ISBN 978-83-60810-22-4

# A MOF based Meta-Model of IIS*Case PIM Concepts

Milan Čeliković,
University of Novi Sad,
Faculty of Technical Sciences,
Trg D. Obradovića 6,
21000 Novi Sad, Serbia,
Email: milancel@uns.ac.rs

Ivan Luković,
University of Novi Sad,
Faculty of Technical Sciences,
Trg D. Obradovića 6,
21000 Novi Sad, Serbia,
Email: ivan@uns.ac.rs

Slavica Aleksić,
Vladimir Ivančević
University of Novi Sad,
Faculty of Technical Sciences,
Trg D. Obradovića 6,
21000 Novi Sad, Serbia,
Email: {slavica,
dragoman}@uns.ac.rs}

*Abstract*—**In this paper, we present platform independent model (PIM) concepts of IIS*Case tool for information system (IS) modeling and design. IIS*Case is a model driven software tool that provides generation of executable application prototypes. The concepts are described by Meta Object Facility (MOF) specification, one of the commonly used approaches for describing meta-models. One of the main reasons for having IIS*Case PIM concepts specified through the meta-model, is to provide software documentation in a formal way, as well as a domain analysis purposed to create a domain specific language to support IS design. Using the meta-model of PIM concepts, we can generate test cases that may assist in software tool verification.**

## I. Introduction

IIS*Case is a software that provides a model driven approach to information system (IS) design. It supports conceptual modeling of database schemas and business applications. IIS*Case, as a software tool assisting in IS design and generating executable application prototypes, currently provides:

- Conceptual modelling of database schemas, transaction programs, and business applications of an IS;
- Automated design of relational database subschemas in the 3rd normal form (3NF);
- Automated integration of subschemas into a unified database schema in the 3NF;
- Automated generation of SQL/DDL code for various database management systems (DBMSs);
- Conceptual design of common user-interface (UI) models; and
- Automated generation of executable prototypes of business applications.

In order to provide design of various platform independent models (PIM) by IIS*Case, we created a number of modelling, meta-level concepts and formal rules that are used in the design process. Besides, we have also developed and embedded into IIS*Case visual and repository based tools that apply such concepts and rules. They assist designers in creating formally valid models and their storing as repository definitions in a guided way. Main features of IIS*Case and the specification of its usage may be found in [10].

There is a strong need to have 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 two related approaches to formally describe IIS*Case PIM Concepts. One of them is based on MOF and the other one on a textual Domain Specific Language (DSL). In [1], we give a specification of the IIS*Case textual modelling language, named IIS*CDesLang that formalizes IIS*Case PIM concepts and provides modelling in a formal way. IIS*CDesLang meta-model is developed under a visual programming environment for attribute grammar specifications named VisualLISA [17].

In this paper, we propose a meta-model of IIS*Case PIM concepts, which is based on the Meta Object Facility (MOF) 2.0. MOF 2.0 is a common meta-meta-model proposed by Object Management Group (OMG) where a meta-model is created by means of UML class diagrams and Object Constraint Language (OCL) [14]. As we could not find standardized implementation of MOF, we decided to use Ecore meta-meta-model. Ecore is the Eclipse implementation of MOF 2.0 in Java programming language which is provided by Eclipse Modelling Framework (EMF) [9]. Ecore concepts are not always identical to MOF 2.0 concepts, but they are expressive enough to create our IIS*Case meta-model. A benefit of such a meta-model is providing software documentation in formal way. Besides, created meta-model can be used for the software tool verification in EMF environment. It also represents a domain analysis specification necessary to create IIS*CDesLang, as a textual DSL to support IS design.

In Figure 1 we illustrate the four layered architecture of our solution, which is tailored from OMG four–layered architecture standard. Level M3 comprises meta-meta-model (MOF 2.0) [7] that is used for implementation of the IIS*Case meta–model (M2). M2 level represents the IIS*Case PIM meta-model specified by MOF specification

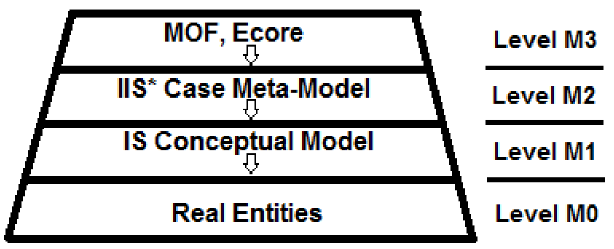

Fig 1. Four layered meta-data architecture

and implemented in EMF. Using the IIS*Case PIM meta-model, a designer can specify and implement a conceptual model of an IS that is placed at the M1 level of the four-layered data architecture from Figure 1. By using applications of an IS generated by IIS*Case, end users manipulate real data, i.e. they create and use models of entities from real world (M0), using the conceptual model (M1).

