Towards a full implementation of a robust solution of a Domain Specific Visual Query Language for HEP Physics analysis

Vasco Sousa[1], **Vasco Amaral**[2] **and Patricia Conde**[3]

[1,2] FCT-UNL, Universidade Nova de Lisboa, [3]LIP, Lisbon, Portugal

E-mail: `vasco.sousa@gmail.com`, `vasco.amaral@di.fct.unl.pt` and `Patricia.Conde.Muino@cern.ch`

Abstract.

In our previous work, project PHEASANT, we have proposed a Domain Specific Language (DSL) for the purpose of providing a the HEP community with tools that could increase user's productivity while producing query code for HEP physics data analysis. The main purpose of this project was a proof concept including methodology feasibility by introducing the concept of DSLs. We are now concentrated on implementation issues in order to deploy a final tool.

The concept of domain specific languages has always been implicit in Software Engineering although the development of such languages was never done in a systematic way. The main goal of having DSLs is to rise the level of abstraction. The idea is to provide the final user (stakeholder) with tools to reason and model the solution by using concepts of the problem domain instead of having to reason with concepts of the problem domain (meaning the implementation details like programming concepts and hardware restrictions). Once we have the model specified, we can use Model Driven Development and Software Product Lines techniques to deploy artifacts in a automatic way (meaning: software products, code, documentation etc).The Software Engineer community has been focusing its attention to methodologies and deploy tools for helping DSL developers in their effort to help productivity and efficency at several application domains such as HEP. A comparative study of these tools should be done to determine their capability to answer the specificity of HEP Physics Analysis requirements.

In this communication we will present the several technologies for DSLs meta-modeling studied in order to implement the DSL proposed by the PHEASANT project.

1. Introduction

In our previous work [2, 3, 4, 5, 6, 7], we have proposed a new approach for HEP physics data analysis by using a diagrammatic language to equip the physicist with a efficient way to produce his analysis query code by using notations not related to the code or specific analysis framework underneath it. in other words we have proposed to higher the abstraction level from the General Purpose Language (GPL like C, C++, Java, etc).

This need came from the observation that to code thousands of lines of code is time consuming and error prone (specially if the physicist is not proficient with programming or even with the typically complex analysis framework). Bad code has as a side effect a strong penalty over the analysis framework, since a badly conceived algorithm for implementing that query can consume unnecessary resources (that are typically scarce for the amount of data used in this domain).

