A Contribution to User Interface Modelling Based on Graph Transformations Approach

Martin Molhanec

Department of e-Technology, Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2, 166 27 Praha 6, Czech Republic
Phone (++420) 224 352 118, Fax (++420) 224 353 949
molhanec@fel.cvut.cz
http://technology.feld.cvut.cz; http://martin.molhanec.googlepages.com

Abstract At present time a very big progress proceeds in all branches of the software engineering field. Anyhow, one particular area is not under the enhanced focus of software developers and researchers. It is the area of the GUI formalization and modelling. The aim of our article is to overcome this deficiency by the suggestion of lightweight formal method for the design of user interface by the GUI modelling based on the graph transformations of underlying conceptual data model. The proposed method gives us a very visual tool for our objective. In addition, our method is an innovative and original specification and modelling technique targeted primarily for a utilization of large and complex modern-day web based applications.

Key words: user interface modelling, conceptual data model, graph transformations

1 Introduction

The aim of our article is a suggestion of lightweight formal method for the design of user interface by the modelling of the GUI based on the graph transformations of underlying conceptual data model. The particular problem resides in an insufficient methodical support for the user interface design phase. Notwithstanding, that exist some user interface design methods; none of them is based on a conception of deriving the user interface schemes from the conceptual data model specifications. This conceptual deficiency makes a semantic gap between the database content and information presented by a user interface.

In this contribution we intend to demonstrate an alternative formal approach how we can compose a user interface model scheme as a result of transformations from the conceptual data model scheme. Our method is based on formal descriptions of user interface and conceptual data models and set of rules describing how to set up the resulting GUI with relevant and valid data. This approach allows constructing only the correct graphics user interfaces. This is the main advantage of herein presented method. Furthermore, the approach presented in our article is a very important for the design of sophisticated web based information systems in consideration the fact that quality of this kind of applications is strongly joined with the well-designed user interface working in cooperation with a complex database system. The issues addressed in this article, inter alia, include:

- The description of our method.
- The definition of conceptual data model.
- The definition of logical data model.
- The definition of user interface model.
- The description of the transformations between the models.

This paper is structured as follows: Section 2 discusses alternate approaches to formal specifications of user interface with reference to the utilization of graph transformations method. Subsequently, Section 3 introduces our approach and sections 4, 5 and 6 are concerned with the formalization of conceptual, logical and user interface model, respectively. Section 7 introduces the suggested transformation, i.e., the essential part of our work. Section 8 briefly describes an AGG, i.e., the open graph transformation tool we use to realize and test the proposed graph transformations. Finally, section 9 proposes some conclusions and an overview of possible future works.

2 Related works

Up till now only some few works has been made on the formalisation of the user interface. However, they do not result from the idea of feasibility to derive the user interface model from the underlying conceptual data model. The author method proposed in this paper is based on his prior works [1,2]. The former work, published only in Czech, is not based on the utilization of the graph transformations yet. Notwithstanding, the principal idea of the possibility to derive a user interface model from the underlying data model was introduced therein. The latter one is an immediate predecessor of the herein presented contribution.

Another approach of the user interface formal specification and design has been published in [3,4]. The deficiency of this approach consists of the fact that any context between the user interface elements and the underlying database is not taken under consideration.

Further, the graph transformations utilization has been inspired by the works [5,6]. However the aim of these works consists in the application of the graph transformations into another area of the field of informatics engineering. But, many concepts introduced therein were adopted in our work.

Furthermore, our work is closely joined with approaches engaged in the web site development as well. Almost all web methods propose a technique for the user interface design. The insufficiency of this approaches consist in the unsatisfactory interconnection between the user interface model and the underlying data model. The majority of these web methods only focus in deriving of the navigational model from the underlying data model, but that methods usually are not concerned with the type of the derivation proposed in our contribution.

3 Overview

The proposed approach by us is a very intuitive. Almost all GUI developers have a sense that there is a strong connection between the particular components of GUI and the data from the database system represented by these components. The way of interconnectivity of the data in a database system predestines the interconnectivity of the relevant components of GUI as well. The underlying database scheme determines not only the interconnectivity of GUI components, but their mutual behaviour and style, with respect of possibility to display single-valued or multi-valued data, as well.

