Implementing the Unifying Meta-model for Enterprise Modeling and Model-Driven Development: An Experience Report

Iyad Zikra

Department of Computer and Systems Sciences, Stockholm University,
Forum 100, SE-164 40 Kista, Sweden
iyad@dsv.su.se

Abstract. Model-Driven Development (MDD) is becoming increasingly popular as a choice for developing information systems. Tools that support the principles of MDD are also growing in number and variety of available functionality. MetaEdit+ is a meta-modeling tool used for developing Domain Specific Languages and is identified as an MDD tool. The Eclipse Modeling Framework (EMF) and Graphical Modeling Project (GMP) are two Eclipse projects that provide plug-ins to support the principles of MDD. In this paper, we report on our experience in using MetaEdit+ and the Eclipse plug-ins for developing a graphical editor for the unifying meta-model, which is an MDD approach that extends the traditional view of MDD to cover Enterprise Modeling. The two modeling environments are reviewed using quality areas that are identified by the research community as necessary in MDD tools. This report will provide useful insights for researchers and practitioners alike concerning the use of MetaEdit+ and the Eclipse plug-ins as MDD tools.

Keywords: Model-Driven Development, Tools, MetaEdit+, Eclipse Modeling Framework, Graphical Modeling Project.

1 Introduction

The increasing reliance on Model-Driven Development (MDD) for creating Information Systems (IS) in recent years has led to a stream of research projects, approaches, and tools that support the use of models as the main development artifacts. Models in MDD are used to capture various aspects of an IS, and (automatic) transformations enable the derivation of models from each other and generate executable code.

Attempts to extend the use of MDD to describe the organization are starting to emerge [2, 6, 9]. Models can be used to capture the underlying motivation for which IS are developed, which provides deeper understanding of the IS models and improves the design decisions that are made. In order to capitalize on the full potential of MDD principles, MDD approaches need to be supported with tools to facilitate the creation and management of models, meta-models, and transformations. Tools also enable the execution of model transformations and, eventually, code generation. Aside from that, tools should offer practical and usable functionalities that simplify the complexity associated with managing models and transformations [4].

This paper presents an account of using two tooling environments that are associated with MDD; namely, MetaEdit+[1] and the modeling environment available in Eclipse[2] through the Eclipse Modeling Framework (EMF)[3] and the Graphical Modeling Project (GMP)[4] plug-ins. The aim of this paper is to relay the experience of using MetaEdit+ and EMF/GMP for creating a tool for the unifying meta-model for Enterprise Modeling (EM) and MDD [14], thus helping researchers and practitioners alike in gaining useful insights into the two modeling environments.

The remainder of the paper is organized as follows. Section 2 gives an overview of the quality areas that MDD tools should have as described in the literature. Section 3 presents the unifying meta-model. Sections 4 and 5 include reflections on how MetaEdit+ and the Eclipse plug-ins fulfill the properties in relation to implementing the unifying meta-model. Finally, section 6 gives concluding remarks.

2 Qualities of Model-Driven Development Tools

The importance of MDD tools that is emphasized in the literature has yet to lead to the creation of a tool that covers all the necessary aspects of MDD. Several authors have addressed the requirements for MDD tools, pointing out the drawback of existing MDD approaches in terms of tooling, and highlighting the lack of tools that can realize the theoretical benefits of MDD. As pointed out in [1], current works that tackle MDD tools are limited to superficial comparisons of feature lists, with no deep insights to select a tool based on realistic project requirements and available technical abilities and standards compliance. As a result, [1] suggests a conceptual framework for MDD tool architectures that covers major existing MDD theoretical architectures. The proposed tool architecture captures a three-dimensional view of modeling levels by identifying three types of modeling abstractions: form instantiation, where instances of one level follow the structure defined by concepts of a higher level; content instantiation, where instances capture content instances of the content types described on a higher level; and generalization, where instances are themselves types that extend the definition of the types of a higher level. By introducing the notions of embedded levels and spanning, it becomes possible to project the three-dimensional architecture as a two-dimensional view. Using this architecture, MDD tool developers can plan the number and types of modeling levels that will be supported by the tool.

Aside from the core MDD principles which tools must support (i.e. the creation and management of models and meta-models, and the creation and execution of transformations), the following quality areas are highlighted in the literature as necessary in MDD tools to enhance the process of creating and managing the models:

- Understandability: this refers to the ease with which tool users are able to create meta-models and models [10]. Understandability can be enhanced by tools through facilities that highlight and explain the intended purpose of parts of the

[1] `http://www.metacase.com/`
[2] `http://www.eclipse.org/`
[3] `http://www.eclipse.org/modeling/emf/`
[4] `http://www.eclipse.org/modeling/gmp/`

models to tool users [4]. The graphical notation, which is generally part of the definition of the modeling language, can be enhanced by the graphical editor in the tool [10]. Furthermore, tools should support the representation of details on several levels of abstraction [4], which can occur along multiple axes [1]. A usability framework, including a conceptual model for capturing tool usability and an experimental process for conducting the usability evaluation, is proposed in [8] to measure the satisfaction, efficiency, and effectiveness of an MDD tool.

