User Interface Modelling Based on the Graph Transformations of Conceptual Data Model

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://martin.molhanec.googlepages.com

Abstract The aim of our article is a brief description of user interface modelling based on the graph transformations of conceptual data model. We use the lightweight formal method intended for deriving the user interface schemes by graph transformations from conceptual data model specifications. The presented method based on graph transformation theory gives us a very visual tool for our objective. In our paper the presented method is notwithstanding an innovative and original specification and modelling technique targeted primarily for utilization as a part of BORM II Agile methodology, especially for the design of large and complex web based applications.

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

1 Introduction

Advanced web based information applications are usually constructed above large relational or object oriented databases. The design of those databases is based on the well-known conceptual modelling methodology. The particular problem resides in an insufficient methodical support for user interface design phase. Notwithstanding that exist some user interface design methods; none of them is based on a conception of deriving user interface schemes from conceptual data model specifications. This conceptual deficiency makes a semantic gap between the database content and information presented by a user interface.

In our article we show an alternative formal approach to 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 building up the resulting proper user interface scheme containing only of relevant and valid data. This allows constructing only the correct user interface schemes. This is the main advantage of herein presented method. Furthermore, the approach presented in our article is important for the design of sophisticated web based information systems in

consideration of fact that quality of this kind of applications is highly linked with a well-designed user interface working together with a complex database system.

This paper focuses on formal description of particular user interface design method based on principle of deriving user interface model from conceptual data model using graph transformations theory. The issues addressed in this article include:

- Definition of conceptual data model.
- Definition of logical data model.
- Definition of user interface model.
- Description of our method.

This paper is structured as follows: Section 2 discusses alternate approaches to the formal specification of user interface with reference to utilization of graph transformations. Subsequently, section 3, 4 and 5 are concerned with formalization of conceptual, logical and user interface model, respectively. Section 6 introduces the proposed formal transformation the major part of our work, and section 7 presents some conclusions and an overview of possible future work.

2 Related Work

Some work has been done on the formalisation of user interface. It does not, however, focus on the deriving user interface model from the underlying conceptual data model. The author work proposed in this paper is based on his prior work [1] published on local Czech informatics conference. This former work is not based on the utilization of graph transformations, but the main idea subsisted in the possibility to derive user interface model from the underlying data model was introduced therein.

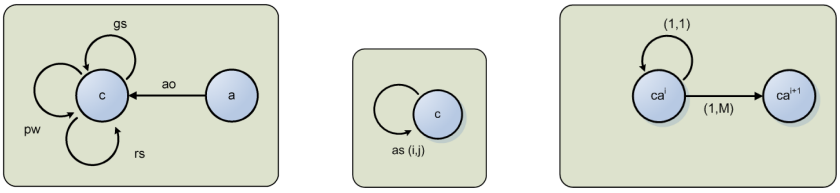
Another approach of user interface formal specification and design was published in [6] and [7]. The deficiency of this approach consists of the fact that any context between user interface elements and the underlying database or more precisely conceptual data model does not exist. Further, the graph transformations used by similar way are published also in [8] and [9]. However the aim of these works is in the use of graph transformations for the *role based access control*. But, many concepts introduced there were adopted in our work.

Also, our work is closely joined with approaches engaged in the web site development. Almost all web methods propose a technique for user interface design. The insufficiency of this approaches consist in unsatisfactory interconnection between user interface model and underlying data model. We can note the *WebML* method [2] with concept of the derivation model derived from the structure model (a data model in the WebML technology). Though, the derivation of the user interface from the derivation model is not described in the WebML method. The majority of 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 derivation proposed in this paper.

Our work we consider as a part of *BORM II* Agile method. The original BORM method [3] was born in 1993 and was intended to provide seamless support for the building of object oriented software systems based on pure object-oriented languages together with object databases, such as Gemstone. It is now realized that this method also has a significant potential in capturing knowledge of business processes, business data and business issues. At the present time, the BORM II method continues in concepts based on the original BORM method, but evolves into the new methodological framework enriched in many ways (e.g., ontology, knowledge management, object normalization, MDA). The first overview of the BORM II method will be published within short.