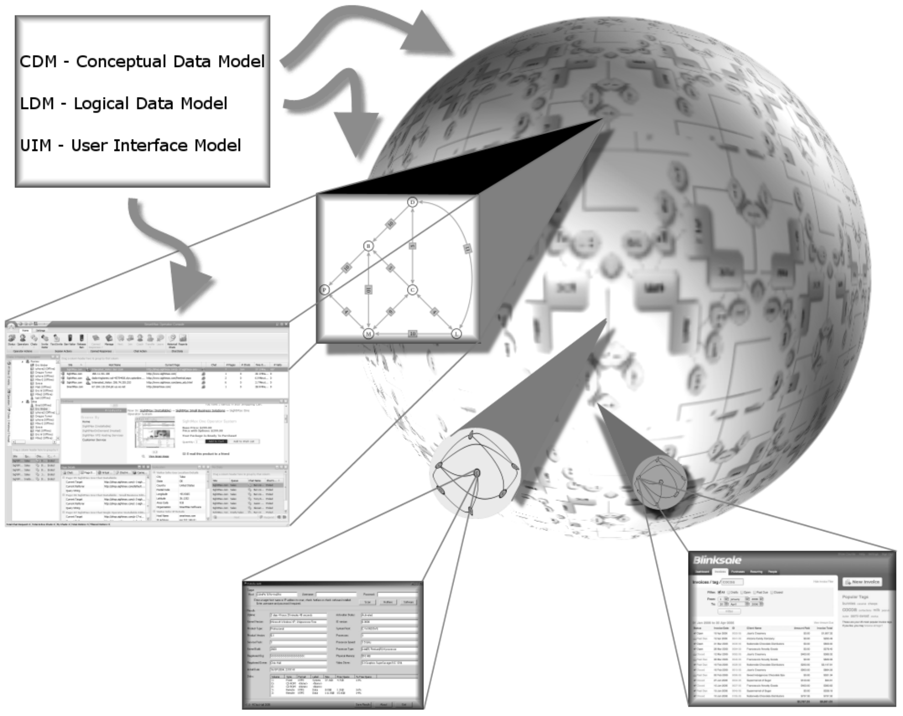


Figure 1. The illustration of an idea of herein presented method.

The interconnectivity between the components of GUI may be express by a model. Regrettably, there do not exists any generally accepted model yet. The proposed model used in this work is based on the graph theory. The model of GUI is not only a segment of data model of the underlying data, not even a model of GUI components separated from them. The model proposed by us is a model both independently designed by developer and derived from the underlying data as well. In other words, at developing GUI, the developer is restricted as well as conducted by underlying data. Our idea is symbolically illustrated by Fig. 1.

The general concept of our approach consists in the following three basic steps:

– The transformation of UML conceptual data model into the relevant graph representation.
– The simplification of conceptual data model into the logical data model.
– The interactive and incremental construction of the GUI model.

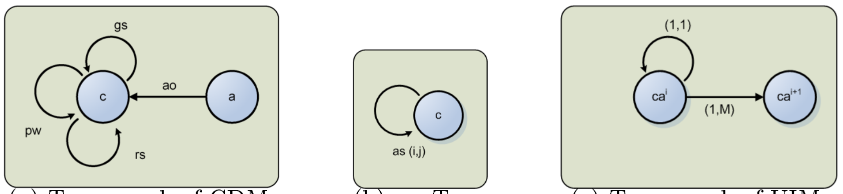
The resulting model of GUI may be used for the smart generation of GUI components by means of developer IDE or for a construction of SQL or other DML commands for the GUI components data filling eventually.

Reason for the utilization of the graph based model approach consists in the possibility to use the graph transformation method in order to transform them. This method is very advantageous for us by the reason that provides a powerful tool for the construction of transformation grammars with the respect to the inspectional and comprehensible visual view of the generated transformation rules. Eventually, the graph transformation method has a robust and confirmed fundamentals based on the graph theory[7].

Indeed, in subsequent chapters firstly we made definitions of the conceptual and logical graph based models and the graph transformation between them. Secondly, we define the proposed GUI graph based model as well. And finally, we describe our algorithm for the construction of the GUI model from the logical model based on the graph transformations rules.

4 Formalization of conceptual data model

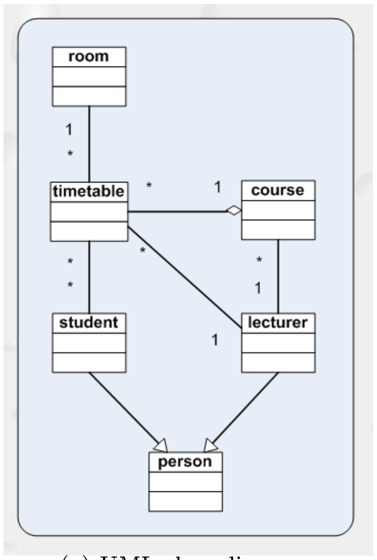
This section gives a brief description of the conceptual data model (*CDM*) we used in our method. The CDM considered in this work is composed of following concepts: class, attribute and association (*generalization-specialisation, part-whole and relationship*) in the usual sense and compliance with the object oriented modelling paradigm.



