A Comparison of Ecore and GOPPRR through an Information System Meta Modeling Approach

Vladimir Dimitrieski, Milan Čeliković, Vladimir Ivančević and Ivan Luković

University of Novi Sad, Faculty of Technical Sciences
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia
{dimmy, milancel, dragoman, ivan}@uns.ac.rs

Abstract. In this paper we present a comparison between the main concepts of the Ecore meta-meta-model and the GOPPRR meta-language. Through our previous research we have specified the PIM concepts of our model driven software development tool for information system design IIS*Case using Ecore implementation of Meta object Facility 2.0 in Eclipse Modeling Framework (EMF). We have also modeled the same concepts by Graph-Object-Property-Port-Role-Relationship (GOPPRR) meta-modeling language provided by the MetaEdit+ meta-modeling environment. Both MetaEdit+ and EMF provide the environment for the meta-models specification. In this paper we give a brief overview of MetaEdit+'s and EMF's main concepts and syntax, as well as an example of IIS*Case PIM concepts modeling using Ecore and GOPPRR meta-meta-models. We also present the main differences between two meta-modeling environments, EMF and MetaEdit+, from the PIM concept modeling point of view.

Keywords: Model Driven Approaches, Domain Specific Languages, Domain Specific Modeling, EMF, MetaEdit+, Information System Modeling.

1 Introduction

Domain-specific languages (DSLs) are special-purpose languages designed to solve a particular range of problems. DSLs are tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use in their domain of application, compared with general-purpose programming languages.

Nowadays, DSLs are of increasing importance for the development of software and other systems. In specifications of DSLs, visual notations are often used. They are to be supported by a tool environment consisting of visual editors, simulators and model transformers. Existing approaches for generating or adjusting the desired tool environments rely on meta-modeling concepts, grammars, or some kind of logics. Dependent on the underlying concepts, different kinds of editors are generated.

Several Eclipse projects are heading towards meta-technology to define DSLs. Eclipse Modeling Framework (EMF) [1] mainly generates the underlying models of visual and textual editors that may be extended by additional syntax checks, implementing certain rules, using Object Constraint Language [2]. A graphical view of

visual editors can be hand coded on the basis of Eclipse Graphical Editor Framework (GEF) [3] or generated using Graphical Modeling Framework (GMF) project [4].

MetaEdit+ [5] is the metaCase tool for development of DSLs. MetaEdit+ is an integrated, repository-based tool aimed at creating and using modeling languages and code generators. It provides a tool support for different modeling languages by configuring the generic tool set with meta-models.

DSLs may take a distinguished role in the modern information system (IS) development process. In IS development DSLs may be used for various purposes, such as: conceptual modeling, specification of rules and constraints, code generation, generation of test cases, specification of transformations between models etc. A detailed overview of DSL usage in the context of Model Driven Software Development (MDSD) and IS development may be found in [6]. Through our 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.

In order to provide design of various platform independent models (PIM) by IIS*Case, we have created a number of modeling, meta-level concepts and formal rules that are used in the design process. Our experience from previous research [6, 8, 10], leads to the conclusion that there was 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 three related approaches to formally describe IIS*Case PIM Concepts. The first one is based on the attribute grammars through which we are developing the textual DSL, named IIS*CDesLang. In [8], we present IIS*CDesLang. It formalizes IIS*Case PIM concepts and provides modeling in a formal way. IIS*CDesLang meta-model is developed under a visual programming environment for attribute grammar specifications named VisualLISA [9].

The second approach is based on Meta Object Facility (MOF) [7]. 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). This approach is presented in [10]. As we could not find standardized implementation of MOF, we selected Ecore meta-meta-model to implement PIM model, since MOF 2.0 is widely used meta-modeling framework. Ecore is the Eclipse implementation of MOF 2.0 in Java programming language which is provided by EMF. We deploy it to implement a meta-model as a basis for textual and graphical DSL we plan to build in the future.

In the last approach, we deploy MetaEdit+'s GOPPRR [5, 14, 15] as a meta-modeling framework to describe our PIM concepts. 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. Through our previous research, we have gained a valuable experience in the practical application of Domain Specific Modeling (DSM) for creating our PIM meta-model using different environments and paradigms.