- Model evaluation: tools can offer support for model analysis and evaluation [7], Evaluation is referred to as observability or "model-level debugging" [12] when the reporting of warnings and errors is done during the creation of the model and the execution of the transformations—similarly to how compilers do for programming languages.
- Executability: the use of model-to-text transformations to derive (generate) executable program code from models that describe the desired IS [4, 7, 10].
- Model repositories: tools must provide mechanisms for serializing models in order to transport them to other tools or store them for later reuse [4, 7, 10]. Serialization is another form of model-to-text transformations. However, it does not generate executable programming code. Storing models in repositories also raises the question of model integration [4].
- Traceability and change management: MDD involves the use of transformations to advance from one stage of modeling to the next. Changes in earlier models need to be tracked and reflected in later models, and tools must provide the necessary facilities to realize that [7].
- Other Software Engineering (SE) activities: since MDD tools are used in IS development projects, activities related to the development process itself are not isolated from MDD activities and must be supported by the tool. Project planning [7], collaborative development [4], and Quality of Service (QoS) management [7] are some of the activities highlighted in the literature.
- Tool documentation: finally, tools need to provide self-documentation—guidelines for using the tool itself.

The list of quality criteria described above is by no means complete. There could be other desirable qualities for MDD tools. However, we focus on criteria which are relevant for reporting our experience with MetaEdit+ and the Eclipse MDD plug-ins.

3 The Unifying Meta-model

With the increased popularity of MDD, several efforts have attempted to exploit models and transformations to cover activities that precede the development of IS [13]. Such efforts try to enhance the content captured by design models to represent aspects that affect IS development but is not directly related to it, such as the intention and motivation expressed in enterprise models. However, the survey presented in [13] highlights the hitherto existing need for an MDD approach that spans aspects of both Enterprise Modeling (EM) and MDD. The unifying meta-model was proposed in [14] as a response to that need.

The unifying meta-model provides an overall platform for designing enterprise models, which are then used to derive IS models that can subsequently be used to generate a functioning and complete system following MDD principles. Six complimentary views, illustrated in Fig 1, constitute the unifying meta-model and cover aspects that describe enterprise-level information in addition to information relevant for IS development. Organizational business goals are captured by the Goal Model (GM) view, and the internal rules and regulations that govern the enterprise and its operations are provided by the Business Rules Model (BRM) view. The Concepts Model (CM) view covers the concepts and relationships that describe the static aspects of the enterprise and its supporting IS, while business process that describe the activities needed to realize the business goals are part of the Business Process Model (BPM) view. A Requirements Model (RM) view captures the high-level requirements that are associated with the development of the IS and relates them to components of other views. The IS Architecture Model (ISAM) covers the implementation architecture which will be used to realize the IS, describing how the components that implement the business process, business rules, concepts, and requirements will work conjointly together in an operational system.

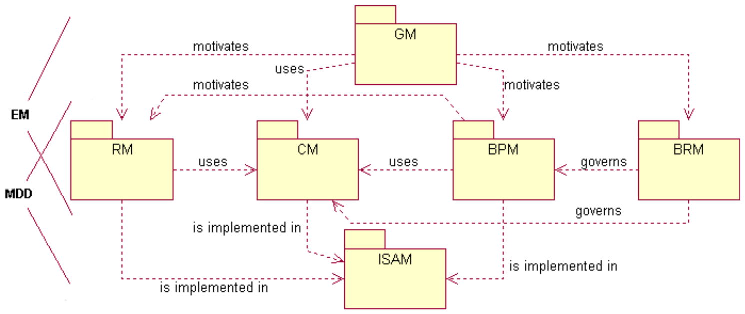


Fig. 1. The Complimentary Views of the Unifying Meta-Model

The GM, BRM, CM, BPM, and RM views represent the EM side of the unifying meta-model. Simultaneously, the RM, CM, BPM, BRM, and ISAM represent the IS and are part of the MDD side of the unifying meta-model. This overlap between the EM and MDD views, supported by inter-model relationships that relate components across the different views and provide built-in traceability support, guarantees the combined overview offered by the unifying meta-model.

A complete description of the complementary views of the unifying meta-model and how they can be utilized to develop enterprise-aligned IS can be found in [14]. The work is still ongoing for implementing all views of the unifying meta-model. In this paper, the implementations of the CM and BPM views are discussed.

4 Implementation in MetaEdit+

MetaEdit+ is a tool for developing Domain Specific Languages (DSLs), which are non-generic modeling languages that are designed for a specific application domain

[5]. DSLs substitute the generic modeling capabilities offered by general-purpose languages (GPLs), such as UML[5], for more expressiveness that results from tailoring the language to the needs of a defined domain. Notations that are familiar to domain experts are utilized instead of generic shapes with broad semantics. Domain conventions and abstractions are also incorporated in DSLs. Generally, DSLs are not required to be executable [5]. However, when used in the context of MDD, a DSL needs to be transformed into other models and eventually into an executable form, following the MDD principles.

The modeling environment of MetaEdit+ is divided into two main parts: MetaEdit+ workbench and MetaEdit+ modeler. The workbench, shown in Fig. 2, includes the necessary facilities for creating the meta-model of the language (in this case the unifying meta-model), organized as a set of tools. The whole meta-model is called a *graph*, and components of the meta-model are created in the workbench using the object tool, property tool, and the relationship tool.