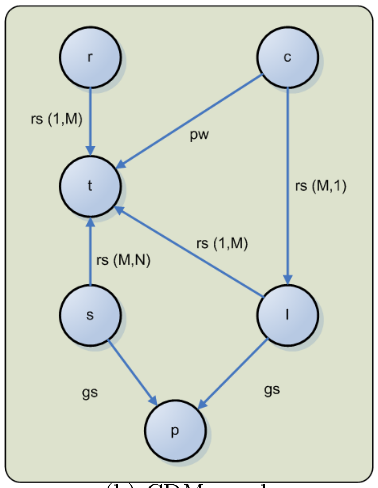
(a) Type graph of CDM. (b) Type graph of LDM. (c) Type graph of UIM.

Figure 2. The type graphs of CDM, LDM and UIM.

The Fig. 2(a) shows the type graph used to describe our CDM. In this type graph each node represent a class or attribute and each edge represents one of the three types of possible associations (*generalisation-specialisation, part-whole and relationship*). This particular type graph contains nodes of type c and a; and edges of type gs, pw, rs and ao. Nodes of type c represent classes and nodes of type a represent attributes. Edges of type gs represent generalisation-specialisation associations, edges of type pw represent part-whole associations, edges of type rs represent relationship associations and edges of type ao represent attribute ownerships between the class and its attributes. However, we do not use the concept of attribute temporarily. The type graph represents a condition which must be fulfilled by all correct graphs that could be constructed.



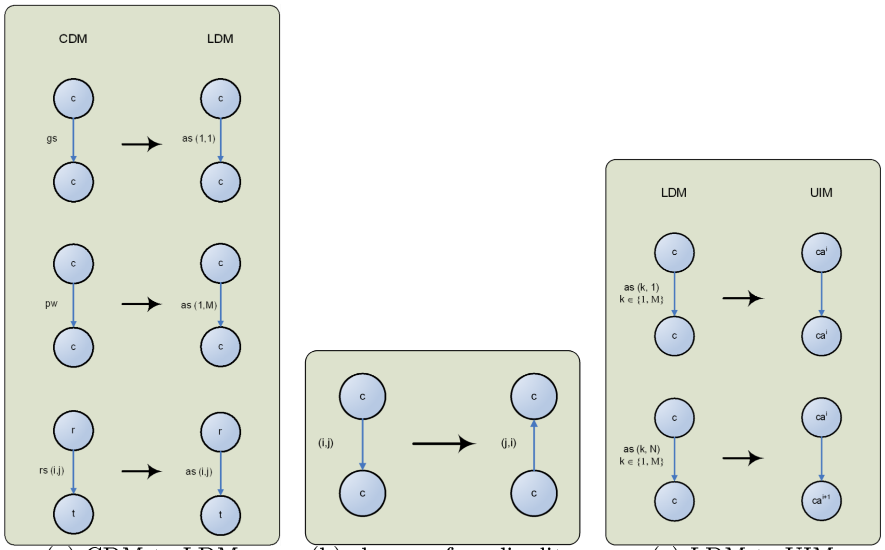
(a) UML class diagram.



(b) CDM graph

Figure 3. The example 1, the university information system.

As an illustration, we demonstrate our ideas by a simple example representing a part of the real world. Our problem domain forms a part of a university information system consisting of courses, time-tables, students, lecturers, teaching rooms etc. The corresponding conceptual data model scheme in UML notation is shown in Fig. 3(a) and the corresponding graph representation in Fig. 3(b). It is evident; we use a directed, typed, attributed and labelled graph. A shorthanded name of the particular class in the corresponding UML diagram is written into the circle representing a node in the resulting graph diagram. The letter c represents a course, t represents a timetable, r represents a room, s represents a student, p represents a person and l represents a lecturer. All nodes in our resulting graph represent only one type allowed in the corresponding type graph – the class. The edges are marked by its type and we use the following shorthands: gs represents a generalisation-specialisation type, pw represents a part-whole type and rs represents a relationship type. The nodes joined by relationship type may have a different cardinality (a participation in the relationship) and we mark these edges by cardinality value as well. The cardinality of the relationship edge means a count of possible occurrences of the instances of both classes from either sides of the particular edge. The possible values of relationship cardinality are defined by the set: $(1, 1), (1, M), (M, 1), (M, N)$. The letters M and N are used as a symbolical count with denotative meaning *many*.