In this paper we present a comparison of EMF and MetaEdit+'s meta-modeling environments and their respective frameworks through modeling of the same PIM concepts. This comparison comprises our previous practical experiences. It is based on the evaluation of environments' meta-language concepts through the comparison of their ease of use. Our goal is to identify the advantages and disadvantages of each environment as both of them are used in implementation of meta-models as a basis for further textual and graphical DSL development. Although we have modeled all IIS*Case PIM concepts using both environments, in this paper we chose to present most representative parts of our PIM concepts, on which we are able to see the true difference between EMF and MetaEdit+.

Apart from Introduction and Conclusion, the paper is organized in three sections. In Section 2 we present related works. In Section 3 we present a comparison of EMF and MetaEdit+'s basic concepts, while in Section 4 we give a presentation of IIS*Case's *FormType* and *FormTypeUsage* PIM concepts specified through the meta-models implemented in MetaEdit+ and EMF.

2 Related Work

Nowadays, meta-modeling is widely spread area of research and there is a huge number of references covering this area. However we could not find a lot of papers, relevant to the comparison of DSM tools. We have found only three of them, presenting EMF and MetaEdit+ concepts and providing their comparison.

In [11] , the author presents the comparison between GEF workbench environment and MetaEdit+. The author also proposes a DSL named Logic Gate Language, for the description of logic circuits models. The language was developed both under MetaEdit+ and GEF. The author reported that, in general, the process of the implementation was much faster in MetaEdit+ than in GEF. GEF also required much more Java language coding, than MetaEdit+.

In [12], the author presents a mapping between MetaEdit+ and EMF meta-meta-level concepts. He proposes the M3-Level-Based Bridges solution that provides interoperability between MetaEdit+ and EMF, i.e. an interface for the exchange of meta-models and models between the two tools. Transformations between models at the M2-level and M1-level have been implemented as the Eclipse plug-in. In this way, the bridge may be used for the model re-usage.

In [13] the authors analyzed a set of meta-modeling languages including Ecore and GOPPRR. To compare the selected meta-meta-models, they defined criteria for their comparison and proposed a comparison framework consisting of abstractions of meta-modeling concepts available in each meta-modeling language. As the last step, the authors evaluated obtained results according to the three aspects: availability of the

meta-modeling concepts, definition of relationships and the concepts of structuring, reuse and modularization in meta-modeling.

In [11], development of a graphical DSL is considered. However, we base our comparison on the modeling of PIM concepts that can be later used for development of either textual or graphical DSLs. In [12] the focus is set to mappings between concepts of two meta-languages, only. In [13], the authors compared meta-modeling languages with respect to diversity of the meta-modeling concepts. However, we focus not only to the similarities and differences of the concepts, but also considerations regarding their practical usage.

3 The Main Concepts of EMF and MetaEdit+

Here we give a brief overview of main Ecore and GOPPRR concepts used in the specification of IIS*Case PIM concepts. We make a comparison between the concepts existing in both meta-meta languages that are used in conceptual specification of meta-models. Although the Ecore provides only conceptual structures, GOPPRR concepts also include information about graphical representation of elements. Concepts used in our PIM specification that are specific to one of the meta-meta languages only, are also described. A presentation of basic concepts of both meta-languages is based on a comparison of the corresponding meta-modeling concept classes.

A *grouping concept* allows to structure meta-model elements in defined parts or modules. Regarding their grouping characteristics, Ecore and GOPPRR contain similar concepts. *EPackage* is the Ecore concept used for the model organization. It groups the instances of all Ecore concepts into one logical unit. EPackage's name need not be generally unique. Instead, a URI is used to uniquely identify the package. GOPPRR's concept for grouping elements is the *Graph* concept. Graph is a collection of objects, relationships, roles and bindings. Graph contains all elements and their explosions to other graphs. Explosion allows each object, relationship or role in a graph to be linked to other graphs. Additionally, Graph can contain properties that describe it further.

A *class concept* defines a class of objects with the same characteristics. A class is the blueprint from which the individual objects are created. *EClass* is an Ecore concept used to define set of model entities. The corresponding concept used in GOPPRR is the *Object*.