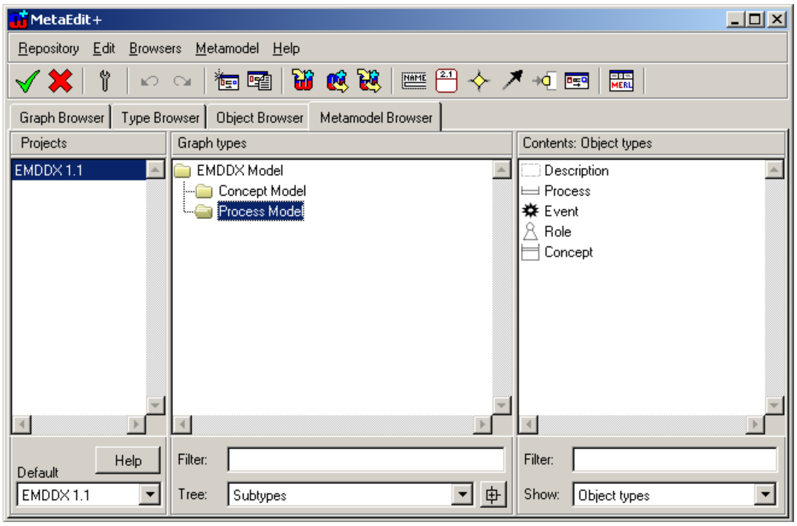
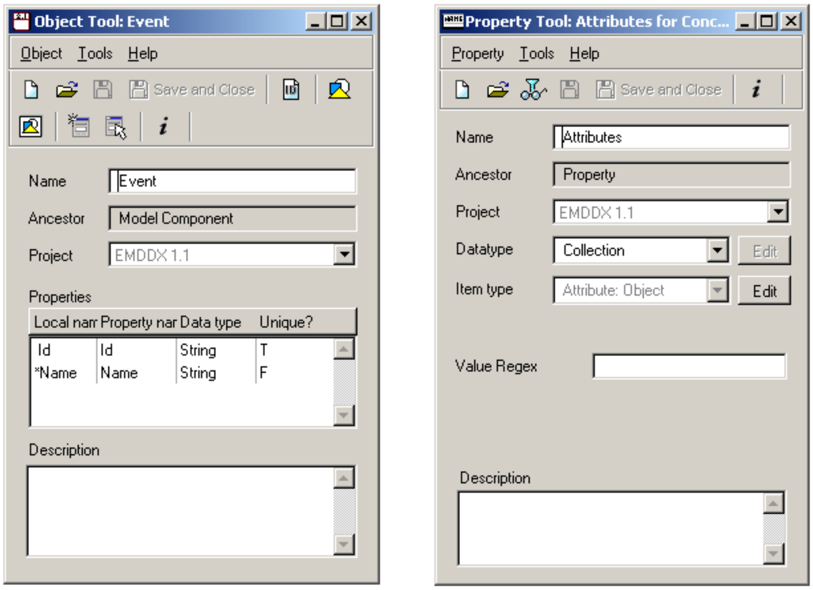


Fig. 2. The MetaEdit+ Workbench

The object tool (Fig. 3-a) is used to define concepts in the meta-model, where each concept is defined as an *object* that has a name, an ancestor (a super type), a description, and a list of properties created using the property tool (Fig. 3-b). In turn, each property has a name and a data type, and it is possible to define the input method for creating property values and constraints on those values. For example, a property can have the type "string," an editable list input widget to indicate that the property can have multiple values, and a regular expression that governs the values that can be entered by the user. Nesting of objects is made possible through allowing objects to be selected as the types of properties.

[5] http://www.uml.org/

Relationships are defined in the workbench using the relationship tool in a similar manner to objects. In fact, relationships are treated in MetaEdit+ as individual modeling components that can have properties of their own. Roles are used to connect concepts to relationships following the principles of Object Role Modeling (ORM)[6], a conceptual modeling method that focuses on the separation of concepts, relationships, and the roles which concepts play in the relationships in which they participate. To realize that, the workbench includes a role tool that can be used to create roles and assign properties to them. In general, and for the purposes of implementing the unifying meta-model, roles can be kept as simple connection points. However, the workbench allows for more control over the definition of roles because they are also treated as standalone modeling components.



Name: Event
Ancestor: Model Component
Project: EMDDX 1.1
Properties

Local nam	Property nar	Data type	Unique?
Id	Id	String	T
*Name	Name	String	F

Description

a

Name: Attributes
Ancestor: Property
Project: EMDDX 1.1
Datatype: Collection
Item type: Attribute: Object
Value Regex:
Description

b

Fig. 3. The MetaEdit+ Object Tool (a) and Property Tool (b)

The graph tool (Fig. 4) is where all modeling components are combined to create the meta-model. In the graph tool, concepts and relationships that are part of the meta-model are selected, and concepts are assigned to relationships using the defined roles. Additional management is also possible, such as creating constraints on how many different relationships a concept can play a role in.