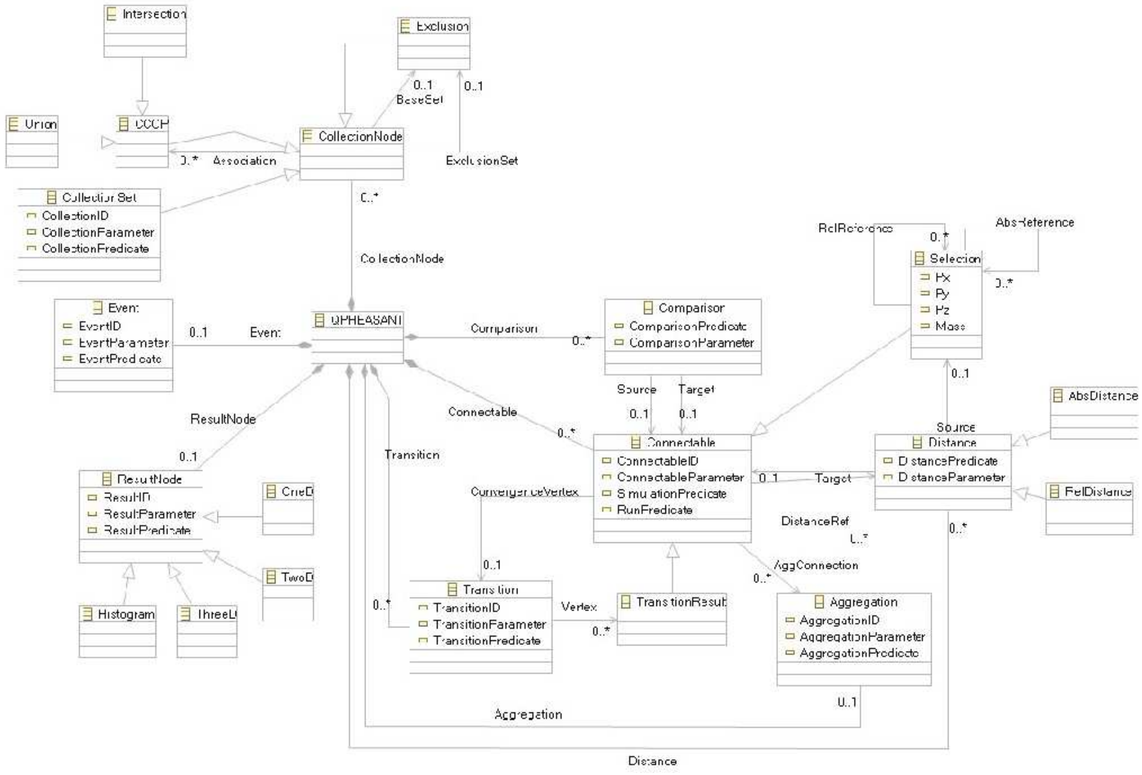


Figure 1. PHEASANT UML class diagram

In order to prove our ideas we have carefully designed and implemented a language named PHEASANT using computer Science methodology. The syntax of the language was defined by using a diagrammatic notation and the exact meaning was given by pattern translation to algebraic operators based on a object oriented algebra specifically defined for this purpose (a technique used in Database kernel construction).

We have then built a prototype framework that contained an editor to interpret a query model and translate it to algebraic operators (which gives us the chance to think on how to optimize the query) and from there to code (by using a code generator that would read the algebraic tree and produce automatically the corresponding code).

Now that we have the proof of concept, our next step is to implement a robust solution that makes use of the current state-of-the-art Software Engineering technology. The advantages over implementing ourselves the language editor are obvious. There should be less effort to build the editor itself and there should be less effort to do language evolution in the future.

In this paper we are going to discuss our comparisons of different tools for designing and implementing diagrammatic DSLs.

On Section 2 we are going to talk about DSLs and give an overall picture of the mission of the existing meta-modeling tools. After that, in Section 3, will concentrate on comparing existing meta-modeling tools and choose the one that best fits to our requirements. Finally, we will conclude with Section 4 and point to directions of our future work.

2. Diagrammatic DSLs and tools for their implementation

Domain-specific languages (DSLs) in opposition to general purpose languages (GPLs) use notations that are closer to the abstractions found in a specific problem domain.

The concept of developing DSLs is already very old. However, the methodology and tools to support their development only recently are becoming more mature.

One of the major difficulties of defining a language is to have a proper method to specify it's syntax (consistent notation) and its semantics and from a description derive the tool to interpret the sentences (that we call models) in order to generate other models, code in a GPL (such as Java or C++), execute or other relevant forms like documentation artifacts.

Experience has shown that typically the DSLs that best fit the needs of a final user specification model in a certain domain have the diagrammatic format (or graphs with nodes and edges). To implement editors to deal with diagrams instead of textual code is a very time consuming process. This is the main reason why those evolved very slowly in the past. However, several tools are evolving to fill this gap.

Several models are required as input to those tools in order to automatically generate a visual editor for parsing and interpret new models specified in a new DSL:

- A DSL grammar serves as the primary artifact for defining DSLs from a higher level of abstraction, typically called as the language meta-model. This syntax description is a model specified in another language, the meta-modeling language like Entity Relationship (ER) modeling language or a Unified Model Language (UML) like the one we have depicted in Figure 1.
- Concrete syntax models are also artifacts that give the visual shape of the language notation. Typically it associates with a certain entity in the syntax model an image format (like jpg,gif, etc). Also some extra information might be added like points of contact to edges, text location of entity properties.
- GUI interface layout and behavior modeling. It supports ways to specify if there should be toolbars, menu-bars, and some of them even specify what to do with specific behavior (like focus, click etc).
- Model restrictions(that the UML model is not capable to express) for the model construction can be typically specified by using OCL(Object Constraint Language)**??**, languages like Java or even Python.
- Transformation rules for dictating the export format of the model that was interpreted by the tool. Sometimes those rules specified in some specific textual language, like for graph transformation grammars, or more generic like python. Basically we can see those rules as algorithms to interpret the graphs in the models and translate them to code or other models.

