Evaluation of Development Tools for Domain-Specific Modeling Languages

Daniel Amyot, Hanna Farah, and Jean-François Roy

SITE, University of Ottawa, Ottawa, Canada
{damyot, hfarah, jroy}@site.uottawa.ca

Abstract. Creating and maintaining tools for domain-specific modeling languages (DSML) demands time and efforts that often discourage potential developers. However, several tools are now available that promise to accelerate the development of DSML environments. In this paper, we evaluate five such tools (GME, Tau G2, RSA, XMF-Mosaic, and Eclipse with GEF and EMF) by observing how well they can be used to create graphical editors for the Goal-oriented Requirement Language (GRL), for which a simplified metamodel is provided. We discuss the evaluation criteria, results, and lessons learned during the creation of GRL editors with these technologies.

1 Introduction

Domain-specific modeling languages (DSML) are high-level languages specific to a particular application or set of tasks. They are closer to the problem domain and concepts than general-purpose programming languages such as Java or modeling languages such as UML. Many companies have such languages developed in-house to satisfy some of their specific modeling, scripting, or testing needs. Improvements in productivity and comprehensibility are often cited as benefits. Still, supporting a development environment for DSML with compilers, (graphical) editors, translators, debuggers and other such tools is often onerous and prevents the rapid adoption and use of DSML.

In the past decade, a strong interest in model-driven engineering has resulted in various theories and technologies that support easier and faster development of DSML environments. The purpose of this paper is to evaluate some of these tool-supported technologies, namely the *Generic Modeling Environment* (GME), *Xactium's XMF-Mosaic*, the combination of the *Eclipse Modeling Framework* (EMF) with the *Graphical Editing Framework* (GEF), and the UML profiling capabilities of *Telelogic Tau G2* and of *Rational Software Architect* (RSA). The general context is one where we want to develop a graphical editor for an evolving graphical modeling language defined by a metamodel. A common case study, based on a simplified version of the Goal-oriented Requirement Language (GRL), is used to assess the maturity of these technologies.

This paper is structured as follows. Section 2 describes our case study and evaluation criteria. Each of the five tools is used in Section 3 to develop simple GRL editors (with lessons learned), and then section 4 summarizes their main strengths and weaknesses. We present our conclusions in section 5.

R. Gotzhein and R. Reed (Eds.): SAM 2006, LNCS 4320, pp. 183–197, 2006.

2 Evaluation Context

Our context is one where we are interested in approaches that can help develop new DSML such as those found in ITU-T and OMG, together with early prototypes for modeling environments. Accordingly, a representative metamodel for such a language and a set of evaluation criteria are suggested to enable comparisons between the vari-ous approaches.

2.1 Simplified GRL Metamodel

Part of the proposal for ITU-T's User Requirements Notation [9], the Goal-Oriented Requirement Language (GRL) is used to specify and reason about business or system goals, alternative means of achieving goals, and the rationale for goals and alternatives. The notation is applicable to non-functional as well as functional requirements. GRL has concepts for various intentional elements including goals, softgoals, tasks, and beliefs. Various types of contributions link these elements into AND-OR graphs used to evaluate strategies that best balance the (often conflicting) goals stakeholders have in a system.

For the purpose of our evaluation, we have created a simple metamodel that includes a subset of the language concepts (Figure 1). The classes and associations were structured to cover the most interesting element notations (named nodes, links between nodes, links attached to links) and situations commonly

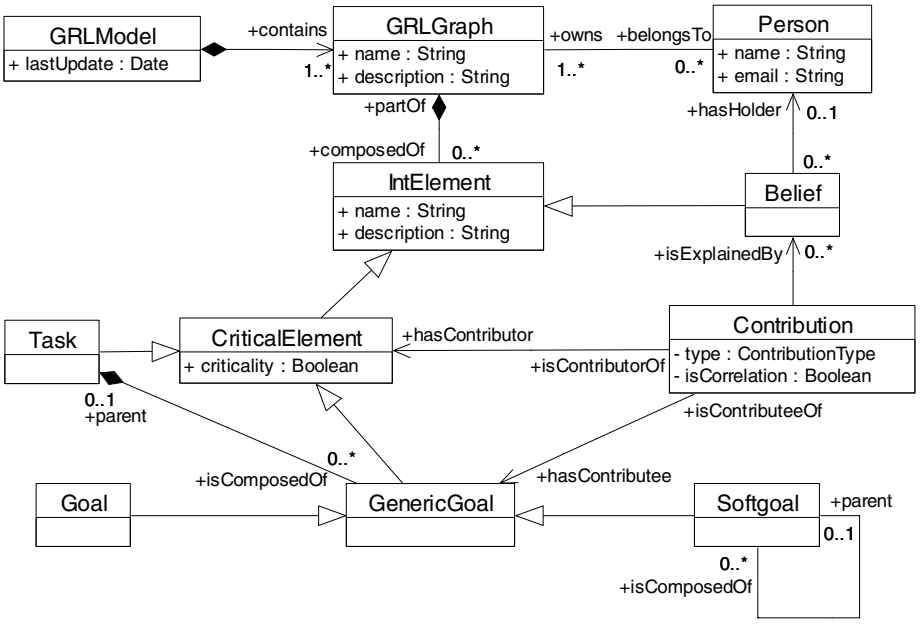
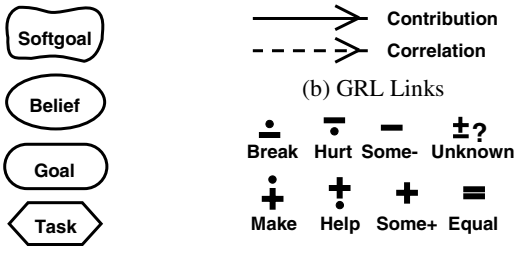


