UNIVERSITY OF ANTWERP

MODEL DRIVEN ENGINEERING

ROLE PLAYING GAME FORMALISM WITH MIC TOOL SUITE

Report II - Implementation part

Daniel Dragojevic

daniel.dragojevic@student.uantwerpen.be

Abstract

This paper describes the Model Integrated Computing (MIC) approach for modelling Role Playing Game (RPG) formalism. Core of MIC approach is on the formal representation, composition, analysis, and manipulation of models during the design process. This paper will give an overview and describe the possibilities of MIC tool suite through the implementation of RPG formalism.

Keywords: MDE, MIC, Generic modelling, GME, Model management, UDM, Model transformation, GReAT, Role playing game, AToMPM

Contents

1 INTRODUCTION **4**

2 GME, UDM and GReAT **5**

2.1 Generic Modeling Environment 6
2.2 Universal Data Model 7
2.3 Graph Rewrite And Transformation 9

3 ROLE PLAYING GAME **10**

3.1 Step 1 - GME 10
3.2 Step 2 - UDM 14
3.3 Step 3 - GReAT 14

4 COMPARISON GME AND AToMPM **18**

4.1 AToMPM 18
4.2 Comparison 19

5 CONCLUSION **20**

List of Figures

1 MIC development cycle [2] 5
2 GME modelling concepts [4] 6
3 UDM framework with modules [5] 8
4 GME interface 11
5 Class diagram 12
6 Attributes 12
7 RPG aspect 13
8 RPG overview 14
9 GReAT configuration elements 16
10 GReAT transformation elements 16
11 Example - 2 inputs and 3 rule elements 17
12 RPG overview in AToMPM 19

1. INTRODUCTION

Model Driven Engineering (MDE) is a software development methodology which promotes an idea of raising the level of abstraction in program specification and improving automation in development. Automation improvement in development is reached by using executable model transformations. Idea is to transform higher level models into lower level models until the model can be made executable using code generation or model interpretation. The MDE principles may be implemented by different standards like *Model Driven Architecture* (MDA) [1], *Model Integrated Computing* (MIC) [2], *Microsoft Domain Specific Languages* (MSDSLs) [3] and many others.

In this paper I will present MIC approach. MIC is developed at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University. Core of MIC is on the formal representation, composition, analysis, and manipulation of models during the design process. Three main elements of this approach are: the technology for the specification and use of domain specific modelling languages (DSML), the fully integrated metaprogrammable MIC tool suite, and an open integration framework to support formal analysis tools, verification techniques and model transformations in the development process.

The most popular MIC tools are the generic model editor named *Generic Modeling Environment* (GME) [4], the model management tool suite named *Universal Data Model* (UDM) [5], the model transformation tool suite named *Graph Rewrite And Transformation* (GReAT) [6], the *Design Space Exploration Tool* (DESERT) [7].

Figure 1 shows the MIC development cycle. Cycle starts with the formal specification of a new application domain. We need to identify concepts, attributes, and relationships. This is process of metamodelling. Metamodel defines the abstract syntax, static semantics and visualization rules of the domain. The visualization rules determine how domain models are going to be visualized and manipulated in a visual modelling environment. With specified domain we are able to generate a Domain Specific Design Environment (DSDE). The DSDE can then be used to create domain specific designs/models. However, to do something useful with these models such as synthesize executable code, perform analysis or drive simulators, we have to convert the models into other formats like executable code, inputs to some analysis tool, or configuration files for simulators. This process is called model interpretation and it includes model interpreters. Model interpreters

can convert models of a given domain into some other format, typically with a different semantic domain. The output of the transformation can be considered as another model that conforms to a different metamodel and thus model interpreters can be considered as tools facilitating a mapping between domains [2].

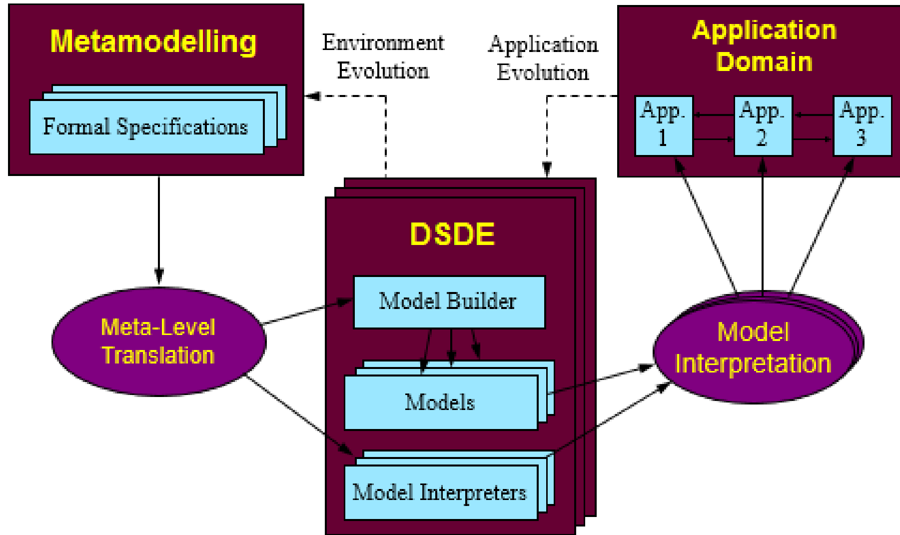

Figure 1: MIC development cycle [2]

Aim of this paper is to present possibilities of MIC tool suite for implementation of *Role Playing Game* (RPG) formalism. In the next section I will give a brief explanation of each tool required for each step of the implementation: GME, UDM and GReAT. Section 3 gives explanation of modelling process of role playing game formalism in GME. Section 4 gives comparison of GME and *A Tool for Multi Paradigm Modeling* (AToMPM) [10]. Section 5 concludes this paper.