Apart from Introduction and Conclusion, the paper is organized in two sections. In Section 2, we present a related work, while in Section 3 we give a presentation of IIS*Case PIM concepts specified through the meta-model that is implemented in EMF environment.

## II. RELATED WORK

Nowadays, meta-modeling is widely spread area of research and there is a huge number of references covering MOF based meta-models. However, we could not find papers presenting formal approaches to specifying meta-model implementation and design of CASE tools, based on MOF or Ecore meta-meta-models.

We found a vast number of meta-model specifications and implementations based on MOF or Ecore specifications. Meta-models based on MOF are presented in [2], [3]. The authors in both papers propose the meta-models of the Web Modeling Language. The meta-model specification and design is implemented under EMF environment. Defining W2000 [2] as a MOF meta-model the authors specify it as an UML profile. In [3], the authors provide a solution for generation of MOF meta-models from document type definition (DTD) specifications [15]. A formal specification of OCL is given in [4]. In their meta-model, the authors precisely define the syntax of OCL, as it is given in [14]. They propose a solution for the presented meta-model integration with the UML meta-model. In [5], the authors propose the Kernel MetaMetaModel (KM3) that represents a DSL for meta-model definition. In [16], the authors propose the UML Profile, EUIS, for the specification of business applications' user interfaces. Their solution provides automatic interface code generation that is based on their own HCI standard. They developed a DSL specified as UML Profile that offers user interface modeling and generation.

There are various meta-modeling tools that are generally based on their own meta-meta-model specifications. One of them is Generic Modeling Environment (GME) [8], a configurable toolkit for domain specific modeling and program synthesis based on UML meta-models. MetaEdit+ [6] allows the creation and the design of meta-models in graphical editor using the Object-Property-Role-Relationship data model. All of these tools can also be used for the IIS*Case PIM meta-model description in a formal way.

## III.IIS*CASE META-MODEL

In this paper, we present the IIS*Case PIM meta-model specified by Ecore meta-meta-model. Hereby we give an overview of the following IIS*Case main PIM concepts: *Project*, *Application system*, *Form type*, *Component type*, *Application type*, *Program unit* as well as *Fundamental concepts* such as *Attributes* and *Domains*. A model of the