3 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 object oriented modelling paradigm. A precise meaning of these concepts is based on general ontology. In our work we emanate from particular ontology (*GOL* [5]); however this fact is not a subject of this article. The reason is that a commonly used object oriented technique UML does not provide a good foundation for a precise description of the conceptual data model semantic we need to use.



(a) Type graph of CDM.

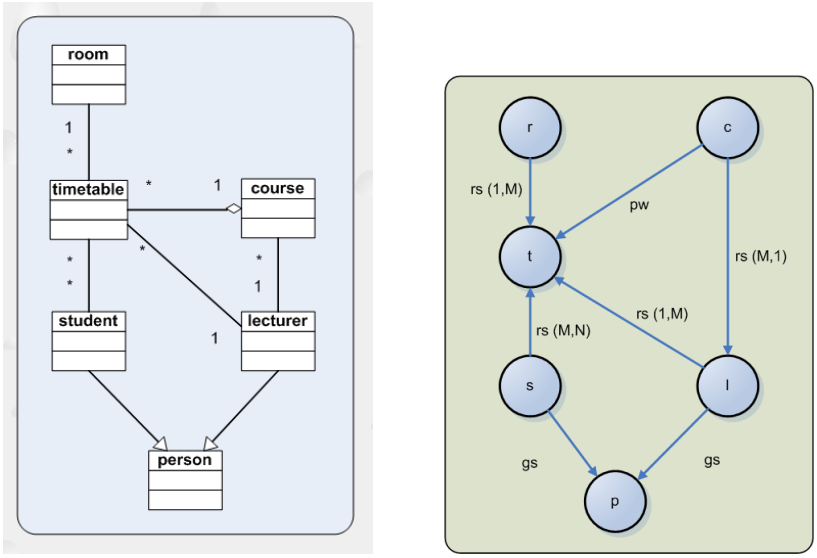
(b) Type graph of LDM.

(c) Type graph of UIM.

Figure 1. Type graphs of CDM, LDM and UIM.

The Fig. 1(a) shows the type graph used to describe our CDM. In this type graph each node represents 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. But, we do not use the concept of attribute in our text temporarily. The type graph represents a condition, which must be fulfilled by all correct graphs that could be constructed.

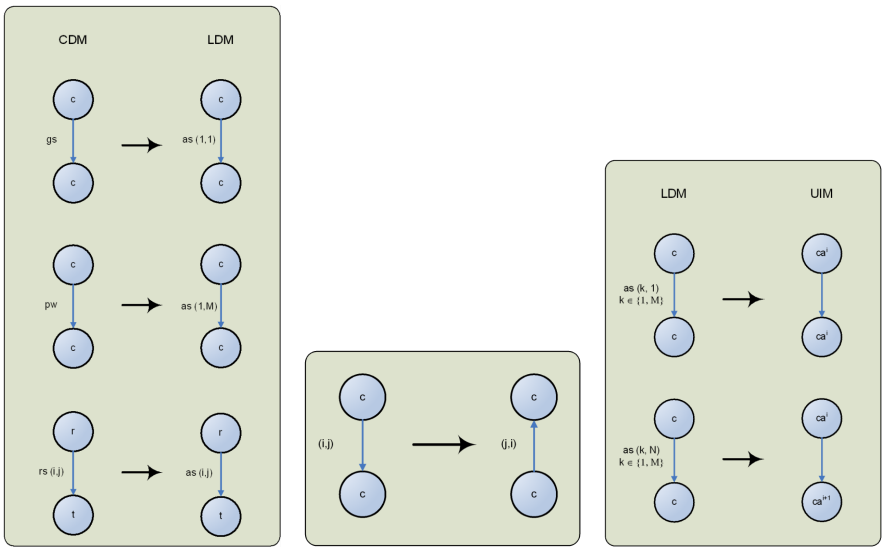


(a) UML class diagram. (b) CDM graph

Figure 2. Simple example - university information system.