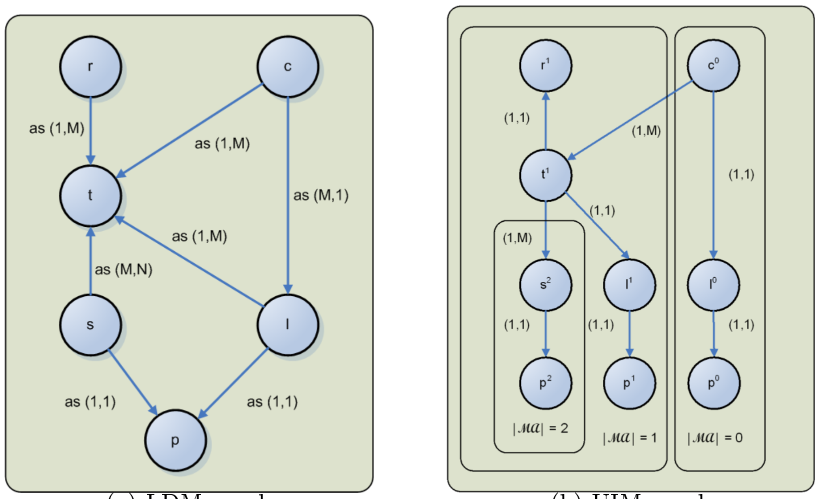
(a) CDM to LDM (b) change of cardinality (c) LDM to UIM

Figure 4. The graph transformation rules.

5 Formalisation of logical data model

For a proper design of GUI we only need to know the cardinality of associations. Because of the simple reason that various meaning of different types of associations disappears in the context of GUI, i.e., we do not need a precise distinction between them in subsequent text. Class attributes can be omitted temporarily from our model as well, due to the fact that the described transformation proceeds only at the level of classes and associations between them. In other words, we do not need to transform attributes. Thus, the type graph for our logical data model is shown in Fig. 2(b). It is evident that we use only one type of labelled nodes – the class and one type of edges marked by cardinality – the general association. The transformation rules for the transition from CDM to LDM (*Logical Data Model*) in commonly used notation are shown in Fig. 4(a).

It is evident that all semantic information about the different types of associations is lost. But, this fact is not too restrictive, because in herein presented method we do not use this semantic information; it is our intent at this moment. In our subsequent work, we intend to propose a more complex transformation method without a loss of the type information. Thus, in our method we omit the type information from the resulting graph and mark edges only with its cardinality by reason that all edges possess only one relationship type – the general association.



(a) LDM graph (b) UIM graph

Figure 5. The example 2, the university information system.

The resulting graph of our example is shown in Fig. 5(a). It is apparent that original and resulting graphs can be a cyclic and doesn't create a tree. Also, the value of cardinality depends on the selected direction we choose. Finally, we can change orientation of edge cardinality by a simple transformation defined in Fig. 4(b).

6 Formalisation of user interface model

The main idea embedded in this work jointly involves the formalisation of the user interface model (UIM) and the definition of the graph transformations rules in order to derive UIM from LDM. Now, we may define the UIM by means of the following rules.

– Only the components of user interface which correlate to the data model (modelled by LDM, respectively CDM) are relevant.
– All components of user interface can display only one value or a list of values.

– All associations between the user interface components are derived from the underlying LDM.

Further, we explain the attributes of the user interface model (UIM) related to our method. First of all, we need to define a few new concepts needed in the subsequent work.

– *Data Model Dependent Area (or Data Area for short).* The concept of Data Model Dependent Area presents a set of user interface components bounded together by a conjunctive dependency on underlying data. Change of data focus of one component can change a data focus of other components. Our work concerns about a proper design of just one Data Area.
– *User Interface Component (or Component for short).* The concept of User Interface Component presents a visual component of user interface which has a visual representation on the screen. User can change the visual presentation of such component, but not its data content.
– *User Interface Data Component (or Data Component for short).* The concept of User Interface Data Component presents a subset of User Interface Components bounded to underlying data. The set of such Data Components forms a Data Area that was hereinbefore defined. Our work concerns only such data components.
– *Data Component Class Area (or Class Area for short).* The concept of Data Component Class Area presents a specific subset of Data Components bounded to exactly one entity in the underlying data model.
– *Data Component Same Multiplicity Area (or Multiplicity Area for short).* The concept of Data Component Same Multiplicity Area presents a specific subset of Data Components having the same multiplicity. We will discuss this concept as well as concept of multiplicity in detail later in this article.

Let us use the following labels for hereinbefore defined concepts; $\mathcal{DA}$ for data area, $\mathcal{C}$ for component, $\mathcal{DC}$ for data component, $\mathcal{CA}$ for class area and $\mathcal{MA}$ for multiplicity area. We may express the concept relations between them as[1]:

$$\mathcal{C}, \mathcal{DC} \subset \mathcal{MA} \subset \mathcal{CA} \subset \mathcal{DA} \tag{1}$$

$$\mathcal{DC} \triangleright \mathcal{C} \tag{2}$$

Let's consider the following definitions of subsequent concepts.

– *Multiplicity of data component (or Multiplicity for short)* is a capability to display a single value or list of values or list of list of values and so on. We will denote a multiplicity by count as a superscript associated with the particular data component. The multiplicity equates to 0 means possibility to display a single value, equates to 1 means possibility to display a list of values and so on.

