A Formal Approach to the Generation of Visual Language Environments Supporting Multiple Views

Esther Guerra, Paloma Díaz
Ingeniería Informática
Universidad Carlos III
Madrid, Spain
{eguerra, pdp}@inf.uc3m.es

Juan de Lara
Escuela Politécnica Superior
Universidad Autónoma
Madrid, Spain
jdelara@uam.es

Abstract

In this paper we present a formal approach, based on meta-modelling and graph transformation, to the generation of environments for visual languages (VLs) supporting multiple views. The VL syntax is defined by means of a meta-model. Views for the VL are created by selecting the classes, associations, attributes and constraints that are part of them. Once the environment is generated, the view models are translated into a global, unique model where consistency checking and further analysis can be performed by means of graph grammars. These ideas have been newly implemented in the AToM^3 tool.

1. Introduction

As software and physical systems become increasingly complex, notations able to describe them using different views become necessary. In this way, modelling is shifting from complex monolithic specifications towards sets of smaller, heterogeneous, partial specifications, each one of them describing some aspect or view of the system.

A usual approach to describe the syntax of a VL is to build a specification showing the concepts and relations present in the language. This specification is called a meta-model, and describes all the valid models the user can build. If the VL includes different diagrams to specify different views of the system, the meta-model should contain all the elements that may appear in each one of them. However, several problems arise. First, the set of elements belonging to each view has to be defined. Note that the same element may be present in different views. Second, customized environments for each view have to be generated and integrated. Finally, the environments must ensure that the system, made up of the different view models, is consistent.

We propose a framework to generate environments supporting VLs with multiple views. The VL is defined with a meta-model, and the views are restricted parts of it. Triple graph grammars [7] are used to translate each view model into a unique model, instance of the whole VL meta-model. Consistency between views is achieved by propagating changes performed in one view to the global model, and from there to the other views. Further checkings can be performed on the global model using regular grammars.

2. A Formal Model of Views and Consistency

VLs are frequently described using meta-models. Meta-modelling allows the definition of the structure of the admissible VL models by defining a model of their (usually abstract) syntax. This model is called a meta-model. It can be expressed as a class diagram containing the entities and relations of the VL syntax, together with their attributes. When the meta-model is equipped with additional information regarding visualization (concrete syntax) and additional constraints, tools can automatically generate modelling environments for the VL.

For VLs with multiple views, in the simpler case, views are (possibly overlapping) parts of the global meta-model. Thus, one has to specify the classes, relations and constraints that conform each view. Moreover, for each class and relation, a view may show only a part of their attributes. In categorical terms, the VL meta-model is the co-limit object of each pair of views and their overlapping parts. This can be seen as a pushout star. The user may build instances of the different view meta-models, and a unique model (instance of the global VL meta-model) can be built by applying some functions (that we implement using graph transformation). The unique model, that we called *repository*, is again the co-limit of all the created view models.

We use graph transformation [3] as a formal means, based on rules, to manipulate graphs. Rules have graphs in the left and right hand sides (LHS and RHS). When apply-