In our article we demonstrate our ideas by a simple example introducing part of the real world. Our problem domain forms a part of the 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. 2(a) and the corresponding graph representation in Fig. 2(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 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, we use the following shorthand: 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 relationship edge means a count of possible occurrences of instances of particular 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

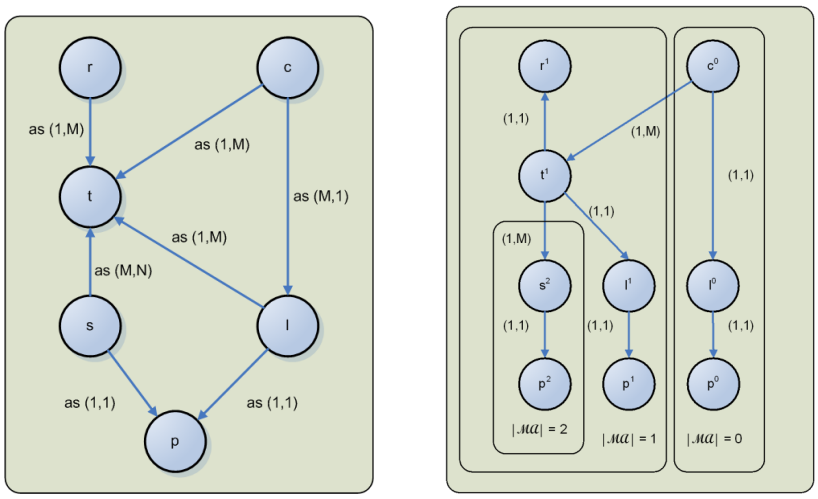
Figure 3. Graph transformation rules.

4 Formalisation of logical data model

In our subsequent work we do not need a precise distinction between different types of associations. For proper design of user interface model we only need to know the cardinality of particular association. Class attributes can be omitted temporarily from our model. The type graph for our logical data model is shown in Fig. 1(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. Transformation rules for transition from CDM to LDM (*Logical Data Model*) in commonly used notation are shown in Fig. 3(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 will 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 4. Simple example - university information system.

The resulting graph of our example is shown in Fig. 4(a). It is apparent that original and resulting graphs can be a cyclic and doesn't create a tree. Also, 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. 3(b).

5 Formalisation of user interface model

The main idea included in our work consists in formalisation of user interface model (*UIM*) and definition of graph transformations rules in order to derive UIM from LDM. Our UIM is based on following concepts:

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

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

Now, we explain the features of user interface model (UIM) related to our method. At first we define a few new concepts we need in the subsequent work.

– *Screen.* The concept of screen presents all properties of user interface seen on computer screen at a time. User can change only hardware dependent characteristics of the screen, for example resolution, brightness and so on.
– *Window.* The concept of window presents explicit area of the screen. Individual windows are mutually independent. The user can change location and visibility of single window on the screen independently.
– *Panel.* The concept of panel presents a contiguous area of the window. The panel impersonates a logically bound entity. We can describe behaviour of panel by means of software engineering abstraction, e.g., activity and state transition diagram. Associations can exist between individual panels.
– *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 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 presentation 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 components bounded to underlying data. The set of 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 underlying data model. We will discuss this concept in detail later in this article.
– *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 and concept of multiplicity in detail later in this article.

Let us use the following labels for hereinbefore concepts. $\mathcal{S}$ for screen, $\mathcal{W}$ for window, $\mathcal{P}$ for panel, $\mathcal{DA}$ for data area, $\mathcal{C}$ for component, $\mathcal{DC}$ for data component, $\mathcal{CA}$ for class area and $\mathcal{MA}$ for multiplicity area. We can consider following relations between them[1]:

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

[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.*

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

Let's consider the following definitions of later used concepts.

