a running example we use a visual domain-specific modeling language to model coarse-grained IT components and their connectors in decentralized IT infrastructures. An infrastructure model has to provide the basis to handle structural security issues, like firewall placements, of such distributed IT components. In order to provide support to model, build, administrate, monitor and control such a local IT landscape, we present a formal, visual domain-specific language family based on attributed type graph hierarchies and views. A simplified visual language for this purpose using typed graphs *without* attributes was first introduced in [5], serving as a basis to transform domain-specific IT infrastructure models to a Reo coordination model [6] for further analysis.

The paper is structured as follows: Section 2 defines the category **GAGraphs** of typed attributed graphs and GAG-morphisms, and introduces the sample VL for IT infrastructures. On this basis, views are defined in Section 3, and the view relations *interaction* and *integration* are given by categorical constructions. Moreover, the interplay of type hierarchies of VLs and views is considered. Section 4 studies models of visual languages and models of views (view-models) and states as main result conditions for the consistency, integration and decomposition of view-models. In Section 5, related work is presented and compared to our approach. We conclude and discuss future work in Section 6.

2 Visual Language Definition by Typed Attributed Graphs

We use the meta-model approach in combination with typed attributed graphs to define visual languages. A meta-model is given by an attributed type graph ATG together with structural constraints, and the corresponding visual language VL is given by all attributed graphs typed over ATG which satisfy the constraints. In the following, we introduce the necessary definitions for typed attributed graphs.

The definition of attributed graphs is based on E-graphs, which give a structure for graphs with data elements. An E-graph $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j \in \{G,NA,EA\}})$ has two different kinds of nodes, namely graph nodes V_G and data nodes V_D, and different kinds of edges, namely graph edges E_G and, for the attribution, node attribute edges E_{NA} and edge attribute edges E_{EA}, with corresponding source and target functions according to the signature on the right.

$$\begin{tikzcd}
 & E_G \arrow[rr, bend left, "source_G"] \arrow[rr, "target_G"'] & & V_G & \\
E_{EA} \arrow[ur, "source_{EA}"] \arrow[drr, "target_{EA}"'] & & & & E_{NA} \arrow[ul, "source_{NA}"'] \arrow[dll, "target_{NA}"] \\
 & & V_D & &
\end{tikzcd}$$

As presented in [2], attributed graphs are defined as E-graphs combined with a $DSIG$-algebra, i.e. an algebra over a data signature $DSIG$. In this signature, we distinguish a set of attribute value sorts. The corresponding carrier sets in the $DSIG$-algebra can be used for attribution. In addition to attributed graph morphisms in [2], generalized attributed graph morphisms are mappings of attributed graphs with possibly different data signatures.

Definition 1 (Attributed graph and generalized attributed graph morphism). *An* attributed graph $AG = (G, DSIG, D)$ *consists of*

- *an E-graph* $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j \in \{G,NA,EA\}})$,
- *a data signature* $DSIG = (S, S_D, OP)$ *with attribute value sorts* $S_D \subseteq S$, *and*
- *a* $DSIG$*-algebra* D *such that* $\dot{\bigcup}_{s \in S_D} D_s = V_D$.

Given attributed graphs $AG^i = (G^i, DSIG^i, D^i)$ for $i = 1, 2$, a generalized attributed graph morphism *(GAG-morphism) $f = (f_G, f_S, f_D) : AG^1 \to AG^2$ is given by*

- *an E-graph morphism $f_G : G^1 \to G^2$,*
- *a signature morphism $f_S : DSIG^1 \to DSIG^2$, and*
- *a generalized homomorphism $f_D : D^1 \to D^2$, which is a $DSIG^1$-morphism $f_D : D^1 \to V_{f_S}(D^2)$ with $f_D = (f_{D,s_1} : D^1_{s_1} \to D^2_{f_S(s_1)})_{s_1 \in S^1}$*

with the following compatibility property*: $f_S(S^1_D) \subseteq S^2_D$ and the diagram on the right commutes for all $s_1 \in S^1_D$, where the vertical (curling) arrows are inclusions.*

$$\begin{tikzcd}
D^1_{s_1} \arrow[r, "f_{D,s_1}"] \arrow[d, hook] \arrow[dr, phantom, "="] & D^2_{f_S(s_1)} \arrow[d, hook] \\
V^1_D \arrow[r, "f_{G,V_D}"'] & V^2_D
\end{tikzcd}$$

A GAG-morphism $f = (f_G, f_S, f_D)$ is called

- injective*, if f_G, f_S, f_D are injective,*
- signature preserving*, if f_S is isomorphic,*
- persistent*, if f_D is isomorphic.*

Attributed graphs with generalized attributed graph morphisms form the category **GAGraphs***.*

Note that AG-morphisms in [2] correspond to signature preserving GAG-morphisms.

For the typing, we use a distinguished attributed type graph ATG. According to [2], attributed type graphs and typed attributed graphs are now defined using GAG-morphisms presented above.

