APPLICATION OF METAEDIT+ TOOL TO SPECIFY INFORMATION SYSTEM MODELING CONCEPTS

Vladimir Dimitrieski, Milan Čeliković, Sonja Ristić, Ivan Luković
{dimitrieski, milancel, sdristic, ivan}@uns.ac.rs
Fakultet Tehničkih Nauka,Univezitet u Novom Sadu

Abstract - *In this paper we present a formal specification of platform independent model (PIM) concepts of the Integrated Information Systems CASE Tool. PIM concepts are described by Graph-Object-Property-Port-Role-Relationshipmeta-modeling language used in MetaEdit+ meta-modeling environment. Through our previous research we have specified PIM concepts using Ecore implementation of Meta Object Facility 2.0 in Eclipse Modeling Framework. As our goal is to create a formal specification of PIM concepts that designers would later use to graphically specify the structure of an information system, MetaEdit+ was the obvious choice. MetaEdit+ provides an integrated environment for definition of abstract syntax as well as the definition of concrete syntax using graphical symbol editor. In this paper we give an example of information systemmodeling using specified PIM concepts.*

1. INTRODUCTION

Abstraction is a powerful modeling mechanism both in Computer Science and Software Engineering. It is a cognitive means according to which we concentrate on the essential features of our subject of thought, and ignore irrelevant details, in order to overcome complexity at a specific stage of a problem solution. Raising the level of abstraction has been the cause of the largest leaps in developer productivity, latest coiming withthe introduction of the Domain Specific Modeling (DSM) followed by a systematic usage of Domain Specific Languages (DSLs).

One of the areas in which the DSM is used, is the development of software development tools specific to the selected problem domains.We are interested in the dimain of Information Systems (IS) development. In this domain, DSLs may take a remarkable role and may be used for various purposes, such as: conceptual modeling, specification of rules and constraints, code generation, generation of test cases etc.Through our previous research, we are developing a textual DSL, named IIS*CDesLang. It is aimed at modeling PIM specifications of an IS. Our research goals are to couple it with our model driven software development tool, named Integrated Information Systems CASE Tool (IIS*Case). IIS*Case provides IS modeling and prototype generation. At the level of PIM specifications, IIS*Case provides conceptual modeling of database schemas and business applications. Performing a chain of model-to model and model-to-code transformations of PIM models, we obtain executable program code of software applications and database scripts for a selected platform. The detailed overview of DSL usage in IS development can be found in [1].

Our previous research leads to conclusion that there was a strong need to have platform independent models (PIM) concepts specified formally in a platform independent way, i.e. to be fully independent of repository based specifications that typically may include some implementation details. Our current research is based on three related approaches to formally describe IIS*Case PIM concepts that can be used in IS design process. The first one is based on the attribute grammars through which we are developing the textual DSL, named IIS*CDesLang [2].IIS*CDesLang meta-model is developed under a visual programming environment for attribute grammar specifications named VisualLISA [3].The second approach is based on Meta Object Facility (MOF) [4].Approach based on MOF is described in [5]. As we could not find standardized implementation of MOF, in that work we decided to use Ecore meta-meta-model to implement PIM model. Ecore is Eclipse's implementation of MOF 2.0 in Java programming language which is provided by Eclipse Modeling Framework (EMF) [6]. The third approach, presented in this paper, is based on MetaEdit+'s Graph-Object-Property-Port-Role-Relationship (GOPPRR) meta-modeling language [7, 8]. In this approach, our goal was to create a formal specification of PIM concepts that designers would later use to graphically specify the structure of an IS. MetaEdit+ provides an integrated environment for definition of PIM concepts as well as the definition of their graphical representation using graphical symbol editor. After the definition of meta-model, specified concepts are to be loaded into the MetaEdit+'s repository and then used to define IS models through graphical representation of these concepts.