Fig. 1. Simplified GRL metamodel

found in metamodels (e.g., associations, generalizations, aggregations, typed attributes, different multiplicities, and navigation). This metamodel is not meant to be a realistic representation of GRL (this is outside the scope of this study). A more complete discussion of the GRL elements and semantics can be found in [1,15].

In terms of syntactical notation elements in the graphical representation, the symbols corresponding to the metamodel in Figure 1 are summarized in Figure 2.



(a) GRL Elements (b) GRL Links (c) GRL Contributions Types

Fig. 2. Graphical symbols for the selected GRL subset

2.2 Evaluation Criteria

Our study puts a particular emphasis on the following evaluation criteria, which are most relevant in our context:

- *Graphical completeness*: Can we represent all the notation elements?
- *Editor usability*: Does the editor generated support undo/redo, load/save, simple manipulation of notation elements and properties, etc.?
- *Effort*: How much time and effort is required to learn the approach and produce DSML tools?
- *Language evolution*: How are older models handled when the language or metamodel evolves?
- *Integration with other languages*: How can we support additional languages (e.g., Use Case Maps in combination with GRL) or integrate with other tools?
- *Analysis capabilities*: Can we easily analyze or transform models produced with the graphical editor?

3 Evaluation of DSML Development Tools

In this section, we study five tools that support the development of DSML environments. Our selection is based on the relative popularity or technical potential of the tools, but many other tools could be studied as well (the DSM Forum discusses some of them [3], including the well-known MetaCase+ [12]).

3.1 Generic Modeling Environment (GME)

The Generic Modeling Environment is a configurable framework developed at Vanderbilt University and used to create domain-specific modeling environments [8]. Version 4.0 was used in our evaluation. Version 5.0 has been released since then but the functionalities we used in our study have essentially remained the same.

In GME, a DSML is described as a paradigm, which is essentially a metamodel. GME comes with a plug-in (actually a DSML) that can be used to describe paradigms with class diagrams. Figure 3 presents our GME paradigm capturing our GRL metamodel.

GME's meta-metamodel offers stereotyped concepts such as *Atom* (elementary object), *Model* (which can have inner parts and structures), *Connection* (relationship between two objects within one model), *Reference*, *Attribute*, *Set* (similar to a UML aggregation) and other *FCO* (first-class objects). Most of the classes in our original GRL metamodel map directly to FCOs and Atoms in

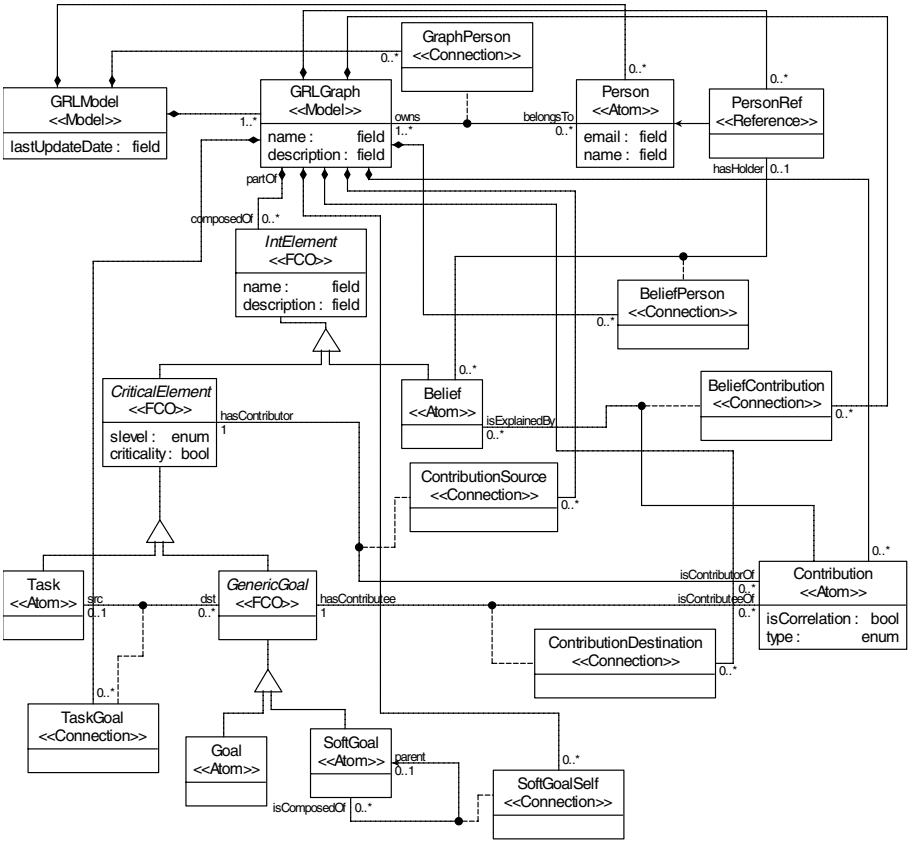


Fig. 3. GRL paradigm (metamodel) in GME

the GME metamodel, but additional Connection classes are also required for the original associations that are meant be manipulated (e.g., TaskGoal). Predefined data types such as *field*, *enum*, and *bool* are also available. An *Aspect* can be used to control the visibility of elements in the editor. OCL constraints can be added to increase the precision of the paradigm and to enable syntactical validation of user models in the target DSML editor.