4.1 MDD Tool Qualities in MetaEdit+

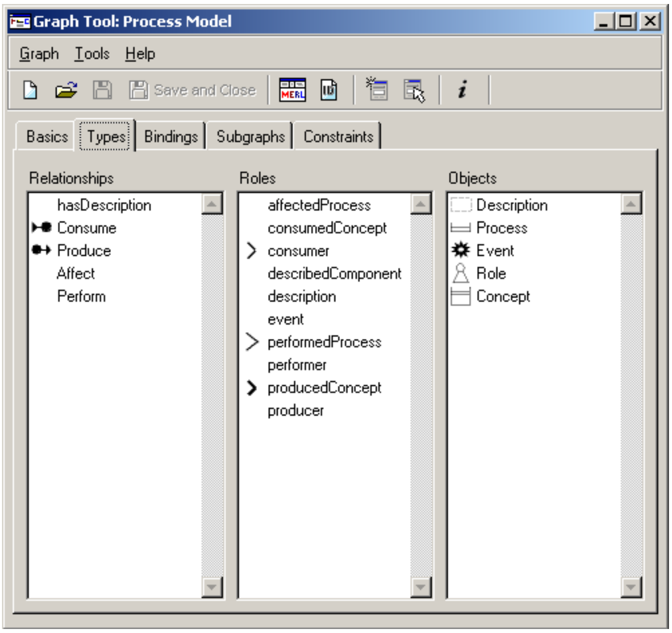
According to [1], MetaEdit+ implements the two-level cascading tool architecture. Tool users create a model on an upper level using the built in tool format in the

[6] `http://www.orm.net/`

workbench, constituting the meta-model of the DSL that is being developed. Then, the tool uses the meta-model to generate another tool, the MetaEdit+ modeler, which is used to create models on a lower level, representing instance models that follow the definition of the DSL. When implementing the CM and BPM views of the unifying meta-model in MetaEdit+, the two views were created using form based interfaces (some of which are shown in figures 3 and 4) in the workbench, and the resulting meta-model was maintained using a proprietary format. In other words, the meta-models were stored in a binary format that is only accessible in MetaEdit+. Then, the workbench generated a new modeler (Fig. 5) that was used to create instance models.

When it comes to understandability, the form based interfaces of the workbench tools hinder the ability of the tool user to gain an overview of the whole meta-model that is being implemented. Lists of objects, properties, relationships, and roles can be viewed separately, and only using the graph tool can they be viewed at the same time. Even then, the connections between objects and relationships using roles can be seen individually, without a full overview. This limitation became a real obstacle during the development of the unifying meta-model because repetitive change cycles meant that the tool user had to maintain a mental image over the way the meta-model is being changed. Eventually, a separate copy of the meta-model was created in MS Visio[7], adding extra complexity for maintaining two versions of the meta-model.

Unlike the workbench, the MetaEdit+ modeler has a graphical interface for creating the models. The graphical representations of modeling components used in the modeler are defined in the workbench using a WYSIWYG tool. However,



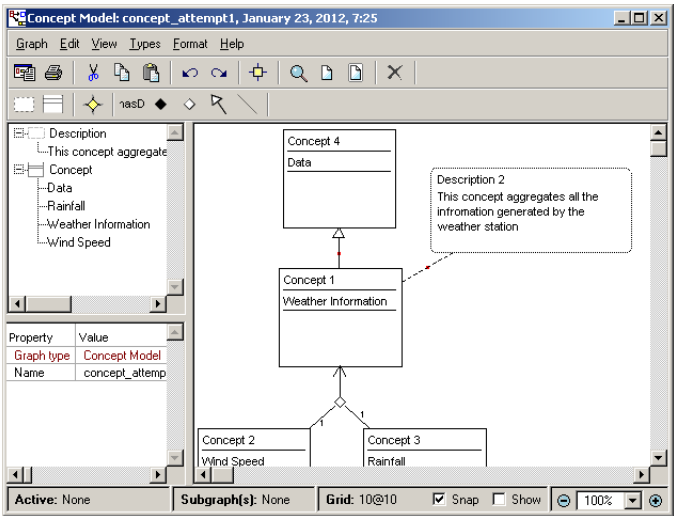
Relationships	Roles	Objects
hasDescription	affectedProcess	Description
Consume	consumedConcept	Process
Produce	consumer	Event
Affect	describedComponent	Role
Perform	description	Concept
	event	
	performedProcess	
	performer	
	producedConcept	
	producer	

Fig. 4. The MetaEdit+ Graph Tool

[7] http://office.microsoft.com/en-us/visio/

no additional support is offered by the tool in terms of explaining what each graphical symbol in the modeler means, limiting the understandability that can be offered during the creation of models.

Meta-models and models created using MetaEdit+ are both stored in a model repository that is part of the tool. The repository is a binary model database, the content of which can only be viewed using the MetaEdit+. Model reuse is supported by the existence of the repository. However, transporting models requires additional user intervention and is not directly supported by the tool. MetaEdit+ includes a scripting language, called MERL, which can be used to traverse models and generate corresponding textual reports. Using MERL, tool users can create generator scripts that can be used to output any text based on the structure and content of the model, including serialization standards (e.g. XMI[8]) or human readable reports (e.g. in HTML). During our work, model executability was enabled using generators which translated BPM models into Java classes and CM models into XML schemas, but this required additional effort since the classes and schemas needed to be constructed from scratch using output statements in MERL.