2. GME, UDM and GReAT

In this section I present three tools for implementation of RPG formalism. I give a brief description and concepts of tools which can be used for the each step of building process.

2.1. Generic Modeling Environment

Generic Modeling Environment was covered in the first paper [2] but because importance of this tool I repeat some of the highlights. GME is a metaprogrammable, domain specific, graphical editor supporting the design, analysis and synthesis of complex software intensive systems. GME supports higher level abstractions than programming languages like C++/C or Java, and modelling languages like Unified Modeling Language (UML). This means that we need less effort and fewer low level details to develop a given system. In addition, GME allows users to define new modelling languages using metamodels to describe the rules, constraints, and concepts useful for modelling a class of problems.

GME metamodels must be created using the MetaGME paradigm, which is installed and registered with GME. Metamodelling level of GME provides generic modelling primitives that assist an environment designer in the specification of new modelling environments. These concepts are directly supported by the framework as stereotypes of the specific classes. Figure 2 shows GME modelling concepts.

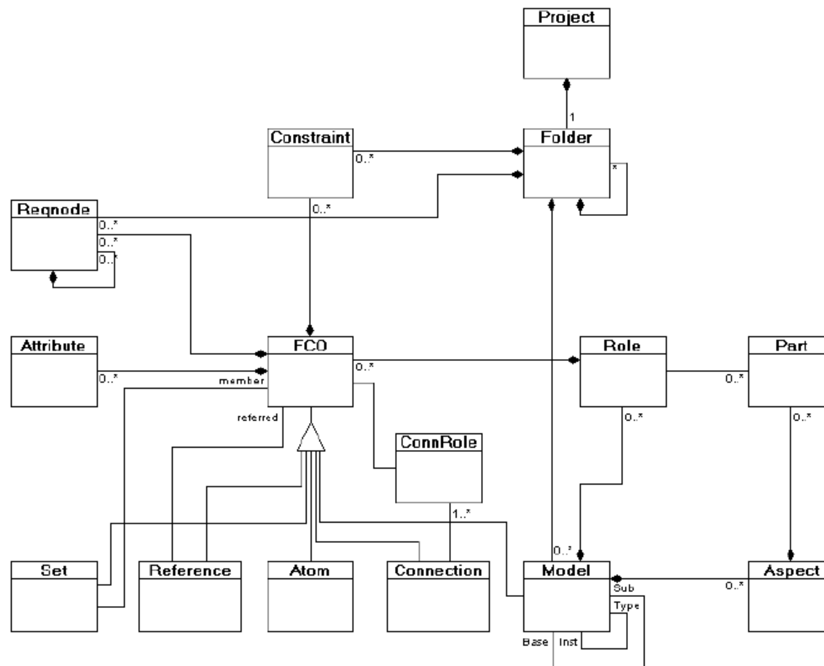

Figure 2: GME modelling concepts [4]

A *Project* contains *Folders*. Folders are containers for easier organization of models. Folder contains *Models*. Models are container elements which can have parts and inner structures. A part in a container Model always has a *Role*. The modelling paradigm determines what kind of parts are allowed in Models acting in which Roles. *Atoms* are atomic elements, i.e. they do not contain any other object. Associations are modelled using the connection primitive that is visualized by the modelling tool as a line between the objects. *Connections* are used to express relationships between objects at the same hierarchy level or one level deeper. To overcome this limitation we can use *References* to associate objects in different model hierarchies. Connections and References model relationships between at most two objects. To group more elements we can use *Sets*.

Atoms, models, connections, references and sets are called the *First Class Objects* (FCO) in GME. FCOs can contain both textual *Attributes* and *Constraints*, which are Object Constraint Language (OCL) based expressions for providing verifiability for the models. Another important concept in GME is the *Aspect* (viewpoint). Every model has a predefined set of Aspects and each part has option to be visible or hidden.

2.2. Universal Data Model

The Universal Data Model is a metaprogrammable tool for providing uniform access to data structures that could be persistent. The tool uses UML class diagram as the language for defining the data structures and it generates C++ or Java class definitions for implementing the classes. Each attribute and association have a corresponding setter/getter method in the generated code. The generated class implementations are pointing to generic objects that provide the real implementation. These generic objects can be persistent and be mapped into: Extensible Markup Language (XML) files, GME project files, database tables, or Common Object Request Broker Architecture (CORBA) structures. For each kind of generic object there is a separate back end library that implements the objects in terms of the underlying technology.