- *Multiplicity of data component (or Multiplicity for short)* is capability to display a single value or list of values or list of list of values and so on. We will denote multiplicity by count as superscript associated with 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.
- *Dependency of data components* means an abstract association between data components laid within the particular data area. Changing of data focus of one component changes data focus of other component. Every data component in particular data area can be mapped to single entity attribute in underlying data model[2].
- *Dependency of class areas* means an abstract association between class areas corresponding to data relationships between entities in underlying model. Changing the data focus of one class area changes the data focus of other class area. Every class area in particular data area can be mapped to a single entity in 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 multiplicity of corresponding data components. It is evident, that multiplicities of dependency are from the set: {(1, 1), (1, M)}. Multiplicity of dependency between data components from particular class area will always equate to pair of values (1, 1). This is the reason why dependencies of data components from particular class areas are not important for us and we can focus on dependencies between different class areas only. By reason that all data components from 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. 1(c). The type graph contents nodes of type ca^{i} and $ca^{\mathrm{i+1}}$ and edges with multiplicity of (1, 1) and (1, M). Nodes of type ca^{i} and $ca^{\mathrm{i+1}}$ represent class areas of multiplicity i and $i+1$ respectively. Edges of type (1, 1) and (1, M) represent dependencies of class areas with multiplicity of dependency equates to (1, 1) and (1, M) respectively. 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.

[2] At presented work we do not consider derived (calculated) data components. This concept will be subject of our future work.

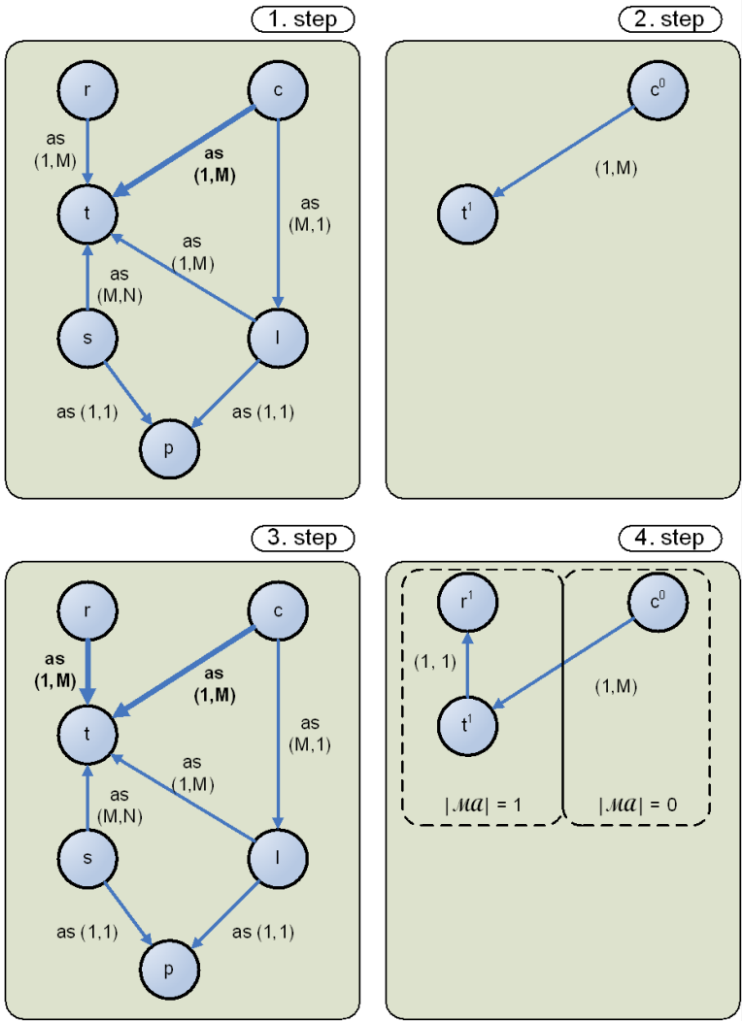


Figure 5. Example 1, graph transformations from LDM to UIM, step by step.

