Exploiting Visual Languages Generation and UML Meta Modeling to Construct Meta-CASE Workbenches

G. Costagliola, ¹ V. Deufemia, ² F. Ferrucci, ³ C. Gravino ⁴

*Dipartimento Matematica e Informatica*
*Università di Salerno*
*Baronissi (SA), Italy*

**Abstract**

In the paper we propose an approach for the construction of meta-CASE workbenches. The approach is based on the technology of visual language generation systems and on UML meta modeling. Visual modeling environments are generated starting from UML class diagrams specifying abstract syntax of the underlying visual language. The meta-CASE generates a workbench by integrating a set of visual modeling environments through inter-consistency constraints defined on the corresponding UML class diagrams.

**1 Introduction**

CASE tools are recognized as useful means to strengthen and support software development since they provide designers with a powerful visual environment which facilitates editing and manipulation of visual modeling languages. Such languages are key elements in the software engineering field since they allow designers to provide suitable models of a software system and effectively enhance the human to human communication, which is essential for cooperative work. CASE tools are especially useful when provide also some kind of correctness and consistency checking and do not support a single task of the software life-cycle, but rather a whole process phase (workbenches) or a considerable part of the software process (environments). In these cases they usually support a specific method and provide a guidance on when tools in the set should be used.

¹ Email: gcostagliola@unisa.it
² Email: deufemia@unisa.it
³ Email: fferrucci@unisa.it
⁴ Email: gravino@unisa.it

©2003 Published by Elsevier Science B. V. Open access under CC BY-NC-ND license.
Although CASE tools are able to speed development, they are not as widespread as one would expect. It is widely recognized that the main difficulty in their large employment derives from their method inflexibility. Indeed, while software development organizations dynamically change the adopted methodologies by tailoring methods to their own requirements, modifying supporting tools is usually impossible or too expensive.

In recent years, the use of meta-CASE technology for developing CASE tools has been proposed as a solution to this problem [14]. Indeed, a meta-CASE allows to automatically generate CASE tools supporting the desired methodology, thus reducing costs of tool construction and/or adaption. Nevertheless, the development of such generators is not easy. The main difficulties are concerned with the generation of suitable modeling environments.

In the paper we propose an approach for the construction of meta-CASE workbenches which profitably exploits the research on the generation of visual programming environments realized in the visual languages research field [2,6,7,15] and the UML meta modeling. The proposed meta-CASEs generate visual oriented workbenches (e.g. analysis and design workbenches) which also include suitable mechanisms to check the consistency of the different diagrams.

The software architecture of the proposed meta-CASE tool consists of two modules: the Modeling Environment Generator (MEG) and the Workbench Generator (WoG). The first allows to generate visual modeling environments starting from a UML class diagram specifying the abstract syntax of the visual language. The second integrates a set of visual modeling environments starting from the visual specification provided by the workbench designer.

The paper is organized as follows. In section 2, the software architecture of the proposed system is presented. Section 3 is devoted to illustrate the methodology underlying the MEG module for the generation of visual modeling environments starting from UML class diagram specifications. In section 4, an outline of the workbench generator module is provided. The section on related work and final remarks concludes the paper.

2 The Meta CASE Software Architecture

In this section we describe the software architecture of the proposed meta-CASE workbenches. It consists of a Modeling language Environment Generator (MEG), a Visual Modeling Environments (VME) Repository and a Workbench Generator (WoG), as shown in figure 1. The MEG module allows to generate visual modeling environments starting from a UML class diagram specifying the abstract syntax of the visual language with additional information on the concrete syntax and the semantic of the language. The visual modeling environments generated by MEG are memorized in a VME repository. Each environment is able to process visual sentences (models) and output a suitable graph representation given in terms of a GXL document. GXL (Graph Exchange Language) is an XML notation which has been pro-
posed to be a standard exchange format for graph-based tools and to facilitate interoperability of reengineering tools [13,18].