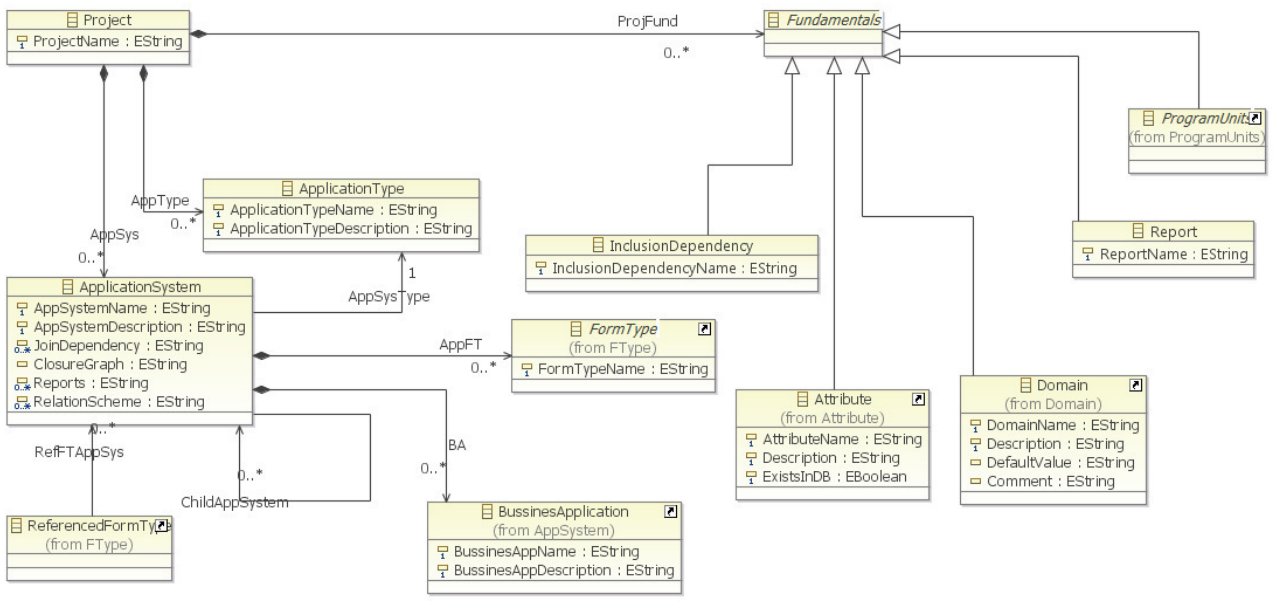

Fig. 2. A meta-model of IIS*Case main PIM concepts

IIS*Case main concepts with their properties and relationships is presented in Figure 2. More information about these concepts may be found in [10] and [11], as well as in many other authors' references.

### *A. Project*

A modeling process in the IIS*Case tool is organized through one or more projects. Therefore, the central concept in our meta-model from Figure 2 is *Project*. For each project, a designer defines the project name as its mandatory property. All existing elements in the repository of IIS*Case are always created in the context of a project. *Fundamental concepts* and *Application systems* are subunits of a *Project*. For each project, we can define zero, or more instances of the *Application system*. A designer of an IS can create application systems of various types. By the *Application type* concept, a designer may introduce various application system types and then associate each instance of an application system to exactly one application type.

In the following example, we illustrate the usage of the application system and application type concepts. We have two application systems created: *Student Service* and *Faculty Organization*. *Student Service* is the child application system of the parent application system *Faculty Organization*. Two kinds of application types are created: a) *System* and b) *Subsystem*. Further, we classify application system *Faculty Organization* as the *System* and *Student Service* as *the Subsystem* application type.

Each project is organized through application systems and fundamental concepts. Fundamental concepts are formally independent of any application system. Fundamental concept instances can be used in more than one application system, because they are defined at the level of a project.

Fundamental concepts comprise zero or more:

- *Attributes*,
- *Domains*,
- *Program units*,
- *Reports* and
- *Inclusion dependencies*.

At the level of a project, IIS*Case provides generation of various types of repository reports.

### *B. Domain*

*Domains* specify allowed values of database attributes. They are classified as:

- Primitive and
- User defined.

Therefore, in our meta-model, there are two classes: *PrimitiveDomain* and *UserDefinedDomain* that are subclasses of a *Domain* class.

Primitive domains represent primitive data types that exist in formal languages, such as string, integer, char, etc. The reason for existence of user defined domain concept is to allow designers to create their own data types in order to raise the expressivity of their models. Each domain has its domain name, description and default value. At the level of a primitive domain, a designer may specify *length required* item value. It specifies if a numeric length: must be, may be, or is not to be given. For user defined domains, a designer needs to define a domain type and a check condition. IIS*Case supports two classes of user defined domains:

- Domains created by the inheritance rule and
- Complex domains.

A domain created by the inheritance rule references a specification of some primitive or user defined domain. By the inheritance, all the rules defined at the level of a referenced (superordinated) domain also hold for the specified domain. We call it a child domain.

Complex domains may be created by the tuple rule, set rule, or choice rule. A domain created by the tuple rule we call simply tuple domain, because it represents a tuple of values. The items of such a tuple structure are some of already created attributes. A domain created by the choice rule we call a choice domain. It is specified in almost the same way as a tuple domain. The choice domain concept is the same as the choice type of XML Schema Language. Each value of a choice domain corresponds to exactly one attribute. A set domain represents sets of allowed values over a specified domain.

Check condition is a regular expression that can additionally constrains possible values of a domain created by a designer.

*Domain* concept allows definition of display properties of screen items that correspond to attributes and their domains. Each domain corresponds to exactly one element of type *Display*. The *Display* concept specifies rules, later used by the application generator to generate screen or report items that correspond to some of the attributes, and attributes correspond to some of domains. Technical aspects of the display properties implementation may be found in [12] and [13].

### *C. Attribute*

In Figure 3, we present a meta-model of the IIS*Case *Attribute* concept. Each attribute in an IIS*Case project is identified by its name. It also has a description and a Boolean specifier if it belongs to the database schema. In practice, the most of created attributes belong to the database schema. For attributes representing derived (calculated) values in reports or screen forms a designer may decide if they are to be included in the database schema. By this, we classify attributes as: a) included or b) non-included in a database schema.

