Modelling language engineering with GME
Report I - Reading part

Daniel Dragojevic

University of Antwerp
daniel.dragojevic@student.uantwerpen.be

Abstract

This paper describes the Generic Modeling Environment (GME), a configurable graphical modeling toolsuite that supports the creation of domain specific modeling, model analysis and program synthesis environments. Moreover, it gives a brief explanation of the procedure for the implementation of the role playing game project.

Keywords: Generic modeling environment, GME, modeling, MetaGME, metamodeling, role playing game, RPG

1. INTRODUCTION

Generic Modeling Environment (GME) [1] is a meta programmable, domain specific, graphical editor supporting the design, analysis and synthesis of complex software intensive systems. GME is developed at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University.

GME supports higher level abstractions than general purpose programming languages (such as C++/C and Java) and general purpose modeling languages (such as UML), so they require less effort and fewer low level details to develop a given system. GME also allows users to define new modeling languages using metamodels, which describe the rules, constraints, and concepts applicable and useful for modeling a class of problems.

I will use capabilities of GME to model a role playing game (RPG) similar to one in the course Model driven engineering. My previous modelling experience was with metaDepth [2] and AToMPM [3] tools which gives me opportunity to compare different (or similar) concepts and principles of this tools during the implementation.

To summarize, aim of this paper is to give the brief explanation and concepts of Generic Modelling Environment and an idea for further work with this tool. Section 2 gives GME overview. Section 3 gives description and starting point for project implementation. Section 4 concludes this paper.

2. GME OVERVIEW

In this section, I first give the technical overview of GME. Moreover, I describe the MetaGME paradigm which we can use to create metamodels with GME tool.

2.1. Introduction to GME

The first thing one must do using GME is define a sketch of a metamodel, which is basically a Unified Modeling Language (UML) Class Diagram extended with some additional concepts. These additional concepts include defining any necessary Object Constraint Language (OCL) constraints and also some GME specific features such as configurable model visualization properties. After the metamodel is initially defined, it can be iteratively refined until it reaches a mature state that captures all pertinent features of the domain.

2.2. Technical Overview

In this subsection I will show the technical overview of GME. Figure 1 shows a modular, component-based architecture of this tool. I will explain each of GME components based on information from the paper *The Generic Modeling Environment* [1] and *GME Manual and User Guide* [4].

Storage: The thin storage layer includes components for the different storage formats. Supported formats are proprietary binary file format and an XML format.

Core component: This component implements the two fundamental building blocks of a modeling environment: objects and relations. Among its services are distributed access (i.e. locking) and undo/redo.

MgaMeta and *MgaModel*: This two components use the services of the Core. The MgaMeta defines the modeling paradigm, while the MgaModel implements the GME modeling concepts for the given paradigm. The MgaModel uses the MgaMeta component extensively through its public COM

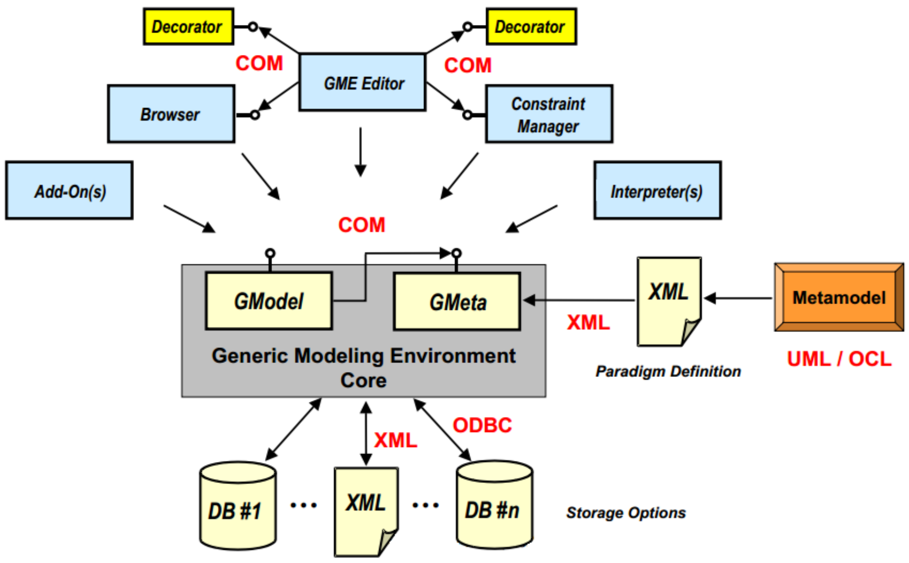

Figure 1: Modular, component-based architecture of GME

interfaces. The MgaModel component exposes its services through a set of COM interfaces as well.