Definition 2 (Typed attributed graph and typed attributed graph morphism). *An* attributed type graph *$ATG = (TG, DSIG, Z_{DSIG})$ is an attributed graph where Z_{DSIG} is the final $DSIG$-algebra, i.e. $Z_{DSIG,s} = \{s\}$ for all $s \in S$, and $V_D = \dot{\bigcup}_{s \in S_D} Z_{DSIG,s} = S_D$.*

Given an attributed type graph ATG, a typed attributed graph *$TAG = (AG, t)$ (over ATG) is given by an attributed graph AG and a GAG-morphism $t : AG \to ATG$.*

Given an attributed type graph ATG and typed attributed graphs $TAG^i = (AG^i, t : AG^i \to ATG)$ over ATG for $i = 1, 2$, a typed attributed graph morphism *$f : TAG^1 \to TAG^2$ is given by a GAG-morphism $f : AG^1 \to AG^2$ such that $t_2 \circ f = t^1$.*

Given an attributed type graph ATG, typed attributed graphs over ATG and typed attributed graph morphisms form the category **GAGraphs$_{\mathbf{ATG}}$***.*

As a special case of GAG-morphisms we obtain generalized attributed type graph morphisms based on attributed type graphs.

Definition 3 (Generalized attributed type graph morphism). *Given attributed type graphs $ATG^i = (TG^i, DSIG^i, Z_{DSIG^i})$ for $i = 1, 2$, a* generalized attributed type graph morphism *(GATG-morphism) $f = (f_G, f_S, f_D) : ATG^1 \to ATG^2$ is given by*

- *an E-graph morphism $f_G : TG^1 \to TG^2$,*
- *a signature morphism $f_S : DSIG^1 \to DSIG^2$, and*
- *a generalized homomorphism $f_D : Z_{DSIG^1} \to Z_{DSIG^2}$, which is uniquely determined by $f_{D,s_1}(s_1) = f_S(s_1)$ for all $s_1 \in S^1$.*

A GATG-morphism f is also a GAG-morphism since the compatibility property is automatically satisfied because $f_{G,V_D}(s_1) = f_S(s_1)$ for all $s_1 \in S^1_D$ and f_D, f_{G,V_D} are uniquely determined by f_S. Moreover, if f is a GATG-morphism then f is persistent.

Now we are able to define visual languages. For simplicity, we consider only visual languages over attributed type graphs, without any constraints. For the case with constraints we refer to [7].

Definition 4 (Visual language). *Given an attributed type graph ATG, the* visual language *VL of ATG consists of all typed attributed graphs $(AG, t : AG \to ATG)$ typed over ATG, i.e. VL is the object class of the category* $\mathbf{GAGraphs_{ATG}}$.

Example 1 (VL for network infrastructures). Fig. 1 shows at the top the attributed type graph ATG_{DSL} which represents a meta-meta model (or schema) for domain-specific languages for IT infrastructures. The DSL schema defines that all its instances (domain-specific languages) consist of node types for components, connections and interfaces. In the center of Fig. 1, the attributed type graph $ATG_{Network}$ defines a simple modeling language for network infrastructures which has component types for personal computers (PC), application servers (AS), and databases (DB). Interfaces are refined into HTTP-client and HTTP-server ports, as well as database client and server ports. Connections may be secure (i.e. with firewall) or insecure, which is modeled by the new boolean attribute secure.

There is a generalized attributed type graph morphism h from $ATG_{Network}$ to ATG_{DSL}, indicated by equal numbering of mapped nodes. Note that in order to be able to define the signature morphism f_S and the $DSIG$-morphism f_D for any GAG-morphisms$f : ATG_1 \to ATG_2$ between different type graphs, we assume that each node type in ATG_2 has at least one sort "*", and one attribute $attr : *$, where all sorts and attributes from ATG_1 can be mapped to which are not already defined in ATG_2. Thus we can have new attributes, sorts and methods at the more detailed type level ATG_1 which need not be defined already in ATG_2. For our sample GAG-morphism h in Fig. 1, this is the case for the new attribute $secure : Bool$ of the type Connection in $ATG_{Network}$. The new sort $Bool$ is mapped by the signature morphism to the sort "*", and the attribute $secure$ is mapped by the $DSIG$-morphism to the constant $attr$.

At the bottom of Fig. 1, a sample computer network is depicted as graph $G_{Network}$ which is an element of the visual $Network$ language since $G_{Network}$ is typed over $ATG_{Network}$: $(G_{Network}, t : G \to ATG_{Network}) \in VL_{Network}$. Obviously, all graphs G in $VL_{Network}$ are also in VL_{DSL}, since every $(G, t : G \to ATG_{Network})$ is also typed over ATG_{DSL} by the composition of typing morphisms: $(G, h \circ t : G \to ATG_{DSL}) \in VL_{DSL}$.