The MEG module can be implemented by using a system that automatically generates visual environments starting from formal specifications of visual languages. As a matter of fact, in the next section we illustrate how the Visual Language Compiler-Compiler (VLCC) system [6,7] can be used to support the construction of a MEG prototype. The WoG module generates the customized workbench by integrating some visual modeling environments present in the repository, thus providing a designer with different languages to model the system from different point of views. Since the developed models can be related and possibly overlapping, the handling of consistency between them is also considered. Different notions of model consistency exist [10]. In particular, two kinds of consistency are taken into account:

1. intra-consistency (syntactical consistency): it ensures that a model conforms to the abstract syntax specified by the meta-model of the language;
2. inter-consistency (horizontal consistency): it is related to diagrams of different languages.

The intra-consistency is addressed by exploiting the formal specification of the visual language which is derived by the meta-model of the language. The inter-consistency properties can be specified by the workbench designer using the WoG module. In particular he/she specifies a visual sentence by:

(i) selecting a set of visual modeling environments from the VME repository,
(ii) defining precedence rules for the use of the selected environments,
(iii) and specifying inter-consistency rules between the models of different environments.

In particular, the consistency rules are given in terms of OCL constraints to be held between elements of the corresponding UML class diagrams. In the generated workbench the checking of such constraints is realized by exploiting the GXL graph representation of each model.
3 The MEG Module

The MEG module supports the workbench designer in the definition and generation of visual modeling language environments. The specification of the visual language is provided in terms of the abstract syntax, given as a UML class diagram, concrete syntax and semantic. It supports a GXL-based methodology for visual language specification and interchanging as described in [8]. GXL is an XML sublanguage aiming to be a standard exchange format for graph-based tools. In the GXL approach, graph classes are defined by GXL documents named *graph schemas*. A graph schema provides the graph structure, i.e. the definition of node and edge classes, their attribute schemas and their incidence structure. Thus, GXL is used to represent an instance graph as well as graph schemas for describing the structure of data [18]. Schema and instance graphs are exchanged by the *same type of document*, i.e. XML documents matching the GXL DTD [18].

The steps of the methodology, supported by the MEG module, are shown in figure 2. The steps automatically carried out are highlighted using bold arrows.

Fig. 2. A methodology for visual language specification.

In step 1 the language designer provides a high-level specification of a visual language in terms of an annotated UML class diagram. This formalism offers a suited declarative language to define visual languages and the annotation provides the concrete syntax of the languages. In step 2.1 the GXL schema for the specified visual language is automatically generated from the UML class diagram. To accomplish this task, a set of general rules for the translation has been defined. The GXL schema so obtained can be used as a general exchange format for the sentences of the language. In step 2.2 a context-free grammar skeleton is constructed from the annotated UML class diagram. In order to automatically carry out this translation, general rules have been defined. In
figure 3 the main rules used to obtain a grammar skeleton from the annotated UML class diagram are explained.

| Rule 1. Each non specialized class produces a terminal symbol of the grammar. |
| Rule 2. Each generalization relationship produces a production skeleton where the names of the specialized classes are the names of the grammar symbols in the right-hand side of the production and the name of the generalized class is the name of the grammar symbol on the left-hand side of the production. |
| Rule 3.1. Each aggregation or composition association between classes produces a production skeleton. In such a production a grammar symbol with the name of the whole class is in the left-hand side and grammar symbols with the names of the part classes are in the right-hand side. Moreover the symbols in the right-hand side are at the same level in the hierarchy of objects and this relationship must be specified by the language designer in step 4 of the methodology. |
| Rule 3.2. The multiplicity of the associations is used to determine the number of the grammar symbols in the right-hand side of the productions and the number of productions with a given grammar symbol in the left-hand side. |
| Rule 3.3. Each composition between a class and a stereotype produces a production skeleton. This production describes a grammar symbol that contains a hierarchy of objects. The names of the grammar symbols in the hierarchy are the names of the classes that “compose” the stereotype. |

Fig. 3. The main rules to obtain a grammar skeleton from a UML class diagram.