Figure 3 shows UDM framework with the following modules:

- *GME UML environment* with the *GME UML paradigm* and *interpreter*: Used to generate equivalent XML files from GME UML models,
- *UDM program*: Used to read generated XML files, and generate the metamodel dependent portion of the UDM API: a C++ source file, a

C++ header file, and an XML Document Type Description (DTD) or XML Schema Definition (XSD),

- *Generic UDM*: Used to include headers and libraries which can be linked to the users program, and
- *Utility programs*: Used to manipulate and query UDM data.

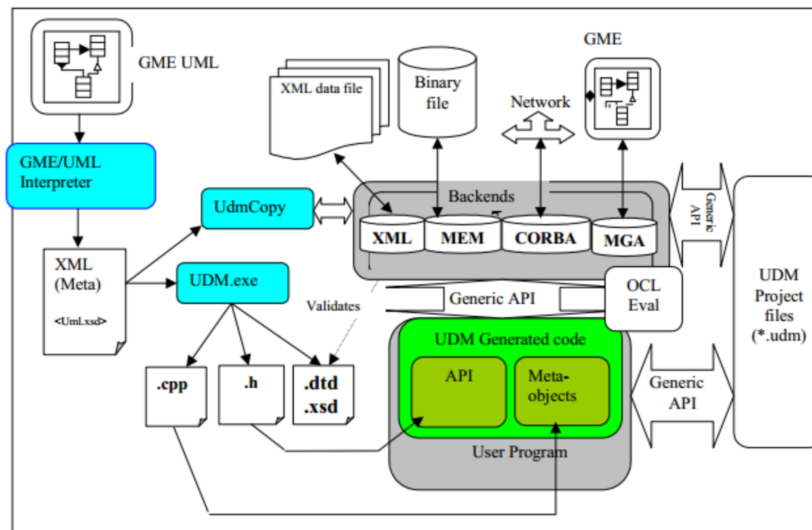

Figure 3: UDM framework with modules [5]

The typical process of using the UDM starts with a UML metamodel which is created in GME. The information in the UML diagram is converted to an XML file using the GME interpreter. The format of these XML files is UDMs representation of UML class diagram information. In the case when the UML diagram contains multiple packages, the diagram is converted to a UDM Project file. We can use the UDM.exe program to generate the paradigm dependent API files. We need to includes these files, along with other, generic UDM headers libraries into a C++ project. Changes in the UML diagrams are followed by the same procedure. Since most changes also change the generated API, modifications in the programs that use it may also be required.

2.3. Graph Rewrite And Transformation

GReAT is the model transformation language for the domain specific modelling tool GME. Based on the type graph, GReAT supports the exogenous transformation and allows the specification of cross domain transformations, where models belonging to different domains can be transformed simultaneously.

GReAT consists of three sublanguages: *the pattern specification language*, *the transformation rule language* and *the control flow language*.

Pattern Specification Language

The pattern specification language is used to specify the graph patterns that will form the *Left Hand Side* (LHS) and *Right Hand Side* (RHS) of the rule. LHS and RHS rules are part of the graph transformation where the LHS specify the precondition and the RHS the postcondition. Unlike some other approaches (e.g. AToMPM) in GReAT both the LHS and RHS are specified together as a single pattern.

Transformation Rule Language

The transformation rule language is used to specify the transformation rules. Each object in the graph pattern plays a specific role in the transformation. There are three types of roles:

- *Bind*: Used to match objects in the graph,
- *Delete*: After match the object is deleted, and
- *New*: After match the object is created.

According to the general form of a the transformation rule, the objects marked as *Bound* can be considered as the LHS and the objects marked as *New* or *Delete* are the RHS. In certain case, the pattern itself is not enough to take decision of the matching, so we need additional nonstructural constraints on the pattern. A C++ code can be placed in Guard to restrict the execution of the rule if certain conditions hold. The rule execution implies the creation or deletion of objects and the values of objects attributes are affected by the transformation, so the *Attribute mapping* specifies the update by a set of assignment statements. The application of a rule depends on the objects (Packets) supplied by the previous rule, the *In* and *Out ports* are used to pass objects between rules.

Control Flow Language

The model to model transformation is the result of sequencing the application of rules, thus the control flow language is used to specify the flow of the application of rules and manage the complexity of the transformation process. The language supports the following features:

- *Sequencing*: Rules can be connected and executed in sequence,
- *Non-determinism*: Rules are executed in parallel where the order of execution is non-deterministic,
- *Hierarchy*: High level or compound rule can contain other compound rules or primitive rules,
- *Recursion*: High-level rule can call itself, and
- *Conditional execution*: Test/Case construct used to choose between different control flow paths.

3. ROLE PLAYING GAME

In the paper [8] I gave an overview of the role playing game. Based on that I implement an idea with the tools mentioned in Section 2. I start with GME, continue with UML and finish with GReAT.

3.1. Step 1 - GME

The first thing one must do using GME is to define a sketch of a metamodel, which is basically a UML Class Diagram extended with some additional concepts. These additional concepts include OCL constraints and also some GME specific features such as configurable model visualization properties. To access metamodel environment we need to use MetaGME paradigm. This is the starting point for implementation of RPG formalism.

