Evaluation of Modeling Tools Adaptation

Amine El Kouhen, Cedric Dumoulin, Sébastien Gérard, Pierre Boulet

Evaluation of Modeling Tools Adaptation

Amine El Kouhen[1, 2], Cédric Dumoulin[2], Sébastien Gérard[1], Pierre Boulet[2]

[1] Commissariat à l'Energie Atomique (CEA) LIST, Laboratory of Model Driven Engineering for Embedded Systems (LISE) Point Courrier 94, 91191, Gif sur Yvette, France
{amine.elkouhen, sebastien.gerard}@cea.fr
[2] Laboratoire d'Informatique Fondamentale de Lille (LIFL) CNRS UMR 8022
U.S.T.L Cité Scientifique, F-59655 Villeneuve d'Ascq Cedex, France
{amine.el-kouhen, cedric.dumoulin, pierre.boulet}@lifl.fr

Abstract. This paper proposes an evaluation for modeling tool's adaptation by observing how well they can be used to customize graphical editors for a sample DSML proposed as a case study. It also discusses the current state of the art, and compares what was done in every tool that we have evaluated, according to relevant criteria. It was perceived; during our research that there is a clear need in term of criteria that supports evaluating Editor's customization quality. For this aim, we propose such criteria in this paper. We review tool's adaptation approaches, adaptation categories regarding different points of view and evaluate tools, with respect to their productivity and expressivity according to the proposed evaluation criteria.

Keywords: MDE, modeling tools, customization, assessment, quality, metrics

1 Introduction

Models are powerful tools to express the structure, behavior, and other properties in all areas of engineering and each of the hard sciences [17]. While models are very widespread, an explicit definition of a Domain-Specific Modeling Language (DSML) and an explicit manipulation of its models are closely connected to some support tools, called Computer-Aided Software Engineering tools or simply "CASE tools".

The design and generation of such tools can be done either using program-based environment or applying model-based tools called Meta-CASE tools [12]. The intent of meta-CASE tools is to capture the specification of the required CASE tool and then generate automatically the tool. In general, meta-CASE tools provide generic CASE tool components that can be customized and instantiated into particular CASE tools [11].

In the past years, a strong interest in model-driven engineering has resulted in many Domain-Specific Languages (DSL's). Among the advantages of DSLs identified in [44], is that DSML allows focusing on the concepts of the considered domain. This also implies that designers who are already acquainted with the domain will use more intuitively the language and tools which support this language. However, these approaches have obvious drawbacks, among which the main one is the additional efforts to design modeling languages from scratch [45]. Consequently, this has resulted in various Meta-CASE tools, which promise to reduce these efforts, and support, in different ways, customization of modeling environments. The purpose

of this paper is to evaluate the most relevant facet of modeling tools customization, which is the graphical editors' customization, by evaluating some of these tools/technologies, namely the IBM Rational Software Architect (RSA), the Generic Modeling Environment (GME), MetaEdit+, Obeo Designer and the Graphical Modeling Framework (GMF). A common case study, based on a simplified version of the Business Process Modeling Notation (BPMN), is used to assess the maturity of these tools.

In Section 3 we evaluate such tools by observing how well they can be used and/or reused (adapted) to customize graphical editors for a BPMN diagram, for which a simplified metamodel is provided, we discuss also customization capabilities and features, for each tool. But before that, we introduce an overview of the evaluation workbench, and retained criteria and metrics to assess tools. In Section 4, we discuss the evaluation criteria, results, and lessons learned during the creation of editors with these tools. Finally, we discuss our future work and conclude respectively in Sections 5 and 6.

2 Evaluation Workbench

We introduce in this section, an overview of the evaluation context and criteria chosen to assess tools quality.

2.1 Case Study

The *Business Process Management Initiative* (BPMI) has developed a standard *Business Process Modeling Notation* (BPMN). BPMN is dedicated to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes [21].
For the purpose of this assessment, we have created a simple metamodel that includes a subset of the language concepts (Figure 2). This metamodel is not meant to be a realistic representation of BPMN (this is outside the scope of this study). A complete specification of the BPM notations and semantics can be found in [4].

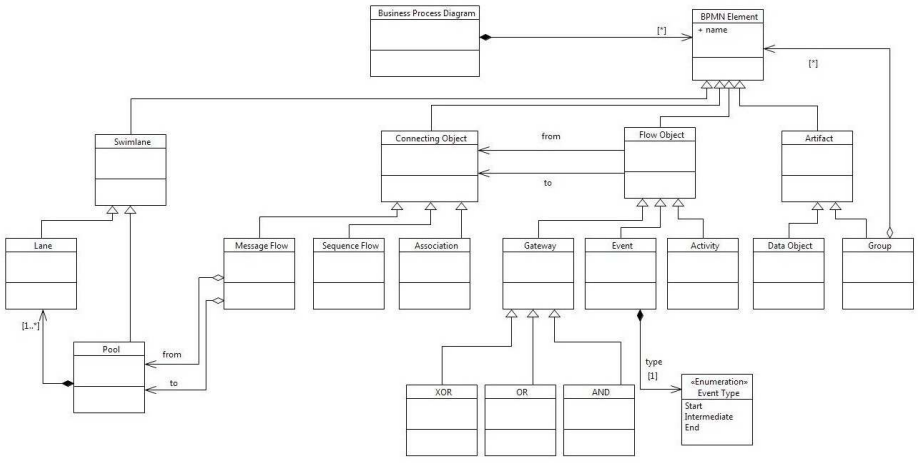

Fig. 1. Simplified BPMN metamodel

BPMN defines a *Business Process Diagram* (BPD), which is based on a flowcharting technique tailored for creating graphical models of business process operations. A Business Process Model, then, is a network of graphical objects, which are activities (i.e., tasks) and the flow controls that define their order of performance [21].

In terms of concrete syntax elements, there are four basic categories of elements: Flow Objects, Connecting Objects, Swimlanes, and Artifacts. The symbols corresponding to them are summarized in Figure 3.

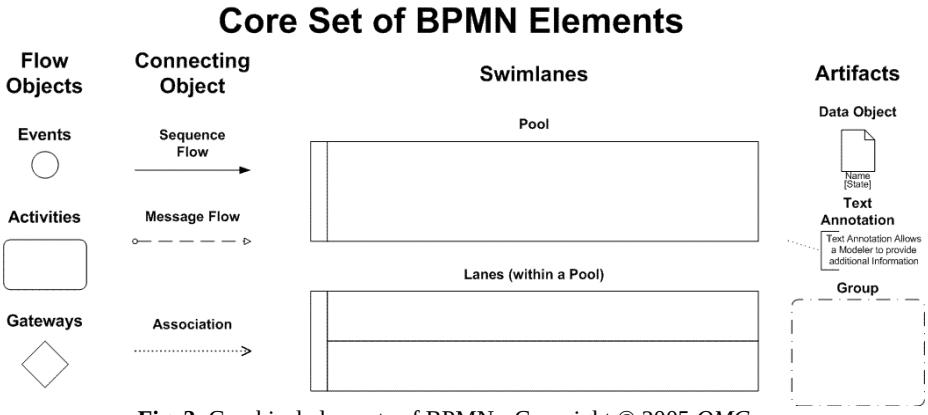

Fig. 2. Graphical elements of BPMN - Copyright © 2005 *OMG.org*

2.2 Evaluation criteria and metrics

Two evaluation approaches were suggested by P. Mohagheghi and Ø. Haugen in [1]:

Qualitative approaches cover case studies, analysis of a language and the tool by experts for various characteristics, and monitoring or interviewing users.

For the quantitative evaluation, they identified several metrics (*effort, understandability, Usability...etc*). In our study we put a particular emphasis on the following evaluation criteria, which are most relevant in our context. For each criterion we propose some metrics to quantify and concretize this evaluation:

- ***Customization level criterion***: what is the proportion of customizable parts in the tool?

  As metric, we propose the adaptability level ***(AL)*** equals to:

  $$AL = 100 \times (C\_functions / S\_functions) \quad (1)$$

  Where:
  C_functions: is the number of customizable functions in a tool.
  S_functions: the total number of functions. This number is determined by identifying the components of the system as seen by the end-user. There are three steps in the process of counting tool functions:

  – Identify the scope and boundary of the count: represent the boundary of the evaluated application.

- Determine the primary process areas. An elementary process is the smallest unit of activity that is meaningful to the user.
- Identify for each process, data functions (external inputs, external outputs and external inquiries) and transactional functions (interfaces to other systems, and internal logical files). They must be unique, user recognizable and non-repeated field functions.

This methodology is based on the best known method of counting functions which is the Function points Analysis. It was defined in 1979 by Allan Albrecht at IBM [29]. Function points (FPs) can be used to estimate the relative size and complexity of software [36].

- ***Graphical expressiveness and completeness criteria****:* Can we represent all the notation elements? Can we use the full range of visual variables [28]? And what's their complexity?

  For the *Graphical expressiveness* we apply the D. Moody's scale [9] to our context to measure this criterion; it consists in assess tool capability to represent the eight visual following variables: retinal variables (shape, texture, brightness, size and color) - planar variables (Horizontal position and Vertical position).
  For the *Graphical completeness* we assess the tools capability to represent all sorts of shapes weighted on a scale of 0 – graphs not presents (textual editors), 1 - minor graphical completeness, to 5 - strong graphical completeness.