[1] We used a symbol $\triangleright$ for concept of inheritance or generalization - specialization. The term $\mathcal{A} \triangleright \mathcal{B}$ we read as *A is inherited from B* or *A is a specialization of B.*

- *Dependency of data components* means an abstract association between the data components laid within the particular data area. Changing the data focus of one component changes the data foci of other components. Every data component in the particular data area can be mapped to the single entity attribute in the underlying data model[2].
- *Dependency of class areas* means an abstract association between class areas corresponding to the data relationships between entities in underlying model. Changing the data focus of one class area changes the data foci of other class areas. Every class area in the particular data area can be mapped to the single entity in the underlying data model.
- *Multiplicity of dependency of data components or class areas (or Multiplicity of dependency for short)* is defined as an ordered pairs of numbers denotative the multiplicity of corresponding data components. It is evident, that multiplicity of dependency may take the value from the set: {(1, 1), (1, M)}. Multiplicity of dependency between the data components from a particular class area will always equate to the value of (1, 1). This is the reason why dependencies of data components from the particular class areas are not so important for us and so we can only deal with the dependencies between a different class areas. By reason that all data components from a particular class area have the same multiplicity, we can define a multiplicity of particular class area as follows.
- *Multiplicity of class area* equates to multiplicity of its data components.

The type graph of our UIM is shown in Fig. 2(c). The type graph contents the nodes of type ca^{i} and $ca^{\mathrm{i+1}}$ and edges with multiplicity of (1, 1) and (1, M). The nodes of type ca^{i} and $ca^{\mathrm{i+1}}$ represent the class areas of multiplicity i and $i{+}1$ respectively. The edges of type (1, 1) and (1, M) represent the dependencies of class areas with the multiplicity of dependency equates to (1, 1) and (1, M) respectively. Further, the type graph represents a condition, which must be fulfilled by all correct graphs that could be constructed. It is good to note that for any LDM can be constructed as many UIM as you like.

7 Transformation from LDM to UIM

The last phase of our method deals with the generating of UIM from LDM. We must to note that our utilization of the graph transformations is done by a somewhat different way as is customary. The graph transformations are usually used to transform a particular graph from the source form to the target form. We use the graph transformations to build the UIM graph by inference from the LDM graph.

[2] In this contribution we do not consider derived (calculated) data components. This concept will be a subject of our future work.

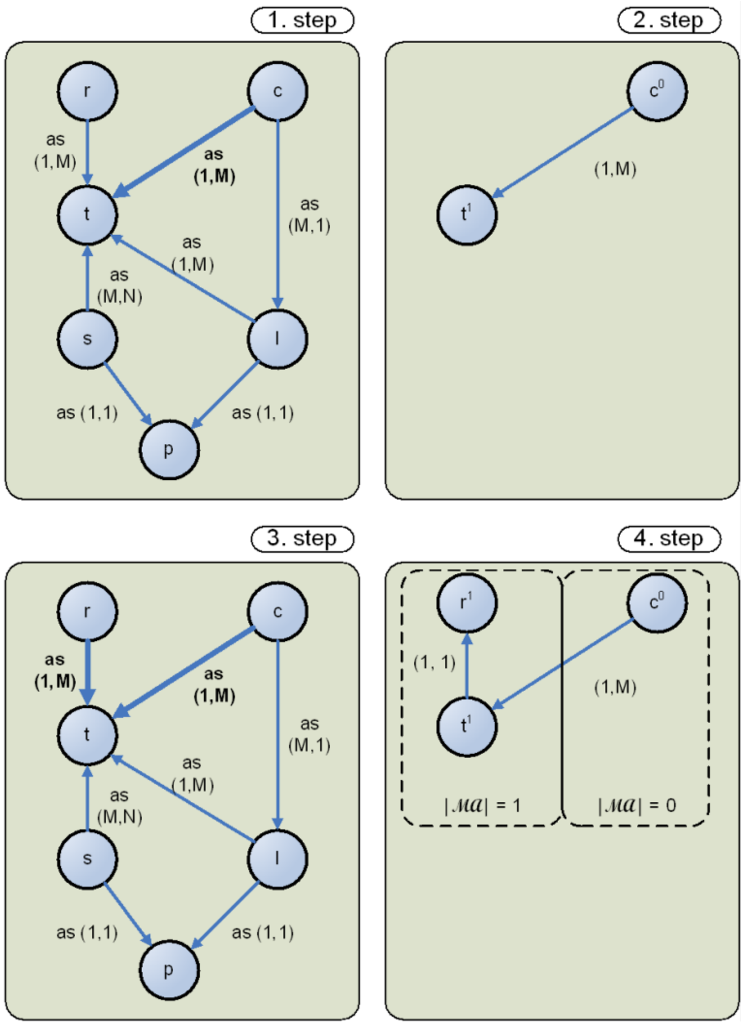


Figure 6. The example 3, the graph transformations from LDM to UIM, step by step.