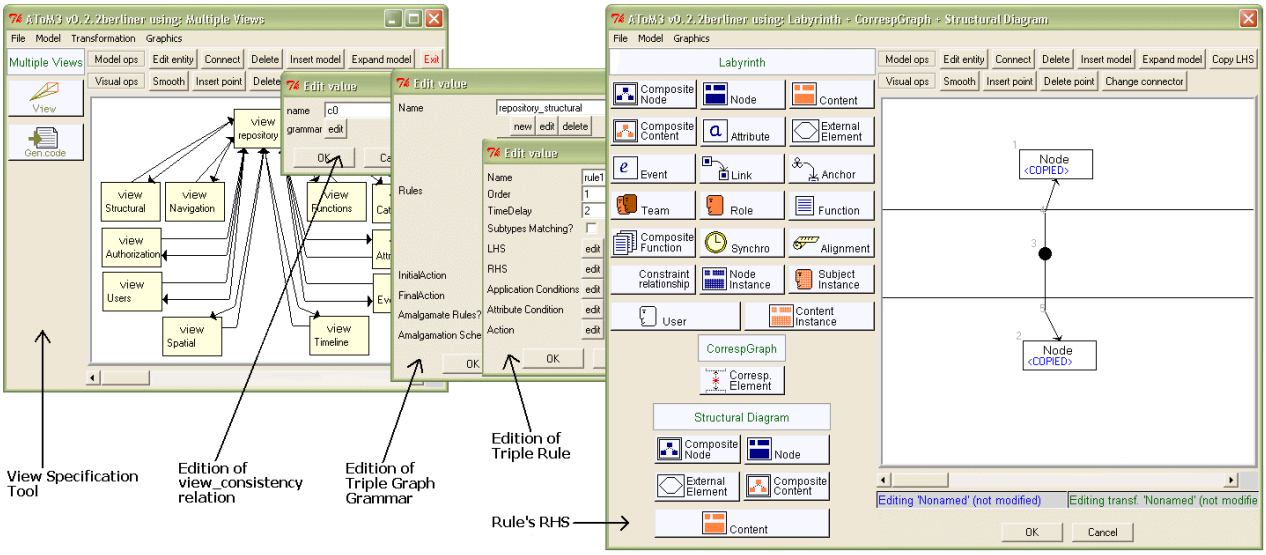

Figure 1. The View Specification Tool.

ing a rule to a graph, a morphism should be found between the LHS of the rule and the graph. Then, the substitution by the RHS can be performed. We follow the Double Pushout Approach (DPO) [3] which is based on category theory to formalize rules and rule applications. Similarly, triple graph grammars [7] rewrite graph triples instead of simple ones. In this work, we build graph triples made of two view models and an intermediate graph that relates objects of the two views. Usually, one of the views is the *repository model*. Thus, the triple graph grammar specifies how the "gluing" of the view models in the *repository* should be done.

3. Implementation in AToM^3

AToM^3 [1] is a meta-modelling tool in which model manipulation (simulation, optimisation, translation into another formalism and code generation) can be expressed by means of graph rewriting. Up to now, AToM^3 only allowed the definition of formalisms with a single view. Different views of the same VL had to be defined in separate meta-models. Thus, a number of different, independent tools were generated. We have improved the tool incorporating a facility to overcome this problem. The new function has been automatically generated from a meta-model built in AToM^3, and then integrated into it. The meta-model was made of a single class (*View*) used to define the view meta-models, and a relationship (*view_consistency*) to specify consistency checkings by means of triple graph grammars. The so-generated View Specification Tool is shown to the left of Figure 1. It can be opened once the whole meta-model of a VL has been defined. A view called *repository*, containing the complete meta-model, is always added by default in the View Specification Tool. The *repository* is used to keep the unique model containing the "glued" views. VL designers can add as many views as necessary. New views initially contain the complete VL meta-model, which can be edited afterwards. A restrictive approach has been implemented, in the sense that view meta-models must be subsets of the complete one.

Consistency mechanisms between views can be provided by means of relationship *view_consistency*, where triple graph grammars [7] can be defined with this purpose. At least two consistency relations between each view and the *repository* must be defined. The first one must copy the changes performed in the view models to the *repository*, in order to build a unique model gluing all the view models. The second one must propagate changes from the *repository* to the other views, if necessary, to maintain the view models consistent. The tool automatically generates all these basic rules for consistency (those which are performed when some element is added, deleted or modified in a view model). They can be complemented with other more complex specific domain consistency rules for checking static semantics (well-formedness). Furthermore, once the *repository model* has been created, we can use regular graph grammars to transform it into a semantic domain, with the purpose of functional verification [5].

Figure 1 shows the edition of the *view_consistency* relation between one view called *Structural* and the *repository*. A dialog box (on top of the window in the background) shows the relation (*c0*) and a button to edit the triple graph grammar. The dialog box to its right is used to edit the

triple graph grammar properties. The dialog box on top of it shows the edition of one of the rules. This rule simply copies a newly created node to the *repository*. The RHS of the rule is being edited in the right-most window. Here we show a triple graph where the lower part is an instance of the Structural view and the upper part an instance of the whole meta-model.