A *relationship concept* describes a connection between elements of the model and is a subset of the Cartesian product over the participating object types. *EReference* is an Ecore concept that defines a set of relations between objects. It establishes the link from one EClass instance to another. As the EReference instance links at most two objects, it represents the binary relationship. GOPPRR owns a similar concept named *Relationship*. An instance of Relationship differs from EReference instance, as it may have own properties describing the relationship. It can also link more than two Object instances, of the same or different Object concept. GOPPRR Relationship instances attach to objects via roles and they can define properties for the objects' connections. They are used to form bindings with objects and roles. *Role* concept exists only in GOPPRR and no direct equivalent concept exists in Ecore. This concept specifies the

lines and end-points of relationships and describes how an object participates in a relationship. *Object Set* and *Bindings* are also the concepts existing in GOPPRR only. An object set describes a collection of objects with the same role in a binding. Bindings contain the information about how the objects, ports, roles and relationships in a Graph are connected. *Inheritance* is a special kind of a relationship that allows creating subtypes of other language concepts. Both Ecore and GOPPRR provide this concept. In Ecore only the class concept can be inherited, whereas in GOPPRR all metatypes can be inherited.

An *attribute concept* is a property of a meta-model element. At a model level an attribute can hold concrete values. *EAttribute* is the Ecore concept used to define the characteristics of the EClass instances. *EDataType* is another Ecore concept used for the specification of the EAttribute instances type. The similar concept to EAttribute in GOPPRR is the *Property* concept. Both of them have the same behavior in the usage for primitive attribute data types. Additionally, Property concept may represent the link to another object, specifying it as an object member. In Ecore this is accomplished using the EReference concept.

4 IIS*Case Meta-model

Through our previous research we have formally described IIS*Case PIM models using Ecore and GOPPRR meta-modeling languages and attribute grammars. As we used Ecore and GOPPPR to describe same PIM concepts we found some similarities and some differences in these approaches. In this chapter we describe those findings through detailed description of *FormType* and *FormTypeUsage* PIM concepts. We also give a brief overview of other main PIM concepts: *Project*, *ApplicationSystem*, *ApplicationType*, *BusinessApplication* and *Fundamentals*. Modeling of the IIS*Case PIM concepts is organized through the package concept in EMF. In order to provide grouping of elements inside a graph in MetaEdit+, we added a ***GraphGroup*** concept to the graphical meta-model of MetaEdit+. It is represented in form of dotted rectangle surrounding concepts that need to be grouped as one logical unit. *GraphGroup* element contains a name of the group shown in the top left corner of the graphical representation.

4.1 A Brief Overview of Main PIM 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 Figures 1 and 2 is the concept of a *Project*. In MetaEdit+ model we restricted the number of *Project* instances to one instance per graph. As one project is one IS specification, a designer may have only one instance of *Project* in a single MetaEdit+'s graph. Also, a designer may not create two projects with the same name as the project's name must have globally unique value.

As it is shown in Figures 1 and 2, *ApplicationSystems* and *Fundamentals* are subunits of a *Project*. Every instance of a *Project* may be connected to zero or more instances of the *ApplicationSystem* and zero or more instances of any descendant of

Fundamentals. *ApplicationSystems* are organizational parts, i.e. segments of a project. Designers of an IS may create application systems of various types. By the *ApplicationType* concept, designers introduce various application system types and then associate each application system instance with one application type.