Creation of project

When we start GME a pop up dialog box gives an option to start a new project or continue from the previous one. In this case we choose the new project and the MetaGME paradigm on which the project will be based on. Application warns us to save our project immediately. Saved file is

in the .mga format (RPG.mga). Till this point we have an empty project named RPG, based on the MetaGME paradigm and containing just a root folder. We continue by creating the paradigm sheet to the root folder which is accessible in the browse window of GME. New paradigm sheet gives us an empty window in the user area where we can start to define entities and relationships (Figure 4).

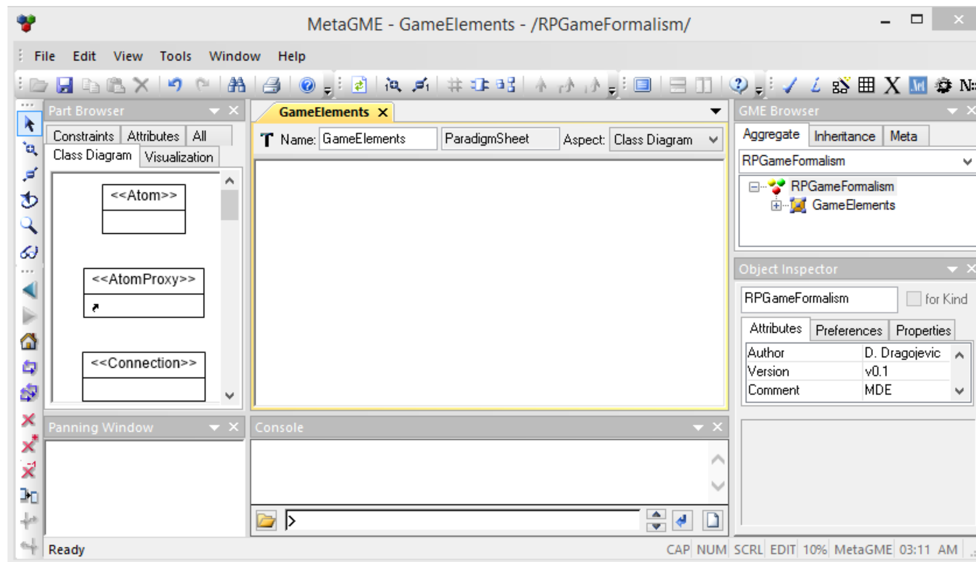

Figure 4: GME interface

The part browser window of GME shows available objects which we use to define our model. Objects are organized in several tabs: *class diagram*, *visualization*, *constraints* and *attributes*.

Definition of entities and relationships

The class diagram aspect tab offers necessary objects to define our RPG formalism. Each of objects has access to the attribute panel where we can edit attributes, preferences and properties (e.g. icons, size, colours, etc.). *Connection* between objects involve: the source, the destination and the connection itself. We need special *Connector* objects to define each type of relationship between objects. Connections are created in *Connect mode* which we choose from the toolbar on the left hand side in GME. Following figure shows the RPG class diagram.

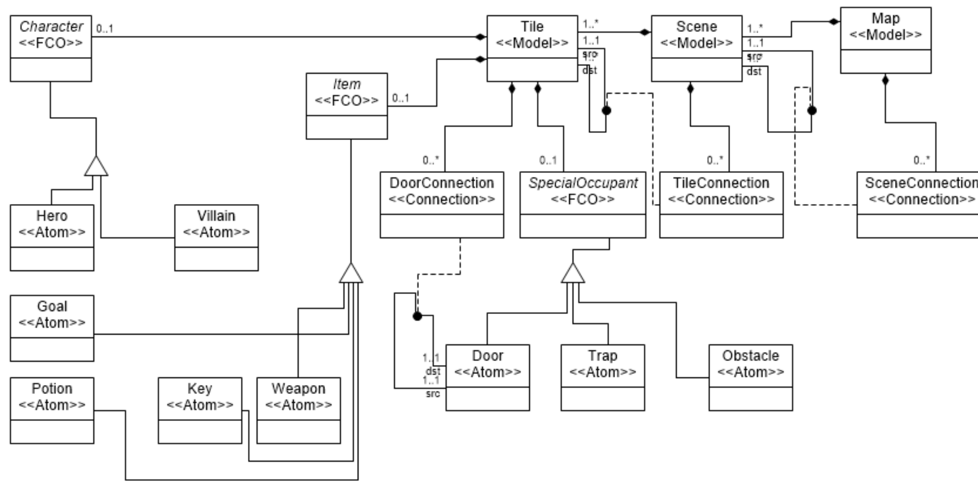

Figure 5: Class diagram

Attributes definition

Next step is to define attributes to our paradigm to make it more realistic. It is necessary to switch to the Attributes aspect tab in the part browser window to access attribute objects. Each attribute needs to be defined in the separate object. Final step is to connect game element(s) with associated attribute. Figure 6 shows the final attribute aspect.