GME supports the visual drawing of an object with a COM object called *decorator*. This allows one to associate the GRL shapes and symbols of Figure 2 to their respective concept in the paradigm. Simple bitmaps can be used as icons, but in this editor (implemented mainly by Y. Chu [2]) COM objects were programmed in C++, with great efforts, to reproduce the symbols correctly and have them automatically resized according to the length of the labels they contain. GME also offers a higher-level C++ interface called Builder Object Network, which is simpler to use than plain COM decorators but which is more limited.

Once a paradigm is created (and the decorators defined), it can be registered in GME and then used as an editor, as shown in Figure 4. The framework provides many features for free, including loading/saving (binary and XML), multiple undo/redo, drag and drop interface for the creation of model elements, validation against the metamodel multiplicities and OCL constraints, printing,

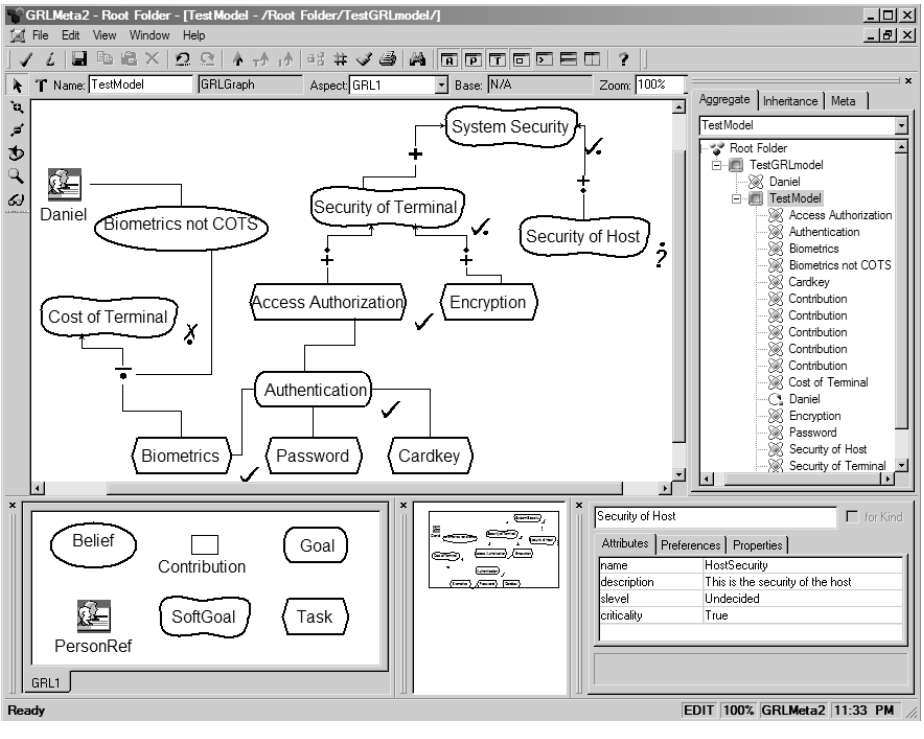


Fig. 4. GRL editor with GME

zooming, overviews, property views, etc. Multiple diagrams from the same model can be viewed and edited at the same time (in different sub-windows). The documentation is very good. However, we have found it difficult to associate decorators to links (e.g., for GRL levels of contributions) and intermediate nodes had to be defined, therefore hurting the usability of the editor. We could not find a way to visualize GRL correlations properly either.

Evolving paradigms can preserve backward compatibility if elements, links and references are added but not renamed or deleted (there is more robustness for attributes). Multiple paradigm versions can be registered, allowing one to open older files. Finally, it is possible to create our own analysis and transformation functions (and interfaces to the model are provided), but at the cost of fairly heavy C++ programming.

3.2 Telelogic Tau G2

Telelogic Tau G2 is a model-driven development environment [14] that supports UML 2.0. It can also be tailored and customized to specific modeling domains such as GRL via UML 2.0 *profiles* [13]. There are two ways of using profiles in this environment:

- *Stereotype Mechanism (SM)*: Stereotypes that extend basic UML elements are used, and extensions include customizations of names, attributes, and appearance. In this way, each GRL element can be implemented as a stereotype of a UML class. Although constructing a profile is relatively simple, the created modeling environment still includes all the basic UML elements that were extended. In essence, this does not lead to a real domain-specific environment, just to the addition of new and more precise modeling elements.
- *Metamodel Extension Mechanism (MEM)*: In addition to the functionality of the previous SM category, this mechanism provides metamodel extensions of non-basic UML element, such as class diagrams, by extending the UML metamodel itself. GRL models can hence be represented as a metaclass extension of UML class diagrams. This mechanism is more powerful but is more complex to implement. However, the resulting environment can be restricted to a domain-specific modeling language, without being polluted by other UML constructs.