6 Graph transformation from LDM to UIM

Graph transformations are usually used to transform particular graph from source form to target form. We use graph transformations to derive UIM graph from LDM graph. This transformation can be described by the following steps:

1. Selection of the initial node in the LDM graph.
2. Addition of the selected node as initial node in the new UIM graph.
3. Marking of the selected node as starting point and simultaneously as already used node in the LDM graph.
4. Selection of another node in the LDM graph, which is in incidence relationship with arbitrary node marked before.
5. Changing of direction of relevant edge by using appropriate graph transformation rule (Fig. 3(b)) if necessary.
6. Transformation of the selected node and the corresponding edge by using appropriate graph transformations rules (Fig. 3(c)) and insertion of this node and relevant edge to our new-created UIM graph.
7. Marking of the selected node and edge from preceding step as already used in the LDM graph.
8. Repeat steps 4 to 7 as you need.

Finally, we document our approach by a simple example shown in Fig. 5, for demonstration of our algorithm described hereinbefore. Our example is broken down to single steps, labelled from 1 to 4. The resulting UIM graph of a little more complex example is shown in Fig. 4(b). For better understanding of our approach the corresponding graphical presentations of user interface screens of these two examples are shown in Fig. 6 and Fig. 7. Concepts of class area and multiplicity area are marked in these examples as well. We have to note that arbitrary node or edge can be used in step 4 a number of time in the course of utilization of our algorithm. We break up cycling of our algorithm when nodes all we required will be placed in the resulting UIM graph.

7 Conclusion and further work

The proposed formalization is based on graph transformations theory [4] with a few modifications. This formalization has some advantages:

- A clear visual interface and intuitive visual description of our problem domain.
- A good theoretical foundation based on graph transformations.
- Graph transformations also provide possibility to verify correctness of the resulting user interface scheme.

However, this work is a first attempt at this field and we used the simplest model of user interface. Thus, for now, we must consider following disadvantages:

- The proposed method is targeted for further elaborating.

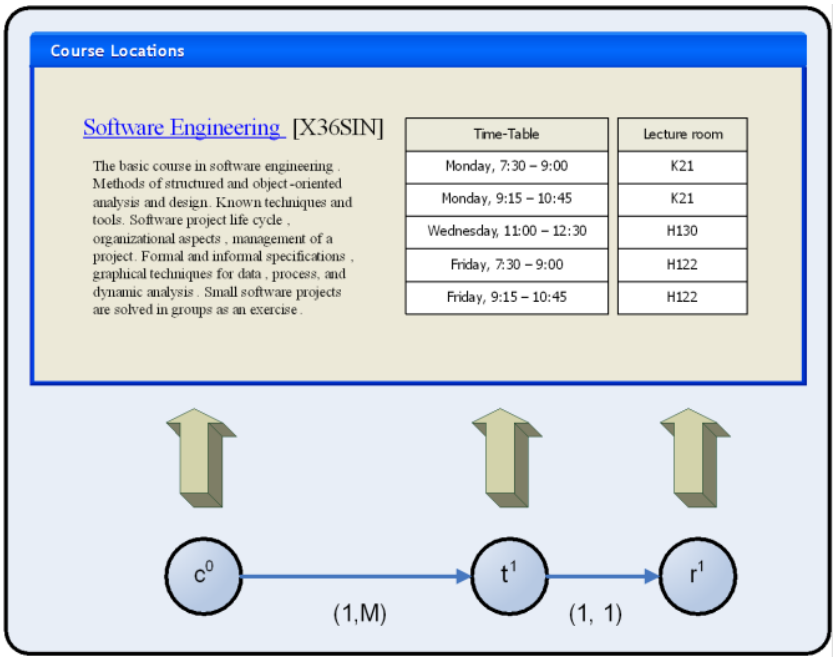


Figure 6. Example 1, GUI presentation.