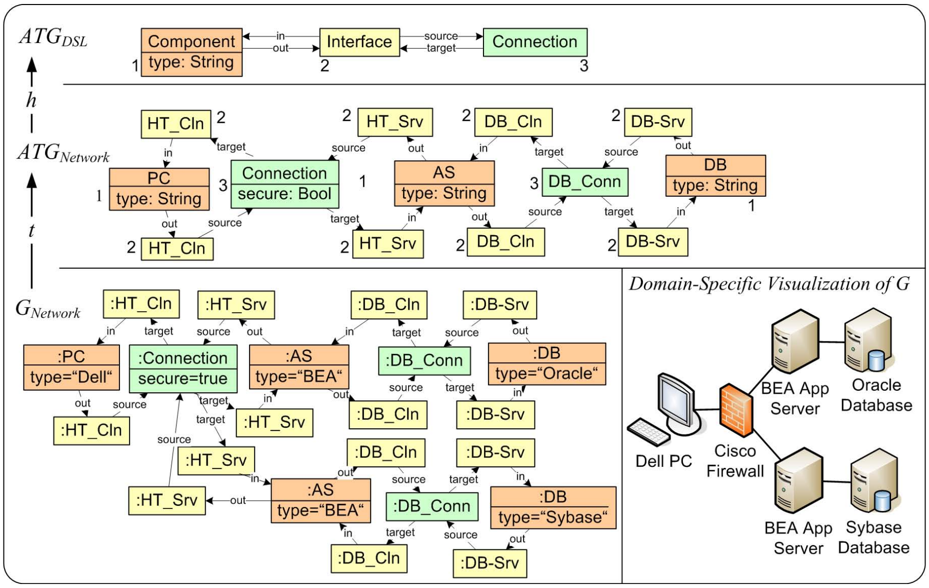

Fig. 1. *Example 1:* Domain-Specific Languages for IT Infrastructures

3 Type Hierarchies and Views of Visual Languages

In this section, we study type hierarchies and views of visual languages based on morphisms in **GAGraphs**, which allow to change not only the graph structure but also the data signature and data type. Note that in this section we only consider the attributed type graphs and their relations, but not yet models over them. This is done in the next section.

A restriction of a visual language to a specific subpart of the language is called a view.

Definition 5 (View). *A* view *of a visual language VL over an attributed type graph ATG is given by an injective $GATG$-morphism $v_1 : ATG_1 \to ATG$.*

For the interaction and integration of views we need the categorical constructions of pullbacks and pushouts in **GAGraphs**. Proofs for the pushout and pullback construction lemmas are given in [7]. Pullbacks are a kind of generalized intersection of objects over a common object.

Lemma 1 (Pullback construction in GAGraphs). *Given GAG-morphisms $f : AG^2 \to AG^3$ and $g : AG^1 \to AG^3$ then the pullback in* **GAGraphs** *is constructed componentwise in the G-, S- and D-components. Moreover, pullbacks preserve injective, signature preserving, and persistent morphisms.*

Pushouts generalize the gluing of objects, i.e. a pushout emerges from the gluing of two objects along a common subobject using the amalgamation of data types in the sense of [15].

Lemma 2 (Pushouts in GAGraphs over persistent morphisms). *Given persistent morphisms $f' : AG^0 \to AG^1$ and $g' : AG^0 \to AG^2$ in* **GAGraphs** *then the pushout* (1) *in* **GAGraphs** *is constructed componentwise in the G- and S-components, with attribute value sorts $S_D^3 = g_s(S_D^1) \cup f_S(S_D^2)$, and in the D-component by amalgamation as $D_3 = D^1 +_{D^0} D^2$. Moreover, pushouts preserve injective, signature preserving, and persistent morphisms.*

$$\begin{tikzcd}
AG^0 = (G^0, DSIG^0, D^0) \arrow[r, "f'=(f'_G, f'_S, f'_D)"] \arrow[d, "g'=(g'_G, g'_S, g'_D)"'] \arrow[dr, phantom, "(1)"] & (G^1, DSIG^1, D^1) = AG^1 \arrow[d, "g=(g_G, g_S, g_D)"] \\
AG^2 = (G^2, DSIG^2, D^2) \arrow[r, "f=(f_G, f_S, f_D)"'] & (G^3, DSIG^3, D^3) = AG^3
\end{tikzcd}$$

Based on the concepts of pullbacks and pushouts, we are now able to define the interaction and integration of views. Roughly spoken, the interaction is the intersection, and the integration is the union of views.

Definition 6 (Interaction and integration of views). *Given views (ATG_1, v_1) and (ATG_2, v_2) over ATG the* interaction *(ATG_0, i_1, i_2) is given by the following pullback* (1) *in* **GAGraphs***, where (ATG_0, v_0) with $v_0 = v_1 \circ i_1 = v_2 \circ i_2$ is a view over ATG and also called subview of (ATG_1, v_1) and (ATG_2, v_2).*

$$\begin{tikzcd}
 & ATG_1 \arrow[dr, "v_1"] & \\
ATG_0 \arrow[ur, "i_1"] \arrow[dr, "i_2"'] \arrow[rr, bend right=15, "v_0"'] & (1) & ATG \\
 & ATG_2 \arrow[ur, "v_2"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & ATG_1 \arrow[dr, "w_1"'] \arrow[drr, "v_1"] & & \\
ATG_0 \arrow[ur, "i_1"] \arrow[dr, "i_2"'] & (2) & ATG_3 \arrow[r, "v_3"'] & ATG \\
 & ATG_2 \arrow[ur, "w_2"] \arrow[urr, "v_2"'] & &
\end{tikzcd}$$

The integration *of views (ATG_1, v_1) and (ATG_2, v_2) with interaction (ATG_0, i_1, i_2) is given by the above pushout* (2) *in* **GAGraphs***. Due to the universal pushout property there is a unique injective GATG-morphism $v_3 : ATG_3 \to ATG$ such that (ATG_3, v_3) is a view over ATG.*

ATG is covered *by views (ATG_i, v_i) with $i = 1, 2$ if v_1 and v_2 are jointly surjective.*

There is a close relationship between covering by views and view integration.