According to the way how an attribute gains a value, we classify attributes as: a) non-derived or b) derived. A value of a non-derived attribute is created by an end user. A value of derived attribute is always calculated from the values of other attributes, by applying some function, i.e. a calculation formula. There is a rule that any non-included attribute must be specified as derived one.

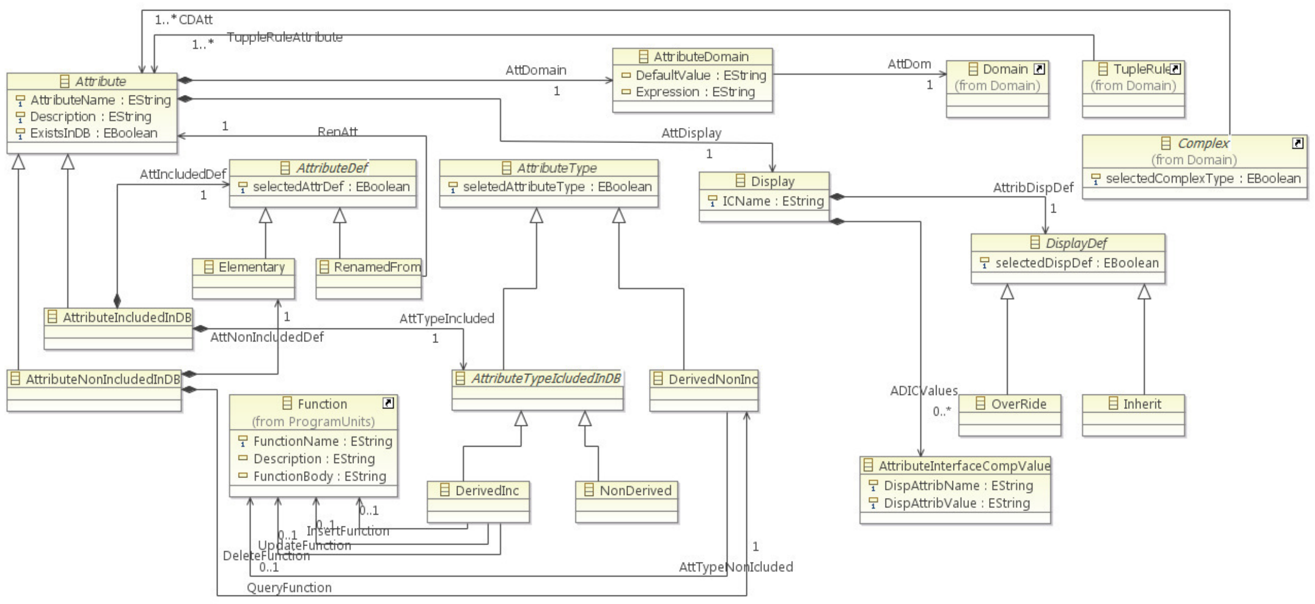

Fig. 3. A meta-model of the IIS*Case Attribute concept

The function that is used to calculate a derived attribute value is formally specified in the IIS*Case repository. Additionally a designer may specify parameters that are passed into the function. The *Function* concept will be presented in the following subsection, Program Units. If an attribute is non-included in a database schema, the function is referenced as a query function. Only derived attributes that are included in a database schema may additionally reference three IIS*Case repository functions specifying how to calculate the attribute values on the following database operations: insert, update and delete.

The attribute may be specified as a) elementary or b) renamed. A renamed attribute references a previously defined attribute. The source of such an attribute is the referenced attribute, but with the different semantics. The renamed attribute needs to be included in database schema.

To each attribute a domain must be associated. This association allows defining a default value and a check condition. If the attribute value is not specified, the default value is assigned to it. Check condition is the attribute check expression that represents the regular expression that additionally constrains the value of the attribute.

At the level of an attribute, we can specify the display properties. The concept of the *Display* properties is same as the one at the level of the *Domain* concept. The values of display properties, specified at the level of the associated domain, may be inherited or overridden according to the requirements of an IS project.

### *D. Program Units*

The *Program unit* concept is used to express complex application functionalities. We classify program units as: a) *Functions*, b) *Packages* and c) *Events*.

The *Function* concept is used to specify any complex functionality that later may be used in other specifications. Each function has its name and return type that are mandatory properties, as well as a formal specification of a function body and a description that are optional. The return type is a reference to a domain. A function specification may include a list of formal parameters. Each formal parameter of a function is specified by its name and a sequence number, as mandatory properties. Exactly one domain is associated to each formal parameter. Any parameter may also have a default value specified. With respect to the ways of exchanging values between the function and its calling environment, we classify formal parameters as: a) In, b) Out and c) In-Out, with a usual meaning as it is in many general purpose programming languages.