Fig. 1. A Meta-Model of the Main IIS*Case PIM Concepts in MetaEdit+

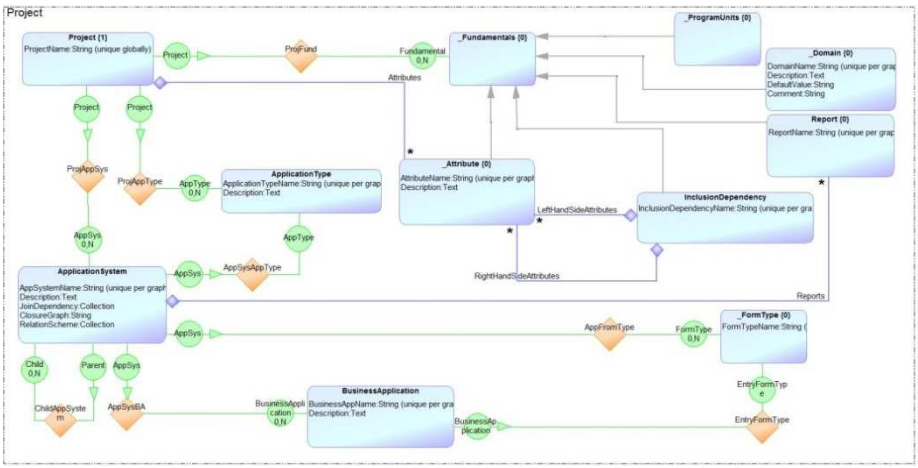
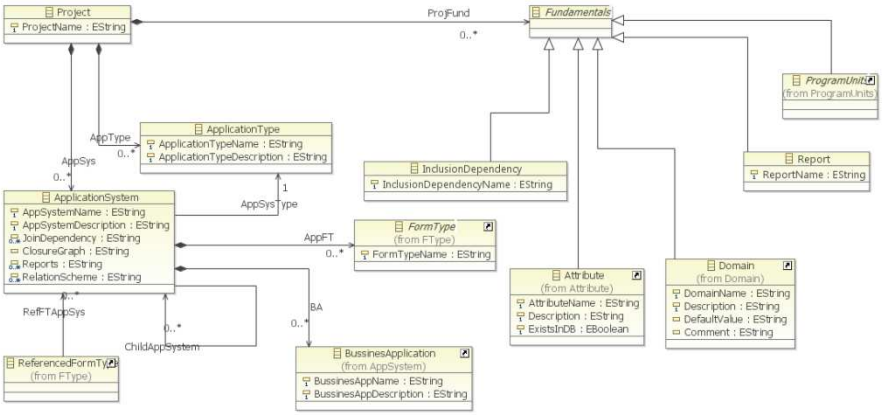


Fig.2. A Meta-Model of the Main IIS*Case PIM Concepts in EMF



Fundamentals (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 latter on. Fundamentals comprise zero or more: *Attributes*, *Domains*, *ProgramUnits*, *Reports* and *InclusionDependencies*.

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

4.2 *FormType*

Form type is the main concept in IIS*Case. The meta-models of this concept are presented in Figures 3 and 4. It abstracts document types, screen forms, or reports that end users of an IS 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 IS.

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. All these properties are mandatory. In MetaEdit+ there is no built-in mechanism to declare mandatory properties. Instead, a user is to specify a regular expression option for every property. In EMF this kind of constraint is easier to specify by setting the lower and upper bounds of cardinality. In MetaEdit+, the regular expression option provides a more powerful mechanism for specifying properties' value characteristics.

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.

Apart from creating form types in an application system, a designer may include 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. In EMF we have modeled the Form Type concept by the Inheritance rule. We have the abstract class named *FormType*. It is superordinated to the classes: *OwnedFormType* and *ReferencedFormType*.

A main advantage of GOPPRR's relationship properties may be used in the modeling of referenced form type concept. Through GOPPRR's concepts we have modeled referenced form type as the relationship named *CallFT*. This relationship is illustrated in Figure 3. The relationship has a property named "Options", which is an instance of *Options* object. Similarly, property named "CalledFTParameters" is a collection of *CalledFormTypeParameter* object instances.