- ***Tool openness criteria****:* This criterion is composed of four sub criteria
  - *Tool building approaches:* what are the approaches used to describe the semantic and graphical parts of the editor? (e.g, Proprietary languages, standard language, open language…)
  - *Extensibility:* is it possible to add additional languages or integrate with other tools? And How?
  - *Reusability:* Can we reuse existing parts/functionalities of our tool? The tool support separation of concerns?
  - *Maintainability:* this aspect was not thoroughly studied in our evaluation; it was difficult to provide metrics to assess this criterion. However, there are some approaches in the software reverse engineering and software quality fields which provide estimations of the tools maintainability (e.g. Total number of code lines, number of code lines per object, methods number per object, total number of methods, ratio of code lines/number of methods, ratio of code lines/number of objects, Ratio comment lines/code lines), specialization index, level of abstraction, cyclomatic complexity etc.).
- ***Tool Usability criteria****:* Does the generated tool's editor support features to achieve the specific goal of the context of use?

  There are many metrics and categorizations that can be used to measure UI usability [31]. Usability is discussed also in [15]. Seffah et al. define usability as "whether a software product enables a particular set of users to achieve specific goals in a specific context of use" and cover the ten usability factors: efficiency, effectiveness, productivity, satisfaction, learnability, safety, trustfulness, accessibility, universality and usefulness for solving problems. Braz et al. in [33] sort some of these metrics in two categories:

- Countable metrics: they are extracted from data collected from observations, interviews, survey, logs…
- Calculable metrics: represent the result from mathematical calculations, algorithms, or heuristics with observational data and countable metrics.

Constantine and Lockwood [32] propose also some UI usability metrics and classify them in three main categories:

1. Structural metric: based on the UI surface properties.
2. Semantic metrics: based on the UI content.
3. Procedural metrics: based on the tasks (user triggered tasks or automatic tasks)

In [15] Seffah et al. identify more than 127 metrics to measure usability. We retain the most used factors of usability and their metrics for our evaluation, which are: efficiency, effectiveness and accessibility. We provide also a feedback for the learnability and satisfaction factors, which are dependents to users' preferences.

a. For efficiency we have chosen the Essential Efficiency (***EE***) [32]:

$$EE = 100 \times (S\_essential / S\_enacted) \quad (2)$$

Where:

EE: Estimates how closely a given user interface design approximates the ideal expressed in the use case model

S_essential = the number of user steps in the essential use case narrative (conceptual steps).

S_enacted = the number of steps needed to perform the use case with the user interface design. Rules for counting the number of the enacted steps [32] are following:

- Entering data into one field by continuous typing that is terminated by an enter, a tab, or some other field separator.
- Skipping over an unneeded field or control by tabbing or by means of any other navigation key.
- Selecting a field, an object, or a group of items by clicking, double-clicking, or sweeping with a pointing device.
- Selecting a field, an object, or a group of items with a keystroke or series of connected keystrokes.
- Switching from keyboard to pointing device or from pointing device to keyboard.
- Triggering an action by clicking or double-clicking with a pointing device on a tool, a command button, or some other visual object.
- Selecting a menu or a menu item by a pointing device.
- Triggering an action by typing a shortcut key or key sequence, including activating a menu item through keyboard access keys.
- Dragging-and-dropping an object with a pointing device.

b. For effectiveness we have chosen the formula proposed by Bevan and Macleod in [34] for calculating task effectiveness (***TE***):

$$TE = Quantity \times Quality / 100 \quad (3)$$

Where:

Quantity is a measure of the amount of a task completed by a user. It is defined as the proportion of the task goals represented in the output of the task and ***Quality*** is a measure of the degree to which the output achieves the task goals.

c. For accessibility we have chosen both next metrics: the task visibility (***TV***) [32] and the visual coherence (***VC***) [35] metrics.

$$TV = 100 \times (\sum V_i / S\_total) \quad (4)$$

Where:

TV: The proportion of interface objects or elements necessary to complete a task those are visible to the user.

S_total = total number of enacted steps to complete the use case.

V_i = Feature visibility (0 or 1) of enacted step i. The visibility depends on the types of enacted steps. There are four different categories of enacted steps:

1. **Hidden:** hidden operations draw on the user's internal knowledge of the application and its use apart from any information communicated by the visible user interface. Hidden steps include:
    - Typing a required code or shortcut in the absence of any visual prompting or cue.
    - Accessing a feature or features having no visible representation on the user interface.
    - Any action involving an object or a feature that may be visible but the choice of which is neither obvious nor evident based on visible information on the user interface.

    Opening a generic context menu by clicking on blank background with the right mouse button or typing a keyboard shortcut without being prompted is an example of hidden step. Hidden enacted steps are assigned a visibility of 0.

2. **Exposing**: An enacted step is exposing if its function is to gain access to or make visible some other needed feature without causing or resulting in a change of interaction context. Exposing actions include:
    - Opening a drop-down list.
    - Opening a menu or submenu.
    - Opening a context menu by right-clicking on some object.
    - Opening a property sheet dialogue for an object.
    - Opening an object or drilling down for detail.
    - Opening or making visible a tool palette.
    - Opening an attached pane or panel of a dialogue.
    - Switching to another page or tab of a tabbed dialogue.

    Exposing actions have an intermediate effect on task visibility and are assigned a visibility of 0.5, unless they are or must be accomplished using hidden features, in this case, they are classified as hidden and given a visibility of 0.
3. **Suspending:** An enacted step is suspending if its function is to gain access to or make visible some other needed feature and it causes or results in a change of interaction context. Suspending actions include:
    - Opening a dialogue box.
    - Closing a dialogue or message box.
    - Switching to another window.
    - Switching to or launching another application.

    Suspending or context-switching actions that occur as the first or last enacted step of extensions or other optional interactions have an intermediate effect on task visibility since they provide access to features that may not be needed in all

enactments; they are assigned a visibility of 0.5, unless they are or must be accomplished using hidden features, in this case, they are classified as hidden and their visibility is set to 0. Context changes that are non-optional, that are required in most or all enactments, have a strong effect on task visibility; these are assigned a visibility of 0.

4. **Direct:** An enacted step is a direct action if it is not hidden, exposing, or suspending. In other words, direct actions are accomplished through visible features whose choice is evident and which do not serve to gain access to or make visible other objects. These are assigned a visibility of 1.

The visual coherence ***(VC)***: shows how well a user interface keeps related components together and unrelated components apart.

$$VC = 100 \times G_k / (\sum N_k \times (N_k - 1) / 2) \quad (5)$$

Where:

G_k: the number of related visual component pairs in the group k. With:

$$G_k = \sum R_{i,j} \quad (6)$$

$R_{i,j}$ = semantic relatedness between components i and j in group k, $0 \leq R_{i,j} \leq 1$

In practice, semantic relatedness can be simplified to just two values: $R_{i,j} = 1$ if components i and j belong to the same semantic cluster and are, therefore, substantially related; $R_{i,j} = 0$, otherwise.

N_k = the number of visual components in the group k. A visual component is:

- Any user interface widget.
- An external label not on or embedded in a user interface widget.
- A pane, panel, or frame around any one or more widgets or labels.

Simple lines separating one part of the visual interface from another are not considered to be visual components in themselves.

- ***Required resources criterion***: How much Time and effort is required to model, debug, and generate artefacts [8]?

  We may also add time and effort to understand models. The adequate metric unit for this criterion is the man-day unit. This measurement was done by a single researcher with a background in modeling field but not necessarily an expert of the evaluated tools.

- ***License nature criterion****:* what is the kind of license required to use the tool? (Commercial, Proprietary, Open Source, Freeware…)
- ***Produced Artefacts criteria****:* what are characteristics of artefacts produced with the graphical editor?

  - *Analysis capabilities*: Can we easily analyze or transform models produced with the graphical editor?
  - *Artefect quality:* what is the quality level of models produced with the graphical editor? One quality benchmark that we found useful is described by T. Clarks et al. in [30]. The authors define five levels of produced artefact quality:

    1. The lowest level: a simple abstract syntax is defined, but not implemented yet in a tool. The static and dynamic semantics of the language is informal and incomplete. There is no specific tool

support: an existing language is repurposed, compliance with the DSL is manually maintained and models are mostly interpreted by users.
2. The abstract syntax and static semantics have been largely defined, implemented in a tool and validated. The dynamic semantics is still informally defined.
3. The abstract syntax is completely implemented and tested. Concrete syntax has been defined for the language, but not implemented yet. Optimization of the language architecture has started.
4. The concrete syntax of the language has been implemented and tested. Users create models either visually and textually. The language architecture has been optimized for reuse and extensibility. Tool support for dynamic semantics begins to appear.
5. The topmost level: all aspects of the language have been modeled, including its semantics. Models written in the language can be processed by the tool. Examples thereof include code generation, execution, simulation, verification. The language architecture is well optimized for reuse.