IIS*Case provides grouping created functions into packages. 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: a) Database server packages, b) Application server packages and c) Client packages. A package is identified by its name, and may have an optional description.

The *Package* concept is modeled by the inheritance rule. We have the abstract class named *Package*. It is superordinated to the classes: *DBServerPackage*, *ApplicationServerPackage* and *ClientPackage*. For each instance of the *Package* class, there may be zero or more references to the instances of the *Function* class.

The *Event* concept is used to represent any software event that may trigger some action under a specified condition. Each event is identified by its name, and may have an optional description. Similar to the packages, by the location of the deployment in a multi-layer architecture, we also classify events as: a) Database server events, b) Application server events and c) Client events. The *Event* concept is modeled in the similar way like *Package*, by applying the inheritance rule.

### *E. Application System*

The *Application System* concept is used to model organizational parts of each Project. Each application system has its name and a description as mandatory properties. Besides, it may reference other, subordinated application systems and we call them child application systems. By this, a designer may create a hierarchy of application systems in a project. Application system hierarchy is modeled by a recursive reference.

Various kinds of IIS*Case repository objects may be created at the level of an application system, but in this paper we focus on two of them only, as PIM concepts: a) *Form type* and b) *Business Application*.

### *F. Form type*

Form type is the main concept in IIS*Case. The meta-model of this concept is presented in Figure 4. It abstracts document types, screen forms, or reports that end users of an information system may use in a daily job. By means of the *Form type* concept, a designer indirectly specifies at the level of PIMs a model of a database schema with attributes and constraints included, as well as a model of transaction programs and applications of an information system.

Apart from creating form types in an application system, a designer may include into the application system form types created in other application systems. Therefore, we classify form types as: a) owned and b) referenced. A form type is owned if it is created in an application system. It may be modified later on through the same application system without any restrictions. A referenced form type is created in another application system and then included into the application system being considered. All the referenced form types in an application system are read-only.

Each form type has a name that identifies it in the scope of a project, a title, frequency of usage, response time and 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: a) menus and b) programs.

Menu form types are used to model menus without data items. Program form types model transaction programs 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 programs is rather complex. A program form type may be designated as *considered in database schema design* or *not considered in database schema design*. Form types considered in database schema design are used later as the input into the database schema generation process. Form types not considered in database schema design are not used in the database schema generation process. They may represent reports for data retrievals only.

Each program form type is a tree of component types. A component type has a name, title, number of occurrences, allowed operations and a reference to the parent component type, if it is not a root component type. Name is the component type identifier. All the subordinated component types of the same parent must have different names.