- We work only with a few simplest elements of user interface.
- At the present time we have not completed a software tool for our approach.

In conclusion, our lightweight formal method intended for deriving the user interface schemes by graph transformations from conceptual data model specifications is not fairly good yet. We use the graph transformations paradigm a little bit different way from the common usage described in [4]. 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 semantic information from the source conceptual graph for the construction of proper user interface. Next, we must complete a software tool supporting hereinbefore proposed transformation. Finally, we will work on the specification of the rules for the construction of relational algebra queries or more precisely 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.

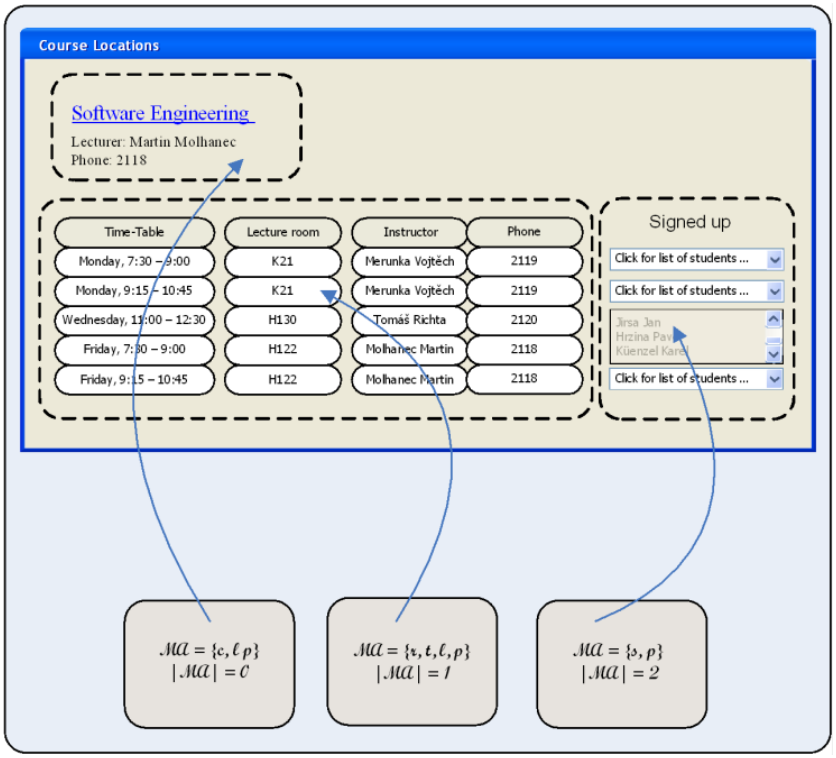


Figure 7. Example 2, GUI presentation.

References

1. Molhanec, M.: Typologie uživatelského rozhraní (Typology of user interface). In: Tvorba software'97, pp. 88–97. Tanger, Ostrava (1997)
2. WebML – Web Modeling Language, `http://webml.org`
3. Knott, R.P., Merunka, V., Polák, J.: The BORM methodology: a third-generation fully object-oriented methodology. In: Knowledge-Based Systems 3(10). Elsevier Science Publishing, New York (2003)
4. Rozenberg, G., editor: Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations. World Scientific (1997)
5. Guizzardi, G., Herre, H., Wagner, G.: On the General Ontological Foundations of Conceptual Modeling. In: Proc. of 21 Intl. Conf. on Conceptual Modeling (ER 2002), Lecture Notes in Computer Science. Springer-Verlag, Berlin, (2002)
6. 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)
7. 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)
8. 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)

9. 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)