3. Comparison Analysis

This section describes in depth a comparative analysis of the existent tools for meta-modeling graphical domain specific languages. The great majority of those are of academic nature, although we can find some commercial tools as well. This analysis has the main goal of selecting the tool or tools that better suit the implementation needs of PHEASANT project and that contains a methodology and technology adequate for easily allowing evolution and/or improvement in the future.

From the array of tools available either for commercial use or academic development, the ones presented in this chapter where the ones that showed to be more accessible and had a minimum maturity level.

3.1. Comparison criteria

During the analysis process of the available meta-modeling DSL tools, in order to have uniform evaluation criteria, the following characteristics where evaluated:

- **Platform** - The OS and requirements of the editor to run
- **License** - The type of licensing the editor and generated products are subject to.
- **Framework** - The base in which the specific language is edited. In most cases, the same base of the meta-modeling process. Customizable UI - If the user interface of the specific language editor is customizable, designing a uniform use pattern, or is it generated automatically without intervention.
- **Import Format** - What formats the editor can interpret to create a model or meta-model from information of external tools.
- **Export Format** - What formats the editor can translate its models to, so other tools can use them. Storage Format - The format in which the models and metamodels are serialized and stored.
- **Metamodel Definition** - The rules by which the metamodel is specified, mostly Class Diagram.
- **Metamodel Construction** - The method of enforcing the Metamodel Definition, in most cases in the form of a schema.
- **Separate Models** - Does the editor use separate models for visual syntax, semantics, and other information the editor needs.
- **Constraints** - Constraint of the model, definition methods, and OCL support.
- **Constraint Verification** - Method of enforcing the defined constraints.
- **Model Construction** - Specifies the use of the metamodeled language. 9
- **Sub Diagramming** - If a given editor allows for hierarchic definition of its models.
- **Multiple DSLs** - The ability of using more that one metamodel in conjunction in the model construction process.
- **Traceability** - The way the editor deals with models they are no longer set to the latest metamodel defined.
- **Report Generation** - the capacity of generating documentation on the model and metamodel automatically
- **Code Generation** - the capacity an method of generating interpretable or compilable code from the model in use.
- **Undo/Redo and Versioning** - Does the editor support undoes redoes and separate versions of the same model and metamodel.
- **Model Transformations** - Can the editor proceed to lower base model transformation, such as representation paradigm shifting to a new metamodel, and optimizations.
- **Editor Documentation** - Classifies the available documentation and examples.
- **Particularities** - Characteristics that the other editor dont present.

3.2. $AToM^3$

The tool $AToM^3$ has been developed at the Modeling, Simulation and Design Lab (MSDL) in the School of Computer Science of McGill University. It was totally conceived in Python scripting language, being supported by a variety of operating systems running Python, including windows, mac OS and a series of flavors of Linux and Unix.

$AToM^3$ is free and downloadable from the development page. Because it runs on an interpreted language the source code is accessible too. The developed languages are edited in the $AToM^3$ editing framework itself. In that process $AToM^3$ adapts its interface by automatically adding language specific buttons in the toolbar, being possible to set python scripts to them, and customize its icons.

The only file types it processes and outputs are in python, although other file formats can be set to be processed by python itself. It defines the language metamodels by means of a visual specification of class diagrams. These metamodel diagrams define all aspects, visual and syntactical of the defined language. On the other end the created models themselves are also stored in a unique file. It is also possible to further restrain the models (besides the syntatical relationship bettwen the diagramatic notations) through python constraints. These constraints and metamodel restrictions are verified at editing time for every element alteration. The model is then constructed in the same manner as the metamodel, in one unique diagram, supporting the possibility to load multiple language definitions at the same time, making them relate and interact in the same diagram.

It was not possible to test $AToM^3$s behavior with models based on previous versions of a particular metamodel. There is no default report generation function available in the editor. It can generate python code from the created models.

There are no Undo or Redo functionalities, because of the restrain maintenance of the model implemented, and no version tracking of the created models and metamodels either.