- *Artefact format*: What kind of persistence format of these models? (Open format, structured support, binary files…)

3 Evaluation Results

We have chosen in this evaluation, only tools which provide the characteristics of a Meta-CASE tool (i.e. they generate modeling tools). These tools present a representative sample of adaptable tools according different approaches (proprietary languages, model-based approach…): RSA represents all tools supporting UML extension mechanism and using the iconic representation of stereotypes (as MagicDraw, Enterprise Architect…). MetaEdit+ and GME represent tools which use proprietary languages to build editors. Obeo Designer represents a model-driven approach to specify modeling tools and finally we have chosen the open source framework GMF, because it is the most known and used technology to build editors (used in Papyrus…)

3.1 IBM Rational Software Architect (RSA)

IBM's *Rational Software Architect* (version 8.0) is a UML 2.0 compliant integrated software development environment, built on top of the Eclipse platform [27]. *RSA* provides the UML extension mechanism with the stereotypes for defining profiles, and allows generating editors for such profiles. UML's Profiles mechanism makes RSA's strength: based on UML, it benefits from its genericity, its reputation and its concrete syntax. At the same time, it makes also its weakness: UML contains a lot of concepts not always appropriate to particular needs of the DSML.

Creating a profile for a BPMN diagram in *RSA* is quite simple. A user needs to create a UML profile project (directly supported through the Eclipse New Project wizards), select metaclasses to be stereotyped, (optionally) specify icons and images,

and release the profile. In our example, as shown in figure 4, BPMN Process elements are stereotypes of the UML Activity metaclass, the BPMN Activity elements are stereotypes of the UML Action metaclass, and BPMN Association are stereotypes of the UML Dependency metaclass. The actual BPMN diagram is simply a UML class diagram with the extra BPMN stereotypes. For the BPMN elements, custom icons and shapes were used, but no such graphical customization exists for link styles (Connecting Objects). The transformation between BPMN DSL and UML metamodels was based on the OMG RFP (request for proposal) [5] discussing the construction of a UML profile for BPMN.

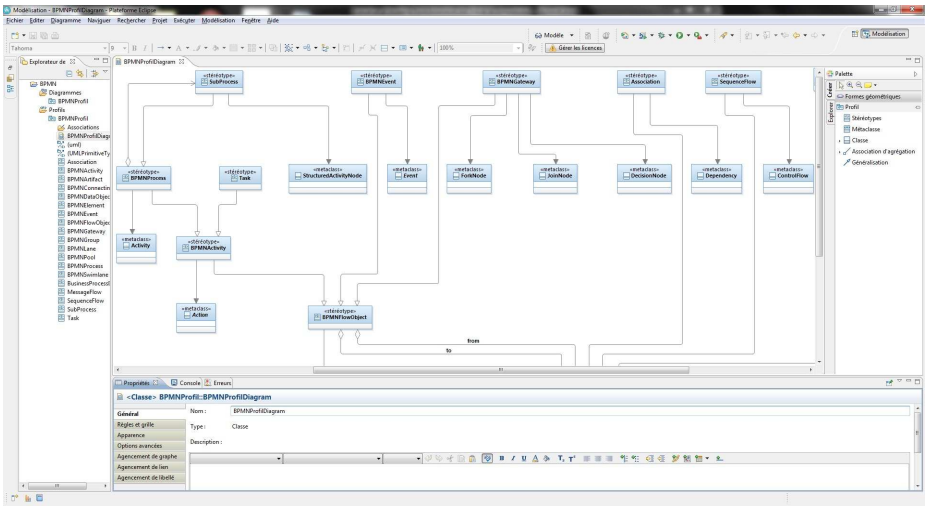

Fig. 3. UML Profile for BPMN simplified metamodel in *RSA*

The tool environment offers many features, including loading/saving, multiple undo/redo, filters, elements drag and drop from explorers to editors, validation, printing, zooming, property sheet, etc. The documentation is very good and abundant in the web. However, the usability of the editors generated in *RSA* is rather weak.

Other issues have been observed. *RSA* does not support custom restrictions on links' styles, and custom relationship types cannot be created (and hence class diagram elements can get mixed to the BPMN diagram, for instance multiplicities are shown by default, as shown in the red circles in Figure 5) because the generated edit parts extends GMF's "ConnectionNodeEditPart" which already defines a default figure (a line with a name label and multiplicity label).

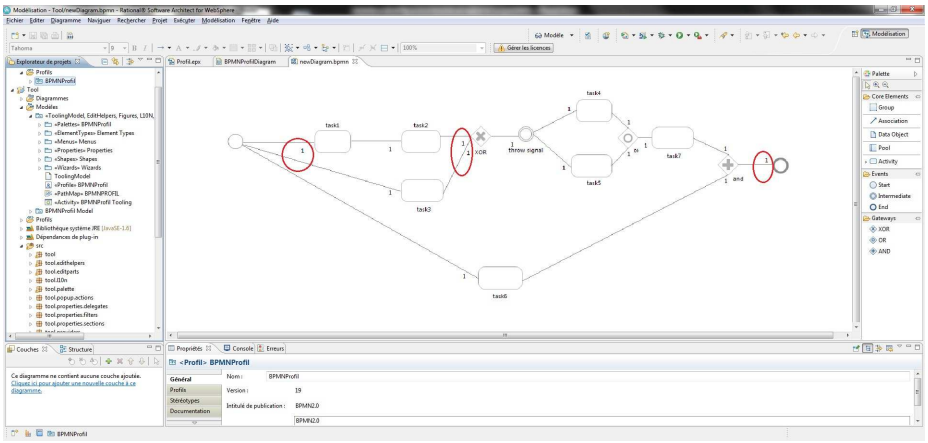

Fig. 4. Iconic representation of BPMN elements with *RSA*

RSA also refers to a diagram editor, called Tooling Model, to create custom palette entries, menu items, creation assistants (wizards), and property sheets. However, RSA confronts some gaps, on offered possibilities to customize the field's graphical presentations; it was among the reasons which explain why RSA pushes users to create a UML profile for their domain, since in the background, the possibilities of graphical customization can be summarized as:

- By default, the reuse of the concrete syntax of UML for the specific domain concepts (each concept is represented by the visual notation of its associated metaclass).
- The use of iconic representations of stereotypes (stereotype-associated image).
- Therefore, when generating custom shapes for non-relationship edit parts (nodes), a simple edit part, figure, and view in the shape of a standard rectangle is generated [10].
- For relationship edit parts (Links), code for a simple connector and label are generated.

However, if the user needs more complex forms, the tool offers only to modify the generated files and that requires a solid knowledge of the *Graphical Modeling Framework* (GMF), the *Graphical Editing Framework* (GEF), the *Eclipse Modeling Framework* (EMF) and the Eclipse development.

The user interface cannot be customized directly via the profile. But since *RSA* is based on Eclipse IDE, it inherits all Eclipse customization capabilities via its API.

3.2 Generic Modeling Environment (GME)

The *Generic Modeling Environment* (GME) is a configurable Meta-CASE tool developed in C++ at Vanderbilt University, providing toolkits for creating a Domain Specific modeling environment. Configuration is done by specifying the modeling paradigm metamodel that represents the modeling language of the application domain (in our case BPMN metamodel). The modeling paradigm contains, besides semantic parts; presentation informations regarding the domain [14].

The vocabulary of the domain-specific languages implemented by different *GME* configurations is based on a set of generic concepts built into *GME* itself. Folders, First-Class Objects (FCO) like Models (which can have inner parts and structures), Atoms (elementary objects), Sets (similar to UML aggregations), References, Connections (relationship between two objects within one model), Roles, Constraints and Aspects (provide primarily visibility control) are the main concepts that are used to define a modeling paradigm (figure 6). In other words, the DSL is made up of instances of these concepts. The choice of these generic concepts is certainly, the most critical design decision. Models in *GME* are similar to classes in Java; they can be instantiated. When a particular model is created in *GME*, it becomes a type (class). It can be subtyped and instantiated as many times as the user wishes [16].

This concept supports the reuse and maintenance of models because any change in a type automatically propagates down the type hierarchy. Also, this makes it possible to create libraries of type models that can be used in multiple applications in the given domain.

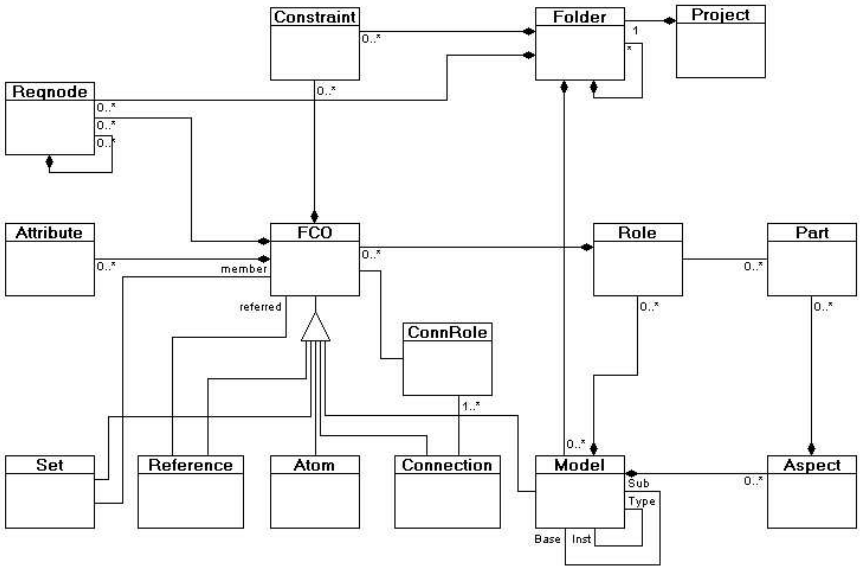

Fig. 5. *GME* modeling concepts

BPMN metamodel elements match directly to FCOs, Atoms and Connections in the *GME* metamodel (figure 7). Aspects can be used to control visibility of parts in the editor.