Fact 1 (Integration of views). *If ATG is covered by views (ATG_i, v_i) for $i = 1, 2$ then the integration ATG_3 is equal to ATG up to isomorphism.*

Proof. According to Def. 6, there is a unique morphism v_3 with $v_3 \circ w_1 = v_1$ and $v_3 \circ w_2 = v_2$. This morphism is injective in the G- and S-components due to general properties of graph and signature morphisms, and v_3 is injective in the D-component as a general property of GATG-morphisms. Surjectivity of v_3 follows from joint surjectivity of v_1 and v_2.

Example 2 (Interaction and integration of views on IT networks). Fig. 2 shows two views $(ATG_{Components}, v_1)$ and $(ATG_{Connections}, v_2)$ of the visual language over ATG_{DSL} (see Fig. 1). The type graph $ATG_{Components}$ consists of a node type for Computer linked to a node type for Port, whereas the type graph $ATG_{Connections}$ contains a node type Channel which is linked to a node type ChEnd. The view embedding v_1 maps Computer to Component and Port to Interface, and v_2 maps Channel to Connection and ChEnd to Interface. Edges are mapped accordingly. The interaction $(ATG_{interaction}, i_1, i_2)$ is constructed as pullback (1) in **GAGraphs** which is the intersection of v_1 and v_2 with suitable renaming. Given the interaction, the integration of the views $(ATG_{Components}, v_1)$ and $ATG_{Connections}, v_2$ over $(ATG_{Interaction}, i_1, i_2)$ can be constructed as pushout (2) in **GAGraphs**, resulting in the type graph $(ATG_{Integration})$. According to Fact 1, $(ATG_{Integration})$ is isomorphic to ATG_{DSL}, since ATG_{DSL} is covered by $(ATG_{Components}, v_1)$ and $(ATG_{Connections}, v_2)$.

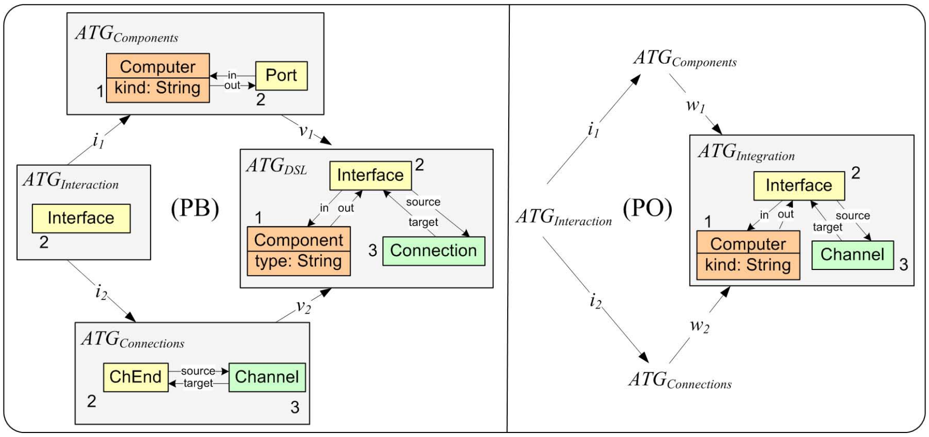

Fig. 2. *Example 2:* Interaction and Integration of two Views on ATG_{DSL}

In order to support stepwise language development, visual languages can be structured hierarchically: one attributed type graph ATG may specify the abstract concepts a set of visual languages VL_i have in common, and different type graphs ATG_i for these visual languages refine the types in ATG by specifying multiple concrete subtypes for them. The type hierarchy relation is formalized by GATG-morphisms h_i from ATG_i to ATG. The morphism $h : ATG_{Network} \to ATG_{DSL}$ depicted in Fig. 1 is such a type hierarchy morphism. The next step is to define the restriction of views along type hierarchies by pullbacks.

Definition 7 (Type hierarchy and restriction of views). *A type hierarchy of visual languages VL and VL' given by attributed type graphs ATG and ATG', respectively, is a GATG-morphism $h : ATG' \to ATG$.*

Given a type hierarchy morphism $h : ATG' \to ATG$ and a view (ATG_1, v_1) over ATG then the restriction *(ATG'_1, v'_1) of this view along h is defined by the pullback (1) in* **GAGraphs**.