One of the important reasons for having three different approaches in our research is a need to experiment and analyse the characteristics of all these approaches. We need to recognize to what extent the approaches meet our needs in modeling of an IS. IIS*CDesLang, as a textual DSL, is needed to provide more experienced users with a textual language and a tool for creating PIM specification more efficiently. From MOF based approach, we obtain a sound domain analysis specification, necessary to precisely understand and present the application domain, and verify the structure of the IIS*Case repository being already developed. We are also researching the use of GOPPRR, as we want to give modelers an additional possibility of using graphical DSL in designing an IS. In [9] we give a comparisson between MOF and GOPPRR based appraches.

Apart from Introduction and Conclusion, the paper is organized in three sections. In Section 2 we present related works, while in Section 3 we give a presentation

of IIS*Case PIM concepts specified through the meta-model implemented in MetaEdit+. In Section 4 we present an example of IS specification through MetaEdit+ by usage of concepts described in this paper.

2. RELATED WORK

As meta-modeling is widely spread area in many recent research activities, there are number of papers covering this area. As MetaEdit+ is popular graphical meta-modeling tool, we have found a number of papers covering the usage of MetaEdit+ in meta-modeling. However, none of those papers presented formal approaches to specifying meta-model implementation and design of CASE tools in MetaEdit+.

A number of DSM examples and their meta-models may be found in [10]. This book presents a practical design and development lessons covering topics including domain modeling, language definition, code generation and DSL tooling. The book also contains case studies in telephony, insurance, home automation and mobile applications that clearly illustrate the use of DSLs in different domains and are based on actual client experiences. Authors of [11] present how domain-specific languages, along with domain-specific frameworks and generators, can support formal specification and document rendering in directory publishing. For that purpose they have developed a graphical DSL named DVDocLang that is highly applicable for user-driven conceptual modeling. In [12] authors depict the implementation of the basic concepts and tools required in modeling a Process Modeling Language (PML) and the integration of user and environment process models in MetaEdit+. The concepts and tools introduced aid in providing full customizability through a project to adapt its own PML for designing models of a process to be enacted. Authors of [13] describe the fundamental concepts and benefits of DSM, and discuss the implementation of a DSM language for a home control system using MetaEdit+.

3. IIS*CASE META-MODEL

In this paper we are focusing only on PIM concepts of the IIS*Case tool specified in MetaEdit+. Hereby we give a brief overview of following concepts: *Project*, *ApplicationSystem*, *ApplicationType*, *_ProgramUnit*,*BusinessApplication*, *_FormType*, *ComponentType*, as well as an overview of *_Fundamental* concepts: *Attribute* and *_Domain*. In this paper we are using MetaEdit+'s syntax for naming our concepts. Main IIS*Case concepts with their relationships and properties are illustrated in Figure 1.

3.1. PROJECT, APPLICATION SYSTEM, DOMAIN AND ATTRIBUTE CONCEPTS

Everything that exists in IIS*Case's repository is always stored in a context of a project. Therefore, the central concept of the meta-model illustrated in Figure 1 is the concept of *Project*. As one project is one Informational System specification, a designer may have only one instance of *Project*in a single MetaEdit+'s graph. If a designer needs to specify another IS, he or she may create another graph with new instance of the *Project*on it. However, a designer may not have two projects with the same name as project's name must have globally unique value. Every instance of a *Project* can be connected to zero or more instances of the *ApplicationSystem* and zero or more instances of any descendant of *_Fundamentals*. Therefore *ApplicationSystems*and *_Fundamentals* (Fundamental concepts) may be seen as subunits of a *Project*.