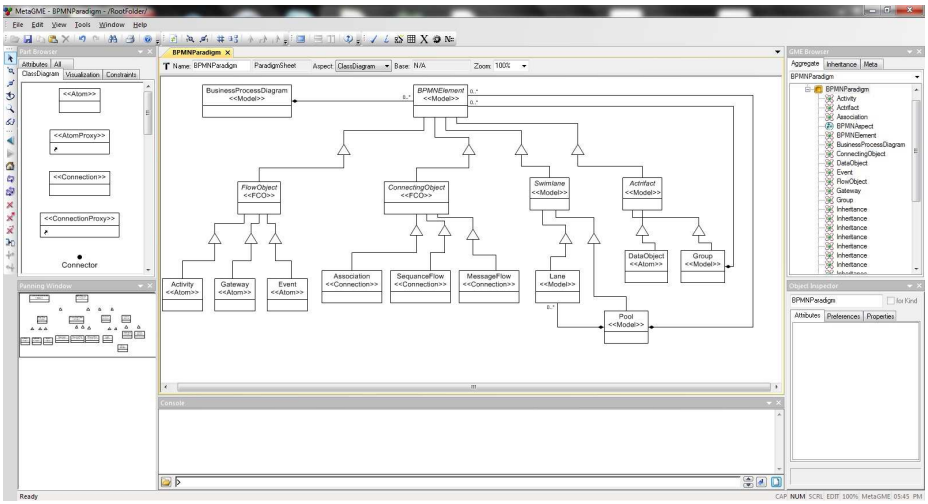

Fig. 6. BPMN metamodel represented as *GME* paradigm.

Once a paradigm is created and the decorators defined, it can be registered in *GME* as a new paradigm and then used as an editor, as shown in Figure 8. The tool provides many features: loading/saving (XMI), undo/redo, drag and drop interface for the creation of model elements, metamodel validation against multiplicities and OCL constraints, printing, zooming, overviews, property views, etc. We can also use the Aspect mechanism to create a sort of perspectives who restrict the metamodel concepts' viewing. The project is very well documented [14]. However, we have found it difficult to create custom styles for links; there are only two available styles (solid line, dash line) and arrows shapes are limited to ten. For nodes representations, we cannot create complex shapes besides iconic representations (Bitmap Image).

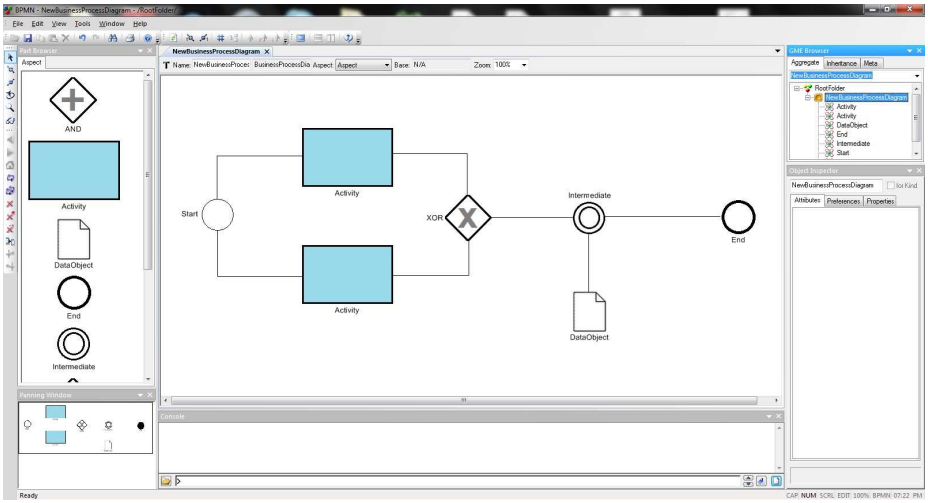

Fig. 7. BPMN editor produced using *GME*

GME has a modular, extensible architecture that uses MS COM for integration. *GME* is easily extensible; external components can be written in any language that supports COM (C++, Visual Basic, C#, Python etc.). *GME* has many advanced features. A built-in constraint manager enforces all domain constraints during model building. *GME* supports multiple aspect modeling. It provides metamodel

composition for reusing and combining existing modeling languages and language concepts. It supports model libraries for reuse at the model level. All *GME* modeling languages provide type inheritance. Model visualization is customizable through decorator interfaces [13], [14]. *GME* supports the visual drawing of an object with a COM object called decorator. This allows (with several limitations) one to associate the BPMN shapes and symbols of Figure 3 to their respective concept in the paradigm.

GME provides a major advantage which is the "modeling paradigms". It supports the reuse and maintenance of models (any change in a type automatically propagates down the type hierarchy). It is possible to create our own transformation, but at the cost of heavy C++ programming.

3.3 MetaEdit +

MetaEdit+ is a completely integrated environment/Meta-CASE tool developed in Jyväskylä University, as part of the *MetaPHOR* project [18] for building and using Domain-Specific Modeling (DSM) solutions. *MetaEdit+* provides the standard set of CASE tool functionality, including graphical editors, design data management, and integration with other tools via its API.

As *GME*, *MetaEdit+* is based on a proprietary metamodeling language which is the GOPPRR metamodeling language [19]. GOPPRR is an acronym formed from language's base types which are Graph, Object, Port, Property, Relationship and Role. Graph is the top-level structure of the metamodel. It defines one language or diagram technique such as Class Diagram or State Transition Diagram. The actual semantics of the graph are defined as the bindings of objects, relationships, roles and ports within the graph. Properties are characterizing attributes that can be attached to each of these other types [20].

A graph (similar to the "Model" concept in *GME*) denotes an aggregate concept which contains a set of objects and their relationships, with specific roles. An example of a graph in our evaluation context is a whole Business Process Diagram (as a whole or just one level of it). In use, the Graph concept is fundamentally a generalized decomposition graph: it can be included in a parent graph, attached to an object, role or relationship therein [19].

The modeling tool building process in *MetaEdit+* is quite simple. First, we design the modeling language and its concrete syntax with *MetaEdit+ Workbench* and then we use the produced editor in *MetaEdit+ Modeler* (figure 9).

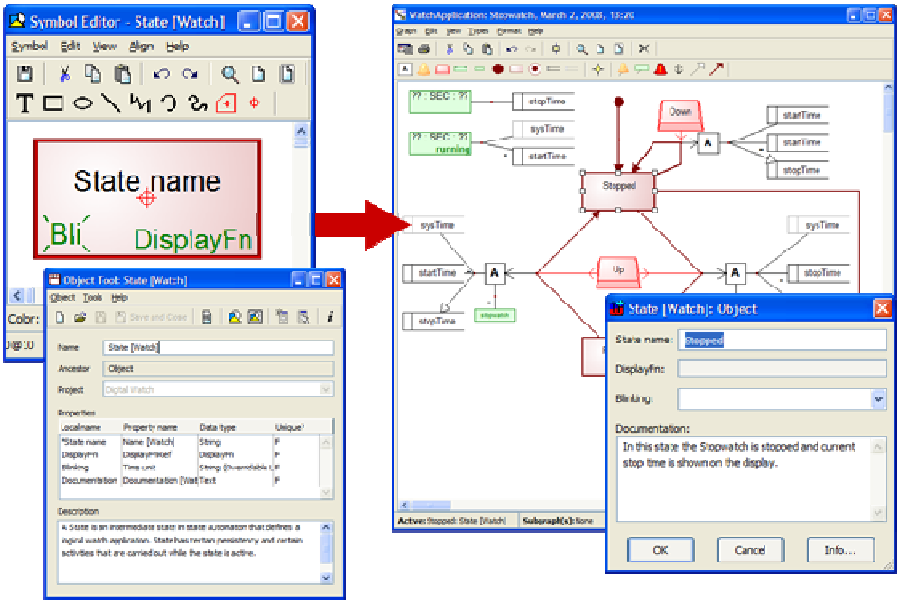

Fig. 8. *MetaEdit+* environment: *MetaEdit+ Workbench* (left) and *MetaEdit+ Modeler* (right) - Copyright © 2011 *MetaCase*

MetaEdit+ Workbench is a tool for designing modeling languages: their concepts, rules, graphical notations and generators. The language definition is stored as a metamodel in the MetaEdit+ repository.

MetaEdit+ Modeler follows the definition of data modeling language defined previously in *MetaEdit+* Workbench by extracting it from the repository and offers automatically, full functionalities of modeling tools: diagram editors, viewers, generators, multi-user support, etc...

To create our BPMN editor with *MetaEdit+*, we defined a Project with a Business Process Diagram as a MetaEdit graph, we added all concepts of our BPMN metamodel as MetaEdit's Objects, relationships and properties (names, types…) and then we proceeded to bind Objects and relationships. Finally, we defined the graphical notations of each language elements, with the symbols editor which is a user-friendly drawer of SVG (Scalable Vector Graphics).