Developing a MEM profile requires the creation or modification of dozens of classes and diagrams, which are too complex to be presented here (suffice it to say that most of the GRL concepts became extensions of the Class and Association base metaclasses in UML). Advanced knowledge of the UML 2.0 metamodel itself is also required. In addition, the process demands many manual steps inside and outside the modeling environment, for instance: installing the TAU SDK with FIDebugger, creating the profile directory structure and then the profile project in TAU, creating sub-packages for the metamodel profile, adding the metamodel classes representing the core UML structure and then classes representing the GRL customizations, and finally creating the TCL script that

must accompany the profile. The SM approach is more straightforward, yet is it still not trivial; Tau's usability for creating and deploying profiles is still rather weak (but improving with each new version).

Two GRL editors were created with Tau G2 2.4 using both profile approaches (version 2.7 has been released since then). Figure 5 shows a GRL model example created with our metamodel-extended profile.

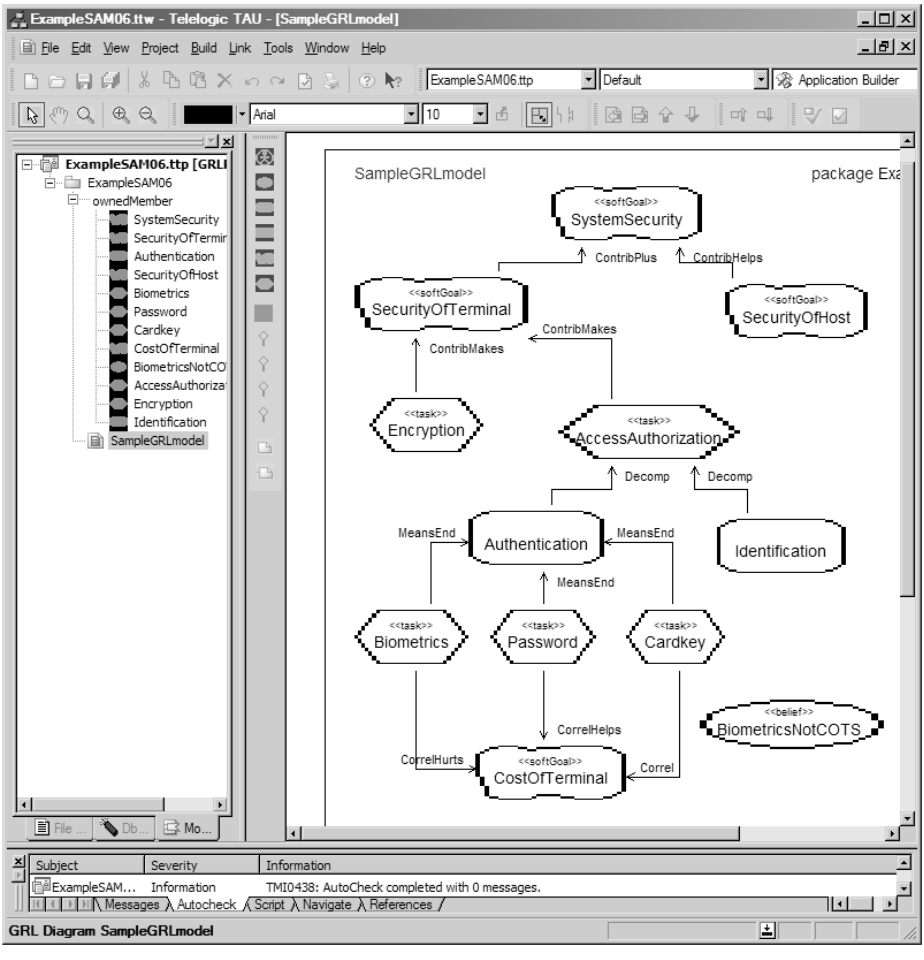


Fig. 5. GRL editor based on a UML 2.0 profile with Tau G2

The MEM approach is superior to the SM approach in many ways. For instance, the former enables one to customize diagram types as well as the user interface itself. Using this mechanism, a custom GRL diagram type was created along with a customized palette and model view, hence preventing one from mixing elements from different notations. This palette can be used to create GRL

elements in the model directly, whereas in the SM approach classes need to be created and then their stereotype changed via menus.

Editors implemented with Tau G2's profiles get many functionalities for free, including loading/saving models, printing, multiple undo/redo, zooming, property sheets, and some validation against the UML 2.0 metamodel. But the best benefit is likely the integration to the rest of UML 2.0 models (possibly with other profiles), something that is not available with GME.

There are however several limitations for the support of the graphical syntax: the appearance of links cannot be customized in TAU (which prevents the visualization of correlations, and more advanced types of GRL links not studied here) and restrictions on end points (constraints) require the programming of Tau *agents* in C++. Additionally, other GRL concepts like actor boundaries, which encompass intentional elements and links, cannot be visualized either (this is also the case for GME). Documentation on how to create profiles was lacking at the time this study was done, but we acknowledge the help of Tau's developers, who provided guidance and answers.