Property	Value
Graph type	Concept Model
Name	concept_attemp

Fig. 5. The MetaEdit+ Modeler for the CM view

The need for observability is eliminated by the tool architecture used in MetaEdit+, since models are not allowed to deviate from the way they are supposed to be constructed as dictated by the meta-model. For example, attempting to create a relationship between two instances of the wrong classes would prompt the tool to display an illegal operation message. While being an advantage when creating models, the inability to arbitrarily connect instances or assign attributes limited our experimentation ability during the development of the unifying meta-model, because testing new structures was not possible. This limitation is also stressed by [1] while discussing the tool architecture.

[8] `http://www.omg.org/spec/XMI/`

Among the various SE activities that can be supported by MDD tools, only collaborative development is available in MetaEdit+, realized using multiple user accounts which can simultaneously access the same model repository.

The available documentation for using different parts of MetaEdit+, including the tools in the workbench, the modeler, and the scripting language, is quite extensive. Tutorials are available to guide beginners, and there is an active community that can provide help when needed. MetaEdit+ offers additional modeling functionalities that can be helpful for complex DSLs, such as model embedding, where a single modeling component can be further describe using another model, as well as the ability to reuse modeling components across multiple models. However, these advantages are overshadowed by the inflexibility of the form-based interfaces and the two-level cascading architecture.

5 Implementation in EMF/GMP

Since its introduction as an Integrated Development Environment (IDE) over a decade ago, Eclipse has grown to be one of the major software development platforms available today. The open source and free software nature and the plug-in based architecture contribute to a wide and highly customizable range of modules that are available in Eclipse. A large community is involved in developing Eclipse plug-ins, and the development is organized in projects that focus on certain domains. The Eclipse modeling project[9] focuses on modeling standards, frameworks, and tooling, and includes the Eclipse Modeling Framework (EMF) and the Graphical Modeling Project (GMP), which were used to implement the unifying meta-model and develop its graphical editor.

EMF provides facilities for building applications using models and model transformations. Meta-models can be created in EMF using Ecore, which is an implementation of Essential MOF (EMOF), a subset of MOF[10] that is aligned with implementation technologies and XML. This highlights the mindset assumed by EMF as component in an implementation of the Model-Driven Architecture (MDA)[11] [11]. Ecore meta-models can be acquired from models written in annotated Java code, XML, or UML. The meta-models are transformed into Java code that can be used to create, edit, and serialize models. EMF is also able to generate a basic editor for the models. However, creating a rich graphical editor for models is made possible through the functionalities of the GMP plug-ins [3]. The combination of EMF and GMP enables the development of DSLs and accompanying rich graphical editors in a manner that is aligned with MDD principles.

The process for implementing the unifying meta-model using EMF and GMP is outlined in Fig 6, and is based on the recommended process in GMP. The first step of the process is to create the domain Ecore model, which in our case covers the CM and BPM views of the unifying meta-model. The domain model is acquired from a UML

[9] `http://www.eclipse.org/modeling/`
[10] `http://www.omg.org/mof/`
[11] `http://www.omg.org/mda/`

model designed in Papyrus[12], which is an Eclipse plug-in for creating UML models. EMF includes a GMP-based graphical editor that can be used for directly creating the domain model. However, it suffers from synchronization problems and changes are not always correctly reflected in the Ecore model. Papyrus represented a good alternative, especially since it offers a similar modeling experience, itself being built using GMP.

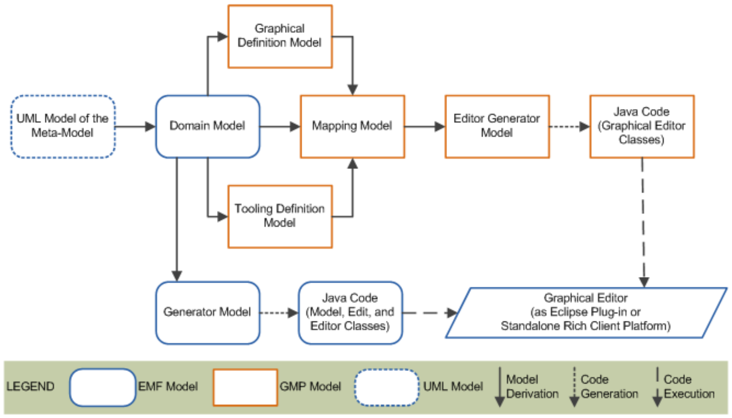
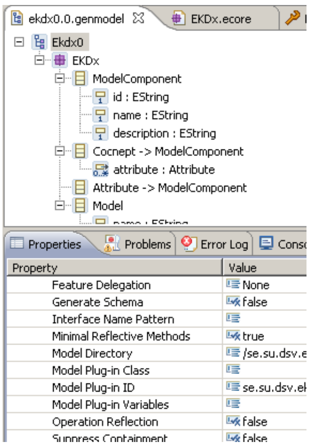

Graphical Definition Model
UML Model of the Meta-Model
Domain Model
Mapping Model
Editor Generator Model
Java Code (Graphical Editor Classes)
Tooling Definition Model
Generator Model
Java Code (Model, Edit, and Editor Classes)
Graphical Editor (as Eclipse Plug-in or Standalone Rich Client Platform)
LEGEND
EMF Model
GMP Model
UML Model
Model Derivation
Code Generation
Code Execution

Fig. 6. The Implementation Process of the Unifying Meta-Model Using Eclipse Plug-ins

The domain model is then used to create the generator model, which is a facility available in EMF to extend the domain model with implementation-specific details which lie outside the scope of the meta-model. These details include features that can be enabled, disabled, or customized during code generation, such as interface naming pattern and operation reflection. The generator model is the one used in EMF to actually generate the Java classes for creating and editing models. An excerpt from the generator model is shown in Fig. 7.