The strongest point in $AToM^3$ is its model transformation capacities, capable of performing graph transformations and being developed transformations that allow to transfer a model between two metamodels.

The available documentation is not very helpful, for at certain tutorial passages are assumed knowledges not yet described or properly referenced, making difficult the initial work with this editor.

One of the particularities that we observed as a characteristic that the other DSL tools do not present is the vector drawing editor for bulding the pictures for the symbols.

3.3. DSL Tools

DSL Tools is the MicroSofts approach to the Domain Specific paradigm. It runs as a Visual Studio 2005 toolkit, running only on windows XP, 2000 and Vista, requiring also the installation of the Operating Systems and Visual Studios service packs.

Although the SDK is in itself free, all work is subject to the Visual Studio license and development restrictions. The developed languages are used in DSL Tools developing environment, and not as a stand alone application.

It automatically makes available the created languages functionalities and elements through a sidebar. It is capable of importing from XML formats but stores it a proprietary XML format defined by microsoft, giving no indication of automatically exporting to other formats.

The format in which DSL Tools defines its languages, is a cascading relations schema that approaches both class diagrams and use case diagrams, in a visual manner.

On the left side of the metamodel diagrams, is placed the semantic definition of the language, on the right the visual aspects of the language elements. Apart from the metamodel imposed restriction, it is possible to specify further constraints, introducing them directly in the generated code.

In the modeling process the metamodel restrictions are immediately verified. The modeling process is made a similar environment as the metamodeling, but without the positioning restrictions present at that phase. It is not defined if it is possible to sub-diagram a model, the developers leaving that possibility in the inclusion of multiple DSLs.

DSL Tools is backwards compatible, meaning that is is possible to read models made with older versions of the defined language DSL, if the updated meta-model still makes sense of that information stored. It can also generate reports of the created models and metamodels through the use of templates and export the diagrams to image format. Like the report generation, code generation is also made by means of the use of templates. It offers Undo and Redo functionalities, but no version tracking of the created models and metamodels. No transformation capabilities are supported in DSL Tools. Documentation for this editor is good, including video showcase of its capacities that can be followed, but all in all, not very abundant and thorough relying more in the discussion groups.

3.4. GEMS

The Generic Eclipse Modeling System (GEMS) is being developed by the Distributed Object Computing (DOC) Group at Vanderbilt Universitys Institute for Software Integrated Systems (ISIS) and other collaborators, such as Siemens Corporate Technology.

Being an Eclipse plug-in, and developed in java, it runs on any machine capable of running the java virtual machine, such as Windows, Mac OS and a series of flavors of Linux and Unix systems.

Working under the Open Source license, it is free to be used, distributed, as are all subsequent products developed in it. The developed languages are run in GEMS itself.

The elements and connections of the created language are automatically made available through a sidebar palette. The only import and export options available are those set for eclipse itself.

It uses a visually manipulated adaptation of UML 2.0 to ease the editing and interpretation processes. The created metamodel diagram define all aspects, visual and syntactical of the defined language, in the same diagram. It is possible to place OCL constrains and Prolog expressions to specify restrictions indescribable by means of the UML class diagram only.

All constraints, metamodel driven or explicit are verified at modeling time at every element. The model construction itself is made in the same manner as the metamodel, in a single diagram.

With alterations in the metamodel definition previously created models are completely rejected.

3.5. GME

The Generic Modeling Environment (GME) is also being developed at Vanderbilt Universitys Institute for Software Integrated Systems (ISIS).

GME is a Windows based Visual language modeling tool, and is now in its fifth version. Working under the Open Source license, it is free to be used, distributed, as are all subsequent products developed in it.

The developed languages are edited in GME itself. For the use of the language, GME makes available a set of separators with the node elements of the created language, being all relations and connections done by the default tool.

GME can import from XML format and stores files in XME, being capable to export to a variety of formats by means of its pattern processing function. It uses an adaptation of UML 2.0 that is visually manipulated to ease the editing and interpretation processes.

The created metamodel diagram define all aspects, visual and syntactical of the defined language, in the same diagram. The model itself keeps the semantic and visual properties in the simple file.