3.3 Rational Software Architect (RSA)

IBM's Rational Software Architect (version 6.0) is a UML 2.0 compliant integrated software development environment, built on top of the Eclipse platform [7]. Unlike Tau G2, RSA only provides the stereotype mechanism for defining profiles, which leads to less sophisticated editors than Tau's.

Creating a profile for GRL in RSA is simpler than with Tau. A user needs to create a UML profile project (so this is directly supported at the user interface level), select metaclasses to be stereotyped, (optionally) specify icons and images, and release the profile. In our example, GRL intentional elements are stereotypes of the UML Class metaclass, and GRL links are stereotypes of the UML Association metaclass or the Association Class metaclass. The actual GRL diagram is simply a UML class diagram with the extra GRL stereotypes. For the intentional elements, custom icons and shapes were used, but no such graphical customization exists for link styles. For GRL contribution and correlation links, Association Class links were used to enable the use of contribution types (see Figure 6).

As with the previous tools, loading/saving, multiple undo/redo, zooming, and property sheets are provided by the tool environment. This approach also benefits from an integration to UML 2.0, metamodel and diagrams alike.

The usability of the GRL editor produced in RSA is rather weak. For intentional elements (extensions of Class), the palette provides easy access by clicking on the Stereotyped Class icon and then selecting the desired stereotype from a list. However, for stereotypes that do not extend the UML Class metaclass (such as GRL Correlation, which extended the Association Class metaclass) these stereotypes have to be applied manually using the Properties view.

Other issues similar to the SM approach in Tau have been observed. RSA does not support custom restrictions on the end points of UML links, and custom diagram types cannot be created (and hence class diagram elements can get

mixed to the GRL diagram, for instance multiplicities are shown by default, as shown in the dashed circles in Figure 6). The user interface cannot be customized directly via the profile, however RSA allows for customization via its Eclipse-based Java API (but this was beyond the scope of this study).

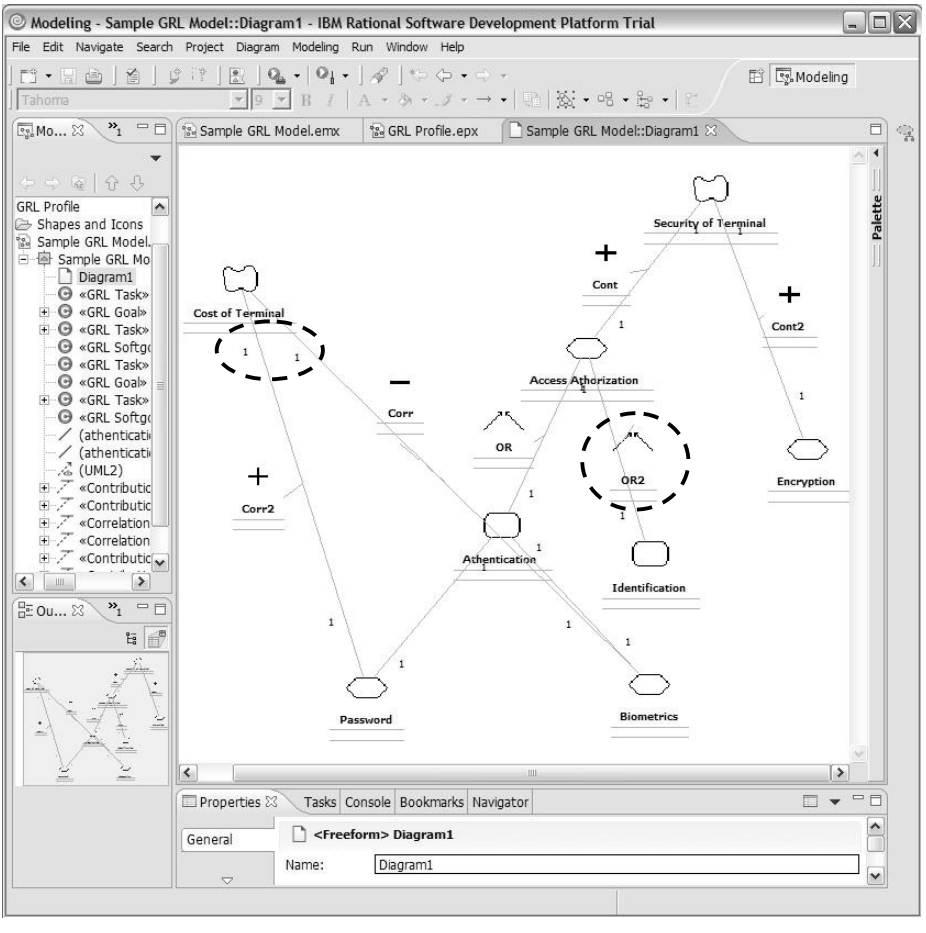


Fig. 6. Example of GRL diagram produced using a UML 2.0 profile with RSA

3.4 XMF-Mosaic

Xactium XMF-Mosaic is an integrated, Eclipse-based, extensible development environment for domain-specific (modeling) languages [16]. Building on standards such as MOF and OCL, it supports the definition of grammars and the generation of parsers. It also supports domain model design with constraints, model transformations, and editor generation by providing the DSML metamodel to the Xtools module. This tool also has a unique feature: concrete textual and graphical syntaxes can easily be provided and supported for the same language.