Each instance of the superordinated component type in a

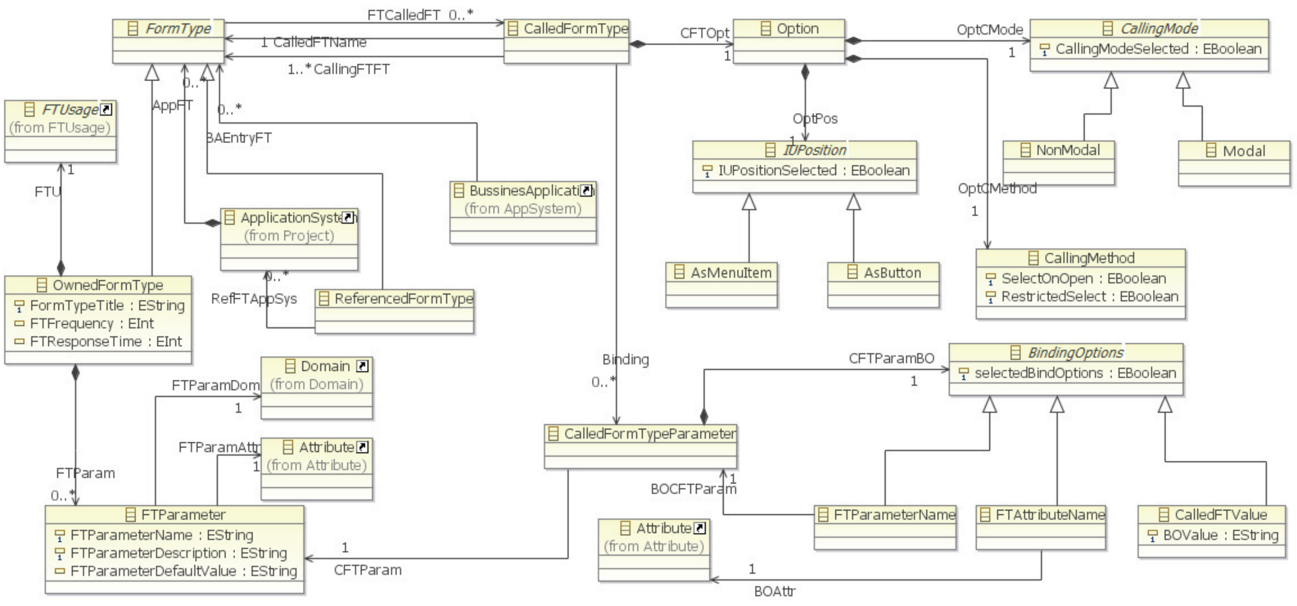

Fig. 4. A meta-model of the IIS*Case Form Type concept

tree may have more than one related instance of the corresponding subordinated component type. The number of occurrences constrains the allowed minimal number of instances of a subordinated component type related to the same instance of a superordinated component type in the tree. It may have one of two values: 0-N or 1-N. The 0-N value means that an instance of a superordinated component type may exist while not having any related instance of the corresponding subordinated component type. The 1-N value means that each instance of a superordinated component type must have at least one related instance of the subordinated component type.

The allowed operations of a component type denote database operations that can be performed on instances of the component type. They are selected from the set {*query*, *insert*, *update*, *delete*}.

A designer can also define component type display properties that are used by the program generator. The concept of component type display is defined by properties: window layout, data layout, relative order, layout relative position, window relative position, search functionality, massive delete functionality and retain last inserted record.

Window layout has two possible values: "New window" and "Same window" and specifies if the component type is to be placed in a new window or in the same window as the parent component type. Data layout specifies the way of component type representation in a screen form. Two values are possible: "Field layout" or "Table layout". By the "Field layout", only one record at a time is displayed in a form. By the "Table layout", a set of records at a time is displayed in a screen form, in a form of a table. The relative order is a sequence number representing the order of a component type relative to the other sibling component types of the same parent in a form type tree. The layout relative position represents the component type relative position to the parent component type. We may select "Bottom to parent" value if we want to place the component type below the layout of the parent component type in a generated screen form, or "Right to parent" value if it is to be placed right to the parent one. Window relative position is to be specified only when "New window" layout is selected. A designer may specify one of the three possible values: "Center", "Left on top", or "Custom". The "Center" value denotes that the center of a new window is positioned to match the center of the parent window. "Left on top" specifies that the top left corner of the new window will match the top left corner of the parent window. By selecting the "Custom" value, a relative position of the new window top left corner to the top left corner of the parent window is explicitly specified by giving X and Y relative positions.

"Search functionality" represents the Boolean property that enables generation of the filter for data selection. If search functionality is enabled, end-users are allowed to refine the WHERE clause of a SQL SELECT statement. If checked, "massive delete functionality" provides generating a delete option next to each record in a table layout. The "retain last inserted record" property specifies if the last inserted record is to be retained in the screen for future use.

Each component type includes one or more attributes. A component type attribute is a reference to a project attribute from the Fundamentals category. It has a title that will appear in the generated screen form. Also, it may be declared as mandatory or optional on the screen form. The allowed operations of a component type attribute denote database operations that can be performed on the attribute, by means of the corresponding screen item. They are selected from the set {*query*, *insert*, *update*, *nullify*}. For a component type attribute a designer may also specify display properties and by this define its presentation details in the screen form. The display properties are specified in the same way as it is for attribute specifications. Values of the display properties may be inherited from the attribute specification or overridden.

So as to unify the layout formatting rules of selected component type attributes, a designer may group them into items groups. Each item group may include one or more component type attributes or other item groups from the same component type. Any item group has its name, title, context and overflow properties. The name and title are mandatory properties. Context and overflow are Boolean properties, specifying if an item group is to be used for presenting layout contextual information or as a layout overflow area.