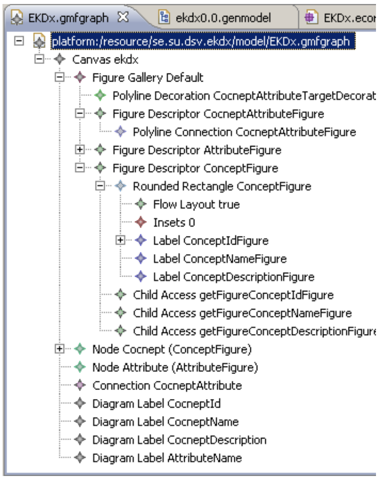
```
Ekdx0
  EKDx
    ModelComponent
      id : EString
      name : EString
      description : EString
    Cocnept -> ModelComponent
      attribute : Attribute
    Attribute -> ModelComponent
    Model
```

Property	Value
Feature Delegation	None
Generate Schema	false
Interface Name Pattern	
Minimal Reflective Methods	true
Model Directory	/se.su.dsv.e
Model Plug-in Class	
Model Plug-in ID	se.su.dsv.el
Model Plug-in Variables	
Operation Reflection	false
Suppress Containment	false

Fig. 7. The Generator Model of EMF

[12] www.papyrusuml.org/

Two complementary models that are part of GMP are derived from the domain model. The graphical definition model describes the graphical notation that is used to create instances of the meta-model which is being implemented, while the tooling definition model describes the palette that will be generated as part of the graphical editor to select and create modeling components. By using these models, GMP separates the meta-model from its visual representation and from the functionality used to create it. The graphical definition model and the tooling definition model can be edited separately from the domain model to customize the graphical editor. For instance, Fig. 8 shows the graphical notation of the "concept" modeling component of CM, defined as a rounded rectangle that includes labels for the id, name, and description properties of the concept. Similarly, the graphical representations of other concepts and relationships can be defined, and GMP offers a range of basic shapes with the ability to customize them and embed them in each other, contributing to a flexible notation definition tool.



```
platform:/resource/se.su.dsv.ekdx/model/EKDx.gmfgraph
  Canvas ekdx
    Figure Gallery Default
      Polyline Decoration CocneptAttributeTargetDecorat
      Figure Descriptor CocneptAttributeFigure
        Polyline Connection CocneptAttributeFigure
      Figure Descriptor AttributeFigure
      Figure Descriptor ConceptFigure
        Rounded Rectangle ConceptFigure
          Flow Layout true
          Insets 0
          Label ConceptIdFigure
          Label ConceptNameFigure
          Label ConceptDescriptionFigure
        Child Access getFigureConceptIdFigure
        Child Access getFigureConceptNameFigure
        Child Access getFigureConceptDescriptionFigure
    Node Cocnept (ConceptFigure)
    Node Attribute (AttributeFigure)
    Connection CocneptAttribute
    Diagram Label CocneptId
    Diagram Label CocneptName
    Diagram Label CocneptDescription
    Diagram Label AttributeName
```

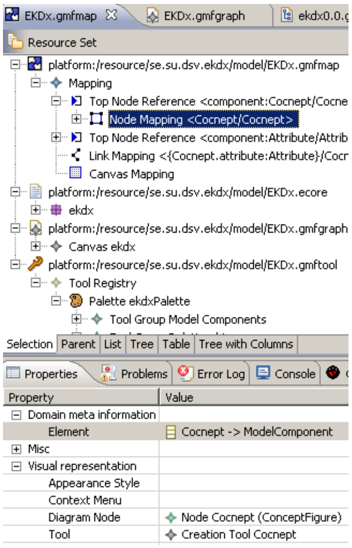
Fig. 8. The Graphical Definition Model of GMP

A mapping model is created to integrate the domain, graphical definition, and tooling definition models. The integrated model is finally used to derive an editor generator model that adds the necessary implementation-specific details, and has a role similar to that of the generator model in EMF. Fig. 9 illustrates the mapping model, showing how a node mapping is created for the concept modeling component to associate the concept definition in the domain model with its graphical notation in the graphical definition model and its tool in the palette that is created from the tooling definition model.

The code generated from the editor generator model is combined with the code generated from the EMF generator model to create the graphical editor. GMP offers a choice to generate the editor as a standalone Rich Client Platform (RCP) that can be used solely to create and edit models, or as an Eclipse plug-in that can be installed in an Eclipse environment and used in combination with other plug-ins.