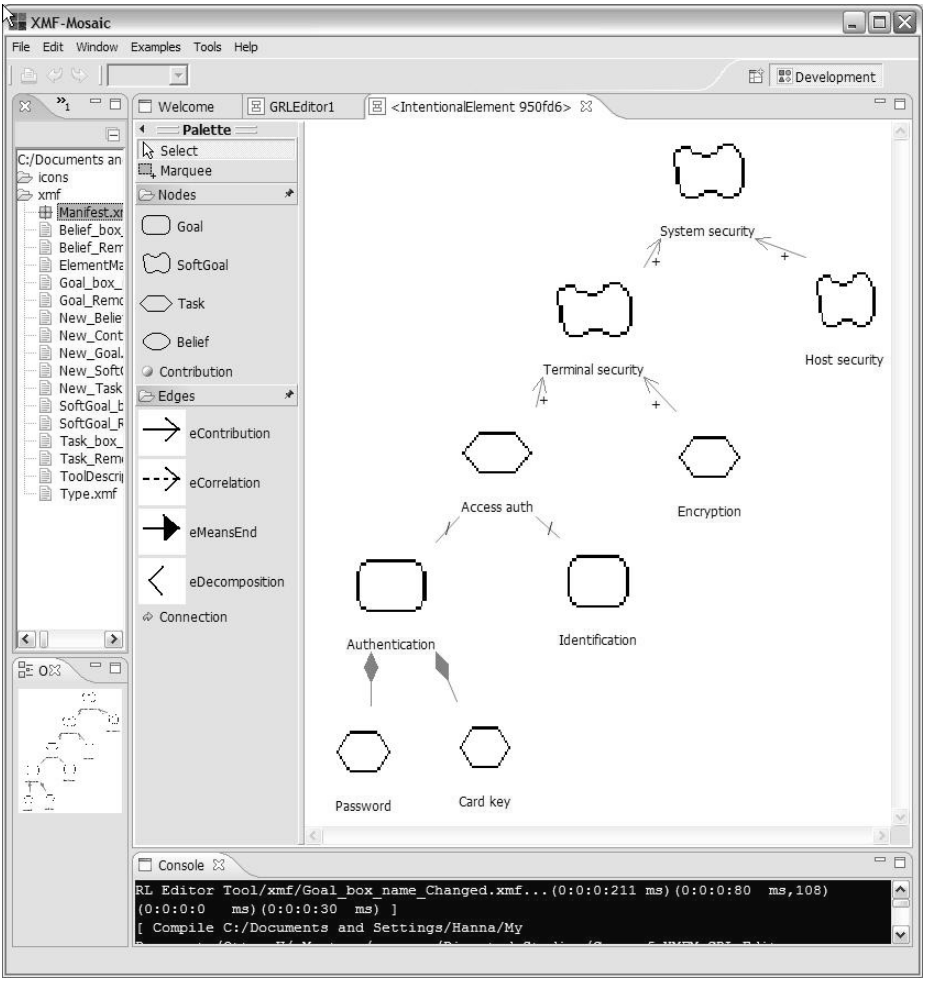


Fig. 7. GRL editor with XMF-Mosaic

In XMF-Mosaic, the domain model (metamodel) is defined with a class diagram in MOF/XCore, and OCL constraints can be added (via menus) to improve its precision. The environment supports the creation of *snapshots*, which are essentially object diagrams allowing one to test the metamodel and its constraints at an early stage. This is useful in our context, where a new language is being developed.

A graphical editor can be generated automatically from the domain model, however this feature still contains many limitations and bugs. For instance, if a superclass has an association with another class, an automatically generated editor supports creating the link for the superclass but not for its subclasses. An additional problem is that this approach generates visual items/nodes for every class in the domain model (including link classes) as the tool has limited

understanding of the semantics. Potential solutions include coding the necessary elements manually, or generating the whole code first and deleting the parts corresponding to unnecessary elements (the first option was selected in our editor).

Different icons can be associated to classes in the palette by editing the *type.xmf* file, and the shapes of the GRL elements in the model can also be changed to bitmaps (see the example in Figure 7). For GRL beliefs, connecting a node to an existing link seemed to be impossible and a workaround (involving an invisible node) had to be used. Also, we could not find a way to modify link ends beyond symbols used in class diagrams.

XMF-Mosaic provides good feedback during the development of the domain model and of the editor. Additionally, the building process is incremental and not everything needs to be recompiled upon modifications, which accelerates the development of editors. The text console, which offers a different mode of interaction, was well appreciated.

Although the approach suggested by this tool is very interesting in theory, the early age of XMF-Mosaic (version 0.7 was used in this experiment) results in several weaknesses. For instance, there is no undo/redo in the GRL editor produced, and one cannot load/save models; this prevented us from evaluating how well the evolution of metamodels is supported. Also, the OCL constraints in the domain model are not transferred to the editor generated (and cannot be used for validation). Documentation was severely lacking, but we acknowledge the help of Xactium's support team who answered many questions. We have quickly looked at version 1.0 (released at the end of this study) and, although the editor generation works better with an attempt at supporting the saving of models, most problems cited here still remained.

3.5 Eclipse EMF+GEF

Eclipse is an open source and extensible Java-based platform that provides many useful services for the creation of textual and graphical editors. Versions 3.0 and 3.1 were successively used, and now version 3.2 has been released. For building graphical editors, two Eclipse plug-ins are especially relevant.