Each component type attribute provides defining a "List of values" (LOV) functionality. To do that, a designer needs to reference a form type that will serve as a LOV form type. He or she should also define how an end user can edit attributes: "Only via LOV" or "Directly & via LOV". "Only via LOV" property means that attribute value may not be inserted or edited using a keyboard, but only using the LOV. "Directly & via LOV" means that inserting or editing attribute values is provided both via keyboard and LOV. "Filter value by LOV" property specifies if all values from LOV will be displayed, or only those filtered according to the pattern given by an end user. Restrict expression represents the where clause that is concatenated to the rest of where clause in the SQL statement supporting the LOV.

Each component type has one or more keys. Each component type key comprises one or more component type attributes. It represents the unique identification of a component type instance but only in the scope of its superordinated component instance. Uniqueness constraints may be defined for each component type also. Each component type uniqueness constraint comprises at least one component type attribute, but may have more than one. If uniqueness constraint attributes have non-null values, it is possible to uniquely identify a component type instance but only in the scope of the superordinated component instance.

In Figure 5, we illustrate the usage of the Form Type concept. We have the form type *Student_Grades,* that has two component types *Students* and *Grades*. *Student_Grades*

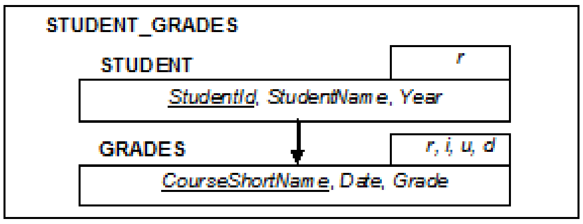

Fig. 5. A form type Student_Grades

form type refers to the information about student grades. *Student* component type represents instances of students, while *Grades* represents instances of grades for each student. *Student* component type is the parent to the *Grades* component type.

Allowed database operation for *Student* is read, while the allowed operations for *Grades* are read, insert, update and delete. The end user of the generated transaction program specified through the form type *Student_Grades*, is able to read data from the set of student instances. He or she can read, update and delete existing grades for each of the students, but can also insert new instances of the grades. *Student* component type should be positioned in *New Window* layout and *centered* to its parent window. Data layout of *Student* component type is in the form of *field layout* style. *Multiple deletions* and *retaining last inserted record* for student records in the screen form are not allowed, while *search functionality* for student records is enabled. Search functionality property allows generation of the filter for the selection of student instances in the generated screen form, so that the end users are able to refine the SQL SELECT statement. *Student* component type owns three attributes: StudentId, StudentName and Year. StudentId is the key of the *Student* component type. For each of *Student* component type attributes, we may specify values of the properties, previously presented.

In the similar way we can give the specification of the *Grades* component type with attributes CourseShortName, Date and Grade. CourseShortName is the key of the Grades component type.

### *G. Business Application*

*Business Application* concept represents the way to formally describe an IS functionality and is organized through a structure of form types. Each business application has a name and a description. One of the form types included into the 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. Each business application must have the entry form type. To create the form type structure of an application, a concept of the form type call is used. By the form type calls, designers model execution of calls between generated transaction programs. They are also used to model parameters and passing the values between two transaction programs during the call executions. The concept of a form type call comprises two form types: a calling form type and a called form type.

Any form type may have formal parameters defined. Each formal parameter has a mandatory name as the identifier. It must be related to exactly one domain. In the specification of a form type call, it is possible to associate each parameter to a called form type attribute. By this, a designer specifies to which attributes real parameter values will be passed during the call execution.

For a called form type in a call we need to specify Binding and Options properties. Binding property comprises formal parameters of a called form type. For each parameter a designer specifies how a real argument value is to be passed to the parameter. There are three possible options: “value”, “attribute reference”, or “parameter reference”. The value is a constant that will be passed during a call execution. The “attribute reference” provides a relation to a calling form type attribute that gives a value to be passed to the parameter during a call execution. The “parameter reference” provides a relation to a calling form type parameter that gives a value to be passed to the parameter during a call execution.

The Options properties comprise: calling method, calling mode, and UI position. Calling method comprises two Boolean properties: a) “Select on open” and b) ”Restricted select”. “Select on open” means that the called form type is opened with an automatic data selection. “Restricted select” allows the data selection in the called form type restricted just to the values of passed parameters. Calling mode specifies a general behavior of the calling form type during the call execution. Three possibilities are allowed: “Modal”, “Non-modal” or “Close calling form”. “Modal” means that a user cannot activate the calling form type while the called form type is opened. “Non-modal” means that both the calling and the called form type are simultaneously active in the screen. “Close calling form” is used to cause the closing of the calling form type during the call execution. UI position specifies how a call will be provided at the level of UI: as a menu item or as a button item.