5.1 MDD Tool Qualities in EMF/GMP

According to the tool architecture proposed by [1], the combination of EMF and GMP constitute an MDD tool that realizes level compaction while providing spanning at the same time. Level compaction refers to the use of one representation format to capture two modeling levels [1], and this is achieved in EMF by using Ecore to describe the structure of domain and instance models. As mentioned earlier, Ecore is an implementation of a subset of MOF called EMOF. Since MOF represents the format abstraction of both meta-models and models, the models can serve as meta-models that can be further instantiated. In other words, the graphical editor generated for the unifying meta-model can be used to design models that can serve as domain models for creating new graphical editors, and this cycle can be repeated endlessly. As for spanning, the use of MOF and the model editing framework provided in EMF enable the creation of different modeling languages, be it a DSL or a GPL, using the same generic mechanisms. A desired side-effect is the increased understandability by tool users; developing a new DSL does not require additional knowledge of the tool. Understandability is further enhanced by using a standardized language for meta-modeling, i.e. MOF, because the semantics of the language are well-defined.


EKDx.gmfmap | EKDx.gmfgraph | ekdx0.0.g

Resource Set

```
platform:/resource/se.su.dsv.ekdx/model/EKDx.gmfmap
  Mapping
    Top Node Reference <component:Cocnept/Cocne
      Node Mapping <Cocnept/Cocnept>
    Top Node Reference <component:Attribute/Attrib
    Link Mapping <{Cocnept.attribute:Attribute}/Cocr
    Canvas Mapping
platform:/resource/se.su.dsv.ekdx/model/EKDx.ecore
  ekdx
platform:/resource/se.su.dsv.ekdx/model/EKDx.gmfgraph
  Canvas ekdx
platform:/resource/se.su.dsv.ekdx/model/EKDx.gmftool
  Tool Registry
    Palette ekdxPalette
      Tool Group Model Components
```

Selection | Parent | List | Tree | Table | Tree with Columns

Properties | Problems | Error Log | Console

Property	Value
Domain meta information	
Element	Cocnept -> ModelComponent
Misc	
Visual representation	
Appearance Style	
Context Menu	
Diagram Node	Node Cocnept (ConceptFigure)
Tool	Creation Tool Cocnept

Fig. 9. The Mapping Model of GMP

Another factor effecting understandability is the ability to acquire domain models from a variety of sources. Tool users who are familiar with Java code can use annotated java to design the meta-model, while users who prefer XML or UML can continue to work in their usual ways, and EMF will take care of deriving the Ecore domain model from the input meta-model. The existence of a plethora of UML graphical modeling tools, both as standalone environments and as Eclipse plug-ins, allows further flexibility in creating the domain model. In fact, the switch from the

EMF built-in Ecore graphical editor to Papyrus was seamless, made easier by the fact that both editors are built using GMP and present similar user experiences.

Both EMF and GMP include a general Eclipse plug-in for error highlighting and management. Using the wrong value for a property or not entering the value of a required property when editing EMF and GMP models prompts Eclipse to highlight the involved property and display an explanatory message. Suggested solutions are also offered for common and recurring types of problems.

The models that are created in EMF and GMP are encoded in XML and eventually transformed into Java code, making them human-readable as well as transportable to other tools or platforms. The open source nature of Eclipse enables tool users to access the source code of the plug-ins and customize their functionality if necessary, changing the generated code to suit their needs. The Eclipse plug-ins also support model integration, which occurs at two points in the process illustrated in Fig. 6: the mapping model integrates the domain, graphical definition, and tooling definition models, and the final graphical editor is created by integrating code generated from the editor generator model with code generated from the EMF generator model. Wizards exist to guide users through repetitive and common activities.

Traceability is partially managed by the facilities of EMF and GMP through the use of naming conventions. This way, model parts and generated code can be traced back to their sources. However, traceability is limited in parts of GMP due to the separation between the domain model and its graphical notation and tooling. Changes to the graphical definition and tooling definition models will be lost if the models are re-generated following changes to the domain model.

The large community that stands behind the development of Eclipse guarantees a wide range of plug-ins that can support the software development process. Code repositories, task lists, collaboration, project planning, and integration with other tools are only a few domains for which plug-ins can be found. Furthermore, resources associated with Eclipse in general and with EMF are plentiful, both as books and on the internet. However, it was difficult to find up-to-date resource on GMP.

6 Conclusion

The increased interest in MDD as an effective means for developing IS in recent years has led to plenty of research in this domain. Many open questions still stand [13], among which is the availability of tools that are able to support all the principles of MDD. In this paper, we report on our experience of using two MDD tools: MetaEdit+ and the Eclipse plug-ins of EMF and GMP. The tools were used to implement the unifying meta-model, which is an attempt to bridge the gap between EM and MDD and streamline the development of IS that are more aligned with organizational goals. The results of this report and can help both tool developers and practitioners in gaining helpful insights into the investigated tools. Table 1 summarizes the observations of this report.

Our observations show that both tools have advantages and drawbacks. On one hand, the separate tools available in MetaEdit+ for creating and managing concepts, properties, relationships, and ports, in addition to the graph itself (i.e. the meta-model) provided increased control over the definition of the meta-model. MetaEdit+ combined all modeling components in a single graph (albeit using multiple user interfaces). A single transition was then required to generate an editor, and only one modeling technique was involved. But these advantages were limited by the use of a proprietary modeling technique and the lack of a single and complete view of the whole meta-model. The need for the overall view forced us to rely on an external tool (MS Visio), multiplying the effort needed to maintain the meta-model, which was still under development.