Further restrictions not representable in class diagrams, are introduced through OCL element expressions and functions. These restrictions are verified at user defined trigger events, such as element creation or connection, or manually, useful if the model has deadlock restriction situations.

Model construction itself is made in the same manner as the metamodel, supporting if defined at metamodel time, the possibility of defining the model in several separated diagrams, containable or not in other diagrams themselves. As the metamodel is altered, GME will try to adapt the existing models to the new metamodel, failing this it gives the chance of loading it in a previous version of the metamodel still registered and try to convert the models through a more thorough XML interpretation.

The report generating capabilities of GME, pass by means of a scripting of the model and output processing pattern, making it possible to adapt the output to many formats. It is also possible to introduce plug-ins created in Visual Studio 2003, that extend GMEs functionality, allowing it to generate compilable code for specific platforms.

Providing Undo and Redo options, it also allows to maintain a version control of the metamodels, but not of the models created. It can be performed model transformations by means of an external tool that works on GMEs model format, available on its website.

The documentation is the best of the set, thorough, simple and coherent with the versions used, missing only in directing good references to OCL references or documentation, being this at some level external to the editor itself.

3.6. GMF

GMF was initially developed by IBM, it was then offered by this institution to the eclipse project, being now developed by its own community.

Running as an Eclipse plug-in, it runs on any machine capable of running the java virtual machine, such as Windows, Mac OS and a series of flavors of Linux and Unix systems, the adaptation of the GMF project to the Rich Client Platform (RCP) allows it to run independently from eclipse in any of the systems capable of running Eclipse .

Working under the Open Source license, it is free to be used, distributed, as are all subsequent products developed in it. The developed languages are used in a generated editor that runs as an eclipse plug-in.

By default a sidebar palette is made available, in which you can select what tool are available their icon and how to group them, and a popup palette with the elements applicable in the context of where the mouse pointer is, it also makes available the creation of toolbars and popup menus.

GMF stores its files in XMI format leaving other export formats to auxiliary eclipse plug-ins, it can also import the base model from XML, ecore, annotated java and rose, although no always smilingly.

The metamodel definition is made through a tree specification of the defining elements of UML 2.0. Each aspects of the metamodel are defined in different files. The visual definition of the model, can be saved separately or together with the semantic model.

GMF makes available mechanisms to introduce OCL expression constraints, over the diagram class metamodel definition, and java restrictions directly in the generated code.

The OCL and metamodel constraints are enforced immediately on model creation, java restrictions, although triggered like the other restrictions, allow to specify their behavior, allowing more than yes and no situations. From the metamodeling and editor specification, a visual editor is created, initially as an eclipse plug-in, to define models in the specified language. This specification can include a certain level of sub-diagramming, in the manner of including elements inside each other, allowing the structuring and hiding of parcels of the created model.

If the metamodel or the editor definition is altered, it loads the created models elements that still fit the definition and in extreme cases rejecting the old model. Not having internal report generation capacities, it relies such functionalities to available eclipse plug-ins.

Like report generation, it is possible to export the models by means of a third party eclipse plug-ins. The Undo and Redo capabilities are supported. No version tracking is available for metamodels and models.

Although no model transformation is available, it is possible to apply XML transformations over the XMI file, that defines the model.

GMF has the worst documentation of the whole set of tools that we have studied. The examples are hard to follow and are incoherent with the documentation of the corresponding version. Normally, the documentation focuses on beta functionalities. However,after our evaluation, we have noticed very recently that there starts to be a consistent efort to evolve documentation in order to be more precise and complete, which can be a very good plus in the near future.

3.7. Metaedit+

Metaedit+ was developed by Metacase, wich has a long tradition of metamodeling and case tools. It is available for Windows, Mac OS and Linux. Runs on a commercial license, but with introductory and evaluation licenses also available.

The developed languages editor runs on Metaedit+ framework itself, not being able to derive a stand-alone generated language editor. Instead a toolbar is created automatically with the elements and connections of the created language in the same editor for the meta-model.

Importing and storing files in a specific Metaedit XML format, it is able to export to a variety of formats by means of pattern scripting, in some ways pared to GMEs exporting functions.