## IV.CONCLUSION

In this paper we presented the IIS*Case PIM meta-model, based on MOF 2.0 specification. Our intention was not to present all the elements of our meta-model in detail. Instead, we tried to focus just on those meta-model details that are necessary to give a general picture of the model. 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 textual DSL to support IS design and give another view of the IS description.

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 has already been implemented in IIS* Case, we want to raise the process of collision resolving at the PIM level of abstraction.

Our further research will include experiments with other technologies that rely on MOF. The presented meta-model is a good base for a research in the area of Query View Transform (QVT) set of languages. Our intention is to embed into IIS*Case transformations between different data models. Providing data model transformations may play an important role in the IS design process. In the course of data reengineering process, our plan is to provide the data integration from various sources based on different data models. Data transformation rules specified by QVT could be applied at the level of meta-models specified by various data-models, all expressed in a unified manner in MOF. Our intention is to provide transformations of the models specified in IIS* Case to the UML models. Providing such transformations we allow designers to have models specified in UML standard with OCL constraints.

## Acknowledgment

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] I. Lukovic, M. J. Varanda Pereira, N. Oliveira, D. Cruz, P. R. Henriques, "A DSL for PIM Specifications: Design and Attribute Grammar based Implementation", Computer Science and Information Systems (ComSIS), ISSN: 1820-0214, DOI: 10.2298/CSIS101229018L, Vol. 8, No. 2, 2011, pp. 379-403.

[2] L. Baresi, F. Garzotto, M. Maritati, "W2000 as a MOF Metamodel." In Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics - Web Engineering track. Orlando, USA, 2002.

[3] A. Schauerhuber, M. Wimmer, E. Kapsammer, "Bridging existing web modeling languages to model-driven engineering: A metamodel for webML", International Workshop on Model Driven Web Engineering (2nd), Palo Alto, CA, 2006.

[4] M. Richters, M. Gogolla, "A meta-model for OCL" In Proc. of the 2nd international conference on The unified modeling language beyond the standard, ISBN:3-540-66712-1, 1999.

[5] F. Jouault, J. Bézivin, "KM3: a DSL for Metamodel Specification", In Proc. of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy, 2006, Springer LNCS 4037, pp. 171-185.

[6] MetaCase Metaedit+, [Online] Available: http://www.metacase.com/.

[7] Meta-Object Facilty, [Online] Available: http://www.omg.org/mof/.

[8] GME: Generic Modeling Environment, [Online] Available: http://www.isis.vanderbilt.edu/Projects/gme/.

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

[10] I. Luković, P. Mogin, J. Pavićević, S. Ristić, "An Approach to Developing Complex Database Schemas Using Form Types", Software: Practice and Experience, 2007, DOI: 10.1002/spe.820, Vol. 37, No. 15, pp. 1621-1656.

[11] I. Luković, S. Ristić, P. Mogin, J. Pavićević, "Database Schema Integration Process – A Methodology and Aspects of Its Applying", Novi Sad Journal of Mathematics, Serbia, ISSN: 1450-5444, Vol. 36, No. 1, 2006, pp. 115-150.

[12] J. Banović, "An Approach to Generating Executable Software Specifications of an Information System", Ph.D. Thesis, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, 2010.

[13] A. Popović, "A Specification of Visual Attributes and Business Application Structures in the IIS*Case Tool", Mr (M.Sc.) Thesis, University of Novi Sad, Faculty of Technical Sciences, 2008.

[14] Object Management Group (OMG), OCL SpecificationVersion 2.0, [Online] Available: http://www.omg.org/docs/ptc/05-06-06.pdf, June 2005.

[15] Document Type definition (DTD), [Online] Available: http://www.w3.org/TR/html4/sgml/dtd.html.

[16] B. Perisic, G. Milosavljeivc, I. Dejanovic, B. Milosavljevic, "UML Profile for Specifying User Interfaces of Business Applications", Computer Science and Information Systems (ComSIS), ISSN: 1820-0214, DOI: 10.2298/CSIS110112010P, Vol. 8, No. 2, 2011, pp. 405-426.

[17] N. Oliveira, M. J. Varanda Pereira, P. R. Henriques, D. Cruz, B. Cramer, "VisualLISA: A Visual Environment to Develop Attribute Grammars", Computer Science an Information Systems, (ComSIS), ISSN:1820-0214, Vol. 7, No. 2, 2010, pp. 265-289.