ApplicationSystems are organizational parts, i.e. segments of a project. Each application system has its name and description as mandatory properties. Besides, each application system may have many child application systems. In that way, designers may create application system hierarchies in an IIS*Case project. Application system hierarchy is modeled by a recursive relationship. Designers of an IS may create application systems of various types. By the

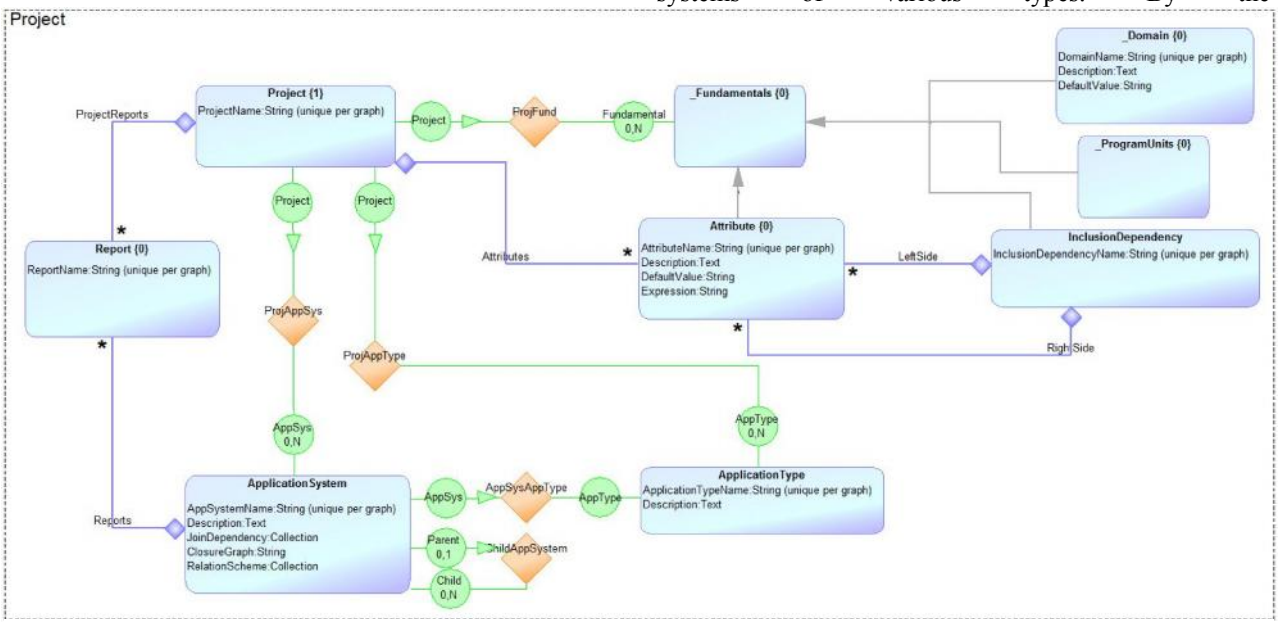

Figure 1. A meta-model of main IIS*Case concepts

*ApplicationType*concept, designers introduce various application system types and then associate each application system instance with one application type. Various kinds of the IIS*Case repository objects may be created at the level of an application system, but in next section we give a brief overview of only two of them, as they are crucial in creating PIM specifications: *_FormType* and *BussinessApplications*.

Fundamental concepts are formally independent of any application system. They are created at the level of a project and may be used in various application systems later on. Fundamental concepts comprise zero or more:*Attributes*, *_Domains*, *_ProgramUnits* and*InclusionDependencies*.

For each project a designer must provide the project's name as its mandatory property. A designer may specify one or more *Attributes*. Any *Attribute* must be specified in a context of the current project defined in the same graph. Therefore, each *Project* has a collection of *Attributes*. Each attribute is identified only by its name. Therefore, we obey to the Universal Relation Schema Assumption (URSA) [14]. Apart from the mandatory name property, each attribute has mandatory description and the *ApplicationType* property specifying if the attribute belongs to the database schema. Most of the project's attributes are to be included into the future database schema. However, we may have attributes representing some calculated values in reports or screen forms that are not included into the database schema. Therefore, we classify attribute types in IIS*Case as:*Included* in database schema and*Non-included* in database schema.