The metamodel is described thanks to a set of tables that define the elements of a class diagram like structure, made available in a set of wizards.

All elements of the created language, visual and syntactical, are defined all together. The constraint to the model can be done by definition methods and OCL support. The metamodel constraints are verified at modeling time. The created language is then modeled on the editor itself, on a visual canvas, in a single diagram.

As the metamodel language created evolves Metaedit+ discards the relation that it can no longer make sense in the metamodel context.

Report generation is made by using a scripting of the output patterns and model analysis, making it adaptable to many report formats.

This tool has the capacity an method of generating interpretable or compilable code from the model in use. Does the editor support undoes redoes and separate versions of the same model and metamodel.

The editor is not able to deal with model transformation. Generally speaking it has documentation and examples.

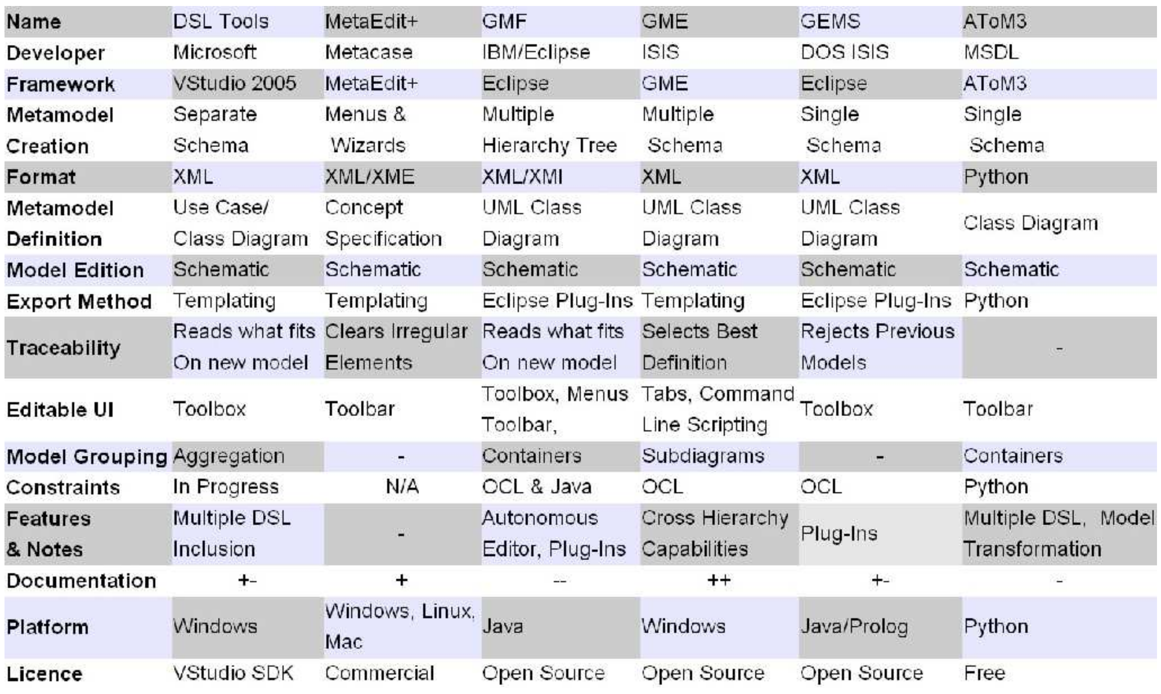
3.8. Comparative results

Figure 2 summarizes our survey. All of the tools evaluated presented some degree of limitation, notably at interface management and definition.

As a result of this analysis, two editors were picked to proceed the implementation of a PHEASANT editor. GME was selected for its ease of use, rapid development and learning curve, working as a good proof of concept tool, and allowing the use of the script console if model developers need a script generated model interface. In Fig. 3 we can an example of a generated editor for pheasant implemented this framework. The second selected was GMF, because it can run on many systems including Linux, which has a broad use by Physicists, it allowed the further development of the editor, including the customization of the interface and in code improvements to the editor, even after the end of this project, and the possibility to make PHEASANTed independent from its building platform, making it possible to progress to a fully professional and seemingly commercial tool.