The *Eclipse Modeling Framework* (EMF) is a framework and code generation facility for building tools and other applications based on a structured data model [4]. From a metamodel specification described as an XML Schema or as a class diagram in Rational Rose (such as the one in Figure 1), 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 editor. 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 [5].

Based on our experience in creating an Eclipse plug-in editor for the Use Case Map notation called jUCMNav [11], which uses GEF and EMF, we decided to add support for GRL to this tool. The metamodel was created as a class diagram with Rational Rose, and then imported into Eclipse by EMF. This mechanism, which we have found reliable and easy to use, generates EMF classes in Java

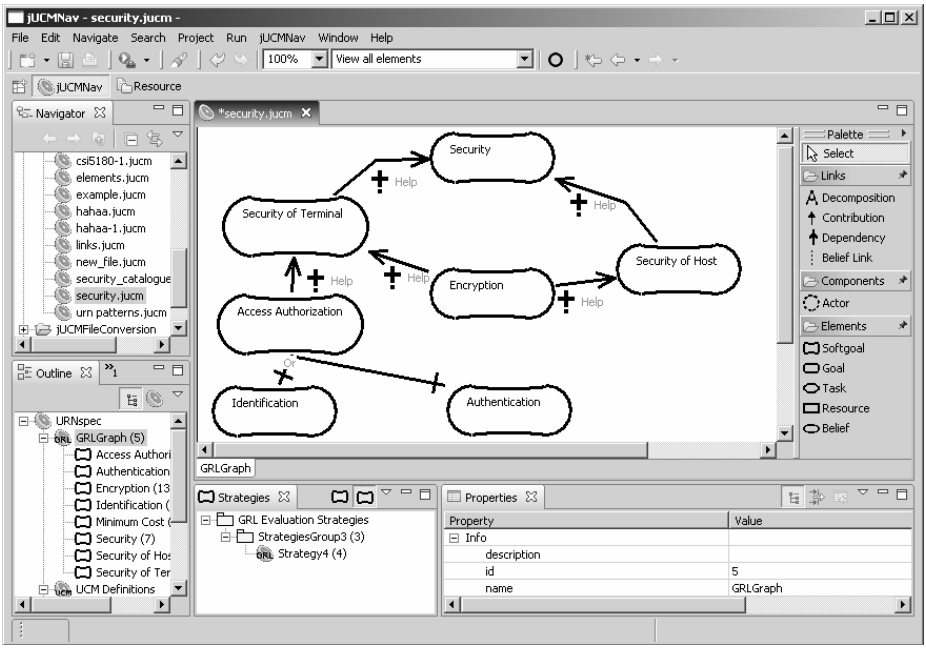


Fig. 8. GRL editor with Eclipse, GEF and EMF

that can be connected to GEF-based GUIs. The only noticeable problem we have observed with this code is that it does not enforce the minimum and maximum multiplicities found in the metamodel. OCL is not supported either.

Much effort is required to learn EMF and GEF and to understand how they are combined. Documentation (including tutorials and books) and useful discussion forums are however available. The quality of the resulting editor is very high, especially from a usability viewpoint. The Eclipse platform, together with EMF and GEF, 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. The entire notation can be supported (see Figure 8), although at this time our prototype does not support beliefs attached to contributions (this proved to be difficult, like for all the other tools).

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

Many items related to the evaluation criteria introduced in section 2.2 were discussed in the section 3, and the current section provides a brief summary with additional insights based on our experience with these tools. Table 1 provides a quick overview of the strengths and weaknesses of each tool.

Table 1. Overview of comparison

	GME	**Tau G2**	**RSA**	**XMF-Mosaic**	**Eclipse**
Graphical Completeness	Medium	Low	Very Low	Low	High
Editor Usability	Medium	Medium	Low	Low	Very High
Effortlessness	Medium	Low	High	Low	Very Low
Language Evolution	High	?	?	?	Medium
Integration	Low	High	High	Low	High
Analysis / Transformation	Medium	Medium	Low	High	Medium

- *Graphical completeness*: The Eclipse approach is the only one that allowed reproducing the GRL notation with fidelity (including more advanced concepts like actor boundaries). GME did well in general, except for a few restrictions. Both required substantial additional programming. RSA offered the least flexibility for this criterion.
- *Editor usability*: The best usability is offered by the Eclipse editor (by far) in terms of user experience, tool feedback, and overall number of features. All tools except XMF-Mosaic support multiple undo/redo and loading/saving of models. The manipulation of elements is somewhat awkward in RSA.
- *Effort*: All these tools require some effort for learning the technology and for creating a DSML editor. The profile creation and usage mechanism in RSA is likely the easiest one among the five studied here, followed by GME, and Eclipse is definitely the worst.
- *Language evolution*: When the language metamodel evolves, Eclipse and GME share many common characteristics regarding backward compatibility (with files saved using the previous version). The time spent for fixing the editor is small in GME and, again, fairly high in Eclipse (although the modifications are not difficult in our experience). This aspect was not tested in XFM-Mosaic because models could not be saved and reloaded.
- *Integration with other languages*: Tau G2 and RSA both offer a direct integration with UML 2.0 as well as with other profiles. The Eclipse solution offers a different integration via extension points and the simultaneous presence of multiple plug-ins (some of which might be related to other languages). Integration appears to be weak with XMF-Mosaic (although it has some potential, being Eclipse-based) and similarly with GME, more isolated.
- *Analysis capabilities*: This aspect was not thoroughly studied in our experiments. Such capabilities appear to be weak in RSA. Tau G2 supports the concept of agents, which can be programmed (in C++ and possibly TCL) to examine/transform models. GME offers interfaces (in COM/C++) to access