The restriction (ATG'_1, v'_1) is a view over ATG' because pullbacks preserve injectivity.

$$\begin{tikzcd}
ATG'_1 \arrow[r, "h'"] \arrow[d, "v'_1"'] \arrow[dr, phantom, "(1)"] & ATG_1 \arrow[d, "v_1"] \\
ATG' \arrow[r, "h"'] & ATG
\end{tikzcd}$$

Fact 2 (Hierarchy and covering views). *Given a hierarchy morphism $h : ATG' \to ATG$ and views (ATG_i, v_i) for $i = 1, 2$ covering ATG, then the restrictions (ATG'_i, v'_i) along h are covering ATG'.*

Proof. In the diagram to the right, v_1 and v_2 being jointly surjective implies that also v'_1 and v'_2 are jointly surjective because (1) and (2) are componentwise pullbacks.

$$\begin{tikzcd}
ATG'_1 \arrow[rr, "h_1"] \arrow[dr, "v'_1"'] & & ATG_1 \arrow[dr, "v_1"] & \\
& ATG' \arrow[rr, "h"] \arrow[r, phantom, "(1)" near end, shift left=4ex] \arrow[r, phantom, "(2)" near end, shift right=4ex] & & ATG \\
ATG'_2 \arrow[rr, "f_2"'] \arrow[ur, "v'_2"] & & ATG_2 \arrow[ur, "v_2"'] &
\end{tikzcd}$$

Example 3 (Hierarchy and covering views). The morphism $h : ATG_{Network} \to ATG_{DSL}$ in Fig. 1 is a type hierarchy morphism. Moreover, we have two views $(ATG_{Components}, v_1)$ and $(ATG_{Connections}, v_2)$ on ATG_{DSL}, shown in Fig. 2, which are covering ATG_{DSL}. Fig. 3 shows the restrictions v'_1 and v'_2 of the views along the hierarchy morphism h which are covering $ATG_{Network}$ due to Fact 2.

4 Models and View-Models of Visual Languages

In this section we study models of visual languages and of views of visual languages, called view-models, and we present our main result on the integration and decomposition of models.

Definition 8 (Model). *Given a meta-model of a visual language VL by an attributed type graph ATG, then a* model *of VL is a typed attributed graph AG, typed over ATG with a GAG-morphism $t : AG \to ATG$.*

The model (AG, t) is called signature-conform *if t is signature-preserving.*

Similar to the restriction of views at the type level we now define the restriction of models at the model level.

Definition 9 (Restriction). *Given a view $f : ATG_1 \to ATG$, i.e. an injective GATG-morphism, and an ATG-model (AG, t) then the* restriction *(AG_1, t_1) of (AG, t) to the view (ATG_1, f) is defined by the pullback (1), written $f^<(AG, t) = (AG_1, t_1)$.*

$$\begin{tikzcd}
AG_1 \arrow[r] \arrow[d, "t_1"'] \arrow[dr, phantom, "(1)"] & AG \arrow[d, "t"] \\
ATG_1 \arrow[r, "f"'] & ATG
\end{tikzcd}$$

The construction $f^<(AG, t)$ is called backward typing and can be extended to a functor $f^<(AG, t) : \mathbf{GAGraphs_{ATG}} \to \mathbf{GAGraphs_{ATG_1}}$, as opposed to the extension of view models defined by forward typing $f^>(AG_1, t_1) = (AG_1, f \circ t_1)$.

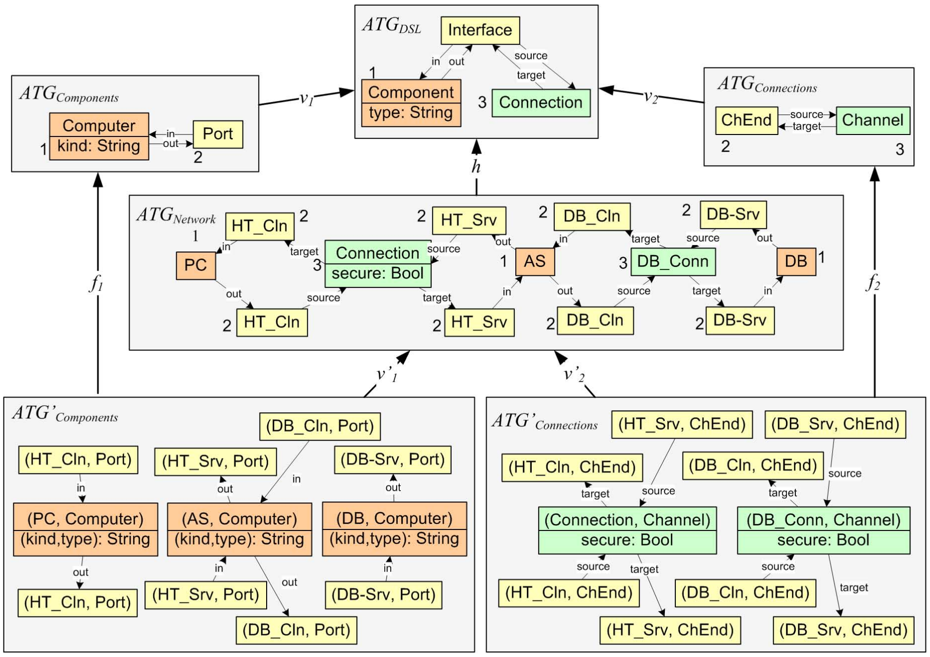

Fig. 3. *Example 3:* Restriction of two Views along Hierarchy Morphism h

In order to state the main result on integration and decomposition of models, we have to define the notions of consistency and integration for models. Roughly, models AG_1 and AG_2 of type ATG_1 and ATG_2, respectively, are consistent if they agree on the interaction type ATG_0. In this case, there is an integrated model AG such that the restrictions of AG to ATG_1 and to ATG_2 are equal to the given models AG_1 and AG_2, respectively.