4. Conclusions and Future work

Altough the state-of-the-art technology for diagrammatic languages meta-modeling is still at its maturing phase, we expect much greater enhancements in the near future. However, considering the specificity of the requirements for Physics Data Analysis, this type of technology has the minimal functionality to be considered usable. In fact, it can rapidly build a prototype although being limited in its expressivness and funcionality.



Name	DSL Tools	MetaEdit+	GMF	GME	GEMS	AToM3
Developer	Microsoft	Metacase	IBM/Eclipse	ISIS	DOS ISIS	MSDL
Framework	VStudio 2005	MetaEdit+	Eclipse	GME	Eclipse	AToM3
Metamodel Creation	Separate Schema	Menus & Wizards	Multiple Hierarchy Tree	Multiple Schema	Single Schema	Single Schema
Format	XML	XML/XME	XML/XMI	XML	XML	Python
Metamodel Definition	Use Case/ Class Diagram	Concept Specification	UML Class Diagram	UML Class Diagram	UML Class Diagram	Class Diagram
Model Edition	Schematic	Schematic	Schematic	Schematic	Schematic	Schematic
Export Method	Templating	Templating	Eclipse Plug-Ins	Templating	Eclipse Plug-Ins	Python
Traceability	Reads what fits On new model	Clears Irregular Elements	Reads what fits On new model	Selects Best Definition	Rejects Previous Models	-
Editable UI	Toolbox	Toolbar	Toolbox, Menus Toolbar,	Tabs, Command Line Scripting	Toolbox	Toolbar
Model Grouping	Aggregation	-	Containers	Subdiagrams	-	Containers
Constraints	In Progress	N/A	OCL & Java	OCL	OCL	Python
Features & Notes	Multiple DSL Inclusion	-	Autonomous Editor, Plug-Ins	Cross Hierarchy Capabilities	Plug-Ins	Multiple DSL, Model Transformation
Documentation	+-	+	--	++	+-	-
Platform	Windows	Windows, Linux, Mac	Java	Windows	Java/Prolog	Python
Licence	VStudio SDK	Commercial	Open Source	Open Source	Open Source	Free