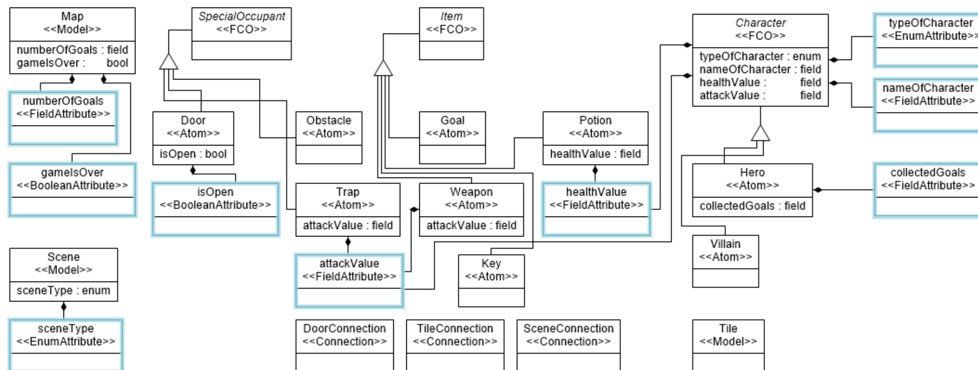

Figure 6: Attributes

Aspect definition

After attributes, we need to define at least one aspect for our paradigm. By switching to Visualization aspect tab in the part browser window, we are able to choose a type of object named Aspect. This object creates visualization tab for our paradigm. It is necessary to include all objects which we want to make available later. Abstract classes are not going to be selected. Following figure shows defined RPG aspect for our paradigm.

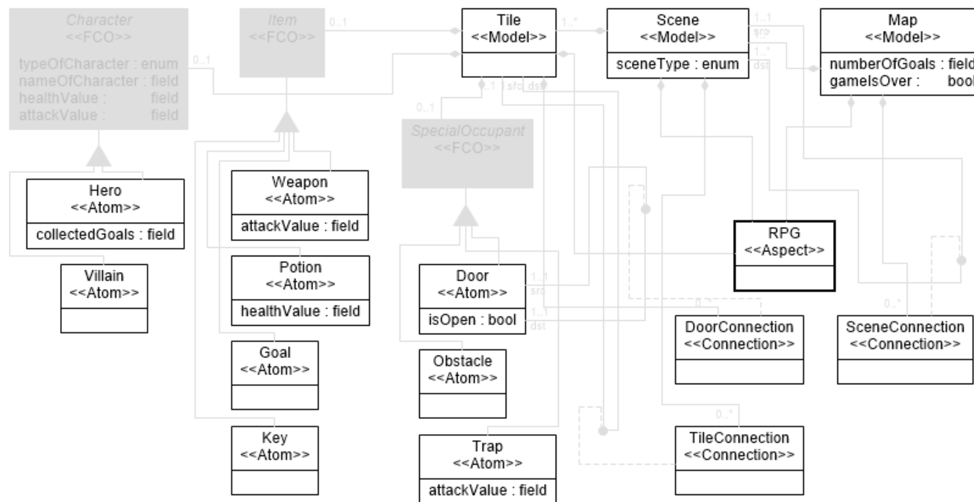

Figure 7: RPG aspect

Interpretation of metamodel and installation of new paradigm

This brief overview of implementation described necessary steps for creation of new paradigm. Before we can create models using our new paradigm we need to interpret the metamodel. To create RPG paradigm we need to use MetaInterpreter. Interpreter generates XMP file containing the paradigm specifications and we are able to register our paradigm, *RPGameFormalism.*

Creation of domain specific model

To create domain specific models we start new project and in the paradigm dialog box this time we need to select our newly created paradigm. We need to add a new Map in the root folder which enables full view of RPG aspect of game elements in the part browser of GME. We continue with adding Scenes and Tiles in Scenes to build our game map. Moreover, we can add other game element into tiles. Figure 8 shows RPG overview.