Once the Project is created and registered in *MetaEdit+* repository as a metamodel it can be used as an editor, as shown in Figure 10. The tool provides many features like loading/saving diagrams to a custom XML, undo/redo, printing, diagrams export to bitmap, GIF, PNG and PICT format, import/export of graphical representations from/to SVG, zooming, property views, models explorer, editing tools for creating and modifying new types based on the base type, Symbol Editor tool for drawing graphical symbols for objects, relationships and roles, constraints and rules definer tool, it offers also to choose between three different representations (diagram, table, matrix) without reloading or regeneration. We have succeeded in generating a BPMN editor in less than a half day which allowed us to evaluate the maturity of this tool and its user-friendliness. However, we have found a minor limitation in the produced editor such as the difficulty to move labels, or to rename graphical elements without

using the property popup menu… but this is due to the nature of the graphical representations that are usually vector graphics (SVG).

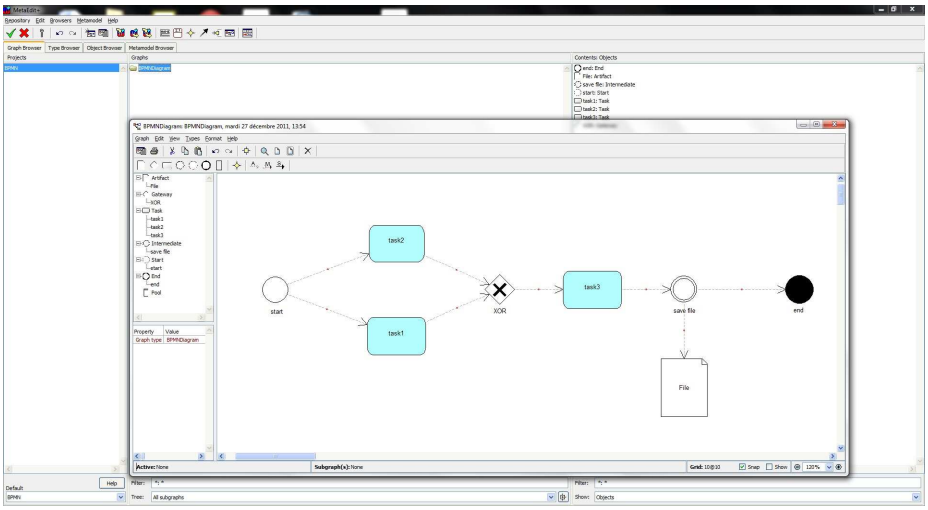

Fig. 9. The BPMN Editor Produced using *MetaEdit+*

MetaEdit+ offers an advanced advantage which is GOPPRR metamodeling language. The goal of such languages is to support the reuse and maintenance of models: any change in a conceptual Graph (semantic model) is propagated between different representational Graphs (graphical representations model), and both types and instances of object, relationship, role, property and graph can be reused within other graphs or projects.

3.4 Obeo Designer

Obeo Designer is an adaptive tool led by points of view. It provides a setting environment to configure viewpoints and their various representations. It is based on the Eclipse Modeling technological base from which it takes part in these expandability capacities and modularity. It is based on the frameworks EMF, GEF and GMF that offer all the elements to build modelers [7].

Obeo Designer (Version 5.0) allows architects to create the graphical modeling workbenches that support their own language, notation, process and technical target. It provides a tooling to easily define graphical representations such as diagrams, tables or trees with rich user interactions hiding the complexity. In *Obeo Designer*, an editor is described in three principle steps:

1. Domain vocabulary definition: It consists to define domain concepts, relationships, and properties by creating a metamodel in ecore, *Obeo Designer* offers an advanced graphical editor for this aim. After the DSL definition, the architect uses ecore to generate the metamodel implementation and releases it.

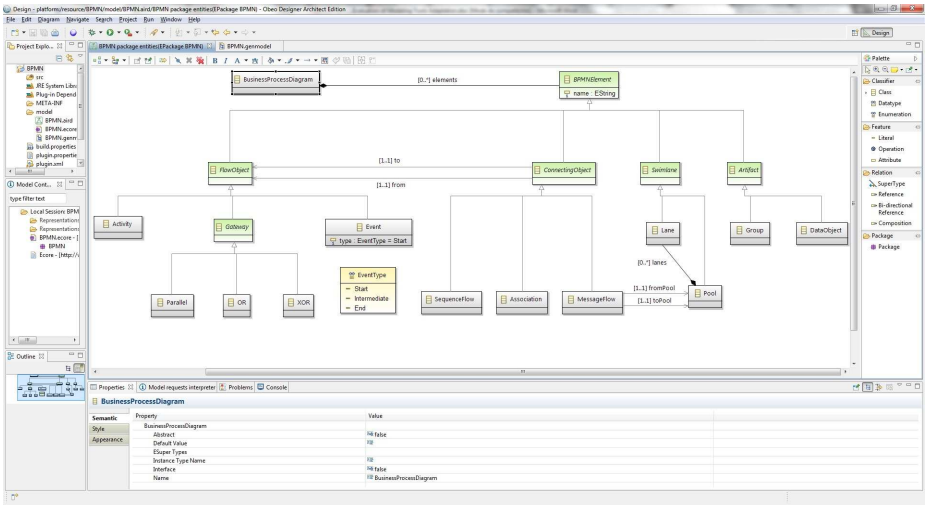

Fig. 10. BPMN metamodel in ecore using *Obeo Designer*

Designer description: the aim of this description is to define viewpoints, their representations and graphical elements' parameters for each one of these representations.

The notion of viewpoints is an abstraction that provides a specification of a system, limited to a particular set of problems. It was introduced in the specification IEEE 1471 [6]. In *Obeo Designer*, viewpoints were used to provide users a set of visual representations focused on a particular concern (figure 7). A point of view provides a set of representations. This concept defines a projection used to view or edit a set of semantic concepts. In *Obeo Designer*, a representation may be presented as a diagram, a table or tree. The same concept is used in the RSA, known as "Layers" and in GME under the name of "Aspect".

2. Semantic model creation and visualization: After the description of our editor has been done, we can create a semantic model and visualize its graphical representation as diagrams, tables, trees...etc.

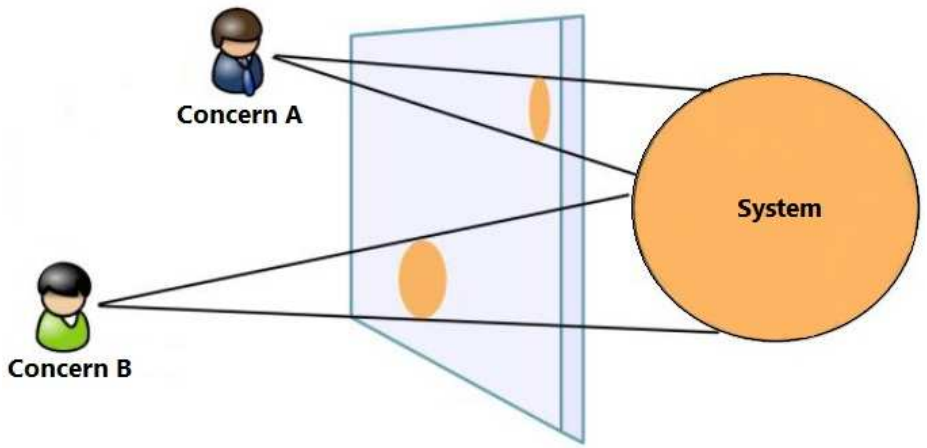

Fig. 11. Viewpoints notion

3. Graphical representations description: this is the last step of the editor building process. Obeo Designer offers a tree-styled editor to describe concrete syntax (figure 12). In this editor we can create many diagrams, trees or tables for the same DSL. The configuration starts with the definition of graphical elements, with custom styles (figures, colors, size …) and the binding with the abstract syntax (semantic part).

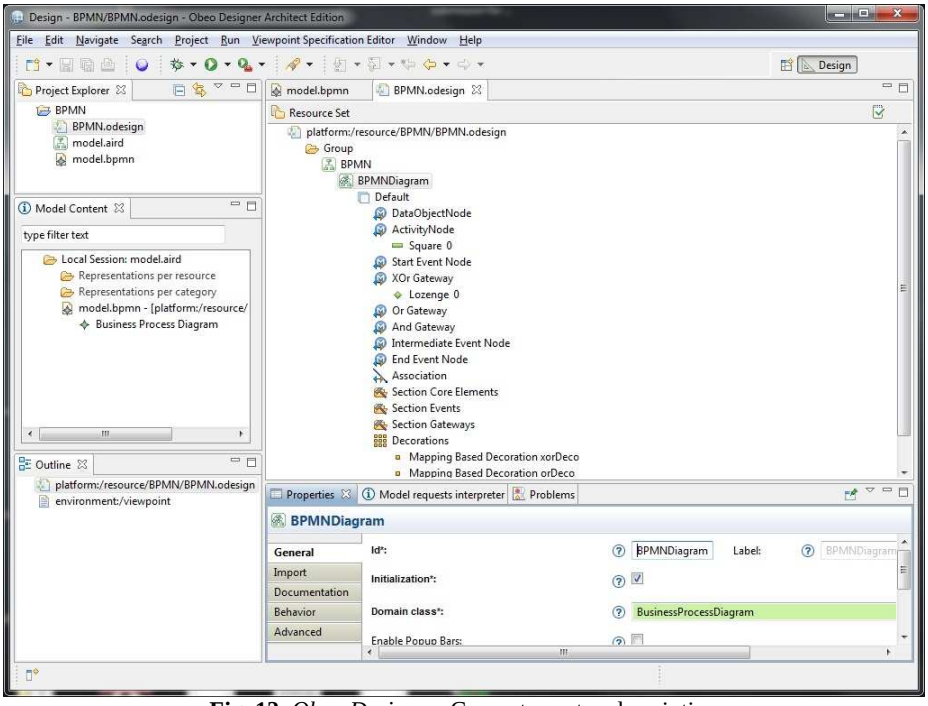

Fig. 12. *Obeo Designer*: Concrete syntax description

In this step, we can make specific validators, model comparators, define tools like palette and layers for filtering the graphical editor elements and we can also create custom code generators using Acceleo.