As a proof of concept of our approach, we show the generated environment for Labyrinth [2], a domain-specific language for hypermedia design. The language has 16 views of the VL meta-model. A snapshot of part of its definition is shown in Figure 1. All the consistency relations between views on the View Specification Tool were automatically generated by the tool. Starting from this definition, the modelling tool in Figure 2 was generated. AToM^3 generates a button for each view, as it is shown on the left of the background window. Users can add a new diagram of a certain view by clicking the corresponding button and then the canvas. The newly created diagram is shown as an icon.

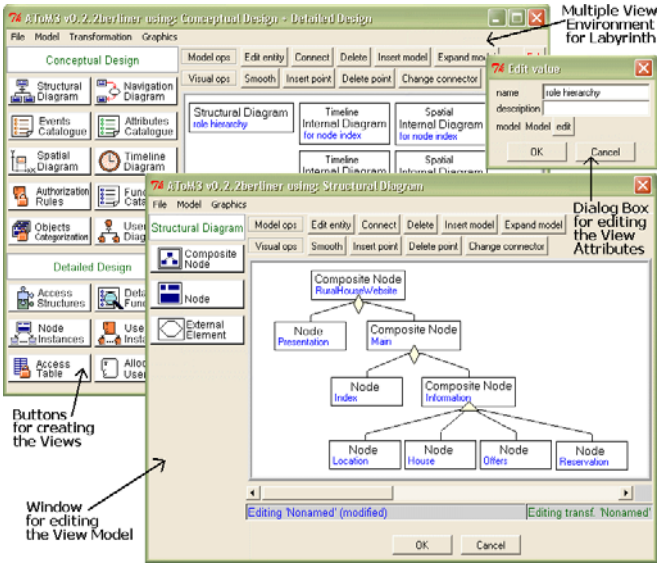

Figure 2. The Generated Environment.

If the user clicks on some diagram icon, a control dialog allows editing the view attributes, as well as the model. The latter is edited in a new AToM^3 window. The upper window on the right of Figure 2 shows the edition a view model. The *repository model* is built by executing the defined graph grammars, which are executed whenever a window containing a modified view is closed.

4. Related Work

Some approaches for consistency checking in VL with multiple views implement the idea of a common *repository* where the different view models are somehow related. The way in which these relations are built can be specified by using either a textual or a graphical notation. Examples of the former include MetaEdit+ [6] and Pounamu [8]. Examples of the latter include JComposer [4] and our tool AToM^3. In JComposer the *change propagation and response graphs* allow visually attach events to the elements of the models, performing actions in response to user actions. In AToM^3 triple graph grammars are used for consistency and change propagation. They are a graphical and natural way to express relations between models, and in addition its theoretical background makes transformations subject to analysis.

5. Conclusions

In this work, we have presented a formal approach for the definition of VLs with multiple views, and subsequent generation of customized modelling environments. Consistency is maintained by gluing the view models into a global model, by means of triple grammars. We have implemented these concepts in AToM^3, and built a modelling environment for a hypermedia VL called Labyrinth.

Acknowledgements: This work has been supported by the Spanish Ministry with project TIC2002-01948.

References

[1] de Lara, J., Vangheluwe, H. 2002. *AToM^3: A Tool for Multi-Formalism Modelling and Meta-Modelling*. LNCS 2306, pp.: 174 - 188. Springer.

[2] Díaz, P., Aedo, I., Panetsos, F. 2001. *Modeling the Dynamic Behavior of Hypermedia Applications*. IEEE Transactions Software Engineering, 27(6). 550-572.

[3] Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. 1999. *Handbook of Graph Grammars and Computing by Graph Transformation*. (1). World Scientific.

[4] Grundy, J.C., Mugridge, W.B., Hosking, J.G. 1998 *Visual Specification of MultiView Visual Environments*. IEEE Symposium on Visual Languages, 236-243.

[5] Guerra, E., de Lara, J. 2003. *A Framework for the Verification of UML Models. Examples using Petri Nets*. Proc. JISBD'03. pp.: 325-334. Alicante.

[6] http://www.metacase.com/

[7] Schürr, A. 1994. *Specification of Graph Translators with Triple Graph Grammars*. LNCS 903, pp.: 151-163. Springer.

[8] Zhu, N., Grundy, J.C. and Hosking, J.G., 2004. *Pounamu: a meta-tool for multi-view visual language environment construction* VL/HCC, pp. 254-256.