and transform models. Eclipse/EMF provides Java interfaces to easily access models, but transformations are manual. XMF-Mosaic is probably the most promising environment in this category, with specific (and standard) languages for analysis and transformations. Note also that the only environment that generates editors where models are checked against the OCL constraints in the metamodel is GME.

5 Conclusions

This paper compared five different tools for the generation of development environments targeting domain-specific modeling languages. A particular emphasis was put on the generation of graphical editors with a case study involving a simple but representative subset of the Goal-oriented Requirement Language whose abstract syntax is specified with a metamodel. Editors were created with each tool, and our experiments helped us compare the approaches against criteria such as graphical completeness, usability, development effort, handling of language evolution, integration with other languages, and analysis capabilities.

For simple prototyping of modeling language editors, GME offers an interesting balance between metamodel precision and validation, ease of editor generation, and usability of the editor. For serious, industrial-strength editors, Eclipse (with EMF and GEF) appears to be the most viable (and multi-platform) solution among those studied here, and this is in part why GRL tools such as jUCMNav [11] and OpenOME [17] are headed this way. However, the development effort will be proportional to the benefits. If the integration with UML 2.0 is a must, then Tau G2 and its metamodel extension mechanism for profiles has several interesting benefits over RSA, which is currently limited to a stereotype mechanism. XMF-Mosaic brings novel and promising ideas in the DSML area, but at this time it still suffers from a lack of maturity.

To alleviate some of Eclipse's weaknesses in terms of required development efforts, a new plug-in called *Graphical Modeling Framework* (GMF) [6] attempts to provide a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF. We plan to study GMF in the near future. We also plan to continue the integration of GRL and UCM in jUCMNav, and to improve its analysis and transformation features.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada, through its programs of Strategic Grants and Discovery Grants. The development of editors with GME and RSA/Tau was done respectively by Yi Chu [2] and Nadir Janmohamed [10], whom we thank. We are grateful to IBM, Telelogic, Vanderbilt University, and Xactium for providing their tools and technical support for this study.

References

1. Amyot, D. and Mussbacher, G: URN: Towards a New Standard for the Visual Description of Requirements. In E. Sherratt (Ed.): Telecommunications and beyond: The Broader Applicability of SDL and MSC (SAM 2002). Lecture Notes in Computer Science 2599, Springer 2003, 21–37.
2. Chu, Y.: Tool Support for the Goal-Oriented Requirement Language. M.C.S. project report, University of Ottawa, August 2005. http://www.site.uottawa.ca/ damyot/students/YiChuReportAndTool.zip
3. Domain-Specific Modeling Forum, http://www.dsmforum.org
4. Eclipse: Eclipse Modeling Framework (EMF), http://www.eclipse.org/emf/
5. Eclipse: Graphical Editing Framework (GEF), http://www.eclipse.org/gmf/
6. Eclipse: Graphical Modeling Framework (GMF), http://www.eclipse.org/gmf/
7. IBM: Rational Software Architect (RSA), 2005. http://www-306.ibm.com/software/awdtools/architect/swarchitect/
8. Institute for Software Integrated Systems: The Generic Modeling Environment (GME), 2004. http://www.isis.vanderbilt.edu/Projects/gme/
9. ITU-T: Recommendation Z.150, User Requirements Notation (URN) – Language Requirements and Framework. Geneva, Switzerland, 2003.
10. Janmohamed, N: Expressing Goal-oriented Requirement Language in UML 2.0: Examining the functionality of UML Profiles. CSI 4900 project report, University of Ottawa, April 2005. http://www.site.uottawa.ca/damyot/students/NadirRep.zip
11. Kealey, J., Tremblay, E., Daigle, J.-P., McManus, J., Clift-Noël, O., and Amyot, D.: jUCMNav: une nouvelle plateforme ouverte pour l'édition et l'analyse de modèles UCM. 5ième colloque sur les Nouvelles TEchnnologies de la RÉpartition (NOTERE'05), Gatineau, Canada, August 2005, 215–222. http://jucmnav.softwareengineering.ca/twiki/bin/view/ProjetSEG/WebHome
12. MetaCase, MetaEdit+, http://www.metacase.com/mep/
13. OMG: Unified Modeling Language (UML), version 2.0, October 2004. http://www.uml.org/#UML2.0
14. Telelogic AB: TAU G2, 2005. http://www.telelogic.com/products/tau/
15. URN Focus Group: Draft Rec. Z.151 – Goal-oriented Requirement Language (GRL). Geneva, Switzerland, Sept. 2003.
16. Xactium: XMF-Mosaic Getting Started Guide, Version 1.0, July 2005. http://www.xactium.com/
17. Yu, E.: OpenOME, an open-source requirements engineering tool, 2005. http://www.cs.toronto.edu/km/openome