Definition 10 (Consistency and integration). *Given views (ATG_i, v_i) for $i = 1, 2$ of ATG with interaction (ATG_0, i_1, i_2) defined by the pullback in the bottom face of the following cube, then the models (AG_i, t_i) of the views (ATG_i, v_i) are called* consistent *if there is a model (AG_0, t_0) of ATG_0 such that the back faces are pullbacks, i.e. $i_1^<(AG_1, t_1) = (AG_0, t_0) = i_2^<(AG_2, t_2)$.*

A model (AG, t) of ATG is called integration *(or* amalgamation*) of consistent (AG_1, t_1) and (AG_2, t_2) via (AG_0, t_0) if the front faces of the above cube are pullbacks, i.e. $v_1^<(AG, t) = (AG_1, t_1)$ and $v_2^<(AG, t) = (AG_2, t_2)$, and the top face commutes.*

$$\begin{tikzcd}
 & AG_0 \arrow[dl, "j_1"'] \arrow[dr, "j_2"] \arrow[ddd, "t_0"] & \\
AG_1 \arrow[dr] \arrow[ddd, "t_1"'] & & AG_2 \arrow[dl] \arrow[ddd, "t_2"] \\
 & AG \arrow[ddd, "t"] & \\
 & ATG_0 \arrow[dl, "i_1"'] \arrow[dr, "i_2"] & \\
ATG_1 \arrow[dr, "v_1"'] & & ATG_2 \arrow[dl, "v_2"] \\
 & ATG &
\end{tikzcd}$$

Example 4 (Inconsistent models). Consider the view models AG_1 and AG_2 in Fig. 4. These models are inconsistent since the squares (1) and (2) are pullbacks corresponding to the back squares of the cube in Def. 10, but the resulting pullback objects AG_0 and AG'_0 are different (and non-isomorphic), so we have $i_1^<(AG_1, t_1) = (AG_0, t_0) \neq i_2^<(AG_2, t_2) = (AG'_0, t'_0)$. In this case, there is no integration (AG, t) s.t. $v_1^<(AG, t) = (AG_1, t_1)$ and $v_2^<(AG, t) = (AG_2, t_2)$.

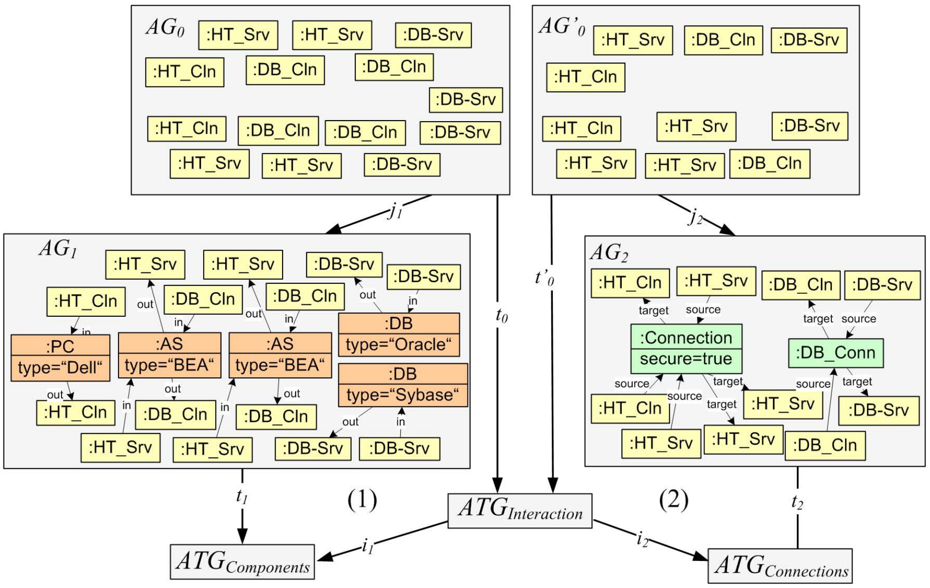

Fig. 4. *Example 4:* Inconsistent View Models

Theorem 1 (Integration and decomposition of models). *Let ATG be covered by the views (ATG_i, v_i) for $i = 1, 2$.*

Integration. *If (AG_i, t_i) are consistent models of (ATG_i, v_i) via (AG_0, t_0) then there is up to isomorphism a unique integration (AG, t) of (AG_i, t_i) via (AG_0, t_0).*

Decomposition. *Vice versa, each model (AG, t) of ATG can be decomposed uniquely up to isomorphism into view-models (AG_i, t_i) with $i = 1, 2$ such that (AG, t) is the integration of (AG_1, t_1) and (AG_2, t_2) via (AG_0, t_0).*

Bijective Correspondence. *Integration and decomposition are inverse to each other up to isomorphism.*

Proof
Integration. Since ATG is covered by (ATG_i, v_i) for $i = 1, 2$ it is also the integration of these views by Fact 1. This means that the bottom pullback is already a pushout in **GAGraphs** with injective and persistent morphisms. Now assume that (AG_i, t_i) with $i = 1, 2$ are consistent models. This means that the back

faces of the cube in Def. 10 are pullbacks with injective and persistent j_1 and j_2. This allows to construct AG in the top face as pushout in **GAGraphs** leading to a unique t such that the front faces commute. According to a suitable van Kampen property (see [7]), the front faces are pullbacks such that (AG, t) is the integration of (AG_i, t_i) for $i = 1, 2$ via (AG_0, t_0). In order to show the uniqueness let also $(AG', t' : AG' \to ATG)$ be an integration of (AG_i, t_i) for $i = 1, 2$ via (AG_0, t_0). Then the front faces are pullbacks with (AG', t') and the top face commutes. Now the van Kampen property in the opposite direction implies that the top face is a pushout in **GAGraphs**. This implies that (AG, t) and (AG', t') are equal up to isomorphism.