Table 1. Summary of our experience in using MetaEdit+ and EMF/GMP

	MetaEdit+	*Eclipse plug-ins of EMF and GMP*
Understandability	+ Consistent interfaces of all tools. + Graphically create models. – No overview of the whole meta-model. – No explanation during the creation of instance models.	+ Uniform usage of models and meta-models because MOF is the meta-meta-modeling language. + Support for many sources and formats to acquire domain models, hence adapting to user skills.
Model Evaluation	+ Tool architecture eliminates the need for evaluation: not possible to create models that do not conform to the meta-model.	+ Ability to use a general Eclipse plug-in that provides error highlighting and suggested solutions during development time.
Executability	+ Scripting language for generating any text from models. – No generation framework (e.g. for Java code).	+ Many Eclipse plug-ins to support executability (e.g. Java Emitter Templates JET).
Model Repositories	+ Model reuse. – Proprietary binary storage format.	+ XML-based: human-readable and accessible by other tools. + Java-based, enabling access and editing of code. + Many Eclipse plug-ins to support model integration.
Traceability and Change Management	*Not applicable since only one model is used.*	+ Partially supported using naming conventions. – Limited in parts of GMP, causing changes to be lost if some models are re-generated.
Other SE Activities	+ Collaborative development.	+ Many Eclipse plug-ins to support a myriad of activities.
Tool Documentation	+ Extensive documentation. + Large support community.	+ Extensive documentation. + Large support community. – No up-to-date resources on GMP.

One the other hand, EMF and GMP implemented well-known and open standards. The visual representation of the domain model in EMF offered the necessary overall view and shortened the cycle of updating the meta-model and testing the changes for suitability. EMF and GMP used many models and automatic transformations between the models, constituting an MDD approach that covered several layers of modeling. But while this separation is necessary to decouple unrelated information, it resulted in a long development process that involved many types of models. Consequently, different types of modeling knowledge were required.

The implementation of the unifying meta-model is part of a research effort that involves developing an MDD approach that extends to cover EM aspects as well. Acquiring a tool that supports the creation and editing of models described by the unifying meta-model is only one step. In terms of tooling, the next step will investigate the possibility of adding executability support to the editors generated using MetaEdit+ and the Eclipse plug-ins. (This is not to be confused with the executability support already available in MetaEdit+ and the Eclipse plug-ins; both tools are able to generate editors from models). Executability in the generated editor is not directly supported in MetaEdit+, but arises as a side effect of MERL, which could be used to generate executable code. Several possibilities exist in Eclipse for extending the generated editor with executability support, such as Java Emitter Templates (JET), which is another part of the Eclipse modeling project.

References

1. Atkinson, C., Kühne, T.: Concepts for Comparing Modeling Tool Architectures. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 398–413. Springer, Heidelberg (2005)
2. da Silva, A.R., Saraiva, J., Ferreira, D., Silva, R., Videira, C.: Integration of RE and MDD Paradigms: The ProjectIT Approach and Tools. IET Software 1(6), 294–314 (2007)
3. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley (2009)
4. Henkel, M., Stirna, J.: Pondering on the Key Functionality of Model Driven Development Tools: The Case of Mendix. In: Forbrig, P., Günther, H. (eds.) BIR 2010. LNBIP, vol. 64, pp. 146–160. Springer, Heidelberg (2010)
5. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-Specific Languages. ACM Computing Surveys 37(4), 316–344 (2005)
6. Navarro, E.: Architecture Traced from Requirements applying a Unified Methodology, PhD thesis, Computing Systems Department, UCLM (2007)
7. Oldevik, J., Solberg, A., Haugen, Ø., Møller-Pedersen, B.: Evaluation Framework for Model-Driven Product Line Engineering Tools. In: 10th International Conference on Software Product Lines, SPLC 2006, pp. 589–618. IEEE (2006)
8. Panach, J.I., Condori-Fernández, N., Baars, A., Vos, T., Romeu, I., Pastor, Ó.: Towards an Experimental Framework for Measuring Usability of Model-Driven Tools. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part IV. LNCS, vol. 6949, pp. 640–643. Springer, Heidelberg (2011)

9. Pastor, O., Giachetti, G.: Linking Goal-Oriented Requirements and Model-Driven Development. In: Intentional Perspectives on Information Systems Engineering, pp. 257–276. Springer, Heidelberg (2010)
10. Pelechano, V., Albert, M., Muñoz, J., Cetina, C.: Building Tools for Model Driven Development. Comparing Microsoft DSL Tools and Eclipse Modeling Plug-ins. In: The Conference on Software Engineering and Databases, JISBD 2006, Sitges, Spain (2006)
11. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley (2009)
12. Uhl, A.: Model-Driven Development in the Enterprise. IEEE Software 25(1), 46–49 (2008)
13. Zikra, I., Stirna, J., Zdravkovic, J.: Analyzing the Integration between Requirements and Models in Model Driven Development. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) BPMDS 2011 and EMMSAD 2011. LNBIP, vol. 81, pp. 342–356. Springer, Heidelberg (2011)
14. Zikra, I., Stirna, J., Zdravkovic, J.: Bringing Enterprise Modeling Closer to Model-Driven Development. In: Johannesson, P., Krogstie, J., Opdahl, A.L. (eds.) PoEM 2011. LNBIP, vol. 92, pp. 268–282. Springer, Heidelberg (2011)