Fig.3. A Meta-Model of *FormType* concept in MetaEdit+

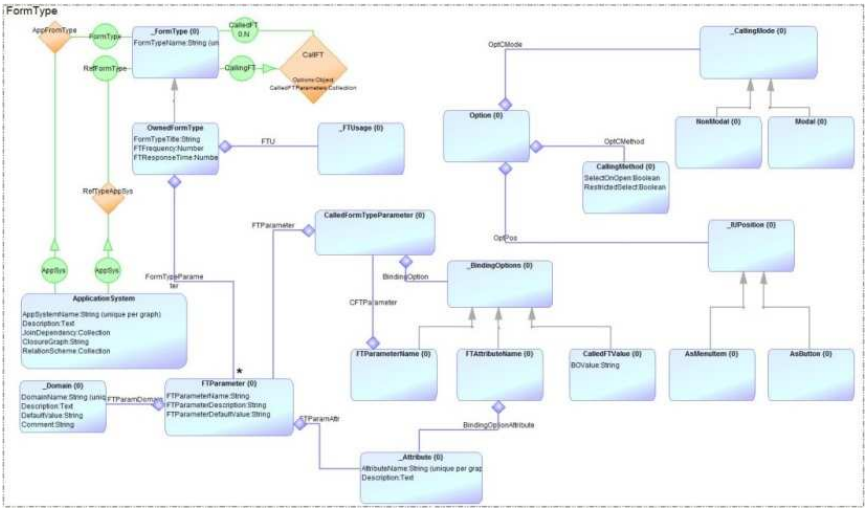
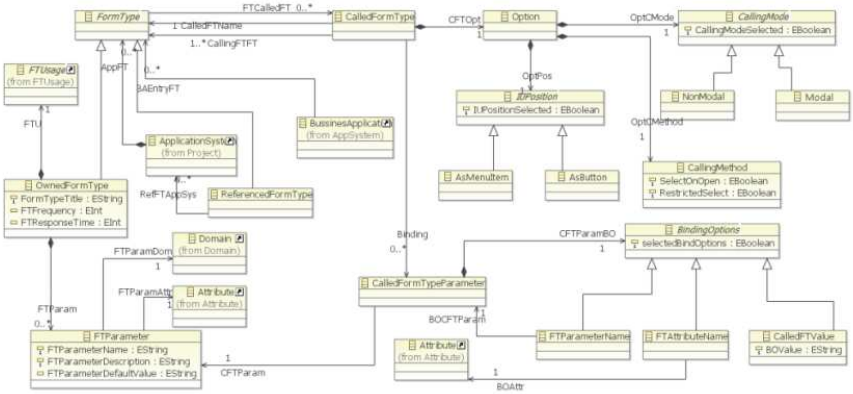


Fig.4. A Meta-Model of *FormType* concept in EMF



Calling referenced form requires some options and form parameters to be set. *Options* contain *CallingMode*, *CallingMethod* and *UIPosition* of the element calling the form. Every form can be called in a modal or non-modal mode. Modal called form must be closed before a user can continue to work in a calling form. Non-modal form specifies that a calling and the called form may exist simultaneously opened on the screen. *CallingMethod* specifies two parameters “Select on open” and “Restricted select”. Select on open specifies if a called program, generated from the called form type, will be opened with an automatic data selection, during the call execution, or not. Restricted select parameter specifies if a called program, generated from the called form type, will be opened in a way to allow only selection of data restricted to the values of passed parameters. *UIPosition* specifies if a called form is shown as a menu item or button in the calling form.

CalledFormTypeParameter in MetaEdit+'s meta-model or *Binding* in EMF's meta-model, is a definition of parameter that is being passed in a form type call. For each form type call parameter we may select this parameter for binding and, if it is selected, to define how a real argument value will be passed to that parameter.

4.3 *FormTypeUsage*

All *FormTypeUsage* concepts are presented in Figures 5 and 6. 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 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. 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.

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". Each component type has one or more keys and uniqueness constraints. Both elements comprise one or more component type attributes. Component type keys and unique constraints with non-null values represent the unique identification of a component type instance but only in the scope of its superordinated component instance.

Due to limited space we omit descriptions of many other properties concerning FormTypeUsage concept. Their detailed descriptions may be found in [10].

Through the specification of *FormTypeUsage* concepts we have modeled the same concepts in EMF and MetaEdit+. Those concepts are modeled in the same way and using similar constructs in both environments. Our goal in this subsection was to show that despite all the differences between environments, both of them can be used equally to model the same concepts.