Decomposition. Vice versa, given a model (AG, t) of ATG we construct the front and one of the back faces as pullbacks such that the remaining back face also becomes a pullback and the top face commutes. This shows that (AG_1, t_1) and (AG_2, t_2) are consistent w.r.t (AG_0, t_0), and, similar to the previous step, (AG, t) is the integration of both via (AG_0, t_0). The decomposition is unique up to isomorphism because the pullbacks in the front faces are unique up to isomorphism.

Bijective Correspondence. Uniqueness of integration and decomposition as shown above implies that both constructions are inverse to each other up to isomorphism.

Example 5 (Integration and decomposition of models). The graph $G_{Network}$ from Fig. 1 is a model, typed over $ATG_{Network}$. From the two views $ATG'_{Components}$ and $ATG'_{Connections}$ given in Fig. 3 we can construct two consistent view models $G_{Components}$ and $G_{Connections}$ in Fig. 5 according to the *Decomposition* in Thm. 1 such that $G_{Network}$ is the integration of $G_{Components}$ and $G_{Connections}$ via $G_{interaction}$. Vice versa, starting with consistent models $G_{Components}$ and $G_{Connections}$, via $G_{interaction}$ we obtain $G_{Network}$ as the integration.

5 Related Work

Viewpoint-oriented software development is well-known in the literature [8, 9, 4], however identifying, expressing, and reasoning about meaningful relationships between view models is hard [10]. Up to now existing formal techniques for visual modeling of views and distributed systems by graph transformation support the definition of non-hierarchical views which require a common fixed data signature [2, 11]. This is in general not adequate for view-oriented modeling where only parts of the complete type graph and signature are known and necessary when modeling a view of the system. Moreover, hierarchical relations between views could not be defined on the typing and data type level resulting in a lack of composition and decomposition techniques for view integration, verification, and analysis.

In [12] domain specific languages are defined using graphical and textual views based on the meta-modeling approach used in the $AToM^3$ tool. In this approach the language designer starts with the common (integrated) meta-model and selects parts of the meta-model as different diagram views. So a common abstract meta-model is missing allowing to define hierarchical relations between the models.

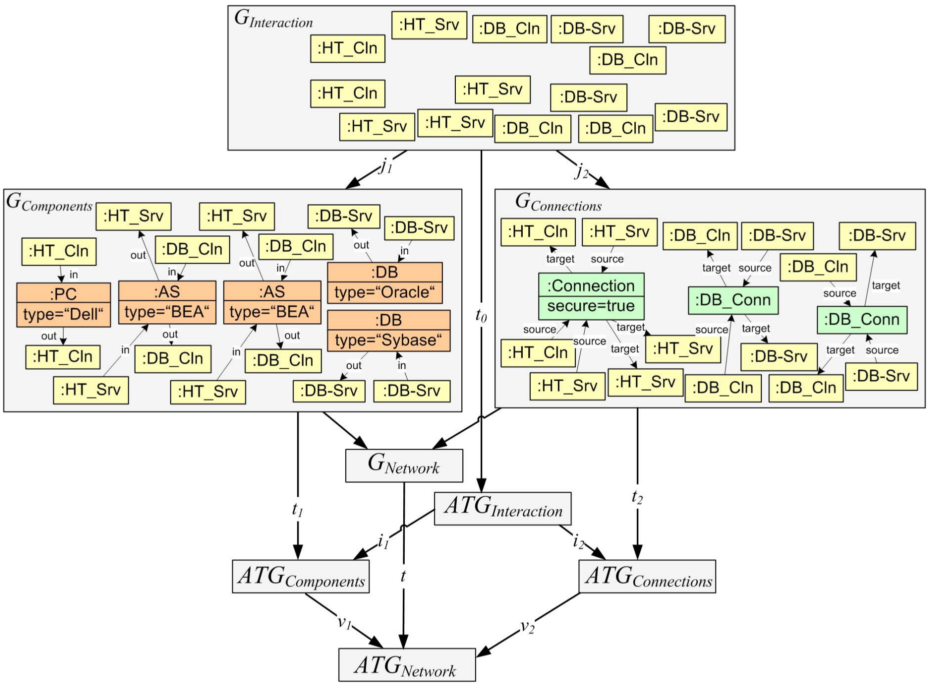

Fig. 5. *Example 5:* Integration and Decomposition of View Models

In [13] abstract graph views are defined, abstracting from specification details allowing a convenient usage of modules. To fulfill this purpose, reference relations have been introduced for the definition of mapping between view elements and abstract model elements (e.g. the database). Given this relations, there are different semantics for modifying view objects which are not studied yet in full detail. In comparison with the presented approach, generalized attributed graph morphisms have a unique formal semantics on the one hand and they provide the flexibility to define hierarchical relations on the other hand.