In step 3 the language designer completes the grammar skeleton to obtain a visual language grammar specification. Successively, he/she adds actions to the grammar productions in order to translate visual sentences into GXL instances. These actions take into account the annotated UML class diagram and the GXL schema generated in step 2.1. Moreover, other routines can be added to check the intra-consistency constraints of the modelled visual language. An example of intra-consistency constraint for the statechart language is: ”the outgoing transitions of each state in each statechart diagram must be disjoint, that is, the behavioral model must be deterministic”.

In step 4 an integrated Visual Modeling Environment (VME) is generated starting from the supplied language specification. The environment encompasses a visual editor and a compiler for the specified language. Using this environment the user can edit a visual sentence by selecting terminals and arranging them on the working window. Finally he/she can compile the input sentence and obtain the corresponding GXL instance of the sentence.

Figure 4 shows the architecture of the MEG module. It consists of the VLCC and the UML Class Diagram environment. The VLCC system is a powerful visual environment generator based on the formalism of eXtended Positional Grammars which are a direct extension of context-free string grammars to the case of visual languages [9]. As a notable result, the VLCC system supports not only the specification of the visual language (step 3) and the automatic generation of the target VME (step 4), but it is also able to generate the suited UML Class Diagram Environment to draw class diagrams and translate them into a GXL schema and grammar skeleton (steps 1 and 2). The VLCC is formed by a Language Design module which supports the designer in the visual and logical specification of the language terminals through a Symbol Editor and in the definition of the language syntax and semantics through
a Production Editor. In particular, the Production Editor is supported by a
textual editor that allows to enhance the skeleton generated from the UML
class diagram. Starting from the supplied grammar specification, the Environment Generator generates an integrated visual modeling environment which encompasses a visual editor and a compiler for the specified language. As an example, in figure 5 a VME for UML state diagrams is shown. In particular, a visual sentence is depicted, together with two windows containing respectively the GXL instance and the GXL schema obtained by compiling the sentence.

Fig. 4. The architecture for the MEG module.

Fig. 5. The VME for state diagrams generated by the MEG prototype.
4 The Workbench Generator

The WoG module is a visual environment which supports the workbench designer in the definition and generation of workbenches. The WoG environment allows a designer to define visual sentences consisting of a set of graphical boxes possibly linked by arrows, where each box refers to a visual modeling environment generated by MEG and the arrows linking the boxes can be precedence arrows or consistency arrows. The precedence arrows, graphically represented by dashed arrows, define precedence rules for the use of the VME in the customized workbench. As an example the arrow between the UML class diagram box and the UML sequence diagram box in figure 6 specifies that the VME for sequence diagrams can be used after the definition of a sentence in the VME for class diagrams. The consistency arrows graphically represented by bold arrows specify inter-consistency constraints between the sentences in the VME associated to the source and target boxes. In particular, WoG allows to define constraints between a sentence (or part of it) of the source language and a sentence of the target language. To facilitate the formal specification of the constraints, the insertion of a new consistency arrow between the boxes $b_1$ and $b_2$ causes the opening of a window visualizing the UML class diagrams of VME($b_1$) and VME($b_2$). In this new window, the designer highlights the classes on which the constraint must be defined and specifies the inter-consistency constraints on them using a formal notation. In particular, we decided to annotate the highlighted classes with an OCL expression [16] that can be specified either textually or visually by using collaboration diagrams as proposed in [4,5].

As an example, the insertion of the link $a_2$ in figure 6 opens a window like the one shown in figure 7 showing the class diagrams of the UML class diagrams and of the UML state diagrams. In order to test the consistency between a class diagram and the corresponding state diagram the following constraint on the Operation and CallEvent classes should be set: "all the methods in the state diagram must have a corresponding class in the class diagram". Thus, the designer highlights the Operation and CallEvent classes and annotates them with the proper OCL expression, as shown in figure 7. By compiling the specified visual sentence, WoG generates a workbench whose structure is shown in figure 8. The Workbench Interface is the module which interacts with the end user and coordinates the use of the VMEs by exploiting
the precedence rules contained in the corresponding repository. Each VME allows a user to edit models, verify their syntax, semantic (which includes the intra-consistency constraints), and compile them to produce GXL documents which are then memorized by the Workbench Interface in a GXL Repository.