The attribute may be specified as elementary or renamed. An elementary attribute is a "simple" attribute that is defined with a given semantics. A renamed attribute references previously defined attribute. The source of such an attribute is the referenced attribute, but with different semantics. If a designer specifies that an attribute A1 is renamed from A, in fact he or she introduces an inclusion dependency of the form $A1 \subset A$ at the level of a universal relation scheme. Accordingly, we automatically introduce that inclusion dependency in IIS*Case repository. In our meta-model, inclusion dependency is defined as the object *InclusionDependency* inheriting *_Fundamentals* in the part of our meta-model, presented in Figure 1.

Each attribute must have a domain from which values can be assigned to the attribute. In the database terminology a notion of domain denotes a specification of allowed values of database attributes. In this paper, we use a notion of domain with a meaning that is common in the area of databases. In our PIM, notion of domain is modeled with *_Domain* concept. Every domain has name, description and default value properties. A name and a description must be specified by a designer. A name of the domain is unique in a model. Domains are classified as primitive domains and user defined domains. Primitive domains represent primitive data types in various formal languages such as string, integer, float, etc. Through this concept we allow a designer to create his or her private domain according to the need of the project thus increasing the expressive value of the model. User defined domains are created by referencing previously created primitive or user defined domains and adding more restrictions on its values using check condition.

In Section 4 we present an example of the specification of aforementioned concepts as a part of an IS segment modeling.