Fig. 5. A Meta-Model of *FormTypeUsage* concept in MetaEdit+

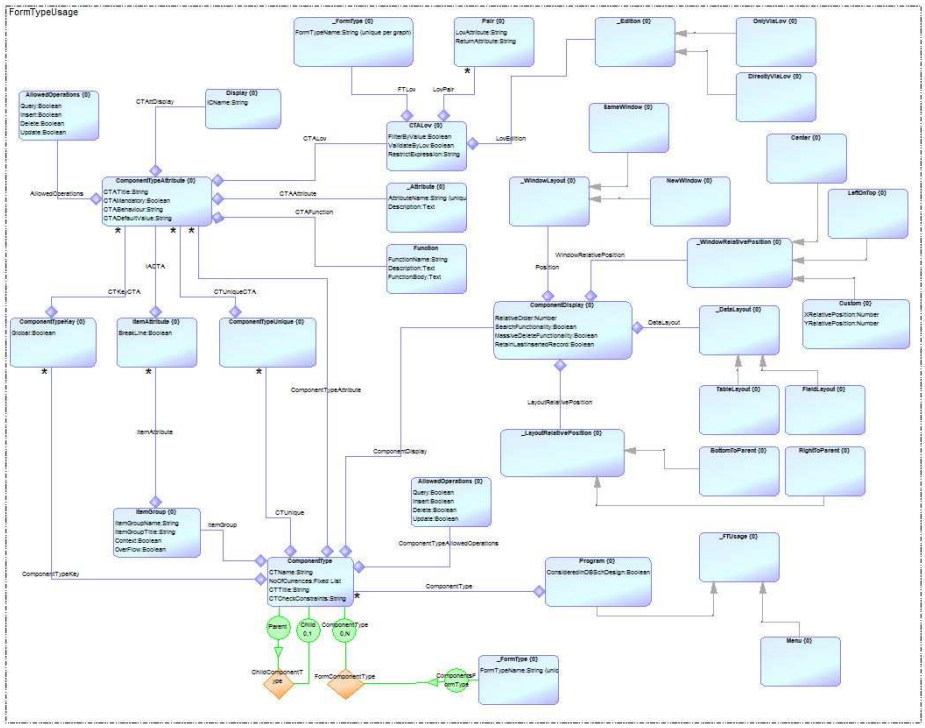
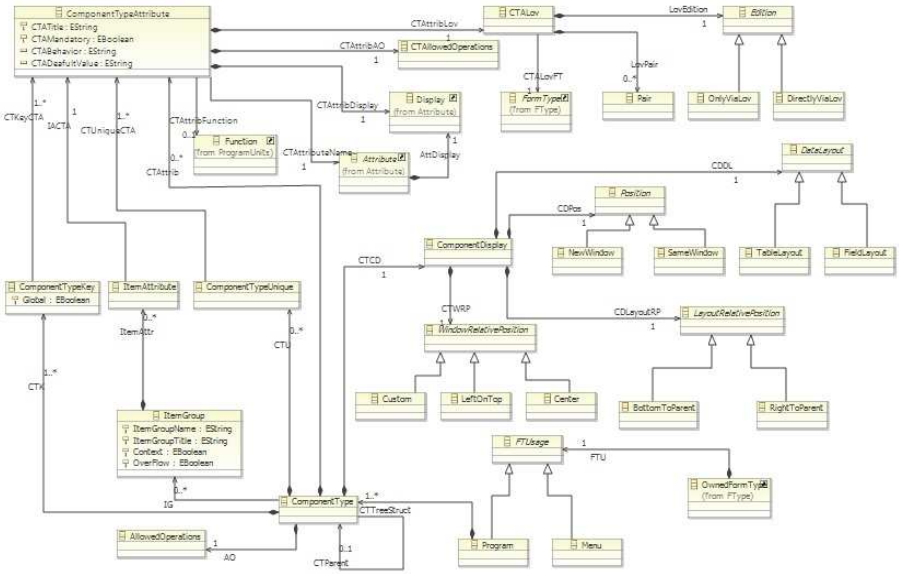


Fig.6. A Meta-Model of *FormTypeUsage* concept in EMF



5 Conclusion

In this paper we presented a comparison between two DSM tools: EMF and MetaEdit+. For this purpose, we explored the MetaEdit+ language definition concepts and Ecore meta-meta-model. The concepts used by MetaEdit+ are described by GOPPRR meta-language that actually represents the meta-meta-model language definition. Ecore is MOF 2.0 implementation used by EMF meta-modeling environment.

Unlike EMF modeling environments that we have worked in, MetaEdit+ allows us to easily generate meta-objects in the repository from our meta-model. Further we can use MetaEdit+'s environment to produce graphical DSL by importing the meta-model. Unlike MetaEdit+'s environment, EMF modeling environment provides only the abstract syntax development. The concrete syntax of some DSLs may be developed under some other Eclipse frameworks, such as Xtext, EMFText or GMF Tooling. These frameworks rely on EMF, and they also use Ecore meta-meta-model. EMF modeling environment is widely used. Using the Ecore meta-meta-model, users have the opportunity to model using the MOF 2.0 concepts as a de facto standard. The meta-model specified under the EMF environment can be further used in the development of some textual or graphical DSLs, using some other Eclipse tools.

Both MetaEdit+'s and EMF's workbenches may be deployed 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 a graphical way. By specifying IS models, designers have better opportunities for mental testing the ideas and checking validity of their models.