The transformations may be described by the following pseudo algorithm:

1. Select the initial node in the LDM graph.
2. Add the selected node as the initial node in the new built UIM graph.
3. Mark the selected node as a new starting point in the UIM graph and simultaneously as already used node in the LDM graph.
4. REPEAT
   (a) Select another node in the LDM graph, which is in an incidence relationship with an arbitrary node marked before as used.
   (b) Change the direction of a relevant edge by using the appropriate graph transformation rule (Fig. 4(b)) if necessary it is.
   (c) Transform the just selected node and the corresponding edge by using the relevant graph transformations rule (Fig. 4(c)) and insert this node and relevant edge to the new-created UIM graph.
   (d) Mark the last selected node and the edge from preceding step as already used in the LDM graph.
5. UNTIL you need.

The key part of hereinbefore presented transformation lie in the step "4-c" and can be expressed by means of the following simple rules:

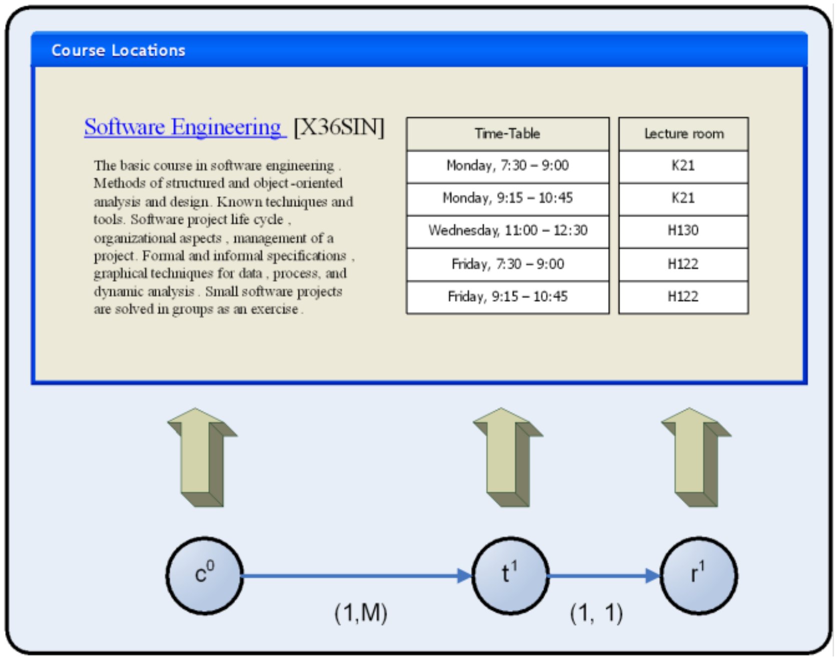
– If the direction of the edge is from the node of multiplicity i to the node of identical multiplicity we use the upper transformation rule shown in Fig. 4(c).
– If the direction of the edge is from the node of multiplicity i to the node of multiplicity of $i+1$ we use the bottom transformation rule shown in Fig. 4(c).

Further, we may demonstrate the presented approach by a simple example shown in Fig. 6 as an explication of the algorithm described hereinbefore. The example is broken down to the single steps, labelled from 1 to 4. Furthermore, the resulting UIM graph of a little bit more complex example is shown in Fig. 5(b). Finally, another illustrations of the presented approach, the corresponding graphical representations of the user interface screens of these two demonstrated examples are shown in Fig. 7 and Fig. 8. The concepts of class area and multiplicity area are outlined in these examples as well. We need to note that an arbitrary node or an edge can be used in the step "4-a" a number of time in the course of utilization of our algorithm. We break up cycling of the algorithm in the case when all nodes we required to process acording to our demand will be placed into the resulting UIM graph as we need.

8 AGG[3] - the used graph transformation tool

One of several improvements that have been achieved over the last year has been selection and utilization of the particular graph transformation tool. In order to continue our work we need a good tool facilitate experiments with ideas proposed in our work. Foremost we stipulate a few requirements.

[3] The AGG is an acronym for "Attributed Graph Grammar".



Software Engineering [X36SIN]

The basic course in software engineering . Methods of structured and object -oriented analysis and design. Known techniques and tools. Software project life cycle , organizational aspects , management of a project. Formal and informal specifications , graphical techniques for data , process, and dynamic analysis . Small software projects are solved in groups as an exercise .