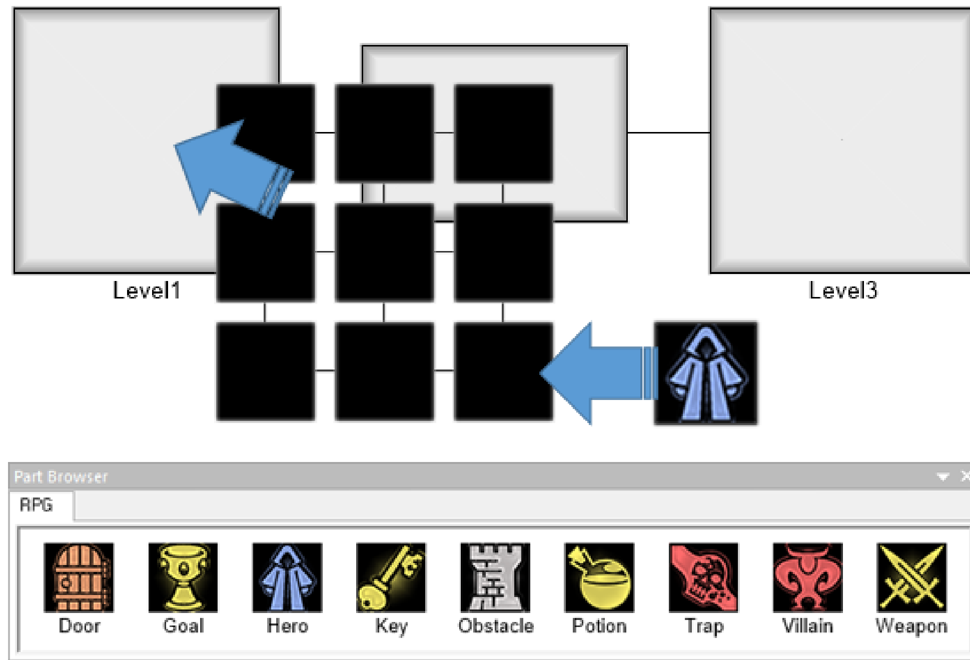

Figure 8: RPG overview

3.2. Step 2 - UDM

Once we have a GME metamodel, we also have to create a meta for the UDM environment. This process is not automated, so UDM meta must be created manually. Reason for this is another tool, GReAT, which we need to use afterwards. GReAT is built on top of UDM, and UDM works with metamodels in the UML paradigm. That is why we need to convert our GME metamodel into UDM style metamodel. The UML class diagram can be generated from the GME meta using the GME2UML interpreter. The next step is to interpret the UML class diagram with UML2XML interpreter that comes with UDM. This interpretation process creates a Document Object Model (DOM) backend of the metamodel.

3.3. Step 3 - GReAT

From this point is possible to say that building a role playing formalism with the MIC tool suite is not effortless process. MetaGME part of implementation goes more or less fluent. Good overview and simple class connection gives an easy and powerful control. Mappings from models to a

semantic domain are performed by model interpreters. Problem comes with this step. Need for different interpreters brings time consuming process with deep knowledge of MIC insights. We can conclude that there is a need for higher level methods and tools for building model interpreters. These generic tools should automatically generate domain specific model interpreters from models. This gap of GME makes implementation of RPG formalisms really hard. Moreover, need to work with several tools on one project makes building process slow and confusing. Till this point I was able to follow all instructions of the implementation but next steps were problematic. All tutorials, papers and knowledge about this tools did not provide any clear answer why my UDM transformation did not provide the correct files for the next step - transformation with GReAT tool. My assumptions are that: tools that I used are not well configured during the last updates, possible changes in the implementation steps, incompatibility with my operating system (Windows 8.1) or version of Visual Studio (2013), or maybe my mistake in the configuration and conjunction of this tools. Even that I am not able to continue with the implementation I will give a possible solution for GReAT part.

As I stated above, next step in implementation needs to continue with GReAT tool. I created a new UMLModelTransformer project in GME. Here we need to use the UMLModelTransformer paradigm. File is named RPG-Transformation.mga and it is available in project source files. Further step is to attach the GME metamodel to our UMLModelTransformer project. In this moment we need to go back to our MetaGME environment and convert our metamodel with GME2UML interpreter. We need to save over UMLModelTransformer project we created earlier. This allows us to attach the UML style metamodel to the UML Model Transformer project. When we are still in MetaGme environment we need to run MetaGME interpreter to save any new change in our metamodel and ensure that that our GReAT transformation wil succeed later.

Now we need to go to our transformation project file, *RPGTransformation.mga*, and continue with implementation. However, this is my last point of implementation. In the moment when when we converted our metamodel with GME2UML interpreter, program does not attach the our metamodel to the UML Model Transformer project. Without this connection I am not able to continue. With next steps I needed to build *Configuration model* with necessary elements. Figure 9 shows configuration elements which are available in the Part Browser of GME/GReAT environment.

Figure 9: GReAT configuration elements

- *File*: This element gives the name and path to the input model(s). This file needs to be connected to a FileType object,
- *FileType*: This element gives the metamodel of the input file attached, along with various other information, and
- *MetaInformation*: This element lists the UDM project file used by the transformation along with the GR file which contains the rewriting rules in an internal format used by GReAT.

Moreover, I needed to specify the transformations. This transformations are are modelled with elements from the Figure 10:

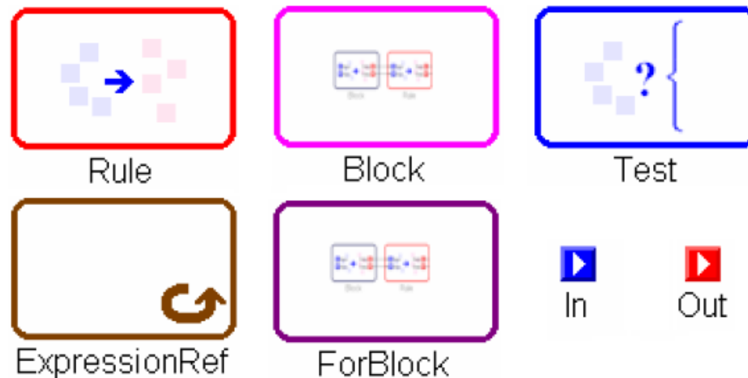

Figure 10: GReAT transformation elements

- *Rule*: The basic rewriting element. Each rule contains a graph pattern composed from UML classes.
- *ExpressionRef*: This element is reference to rules that have already been defined. This elements are useful if we want to express recursion in our transformation.