Add-ons: They are event-driven model interpreters. The MgaModel component exposes a set of events, such as "Object Deleted," "Set Member Added," "Attribute Changed," etc. External components can register to receive some or all of these events. They are automatically invoked by the MgaModel when the events occur. Add-ons are extremely useful for extending the capabilities of the GME User Interface. When a particular domain calls for some special operations, these can be supported without modifying the GME itself.

Constraint Manager: This manager can be considered as an interpreter and an add-on at the same time. It can be invoked explicitly by the user and it is also invoked when event-driven constraints are present in the given paradigm. Depending on the priority of a constraint, the operation that caused a constraint violation are aborted. For less serious violations, the Constraint Manager only issues a warning message.

User Interface: This component has no special privileges in this architecture. Any other component (interpreter, add-on) has the same access rights and uses the same set of COM interfaces to the GME. Any operation that can be accomplished through the GUI, can also be done programmatically through the interfaces. This architecture is very flexible and supports extensibility of the whole environment.

2.3. MetaGME

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

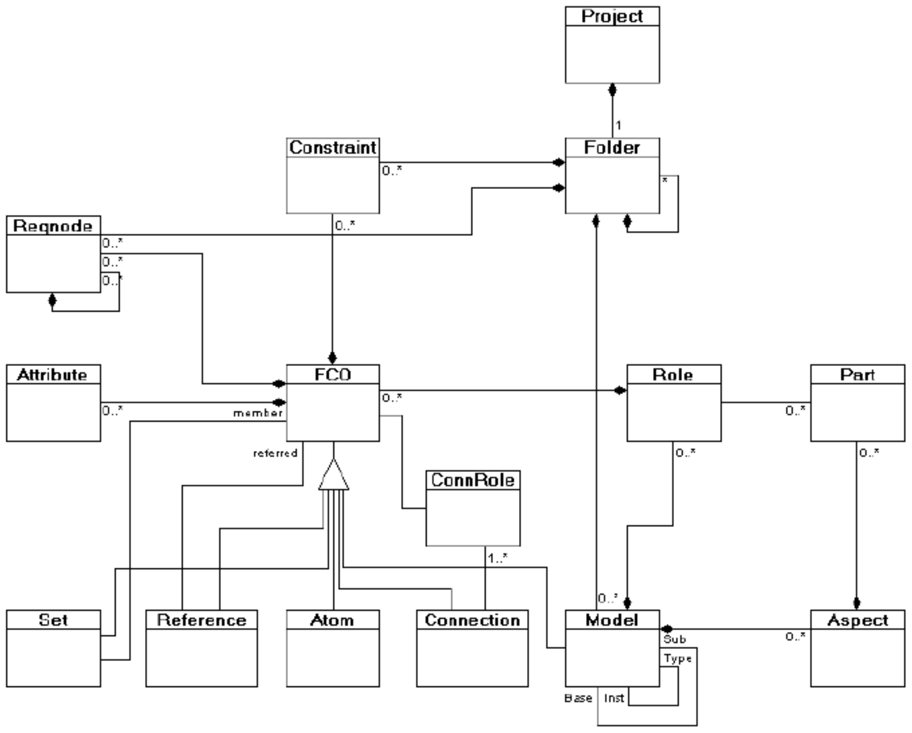

Figure 2: GME modeling concepts

A *Project* contains a set of *Folders*. Folders are containers that help organize models. Folders contain *Models*. Models are used to represent a container element. They can have parts and inner structure. A part in a container Model always has a Role. The modeling paradigm determines what kind of parts are allowed in Models acting in which Roles, but the modeler determines the specific instances and number of parts a given model contains. *Atoms* are used to represent an atomic element, i.e. they do not contain other objects. Associations are modeled using the *Connection* primitive that is visualized by the modeling 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 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. Figure 2 shows GME modeling concepts.