Once these steps were performed for our simplified BPMN metamodel, the designer can be used in Obeo Designer as a BPMN editor, as shown in Figure 9.

The BPMN Editor implemented with Obeo Designer get some functionalities, including Undo/Redo, outline viewer, image export, shortcuts, links routing, diagram elements Show/hide, XMI export, domain classes Auto-completion, Syntax highlighting, direct name edit in diagram, diagram validation, real-time request, zooming, printing, Customizable behavior of creation, deletion, drag and drop…etc.

However, there are some required improvements in the editor's designer; the editor's designer offers advanced functionalities to add parameters and OCL expressions in the editor's graphical syntax, which is not user-friendly for simple users. There are also some bugs in the designer, which we have reported to Obeo's developers.

As RSA, Obeo Designer is based on Eclipse Platform IDE; it inherits also, all Eclipse customization capabilities via its API.

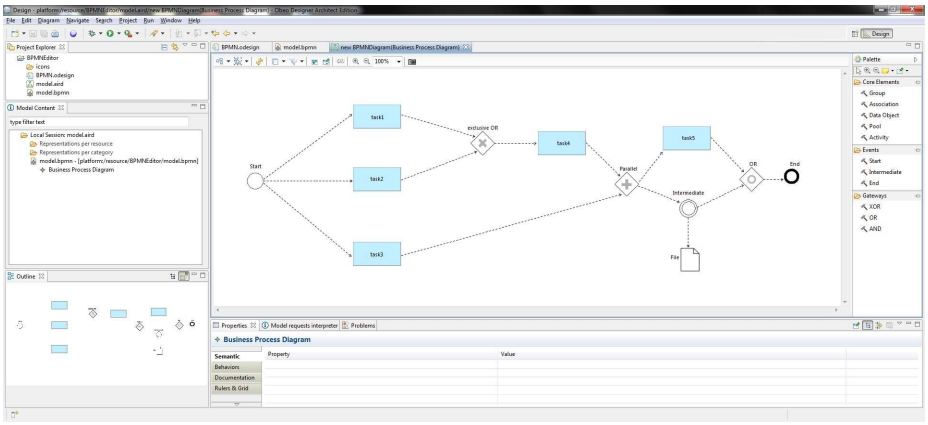

Fig. 13. BPMN Editor produced by *Obeo Designer*

Obeo Designer did well in the graphical editors' customization. However, we have found several limitations for the graphical syntax support:

The editor's designer doesn't allow using the full range of visual variables [9]; it's not possible to create complex graphical shapes for nodes (e.g. compartments, complex polygons, 3D shapes …), there are also some lacks in basic shapes; we can only use six principal shapes (square, lozenge, ellipse, triangle, dot, and ring), besides iconic representations (images). However, the tool provides a mechanism to create multiple conditional graphical styles for the same node. For edges, it is possible to create custom link styles, but we can only choose one between eleven arrow decorations.

Obeo designer offers a good interactivity during the development of the domain model and of the editor description. Additionally, the editor's building process is incremental and there are no need to generate code upon modifications, the advanced transformation engine (in the same context that the Acceleo engine) interprets modifications, and apply them at runtime in the produced editor, which accelerates the development and the handling of editors.

3.5 Eclipse GMF

Eclipse is an open source and extensible Java-based platform that provides many useful services for the creation of textual and graphical editors. Indigo M7 release (Version 3.7) modeling pack was used for our evaluation. For building graphical editors, three others Eclipse plug-ins are especially relevant. The *Graphical modeling Framework* (GMF) is a framework for developing domain specific languages. GMF is built on the Eclipse Modeling Framework (EMF) [22] and the Graphical Editing Framework (GEF) [23].

The *Eclipse Modeling Framework* (EMF) is a framework and code generation facility for building tools and other applications based on a structured data model. From a metamodel specification described as an XML Schema or as a class diagram

in Papyrus [24] (such as the one in Figure 2), EMF provides tools and runtime support to produce a set of Java classes for the metamodel, a set of adapter classes that enable viewing and command-based editing of the model, and a basic tree editor. In the context of GMF, EMF is used to define the metamodel or abstract syntax of languages (expressed in Ecore), and for generating code for creating, editing, and accessing models.

The *Graphical Editing Framework (*GEF) is a framework that allows developers to take an existing application model and quickly create a rich graphical editor for it. It can easily be hooked to EMF metamodels. GEF is a framework that supports the development of graphical editors. In the context of GMF, GEF is used to implement the concrete graphical syntax of languages and for editing of concrete syntax.

To clarify the shaded ideas on GMF and its related projects (runtime, tooling and notation) is held to detail the meaning of each. The GMF project is composed of three interconnected subprojects [40]:

1. GMF Runtime: The GMF Runtime project is an industry proven application framework for creating graphical editors using EMF and GEF. The GMF Runtime provides many features that one would have to code by hand if using EMF and GMF directly.
   - A set of reusable components for graphical editors, such as printing, image export, actions and toolbars and much more.
   - A standardized model to describe diagram elements, which separates between the semantic (domain) and notation (diagram) elements.
   - A command infrastructure that bridges the different command frameworks used by EMF and GEF.
   - An extensible framework that allows graphical editors to be open and extendible.
2. GMF Tooling: The GMF Tooling project provides a model-driven approach to generating graphical editors in Eclipse. By defining a tooling, graphical and mapping model definition, one can generate a fully functional graphical editor based on the GMF Runtime.
3. GMF Notation: The GMF Notation Project provides a standard EMF notational metamodel. The notational metamodel is a standard means for persisting diagram information separately from the domain model. It was based on the principles in the OMG Diagram Interchange Specification [41]

So the creation of a graphical editor with GMF is done with the tools of GMF tooling in an execution environment that is GMF runtime and the diagrams produced with this editor will be persisted according the Diagram Interchange standard defined on GMF notation.

Based on the separation of concerns paradigm, GMF is composed of four basic models for the generation of the graphical editor: Domain model (Semantic model based on EMF), Graphical definition model (based on GEF), Tooling definition model which define tools used in the editor (palettes, contextual menus…) and the Mapping model that make the binding of semantics and graphical representations as shown in the figure 15.

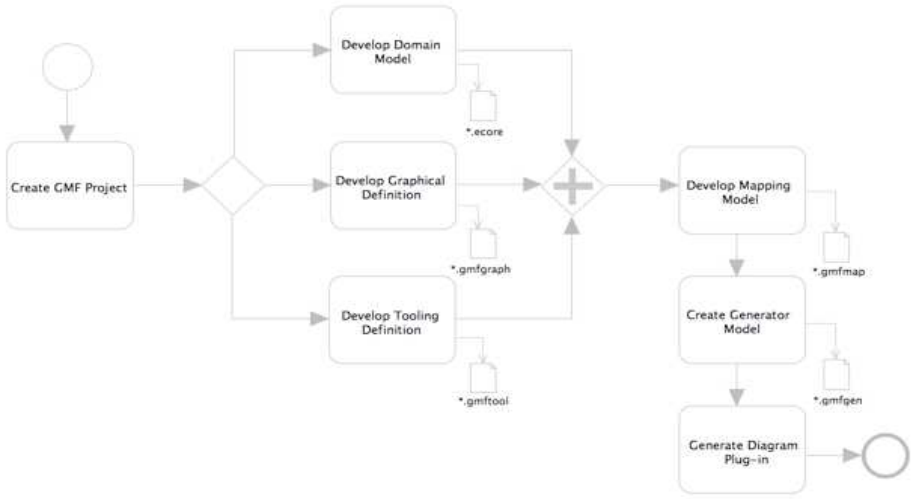

Fig. 14. *GMF*: Editors building process - Copyright 2006 *Borland.com*

Much effort is required to learn GMF and to understand how to make an editor with it. Documentation (including tutorials and books) and useful discussion forums are however available [25]. The quality of the resulting editor is very high, especially from a usability point of view. The Eclipse platform, with GMF, offers several useful services that can be used with little effort: loading/saving (in XMI), zooming, tool palettes, overviews, exporting to images, offering extension points for other applications to access the models created, and multi-platform support. However, much programming effort is required to implement the various shapes and connectors, multiple undo/redo, label editing, and property sheets.