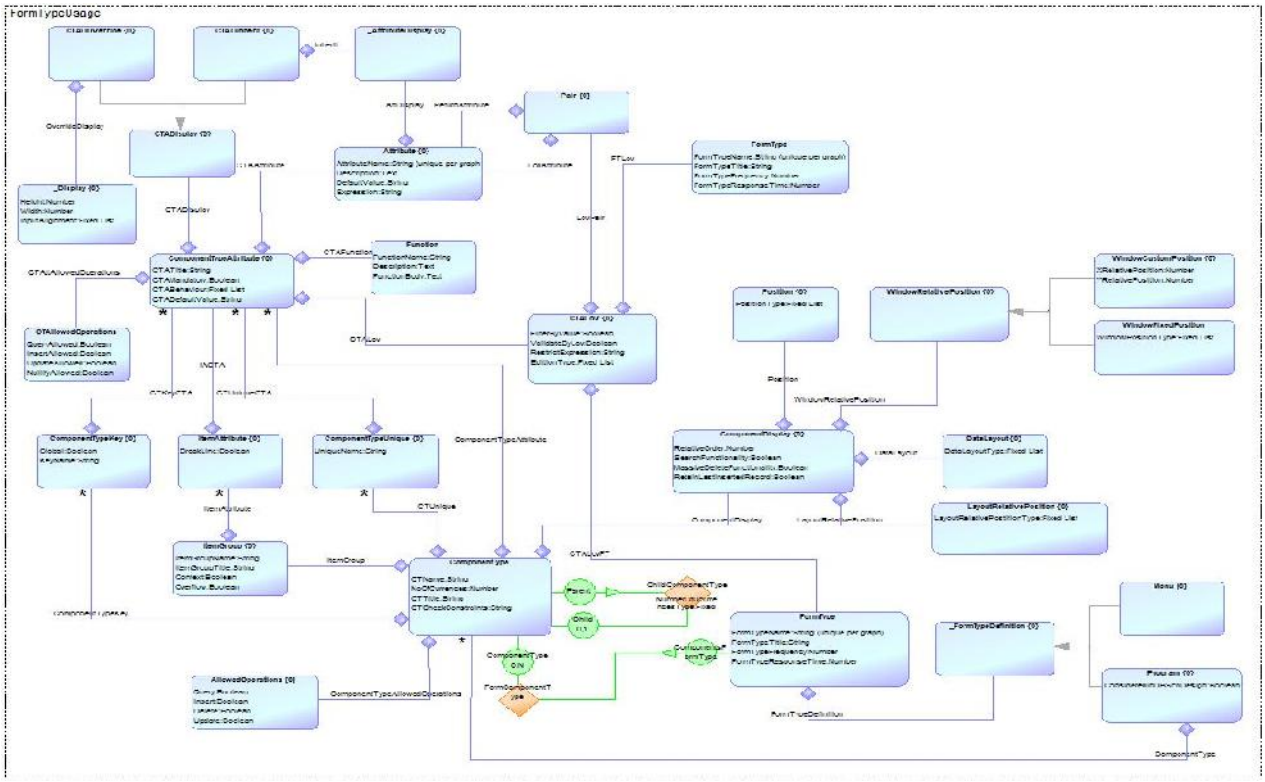

Figure 2. A meta-model of *_FormType* concept

3.2. PROGRAM UNITS, FORM TYPE AND BUSINESS APPLICATION

In this subsection we give a brief overview of *_ProgramUnits*, *_FormType* and *BusinessApplication* concepts.

The *_ProgramUnits* concept is used to express complex application behaviour. We classify program units as:*Functions*, *_Packages* and *_Events*.

The *Function* concept is used to specify any complex functionality that may be used in other project specifications. Each function has its mandatory name and description properties as well as a mandatory return type of a function that is an instance of any object inheriting the *_Domain* object. For each function a function body may be formally specified. In IIS*Case that can be achieved using the Function Editor tool. A function specification may include a list of parameters. For eachparameter a designer must specify the parameter sequence number and the name of a parameter. Each parameter has its own type that is an instance of one of the descendants of the *_Domain* object. A parameter may have its default value specified. With respect to the ways of exchanging values between the function and its calling environment, we classify formal parameters as: In, Out and In-Out, with a usual meaning as it is in many general purpose programming languages.

IIS*Case provides the mechanism for grouping created functions into packages (*_Package* concept). Each function may be included into one or more packages, or may stay as a stand-alone object. By the location of the deployment in a multi-layer architecture, the packages are classified as: Client packages, Database server packages and Application server packages.

The *_Event* concept is used to represent any software event that may trigger some action under a specified condition. Similar to the packages, by the location of the deployment in a multi-layer architecture, we classify events as: Client event, Database server event and Application server event.

A *_FormType* is the main modeling concept in IIS*Case. It generalizes a document types i.e. screen forms or reports by means of users communicate with an IS. It is a structure defined at the abstraction level of a schema. Using the form type concept, a designer specifies a set of screen or report forms of the transaction programs and, indirectly, specifies database schema attributes and constraints. Each particular business document is an instance of a form type. The *_FormType* concept is illustrated in the Figure 2.

Each form type has a name that identifies it in the scope of a project, a title, frequency of usage, response time and a usage type. Frequency is an optional property that represents the number of executions of a corresponding transaction program per time unit. Response time is also an optional property specifying expected response time of a program execution. By the usage type property we classify form types as: Menu and Program. Menu form types are used to model menus without data items. Program form types model transaction programs with UI providing data operations over a database. They may represent either screen forms for data retrievals and updates, or just reports for data retrievals. As a rule, a user interface of such a program is rather complex. Program form type can be either considered or not considered in database schema design. If the form type is considered in database schema design, it is used later as the input into the database schema generation process. On the other hand, if a form type is not considered in database schema design they are not later used in the process of database schema generation. They may represent database report forms only. Each program form type is a tree of component types. Component types are used to specify, display and group data in forms. In Section 5 we present an example of form type specification in the process of IS modeling.

BusinessApplication concept represents the way to formally describe an IS functionality and is organized through a structure of form types. Each business application has a mandatory name and a description. One of the form types included into the application system structure must be declared as the entry form type of the application. It represents the first transaction program invoked upon the start of the application.

4. AN EXAMPLE OF IS MODELING THROUGH METAEDIT+

In this section we present an example of an IS segment specification through MetaEdit+. We present an example of IS as seen in the domain by an IS designer. Later we illustrate the usage of previously described modeling concepts in IS segment specification through MetaEdit+.