- *Block* and *ForBlock*: These elements allow grouping of rules together. The difference between this two elements is in the order incoming packets are passed to the rules which are contained inside. A ForBlock pushes the first incoming packet through all of its contained rules, then pushes the second packet all the way through, etc. A Block pushes all the packets to the first contained rule. The first rule generates a number of packets, which are passed along together to the subsequent rules, etc.

- *In* and *Out*: These elements are ports that allow passing objects from one rule to another.

- *Test*: This element is similar to an If/Else construct in textual programming languages.

Following figure shows an example how rules would need to be organized. Example has two inputs and three Rule elements connected. In the each of Rule element we can define some rules to transform inputs. Output of each Rule element is input of another one.

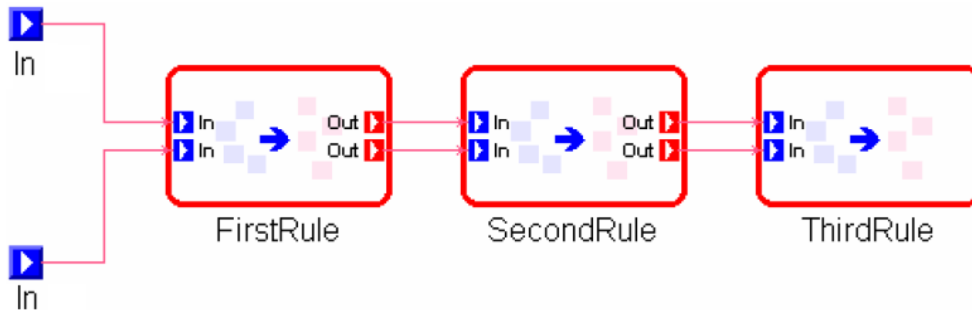

Figure 11: Example - 2 inputs and 3 rule elements

Here I finalize my overview of the implementation. From my research I can conclude that transformation with this tool is complex and highly time consuming. In this overview I gave just brief description of GReAT. For more information I recommend reading paper [6].

4. COMPARISON GME AND AToMPM

This section brings comparison of two powerful tools for metamodelling: GME and AToMPM. Through this paper and previous report we were able to get more familiar with generic modelling environment. This gives us opportunity to see which tool would be a better choice with building RPG formalism and some other similar projects. Next subsection gives brief description of AToMPM. Section 5.2 compares GME with AToMPM.

4.1. AToMPM

AToMPM is a open source research framework for designing DSML environments, performing model transformations, and manipulating and managing models. AToMPM runs completely over the web what makes it independent from any operating system, platform, or device. It provides modern graphical user interface for dening the metamodels, describing rules graphically, controlling structures for model transformations, and executing step by step transformations for given model.

AToMPM supports real time distributed collaboration. There are two modes: *screenshare* and *modelshare*. Developers have option to share the same concrete and abstract syntax. This option ensures fast and easy developing process and representation between more people.

Defining a new formalisms in AToMPM goes fluent. There are three general steps which we need to follow:

- Model definition and compilation of the formalism's abstract syntax,
- Model compilation of the formalism's concrete syntax,
- Model the formalism's operational semantics (meaning).

As I state above in the text, AToMPM's transformations and transformation rules are organized different from the GReAT approach. Transformations and transformation rules in AToMPM are instance models of the Transformation and TransformationRule formalism respectively. Transformations in AToMPM are used to specify how rules and/or transformations should be sequenced together. They are collections of connected transformation steps. Transformation rules consist of one RHS pattern, one LHS pattern and zero or more Negative Application Condition (NAC) patterns.

Following figure gives overview of AToMPM environment for role playing formalism.

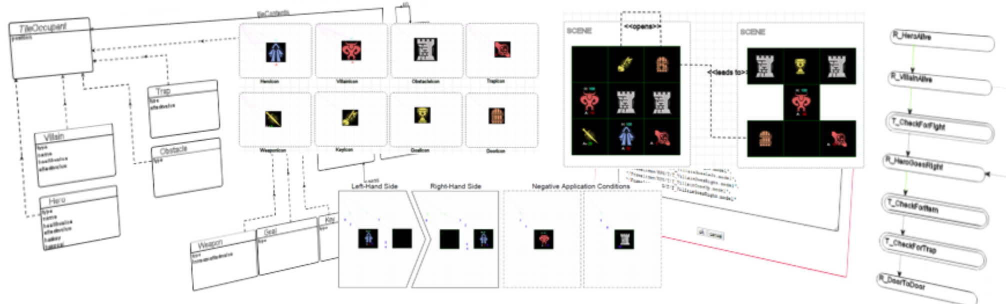

Figure 12: RPG overview in AToMPM

4.2. Comparison

In the previous subsection we were able to see a brief overview of AToMPM framework. This subsection will bring some more details about this tool and compare it with GME. Features, flexibility, interface and implementations vary from one tool to the other. Though I do not intend to provide an exhaustive, feature based comparison and review of this tools, my aim is to see which tool would be a better choice to build RPG and similar formalisms.