We conclude that MetaEdit+ is an environment well suited with concepts to develop fully functional graphical DSLs. It provides all-in-one environment for defining meta-models as well as the representations of graphical DSLs. On the other hand side, EMF provides developing meta-models only. MetaEdit+ is extensible at the level of graphical meta-language, with new concepts that can make modeling more precise and easier. However, the resulting set of concepts still needs to be mapped onto the actual GOPPRR concepts in MetaEdit+. Unlike the MetaEdit+ tool, EMF does not allow introducing new concepts at the meta-meta-level. EMF is the modeling environment, and majority of its concepts are not used when developing a modeling language. Other tools, like GMF, are also needed when specifying the modeling language. On the other hand all GOPPRR is built for the definition of graphical modeling languages and all of its meta-concepts are well suited for fulfilling its purpose. Ecore concepts are supported by many other tools and environments through standard import and export mechanisms, such as XML Metadata Interchange (XMI). This standardization makes a usage of an environment easier for users already familiar with MOF 2.0 concepts.

Our further research will be directed towards the implementation of mapping between the meta-models using GOPPRR and Ecore specification. One goal is to provide a bridge that will support the transformation from the model specified by one meta-model to the other. One of the goals is to deploy MetaEdit+-EMF-Bridge [12] to import IIS*Case GOPPRR meta-model into EMF and then use a transformation engine like Epsilon or XPand, to define a model-to-model transformation. It should provide the users with the ability to deploy both modeling environments utilizing their advantages. This mapping will also allow exchanging models between coworkers

using different environments and thus make them easier work on the same problem with tools they already have.

6 Acknowledgements

The research presented in this paper was supported by Ministry of Education and Science of Republic of Serbia, Grant III-44010.

7 References

1. Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/.
2. Object Management Group (OMG), OCL Specification Version 2.0. http://www.omg.org/docs/ptc/05-06-06.pdf, 2005.
3. Graphical Editing Framework. http://www.eclipse.org/gef/.
4. Graphical Modeling Framework. http://www.eclipse.org/modeling/gmp/.
5. Kelly, S., Lyytinen, K., Rossi, M., MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. CAiSE 1996: pp. 1-21
6. Luković, I., Ivančević, V., Čeliković, M., 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, 2012. (Accepted for publication, Chapter 17)
7. Meta-Object Facility. http://www.omg.org/mof/
8. Luković, I., Varanda Pereira, M. J., Oliveira, N., Cruz, D., Henriques, P. R.: A DSL for PIM Specifications: Design and Attribute Grammar based Implementation. ComSIS, ISSN: 1820-0214, DOI: 10.2298/CSIS101229018L, Vol. 8, No. 2, 2011, pp. 379-403.
9. Oliveira, N., Varanda Pereira, M. J., Henriques, P. R., Cruz, D., Cramer, B.: VisualLISA: A Visual Environment to Develop Attribute Grammars. ComSIS, ISSN:1820-0214, Vol. 7, No. 2, 2010, pp. 265-289.
10. Čeliković, M., Luković, I., Aleksić, S., Ivančević, V.: A MOF based Meta-Model of IIS*Case PIM Concepts. Proceedings, IEEE Computer Society Press & Polish Information Processing Society, ISBN: 978-83-60810-22-4, Szczecin, Poland, 2011, pp. 833-840.
11. Kelly, S.: Comparison of Eclipse EMF/GEF and MetaEdit+ for DSM, Proceedings of the OOPSLA & GPCE Workshop on Best Practices for Model Driven Software Development at OOPSLA'04, 2004.
12. Kern, H.: The Interchange of (Meta) Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges, 8th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA, 2008.
13. Kern, H., Hummel, A., Hühne, S.: Towards a Comparative Analysis of Meta-Metamodels, 11th Workshop in Domain-Specific Modeling, 2011.
14. Welke, R.J.: CASE Repositories: More than another DBMS Application, Challenges and Strategies for Research in Systems Development, Cotterman, W. and J. Senn (eds.), J. Wiley, Chichester, UK, 1992, pp. 181-214.
15. GOPPRR: MetaEdit+ Workbench User's Guide, Version 4.5, MetaCase, [Online] http://www.metacase.com/support/45/manuals/mwb/Mw-1_1.html