Whenever this repository is updated the Checker module accesses the OCL Constraint Repository and verifies the inter-consistency constraints between the new GXL document and the other related documents as specified by the designer in the WoG. Following the approach proposed in [4,5], the OCL constraints can be translated into graph rule expressions, and an OCL evaluator could be implemented on top of a graph transformation machine [11] able to import GXL documents.

The Workbench Interface provides also a common environment for VMEs to interact. As an example it allows a user to relate a sentence (or part of it) from a VME with one or more sentences from other VMEs and adds this information to the OCL Constraint Repository.

5 Related Work and Final Remarks

Several approaches to the construction of meta-CASE have been proposed in the literature. An interesting classification of them is based on the underlying meta-model which usually is ER-based, OO-based or graph based [14]. Most of the commercial meta-CASEs interpret methodologies as a collection of modeling languages and partition the semantic definition in a data model and in a set of constraints applied to the elements of the data model. Thus, the constraints are separated from the definition of the methods. Moreover, most of them do not support the concept of process and do not provide consistency checking on their produced models.
In recent years big efforts have been made in order to investigate the consistency problem in the object-oriented modeling using UML. Many approaches are based on the use of OCL constraints to express consistency properties. In order to improve OCL readability some studies regards the visualization of OCL constraints. Other interesting results have been provided in [1] where the consistency checks between a UML class diagram and a UML sequence diagram are realized by algorithms on the formal basis of graph transformations.

The use of UML metamodel for the specification of visual languages is gaining interest in recent years. As a matter of fact, a metamodel approach is underlying most generators of diagrammatic editors. As an example, Metabuilder [12] automatically generates an editor for a new visual language starting from the class diagram modeling the language. In [3] UML metamodeling has also been exploited to characterize families of diagrammatic languages through an abstract syntax given as a class diagram and a set of constraints in a logical language.

In this paper we have presented the design of a meta-CASE workbench based on the technology of visual language generation systems and on UML meta modeling. The customized workbench is generated by integrating visual modeling environments with constraints and precedence rules for the correct use of the environments. Moreover, consistency constraints are specified on UML meta-models by using the formal language OCL and the integration of the visual environments is supported by consistency routines working on GXL documents.

As future work, we intend to enhance the WoG visual environment by providing further support for other kinds of consistency constraints. Moreover, from the VLCC system point of view, we intend to provide additional facilities to allow the automatic generation of WoG.
References

Diagrams using Attributed Graph Grammars". In Proc. GRATRA 2000 (Ehrig,
Taentzer Eds.), TU Berlin, FB Informatik, TR No. 2000-2, pp. 77-86, March
2000.

Based on Algebraic Graph Grammars", Procs. 1998 IEEE Symposium on Visual

Editors", Procs. of 2nd International Conference on the Theory and Application

and Visualization of OCL Constraints", third International Conference on the

OCL Using Collaborations", Fourth International Conference on the Unified


for the Implementation of Visual Systems", IEEE Transactions on Software

Approach for Supporting Visual Language Specification and Interchanging",
to appear in Proceedings of IEEE International Symposium on Human-Centric
Computing Languages and Environments (HCC’02), Arlington, VA, USA, 3-6
September, 2002.

2000 IEEE Symposium on Visual Languages, Seattle, WA, USA.

Specifying and Analyzing Consistency of Object-Oriented Behavioral Models".
In V. Gruhn (ed.): Proceedings of the 8th European Software Engineering

Grammars and Computing by Graph Transformation. Volume 2: Applications,
Languages and Tools. World Scientific Publisher, Singapore 1999.

Diagrammer", Proceedings Lecture Notes in Computer Science 1889 Springer