Figure 2. DSL Meta-modeling tools comparison table

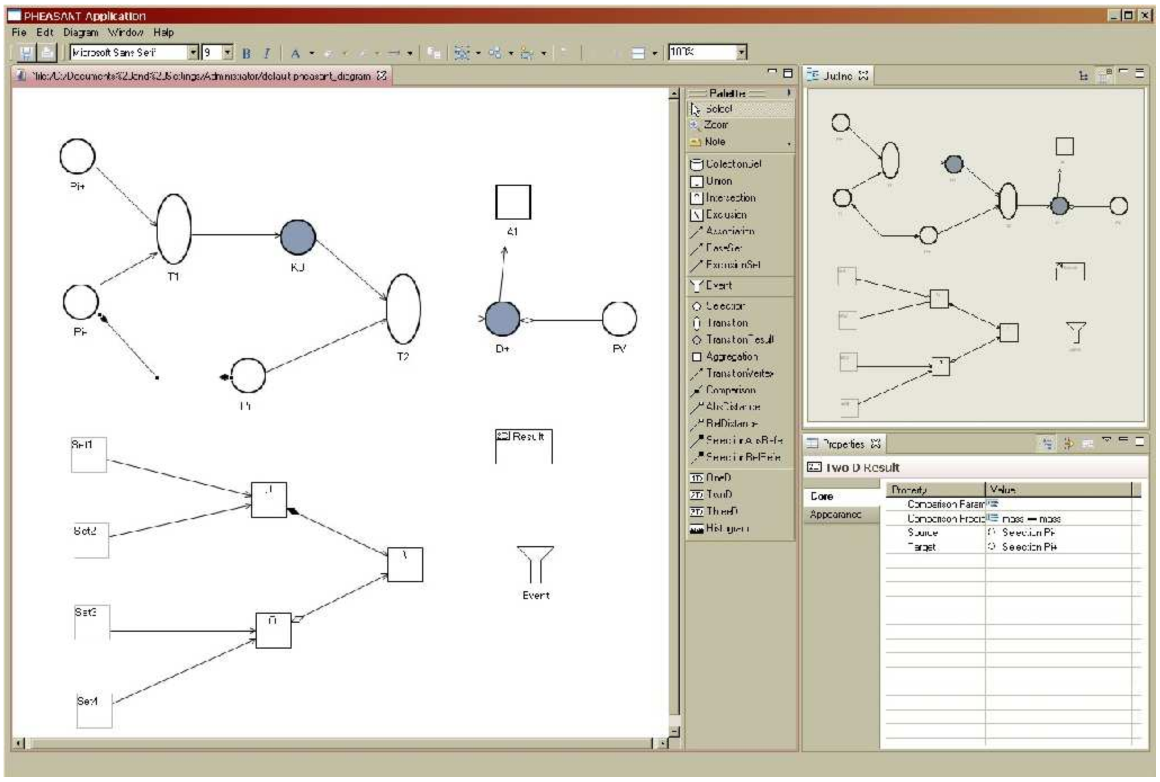


Figure 3. PHEASANT editor generated by GMF. Example of a query modeling the reconstrucion of the decay $D^+ \rightarrow \pi^+ K_s^0 \rightarrow \pi^+\pi^+\pi^-$.

In our case we were capable of generating two editors in chosen DSL technologies: GME and GMF. With these editors it is possible to express, non trivial requirements, such as full reuseability of the user query models and text parsing.

The implementation of an editor in GMF for the PHEASANT language proved that this type of technology can be adequate for the purpose of HEP analysis because it can cope with the majority of the requirements raised for the purpose of the development of a DSL for HEP physics data analysis.

As future work we must first proceed with usability tests in order to extend the proposed language and enhance the language editor's features. We are considering the possibility of creating an Eclipse plug-in capable of dinamicaly collect the data schema of the target framework, and parse the non visual parameters of the query at editor level creating better user interfaces. This future work must also include to work on a good User Interfaces.

With our evaluation, there are still some questions unanswered that should be further investigated with the GMF prototype. For instance, the need for dynamic data model inclusion in the metamodel to face the need of using the same language editor for different HEP experiments with different data-models and even to deal with user defined data sets. All of this work is considered at the editor's level, however further work must be done at the level of the query transformation model that will deliver the generated query code. Once that work is accomplished we will finally be at the position to deliver a stable and reliable tool to the end-users (the physiscits).

References

[1] Sousa V 2007 *A visual Domain Specific Language for Modeling High Energy Physics queries. A comparative study of technologies and PHEASANT implementation* (Lisbon: Universidade Nova de Lisboa).

[2] Amaral V 2005 *Increasing productivity in high energy physics data mining with a domain specific visual query language* (Mannheim:University of Mannheim)

[3] Amaral V, Helmer S and Moerkotte G 2005 *Formally specifying the syntax and semantics of a visual query language for the domain of high energy physics data analysis Proc. Int. Symp. on Visual Languages and Human-Centric Computing (VL/HCC'05)* ed. Erwig M., Schurr A Los Alamaitos,CA:IEEE pp 251-8

[4] Amaral V, Helmer S and Moerkotte G 2004 *PHEASANT: A PHysicist's EASy ANalysis Tool* Lecture Notes in Artificial Intelligence (LNAI) (Berlin:Springer) **3055** 229

[5] Amaral V, Helmer S and Moerkotte G 2004 *Engineering a New Abstraction Layer to Optimize the HEP Analysis Process* TNS **51** 4

[6] Amaral V, Helmer S and Moerkotte G 2004 *Domain Specific Visual Query Language for HEP analysis or How far can we go with user friendliness in HEP analysis? Proc. Int. Conf. CHEP'04*

[7] Amaral V, Helmer S and Moerkotte G 2003 *A Domain Specific Visual Query Language for the High Energy Physics Environment* Proc. 3rd OOPSLA Workshop on Domain-Specific Modeling (DSM) ed. Tolvanen, Gray J and Rossi M (Finland: University of Jyväskylä:) pp 9-16

[8] Warmer, J. and Kleppe, A. 2003 *The Object Constraint Language: Getting Your Models Ready for MDA* (Addison-Wesley)