Therefore, and also to homogenize their tooling, SAP built a framework that hides GMF's complexities from the developer to ease and speed up the development of graphical editors. This framework was called the Graphical Tooling Infrastructure (Graphiti) [42]. After SAP AG decided in 2009 to donate the framework to the eclipse community and then the framework was in the 0.7.0 incubation release in October, 2010.

We were unable to evaluate this framework in this paper for space reasons, but we summarize basic differences [43] between Graphiti and GMF in the following table.

Table 1. Graphiti vs. GMF

	Graphiti	***GMF***
Architecture	Runtime Oriented	Generative
API	Self-contained (independent)	GEF-Dependent
Client Logic	Centralized	distributed functionality
Look & Feel	Sophisticated look defined by SAP usability specialists (highly customizable to the requirements of the tool)	Simple (highly customizable in the generated code)

As paramount differences between the architectures of the frameworks, we observe the two major advantages of Graphiti:

1. Graphiti is strictly focused on the API; a user of the Framework only needs to know how to use the Framework EMF. To build an editor, no knowledge of Draw2D-GEF (or any other graphical framework used) is required.
2. We observe that GMF follows a generative approach, while in Graphiti no generated source code must be manipulated to adapt the editor – in contrast to GMF where one has to change generated sources, which can cause incompatibilities problems when regenerating.

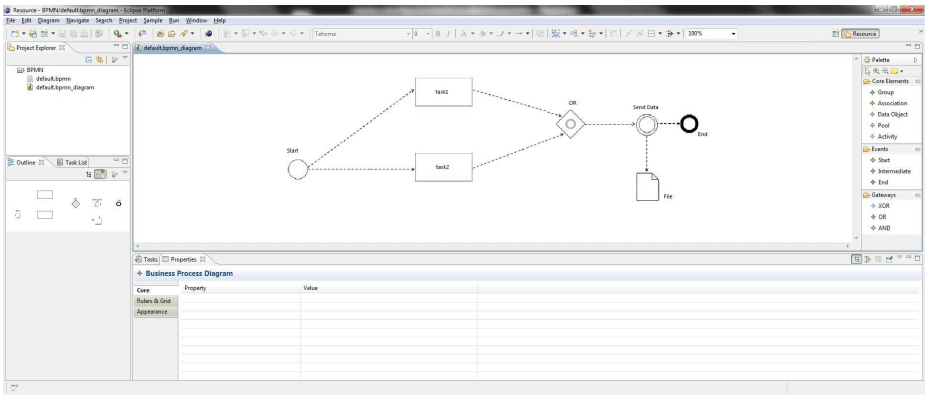

Fig. 15. BPMN Editor generated by *GMF*

Once a basic editor is in place, adding new functionalities becomes efficient. Also, adapting the editor to changes in the metamodel is fairly simple. If new attributes, class, or associations are added to the metamodel, then the editor can still open files created with the previous version. However, deleting or renaming classes or attributes can lead to backward incompatibility problems. Finally, it is important to note that such a plug-in enables the integration of the editor with other modeling and programming tools offered for the Eclipse platforms.

4 Comparison Summary

The current section provides a brief summary with additional highlights and remarks based on our experience with these tools. We would like to point out that all tools used in this evaluation are used on a large scale and have notoriety in the MDE community; each one of these tools provides innovative concepts and methodologies in this field.

– *Customization level*: Obeo Designer and MetaEdit + are the tools that allow customizing their critical parts easily and thoroughly (semantics, behaviors, visual notation, validation tools, editor's pallete), without generating code or reload the model. Actually the customization is done in a dynamic way for the tools following metamodelling paradigms (GME and its notion of Aspect). RSA is doing well for this criterion, it provides a graphical editor to customize the tool, but in terms of ability to adapt the graphic representation, it remains rather low. GME is clearly the weakest tool in terms of tool customization; it doesn’t

support any alternative way to represent concepts besides iconic representation (images), which is discriminating for it on this category. In terms of user interface (UI) customization, Eclipse-based tools are clearly the most advanced tools in this area, their API allows them to create Perspectives, menus, views, property sheets... except that until the moment it is done programmatically, and there's no mechanisms to automate this adaptation (adapt the tool according to the need for the user, the environment and platform) or to restrict it as needed, thus offers opportunities to future researches.

- *Graphical expressiveness and completeness*: According to [9], the visual expressiveness is defined as the number of visual variables used in a notation. The visual variables are: shape, texture, brightness, size, color and the planar positions (horizontal and vertical). The graphical completeness is defined by the capability to use fully the shape variable (the use of any kind of shapes: complex, composites, 2D/3D…). We evaluated the tools' capacity to use the full range of these variables; MetaEdit+ and GMF are the only two tools that allowed reproducing the BPM notation with fidelity and flexibility. Obeo Designer did well in general, except for a few limitations. GMF required substantial additional programming. GME and RSA offered the least flexibility for this criterion.
- *Tool openness*: The tool openness includes extensibility, reusability and maintainability.
    - Tool building approaches: MetaEdit+, GME and RSA are metamodeling-language-based tools, they force users to describe semantic concepts in particular languages: proprietary languages for MetaEdit+ and GME (GOPPRR, GME modeling paradigm), and a standard language for RSA (UML). For graphical description, MetaEdit+ and GME don't allow the separation of concerns, so the user defines the graphical particularities in the proprietary language, and then registers the editor in the case of MetaEdit+ and GME or generate code for RSA. Obeo Designer and GMF are model-based tools, they propose to describe the semantics and graphics as separated models (which is well for reusability) and then users can generate code for GMF, or interpret the model in the case of Obeo Designer.
    - All evaluated tools propose extensibility. Eclipse-based tools offer extensibility through the mechanism of extension points. GME with its MS COM interfaces is able to be extensible with other tools and languages. MetaEdit+ provides the interface to read, create and update model elements via its SOAP/Web Services API, making MetaEdit+ functions accessible from almost any programming language. However, extensibility appears to be weak with all commercial tools on their Editor's generation engines which are the most isolated parts.
    - In term of reusability, MetaEdit+ and GME have a great potential, respectively with the GOPRR language and the GME metamodeling paradigm, being designed for this aim; they are among the best tools which promote reusability. All tools supporting separation of concerns are doing well for this criterion. GMF and Obeo designer offer MDE approaches for editors' design, so they provide possibility to reuse existing models. However, there

is till now, no solution that provides settings inheritance or factorization.

- *Tool usability*: The best usability is offered by far by the Obeo Designer editor in terms of efficiency, accessibility, satisfaction and overall number of features. All tools support multiple undo/redo and loading/saving of models. The manipulation of elements is somewhat awkward in RSA and GME.
- *Required resources*: All these tools require some effort/time for learning the technology and for creating our use case editor. The editor creation and usage mechanism in MetaEdit+ is likely the easiest one among the five studied here, followed by Obeo Designer and GME. GMF followed by RSA are definitely the worsts.
- *License Nature:* for this evaluation we have chosen three commercial tools, a free tool and an open source one. We can distinguish that commercial tools have an advanced degree of maturity comparing to the others. For commercial tools MetaEdit+ and Obeo Designer propose the best quality/price ratio.
- *Artefact characteristics:* criteria for artefacts produced by the tools
    - *Analysis and transformation*: MetaEdit+ provides models' access interface via its SOAP/Web Services API. GME offers an interface (Ms COM) to access and transform models. Eclipse environment provides Java interfaces to easily access models, but transformations are manual. Obeo designer is probably the most promising environment in this category, with a specific language for transformation (Acceleo), which is based on the OMG MTL standard [26]. It provides also capabilities to produce editors' validators where models are checked against the OCL constraints. However, such capabilities appear to be weak in RSA.
    - *Artefact quality level:* all evaluated tools provide overall excellent metamodeling capabilities that enable metadevelopers produce level 5 quality metamodels. However, GME has some limitations in the concrete syntax, for this reason the metamodels products by its editors has a level 3.
    - *Artefact format:* MetaEdit+ is based on a model repository to save editors, this repository has a proprietary format, but the tool provides a web service interface for all external manipulation on it. As MetaEdit+, GME has a particular format of models persistence, the models produced by the GME editors are persisted on binary files. All Eclipse-based tools, have an open format of files which is based on XMI standard, in the case of RSA, there are some particular files which are binary and other that depend to UML syntax.

The following table provides a quick overview of the strengths and weaknesses of each tool, proved by metrics presented in section 2.