A form type defined in the child application system *Teacher service* of the parent application system *Faculty organization* is illustrated in Figure 3. It refers to information about *teacher's courses* (TC). It has two component types: *Teacher*, representing instances of teachers, and *Courses*, representing instances of courses for each teacher. By the form type TC, we allow having teachers with zero or more courses. Component type attributes are presented inside of rectangle representing component types. *TeacherId* is the key of the component type *Teacher*, while *CourseId* is the key of *Courses*. By this, each course is uniquely identified by *CourseId* within the scope of a given teacher. Allowed database operation for *Teacher* is only read (shown in a small rectangle on the top of the rectangle representing the component type), while the allowed database operations for *Courses* are read, insert, update and delete.

In Figure 4 we illustrate this example modeled in MetaEdit+. In top right corner of each element we have specified a type of that element. Central element of this example is *FacultyIS* project. In this project we have specified two application systems: *Faculty organization* and its child application system *Teacher service*.

Bothapplication systems are connected to the project using

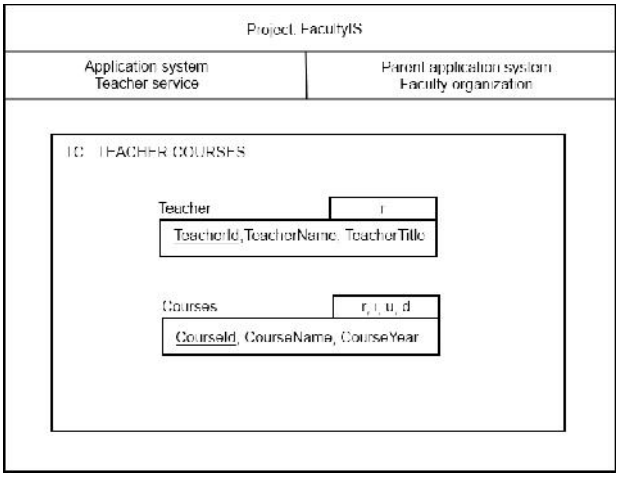

Figure 3. A form type in the TeacherService application system

ProjAppSys relationship. *Teacher service* has been also related to the parent application system *Faculty organization* using the *ChildApplicationSystem* relationship. Child role notation is visible on the relationship, near the child system. For each application but the specification is not shown in Figure 4. To see the specification we need to open *Teacher* component type details. *TeacherId* is specified as the component type key. Only read database operation is allowed over this component type. This is shown in the top right corner of component's graphical representation. The second component type is *Courses* and it contains attributes: *CourseId*, *CourseName* and *CourseYear*. Read, insert, update and delete database operations are allowed over this component type. *CourseId* is the component type key. There are numerous display properties that are to be set in the process of modeling the IS, but are omitted here due to limited space and to avoid burdening our example with too many details.

MetaEdit+ provides a generator for creating meta-objects in the repository from the graph representation of meta-model. These meta-objects can then be assigned with

system we have specified its description as a mandatory property. The description is not illustrated in Figure 4 as it would take a lot of space to display. Each application system is classified to be of one application type. In this example we have created only one type, *Project subsystem*, which is associated with both application systems using the *AppSysAppType* relationship. We have also associated the application type to the project using *ProjAppType* relationship. As we need domains to specify attributes and parameters of forms, we have specified two primitive domains: *Integer* and *String*, and from those primitive domains we have derived two user defined domains: *Positive Integer* and *Short String*. They are related to the project with *ProjFund* relationship. For every domain, description and length required properties are specified but they are not illustrated in Figure 4 to save space.

For *Teacherservice* system we have specified one form type named *Teacher courses*. *Teacher courses*has the similar string as a title and has no parameters. A form type usage has been specified to be program usage but is not illustrated in Figure 4. For this form type, we have specified two component types. The first is *Teacher* component type. It contains attributes: *TeacherId*, *TeacherName* and *TeacherTitle* which are fully specified

graphical symbols and used for specification of IS models. Each instance of a meta-object in the graph is specified by assigning values to properties through generated input forms. Forms are well generated by MetaEdit+ and a designer may easily specify error messages and values that can be presented back to users through input forms. According to our previous practical experiences, we believe that the specification of IS models through MetaEdit+ may be somewhat difficult for a designer. This is clearly obvious when a designer tries to specify object members of other objects. In some cases, the specification of a single object member may require several input forms to be filled. Those objects may have their own object members, requiring a designer to fill in a hierarchy of input forms to specify one object on the graph. This problem is sometimes avoided by reusing already specified objects saved in the graph's repository. Despite that, we are convinced that modeling