Time-Table	Lecture room
Monday, 7:30 – 9:00	K21
Monday, 9:15 – 10:45	K21
Wednesday, 11:00 – 12:30	H130
Friday, 7:30 – 9:00	H122
Friday, 9:15 – 10:45	H122

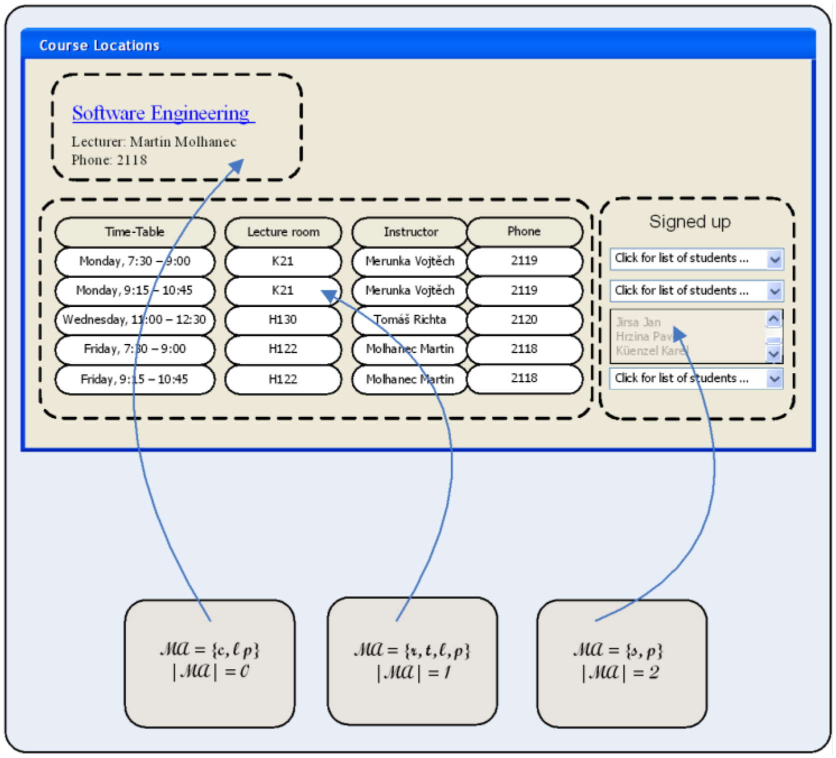
Figure 7. The example 4, the GUI presentation.

– The support of attributed graph grammar.
– The support of graph transformations in wide range.
– The platform independence.
– The open license.
– The user friendly use .

We have chosen one of the following tools:

– *AGG* – “The Attributed Graph Grammar System” a simple and powerful tool developed at the Department of Software Engineering and Theoretical Computer Science in the Technische Universität Berlin[8].
– *AToM3* – “A Tool for Multi-formalism and Meta-Modelling” another powerful tool developed at the Modelling, Simulation and Design Lab (MSDL) in the School of Computer Science of McGill University[9].
– *Groove* – “GRaphs for Object-Oriented VErification”, a simple and modern modelling tool developed at Department of Computer Science, University of Twente[10].
– *Fujuba* - A large Eclipse based project developed at the Software Engineering Group in the University of Paderborn[11].

At present time we elaborate with the AGG tool we have chose because fulfils our conditions mentioned hereinbefore. We may demonstrate this excellent tool


Course Locations

Software Engineering
Lecturer: Martin Molhanec
Phone: 2118

Time-Table	Lecture room	Instructor	Phone
Monday, 7:30 – 9:00	K21	Merunka Vojtěch	2119
Monday, 9:15 – 10:45	K21	Merunka Vojtěch	2119
Wednesday, 11:00 – 12:30	H130	Tomáš Richta	2120
Friday, 7:30 – 9:00	H122	Molhanec Martin	2118
Friday, 9:15 – 10:45	H122	Molhanec Martin	2118

Signed up
Click for list of students ...
Click for list of students ...
Jirsa Jan
Hrzina Pav
Küenzel Karel
Click for list of students ...

$\mathcal{MA} = \{c, \ell\, p\}$
$|\mathcal{MA}| = 0$

$\mathcal{MA} = \{r, t, \ell, p\}$
$|\mathcal{MA}| = 1$

$\mathcal{MA} = \{s, p\}$
$|\mathcal{MA}| = 2$

Figure 8. The example 5, the GUI presentation.

by a few figures from our last work. A scheme of the example used in this article is shown in Fig. 9 and the graph transformation corresponding to the transformation from Fig. 4 is shown in Fig 10.

It is evident that in Fig. 9 is shown the same situation as in Fig. 3(b) only with some differences. The name of the node corresponding to the name of the class in the source UML class diagram is added as a node attribute "*name*" into the diagram. The attribute of edge named "*cardinality*" is added as well. These attributes are used by the AGG tool in subsequent graph transformation steps.