Table 2. Comparison overview

	RSA	*GME*	*MetaEdit+*	*Obeo Designer*	*Eclipse GMF*
Customization level	*38 %*	*35 %*	*51 %*	*57 %*	*70 %*
Graphical expressiveness	*8/8*	*2/8*	*5/8*	*8/8*	*8/8*
Graphical completeness	*2/5*	*1/5*	*5/5*	*3/5*	*4/5*

Openness	*Settings approach**		*Std language: UML*	*Proprietary language: MModeling paradigm*	*Proprietary language: GOPPRR*	*Open Model: EMF*	*Open Model: EMF*
	Extensibility		*Yes*	*No*	*No*	*Yes*	*Yes*
	Reusability		*No*	*No*	*Yes*	*No*	*No*
Usability	*Efficiency (EE)*		*30 %*	*37 %*	*30 %*	*75 %*	*60 %*
	Accessibility (TV & VC)	*Tasks visibility*	*56 %*	*43 %*	*42 %*	*81 %*	*81 %*
		visual coherence	*38 %*	*31 %*	*59 %*	*75 %*	*41 %*
Required resource (man-day)			*12*	*6*	*0.5*	*5*	*25*
License Nature*			*Cial.*	*freeware*	*Cial.*	*Cial.*	*EPL*
Artefact characteristics	*Analysis capability*		*Yes*	*No***	*Yes*	*Yes*	*Yes*
	Artefact quality level		*4/5*	*3/5*	*5/5*	*5/5*	*5/5*
	Artefact Format		*XML*	*Binary*	*Repository*	*XML*	*XML*

* Std.: Standard, MModeling: metamodeling, Cial.: Commercial, EPL: Eclipse Public License
**GME allows models transformation and analysis but at cost of heavy C++ programming

5 Conclusion

The quality of modeling tools has been subject of some research by now. An Evaluation of domain specific language (DSL) solutions according different stakeholders can be found in [1]. Both [37] and [38] compare domain specific language tools to each others, but there are no empirical studies with measurable metrics to evaluate such tools. Based on a literature analysis and our experience with developing modeling environments in research projects (Papyrus Project [24], Gaspard [39]…) we have identified several evaluation criteria and metrics. We tried in this paper to compare five different tools, producing graphical modeling editors. Editors were created with each tool, and our experiments helped us compare the approaches against such metrics. These criteria and metrics allowed us to assess productivity, usability, reliability, reuse and stability through a real case study involving a simple subset of the Business Process Modeling Notation (BPMN) whose abstract syntax is specified with a metamodel.

Given the limitations found in the different tools which we have evaluated and others that we haven't discussed in this paper for space reasons, it turned out that there are many research perspectives for this work, including tools' customization using the Model-Driven approach. *IBM RSA* has made great progress on this issue: it provides a graphical editor to customize tools; but the final graphical rendering remains poor. *MetaEdit+* and *Obeo Designer* have also unrivaled potentials compared to other tools, they provide user-friendly environments for building editors without difficulties and in a shorter time, but there's always limitations in the concrete syntax definition language which leads us to propose solutions on Papyrus modeling tool, benefiting from all evaluated tools advantages and reducing their drawbacks. This will be reflected by the specification of some metamodels which describe all parts of a modeling tool (UI, abstract and concrete syntaxes…), approaches to compose and reuse models and a metamodel representing the methodology that describes the process of tools usage to restrict the environment according to user needs.

References

1. Mohagheghi, P. and Haugen, Ø.: Evaluating Domain-Specific Modelling Solutions, Advances in Conceptual Modeling – Applications and Challenges, Lecture Notes in Computer Science, 2010, Volume 6413, , Pages 212-221
2. Collins English Dictionary, 2011.
3. Lazovik, A. and Ludwig, H.: Managing Process Customizability and Customization: Model, Language and Process, Lecture Notes in Computer Science, 2007, Volume 4831, Web Information Systems Engineering – WISE 2007, Pages 373-384
4. OMG, 2011. Business Process Modeling Notation (BPMN), version 2.0. http://www.omg.org/spec/BPMN/2.0/
5. OMG, 2010. UML Profile for BPMN Processes RFP. http://www.omg.org/cgi-bin/doc?ab/10-06-01.pdf
6. IEEE 1471: http://en.wikipedia.org/wiki/IEEE_1471
7. Juliot, E. and Benois, J.: How to build Eclipse DSM without being an expert developer?, Obeo Designer Whitepaper
8. Kennedy, K., Koelbel, C. and Schreiber, R.: Defining and Measuring the Productivity of Programming Languages. International Journal of High Performance Computing Applications 18(4), 441–448 (2004)
9. Daniel L. Moody: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Trans. Software Eng. 35(6): 756-779 (2009)
10. Wayne Diu : Custom Domain Modeling with UML Profiles «The Basics of Generating Tooling for Elements from a UML Profile», IBM Rational Software 2009
11. Gong, M., Scott, L., Xiao Y., Offen, R.: A rapid development model for meta-CASE tool design, Conceptual Modeling — ER '97, ISBN 978-3-540-63699-1
12. Nuseibeh, B.: Meta-CASE support for method-based software development. Proc. of 1st Int. Congress on Meta-CASE, Sunderland, UK, January (1995)
13. J. Davis, "GME: the generic modeling environment" OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2003
14. Institute for Software Integrated Systems: The Generic Modeling Environment (GME), December 2011. http://www.isis.vanderbilt.edu/Projects/gme/
15. Seffah, A., Donyaee, M., Kline, R.B., Padda, H. K.: Usability Measurement and Metrics: a Consolidated Model. Software Quality Journal 14, 159–178 (2006)
16. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The Generic Modeling Environment, Workshop on Intelligent Signal Processing 2001
17. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Metamodelling - State of the Art and Research Challenges. Model-Based Engineering of Embedded Real-Time Systems 2007: 57-76
18. MetaPHOR project, Jyväskylä University: http://metaphor.it.jyu.fi/metapubs.html
19. Kelly, S., Lyytinen, K., and Rossi, M., "MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment", Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science 1080, Springer-Verlag, pp. 1–21, 1996.
20. Pohjonen R.: Metamodeling Made Easy – MetaEdit+ (Tool Demonstration), Lecture Notes in Computer Science, 2005, Volume 3676, Generative Programming and Component Engineering, Pages 442-446

21. Stephen A. White, Introduction to BPMN-http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf
22. Eclipse Modeling Framework (EMF), http://www.eclipse.org/emf/
23. Graphical Editing Framework (GEF), http://www.eclipse.org/gef/
24. Papyrus - Eclipse: http://www.eclipse.org/papyrus/
25. Eclipse Community Forums: GMF (Graphical Modeling Framework): http://www.eclipse.org/forums/eclipse.modeling.gmf
26. OMG, 2008. MOF Model To Text Transformation Language (MOFM2T), 1.0 http://www.omg.org/spec/MOFM2T/1.0/
27. IBM Rational Software Architect (RSA), http://www.ibm.com/developerworks/downloads/r/architect/
28. J. Bertin, Semiology of Graphics: Diagrams, Networks, Maps. University of Wisconsin Press, 1983.
29. A. J. Albrecht, “Measuring Application Development Productivity,” Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM Corporation (1979), pp. 83–92.
30. Clark, T., Evans, A., Sammut, P. and Willans, J.: Applied Metamodelling: A foundation for Language Driven Development. Version 0.1. Xactium Ltd., 2004.
31. Penichet, V., Calero, C., Lozano, M. and Piattini, M.: using WQM For Classifying Usability Metrics. . ISBN: 972-8924-19-4. 2006
32. Constantine, L.L. and Lockwood, L.A.D. 1999. Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design, New York: Addison-Wesley.
33. Braz, C.; Seffah, A and M’Raihi, D. (2007) “Designing a trade-off between usability and security: A metrics-based model,”Proceedings of the11th IFIP TC 13 Conference on HumanComputer Interaction, Rio de Janeiro, Brazil, Lecture Notes in Computer Science, Vol. 4663. Springer, Berlin, pp. 114–126, September 2007
34. Bevan, N. and Macleod, M. 1994. Usability measurement in context, Behavior and Information Technology 13: 132–145.
35. Seffah, A., Kececi, N., Donyaee, M.: QUIM: A Framework for Quantifying Usability Metrics in Software Quality Models - APAQS '01 Proceedings of the Second Asia-Pacific Conference on Quality Software IEEE Computer Society Washington, DC, USA 2001
36. Software engineering -- IFPUG 4.1 Unadjusted functional size measurement method: ISO/IEC 20926:2003
37. Amyot, D., Farah, H., Roy, J.-F.: Evaluation of Development Tools for Domain-Specific Modeling Languages. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 183–197. Springer, Heidelberg (2006).
38. Pelechano, V., Albert, M., Mu˜noz, J., Cetina, C.: Building tools for model driven development. Comparing microsoft dsl tools and eclipse modeling plugins. In: Proc. of the Actas del Taller sobre Desarrollo de Software Dirigido por Modelos. MDA y Aplicaciones. CEUR Workshop Proceedings, vol. 227 (2007).
39. Gaspard2, http://www.gaspard2.org/
40. Graphical Modeling Project (GMP), http://www.eclipse.org/modeling/gmp/
41. OMG, 2004. Diagram Interchange Specification, v1.0 http://www.omg.org/cgi-bin/doc?formal/06-04-04
42. Eclipse, SAP AG, 2011. Graphiti, http://www.eclipse.org/graphiti/
43. Brand, C., Gorning, M., Kaiser, T., Pasch, J. and Wenz, M.: Graphiti : Development of High-Quality Graphical Model Editor - Eclipse Magazine
44. S. Cook et al, Domain-Specific Development with Visual Studio DSL Tools, Microsoft .net development series, 2007.

45.Robert, S.; Gerard, S.; Terrier, F.; Lagarde, F.; "A Lightweight Approach for Domain-Specific Modeling Languages Design," Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on , vol., no., pp.155-161, 27-29 Aug. 2009.

46.Gérard, S.,Dumoulin, C.,Tessier, P.,Selic, B.: Papyrus: A UML2 Tool for Domain-Specific Language Modeling - Model-Based Engineering of Embedded Real-Time Systems, Lecture Notes in Computer Science Volume 6100 (2011) P. 361-368