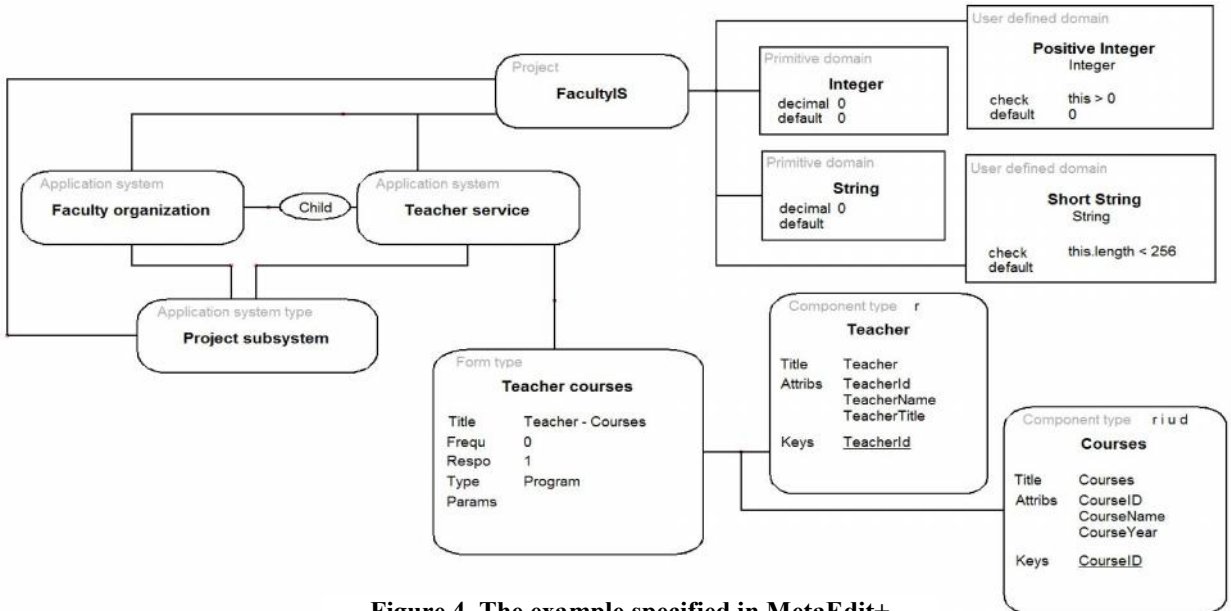

Figure 4. The example specified in MetaEdit+

is done much easier by the IIS*Case tool. The IIS*Case provides the custom input forms for the specification of model elements.

5. CONCLUSION

In this paper we presented the IIS*Case PIM meta-model specified by GOPPRR meta-modeling language used in MetaEdit+. Unlike other meta-modeling environments that we have worked in, MetaEdit+ allows us to easily generate meta-objects in the repository form our meta-model. Then we can use MetaEdit+'s workbench to make an IS model containing this meta-objects. MetaEdit+'s symbol editor also allows instances of meta-objects to have distinctive graphical representation. Therefore designers can easily read and specify models in the graphical way. By specifying an IS model, designers of an IS are able to try out their ideas and check the validity of their models.

The detailed meta-model that we have developed in MetaEdit+ may be used to generate the documentation of PIM concepts. We believe that the formal specification of our meta-model is not for documentation purposes only, but it is a necessary step in creating a graphical DSL to support IS design and give another view of the IS description. MetaEdit+ supports the evolution of languages without destroying previously created meta-objects in the repository. New elements are just merged with already created ones. This allows us to experiment with new approaches and ideas before we implement them in IIS*Case tool.