As a related approach *xlinkit* [14] provides rule-based link generation in web content management systems. In this approach semantics are defined using first order logic allowing automatic link generation to manage large document repositories. According to its purpose, this approach is limited to XML documents using XPath and XLink and thus requires an XML based storage format for models.

6 Conclusion

In this paper we have studied the interaction and integration of views and the restriction of views along type hierarchies. The main result shows under which condition models of these views can be composed to a unique integrated model.

The condition is called *consistency* of view models which means roughly that the models agree on the interaction type of the views. Vice versa, each model can be decomposed up to isomorphism into consistent models of given views. The paper is based on an extended version of typed attributed graph morphisms which allow changes of the type graph including those of data signatures and domains. In this paper we have considered visual languages based on meta-models given by attributed type graphs without constraints. But we claim that most of the results in this paper can be extended to visual languages including constraints and/or generating grammars. Together with full proofs of all technical lemmas used in this paper, some of the extended results are given in our technical report [7].

An important consequence of our work is that we provide the ability to rapidly compose "small" visual languages both at the view (type graph) level and at the view-model level, thus laying the formal basis for multi-view modeling environments. Hence, rather than a "one modeling language does all" approach, we favor a confederation of small, relatively orthogonal visual languages for different system aspects. Future work is planned to investigate the interplay of views and models with behaviour, which is related to the field of merging behavioural models [16, 17].

The concept of type hierarchies should allow a language designer to adapt language definitions by performing model transformations at an abstract hierarchy level and "inheriting" the transformation results at the more concrete levels of the hierarchy. Work is in progress to analyze model transformations for hierarchically structured visual languages.

References

[1] Object Management Group: Unified Modeling Language: Superstructure – Version 2.0, Revised Final Adopted Specification, ptc/04-10-02 (2004), `http://www.omg.org/cgi-bin/doc?ptc/2004-10-02`

[2] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. In: EATCS Monographs in Theor. Comp. Science, Springer, Heidelberg (2006)

[3] Object Management Group: Meta-Object Facility (MOF), Version 1.4 (2005), `http://www.omg.org/technology/documents/formal/mof.htm`

[4] Engels, G., Ehrig, H., Heckel, R., Taentzer, G.: A combined reference model- and view-based approach to system specification. Int. Journal of Software and Knowledge Engineering 7(4), 457–477 (1997)

[5] 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 (2007)

[6] Arbab, F.: Reo: A channel-based coordination model for component composition. Mathematical Structures in Computer Science 14(3), 329–366 (2004)

[7] Ehrig, H., Ehrig, K., Ermel, C., Prange, U.: Generalized typed attributed graph transformation systems based on morphisms changing type graphs and data signatures. Technical report, TU Berlin (2008), `http://tfs.cs.tu-berlin.de/publikationen/Papers08/EEEP08a.pdf`

[8] Goedicke, M., Enders, B., Meyer, T., Taentzer, G.: ViewPoint-Oriented Software Development: Tool Support for Integrating Multiple Perspectives by Distributed Graph Transformation. In: Schwartzbach, M.I., Graf, S. (eds.) TACAS 2000. LNCS, vol. 1785, pp. 43–47. Springer, Heidelberg (2000)
[9] Goedicke, M., Meyer, T., Taentzer, G.: ViewPoint-oriented Software Development by Distributed Graph Transformation: Towards a Basis for Living with Inconsistencies. In: Proc. 4th IEEE Int. Symposium on Requirements Engineering, IEEE Computer Society, Los Alamitos (1999)
[10] Nuseibeh, B., Finkelstein, A., Kramer, J.: ViewPoints: Meaningful Relationships are difficult. In: Proc. Int. Conf. on Software Engineering (ICSE 2003), IEEE Computer Society, Los Alamitos (2003)
[11] Guerra, E., Diaz, P., de Lara, J.: A Formal Approach to the Generation of Visual Language Environments Supporting Multiple Views. In: Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 205), IEEE Computer Society, Los Alamitos (2005)
[12] Andrés, F.P., de Lara, J., Guerra, E.: Domain Specific Languages with Graphical and Textual Views. In: Proc. Third Int. Symposium of Application of Graph Transformation with Industrial Relevance (AGTIVE 2007). LNCS, pp. 79–94. Springer, Heidelberg (to appear)
[13] Ranger, U., Gruber, K., M., H.: Defining Abstract Graph Views as Module Interfaces. In: Proc. Third Int. Symposium of Application of Graph Transformation with Industrial Relevance (AGTIVE 2007). LNCS, pp. 117–133. Springer, Heidelberg (to appear)
[14] Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A.: xlinkit: A Consistency Checking and Smart Link Generation Service. In: of Computer Science, D., ed.: University College London (2007)
[15] Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. In: EATCS Monographs on Theoretical Computer Science, vol. 6, Springer, Heidelberg (1985)
[16] Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., Sabetzadeh, M.: A Manifesto for Model Merging. In: Proc. of the Int. Workshop on Global Integrated Model Management (GaMMa 2006), pp. 5–12. ACM Press, New York (2006)
[17] Uchitel, S., Chechik, M.: Merging Partial Behavioural Models. In: Proc. of the 12th Int.ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 43–52. ACM Press, New York (2004)