Installation and accessibility

Both tools require some kind of the installation of one or more applications on the user's computer. Disadvantage of GME is that this tool restricts the user to the Microsoft Windows operating system and Microsoft Visual Studion environment. AToMPM is developed to run in popular Web browsers (e.g. Google Chrome) and store user data in the cloud. This means that no installation of AToMPM application is required on the user's machine. This makes AToMPM extremely accessible for anyone from expert developer to curious novice.

Design and properties

GME, AToMPM and other modelling language have different approach to specify and design concrete syntax during the metamodelling phase. GME

in contrast to AToMPM has already defined generic types to provide a default concrete syntax. Modification of this property can easily be overridden at the metamodelling level. One of the interesting properties of GME is specification of different aspects (viewpoints). This property permits filtering of the visualization space for an intuitive subset of the design, as partitioned at design time. This makes overview, implementation and understanding of diagrams easier.

Implementation

As I stated in the subsection 3.3, implementation of the role playing game formalism in GME is really complex and time consuming. Implementation with AToMPM is going much faster and easier. AToMPM gives a full solution for all steps in the building process. However, AToMPM is still not 100 % reliable tool. It is new tool which has some issues. Development team needs to work hard on improvements of this tool to make it fully complete. Despite that, even in this stage, AToMPM is more powerful tool and winner for this task.

5. CONCLUSION

This paper presents possibilities of modelling role playing formalism with tools based on the Model Integrated Computing approach. The Generic Modeling Environment is a configurable toolset for creating domain specific modelling and program synthesis environments through a metamodel that specifies the modelling paradigm of the application domain. The metamodel contains descriptions of the entities, attributes, and relationships that are available in the modelling environment, and defines the family of models that can be created using the resulting modelling environment. GME gives powerful options to organize abstract syntax of RPG formalism. Provided interpreters and generic types in GME environment ensures easy design of concrete syntax. Additional properties in this tool provide more visual appealing solutions. Building process with GME goes fluent. However, GME alone can not provide transformations. As a result, there is a need to use other tools: Universal Data Model and Graph Rewrite And Transformation tool. Reason to use UDM is GReAT which is built on top of UDM. That is why there is a need to convert GME metamodel into UDM style metamodel. Implementation fails after interpretation from UDM to GReAT. There are

several assumptions: tools are not well configured or not simultaneous updated, incompatibility with operating system or version of Visual Studio, or maybe even own mistake in implementation steps. Based on everything, it is easy to conclude that building RPG formalism with this tools is complex and highly time consuming process. Need to work with several tools on one project and need to convert from one format to another is something what ensures mistakes and issues. Moreover, this paper compares above tools with another graphical modelling tool - AToMPM. AToMPM is new, modern, web based, open source research framework for designing DSML environments, performing model transformations, and manipulating and managing models. Building a formalism to model RPG in AToMPM is going much faster and easier. Flexibility and numerous properties ensures understandable building process even for person with minimum programming knowledge. This makes AToMPM better choice for this kind of projects.

REFERENCES

[1] J. D. Poole. *Model-Driven Architecture: Vision, Standards And Emerging Technologies.* ECOOP 2001, Budapest, Hungary, 2001.

[2] J. Sztipanovits and G. Karsai. Model-Integrated Computing. IEEE Computer, 1997.

[3] Microsoft Corporation. *Microsoft Domain-Specific Language (DSL) Tools.* Web source: http://msdn.microsoft.com/en-us/library/bb126259.aspx

[4] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle and P. Volgyesi. *The Generic Modeling Environment.* Workshop on Intelligent Signal Processing, Budapest, Hungary, 2001.

[5] E. Magyari, A. Bakay, T. Levendovszky, T. Paka. *The UDM framework.* ISIS, 2004.

[6] A. Agrawal, G. Karsai and F. Shi. *Graph Transformations on Domain-Specific Models.* Workshop on Domain-Specific Modeling, OOPSLA, 2003.

[7] S. Neema. *Design Space Representation and Management for Model-Based Embedded System Synthesis.* 2001.

[8] D. Dragojevic *Modelling language engineering with Generic Modeling Environment.* Report I - Reading part, 2013.

[9] G. Nordstrom, J. Sztipanovits, G. Karsai and A. Ledeczi. *Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments.* In the Proceedings of the IEEE ECBS'99 Conference, Tennessee, 1999.

[10] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo and H. Ergin. *AToMPM: A Web-based Modeling Environment.* MODELS'13 Demonstrations, CEUR, Miami FL, USA, 2013.

ABBREVIATIONS

AToMPM	A Tool for Multi Paradigm Modeling
CORBA	Common Object Request Broker Architecture
DESERT	Design Space Exploration Tool
DOM	Document Object Model
DSDE	Domain Specific Design Environment
DSML	Domain Specific Modelling Languages
DTD	Document Type Definition
FCO	First Class Object
GME	Generic Modeling Environment
GReAT	Graph Rewrite And Transformation
ISIS	Institute for Software Integrated Systems
LHS	Left Hand Side
NAC	Negative Application Condition
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MIC	Model Integrated Computing
MSDSL	Microsoft Domain Specific Language
OCL	Object Constraint Language
RHS	Right Hand Side
RPG	Role Playing Game
UDM	Universal Data Model
UML	Unified Modeling Language
XML	Extensible Markup Language
XSD	XML Schema Definition