We may use meta-model presented in this paper in the verification of relational database schemas. We assist designers to detect conflicts at the level of relational database model, and then we can help them at the level of meta-models to find the appropriate solution of detected problems. Although the algorithms for detection and resolving constraint collisions at the level of relational data model have already been implemented in IIS*Case, we want to raise the process of collision resolving at the PIM level of abstraction.

Our future research will include the development of generators in MetaEdit+ that will be able to check models for errors and at the level of platform independent IS concepts. Our future research will also include a generation of program code that can be imported in our IIS*Case tool. Imported models may be used to enrich IS specification with more details, and to use all benefits of editors and functions of the IIS*Case.

ACKNOWLEDGEMENTS

The research presented in this paper was supported by Ministry of Education and Science of Republic of Serbia, Grant III-44010: Intelligent Systems for Software Product Development and Business Support based on Models.

REFERENCES

[1] Luković, I., Ivančević, V., Čeliković, M. and Aleksić, S., *DSLs in Action with Model Based Approaches to Information System Development*, in the book: Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, IGI Global, USA, ISBN: 978-1-4666-2092-6, pp. 502-532, 2013.

[2] Luković, I., Varanda Pereira, M. J., Oliveira, N., Cruz,D. and Henriques, P. R.,*A DSL for PIM Specifications: Design and Attribute Grammar based Implementation*, ComSIS, ISSN: 1820-0214, Vol. 8, No. 2, pp. 379-403, 2011.

[3] Oliveira, N., Varanda Pereira, M. J., Henriques, P. R., Cruz, D. and Cramer, B., *VisualLISA: A Visual Environment to Develop Attribute Grammars*, ComSIS, ISSN:1820-0214, Vol. 7, No. 2, pp. 265-289, 2010.

[4] *Meta-Object Facility* [Online] : http://www.omg.org/mof/

[5] Čelković, M., Luković, I., Aleksić, S. andIvančević, V., *A MOF based Meta-Model and a Concrete DSL Syntax of IIS*Case PIM Concepts*, ComSIS, ISSN: 1820-0214, Vol. 9, No. 3, pp. 1075-1103, 2012.

[6] *Eclipse Modeling Framework*, [Online] : http://www.eclipse.org/modeling/emf/.

[7] *MetaEdit+* [Online] : http://www.metacase.com/

[8] *GOPPRR* [Online] : http://www.metacase.com/support/50/manuals/mwb/Mw-1_1_1.html

[9] Dimitrieski,V., Čeliković, M., Ivančević, V. and Luković, I.,*A Comparison of Ecore and GOPPRR through an Information System Meta Modeling Approach*, 8th European Conference on Modelling Foundations and Applications (ECMFA), Technical University of Denmark, Joint Proceedings, ISBN 978-87-643-1014-6, pp. 217-228, 2012.

[10] Kelly,S. and Tolvanen, J.-P., *Domain-Specific modeling: Enabling Full Code Generation*, John Wiley & Sons, Inc., ISBN: 978-0-470-03666-2, USA, pp. 93-224, 2008.

[11] Đukić, V., Luković, I. and Popović, A., *Domain-Specific Modeling in Document Engineering*, Federated Conference on Computer Science and Information Systems (FedCSIS), Szczecin, Poland, Proceedings, IEEE Computer Society Press, ISBN 978-83-60810-39-2, pp. 825-832, 2011.

[12] Koskinen, M. and Marttiin,P., *Process support in MetaCASE: implementing the conceptual basis for enactable process models in MetaEdit+*, IEEE, Eighth Conference on Software Engineering Environments, pp. 110–122, 1997.

[13] Hammond, J. L., *Domain-specific modeling significantly reduces development time*, [Online] http://www.metacase.com/papers/ece_april2008.pdf

[14] Luković,I., *From the Synthesis Algorithm to the Model Driven Transformations in Database Design*, Proceedings of 10th International Scientific Conference on Informatics, Herlany, Slovakia, ISBN 978-80-8086-126-1, pp 9-18, 2009.