The Fig. 10 illustrate an example of one graph transformation as it is defined by the AGG tool. The left side of the figure shows the graph pattern searched in the source diagram and the right side shows the graph pattern replacing the just found pattern in the source diagram. In course of the transformation the relevant node and edge attributes are transformed as well. It is evident that the AGG tool provides all needed features, e.g., types, attributes and calculations, for the next development of our method.The AGG tool offer an easy and comfort way for the graph transformation handling. At the present time it is our favourite tool, though our searching for an appropriate tool is not definitely finished.

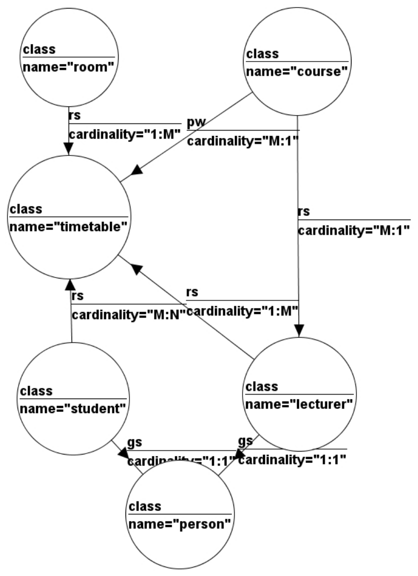


Figure 9. The example of AGG, CDM.

9 Conclusion and further work

The proposed formalization is based on the graph transformations theory [7]. We are convinced that this formalization has many advantages:

- A good theoretical foundation based on graph transformations.
- The possibility to verify correctness of the resulting user interface scheme.

However, this work is not a finished yet and we used the very basic model of possible user interface. Thus, for now, we must consider following disadvantages as well.

- We work only with a few basic elements of user interface.
- We must do a lot of work on the development of an appropriate graph transformation tool.

In conclusion, our lightweight formal method intended for deriving the user interface schemes by graph transformations from the conceptual data model specifications is not fairly good yet. We are convinced that the algorithm described in the preceding section has to be more detailed, formally and purely specified.

Consequently, our future objectives consist in an improvement of our formalization concept and transforming algorithm. Subsequently, we want to use all

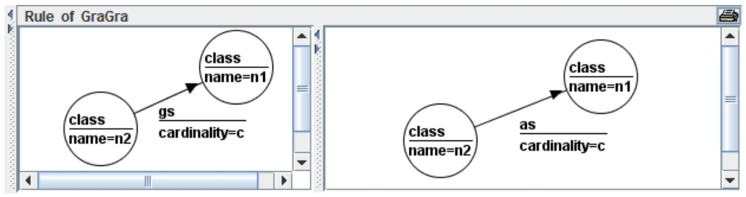


Figure 10. The example of AGG, the graph transformation.

semantic information from the source conceptual graph for the construction of proper user interface. Next, we must a complete a software tool supporting hereinbefore proposed transformation. Finally, we intent to work on the specification of the rules for the construction of relational algebra queries or in other words, on the construction of the SQL *select* statement in order to automate the code generation of corresponding applications.

Acknowledgement

This research (work) has been supported by Ministry of Education, Youth and Sports of Czech Republic under research program MSM6840770017.

References

1. Molhanec, M.: Typologie uživatelského rozhraní (Typology of user interface). In: Tvorba software'97, pp. 88–97. Tanger, Ostrava (1997)
2. Molhanec, M. User Interface Modelling Based on the Graph Transformations of Conceptual Data Model In: CEUR workshop proceedings. 2008, vol. 2008, no. vol-338, p. 79-91. ISSN 1613-0073.
3. Alencar, P.S.C., Cowan, D.D., Lucena, C.J.P.: Abstract Data Views as a formal Approach to Adaptable Software. In: OOPSLA Workshop On Adaptable And Adaptive Software, Proceedings. Austin (1995)
4. Rossel, P., Contreras, R., Bastarrica, M. C.: Graphic Specification of Abstract Data Types. In: Rev. Fac. Ing. - Univ. Tarapacá, vol.12, no.1, pp. 15–23. Tarapacá (2004)
5. Koch, M., Mancini, L.V., Parisi-Presicce, F.: A Graph-Based Formalism for RBAC. In: ACM Transaction on Information ans System Security, Vol. 5, No. 3, pp. 332–365. (2002)
6. Koch, M., Mancini, L.V., Parisi-Presicce, F.: Administrative Scope in the Graph-based Framework. In: SACMAT'04, June 2-4. Yorktown Heights, New York, USA (2004)
7. Rozenberg, G., editor: Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations. World Scientific (1997)
8. AGG – home page, Online: <`http://tfs.cs.tu-berlin.de/agg`>
9. AToM3 – home page, Online: <`http://atom3.cs.mcgill.ca`>
10. Groove – home page, Online: <`http://groove.cs.utwente.nl`>
11. Fujuba – home page, Online: <`http://wwwcs.upb.de/cs/fujaba`>