3. ROLE PLAYING GAME

In this section I present an overview of role playing game and the procedure which I will follow for the implementation of this project. This section will be covered in greater detail in the final project report.

3.1. Role playing game overview

The structure of an RPG is imagined as a world map which contains one or more scenes (levels). In each scene, there are a number of connected tiles. Scenes are connected with doors which can be locked. Tiles can contain one of the characters or other tile occupants as obstacles, weapons, health potions, keys, traps and goals. Simple RPG example is presented on the following figure.

There are two types of characters: hero and villain. Hero is main character which moves through scenes, fights villains and collects items. Game ends when hero collects all goals or dies. Characters cannot pass or step on the obstacles. The weapon gives additional strength to the hero. The health potion renews heros health value. Hero needs keys to open locked doors. Some doors will be unlocked and some will need multiple keys to unlock them. Traps can hurt hero if he steps on them. Hero will need to collect all goals to successfully finish the game.

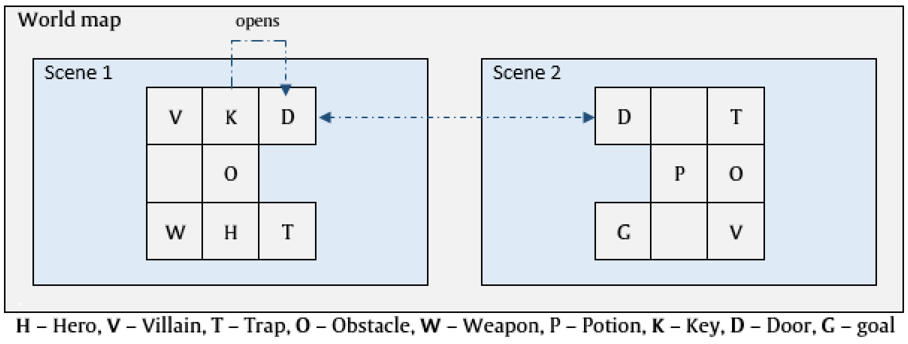
World map
opens
Scene 1
V
K
D
O
W
H
T
Scene 2
D
T
P
O
G
V
H – Hero, V – Villain, T – Trap, O – Obstacle, W – Weapon, P – Potion, K – Key, D – Door, G – goal

Figure 3: RPG example with two scenes

3.2. Constraints

The RPG meta model needs to be enriched with constraints to check if models are well formed. For example some of necessary constraints in the RPG game are that tiles needs to be well connected, there can be only one hero in the game, all values (health, strength) must be positive. More ideas will be implemented and reported during the further work.

3.3. Comparison with other domain specific modeling tools

This subsection is added as a guideline for the further work. Previous modelling experience with metaDepth and AToMPM during the implementation of RPG formalism gives me opportunity to compare concepts and principles of this tools with GME. My findings will be reported in the next paper.

4. CONCLUSION

Generic Modeling Environment is a flexible, generic, modeling environment useful for developing domain specific modeling toolsets. GME has a metamodeling interface for specifying the domain specific languages. It provides model construction facilities and automatic constraint satisfaction verification. GME has varied interfaces for developing model transformation tools. This gives us a freedom to model any project on easy and graphical way. In my case, role playing game will be build with this tool with additional improvements with some other programming languages (Java or Python). In this early stage it is not possible to conclude anything more about implementation process. This part will be covered in the next report.

5. REFERENCES

[1] 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.

[2] J. de Lara and E. Guerra. *Deep meta-modelling with metadepth.* 48th International Conference, TOOLS'10, Mlaga, Spain, 2010.

[3] 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.

[4] Institute for Software Integrated Systems, Vanderbilt University. *GME Manual and User Guide.* Nashville, USA, 2013.