# APPLYING MDA TO GAME SOFTWARE DEVELOPMENT

Takashi Inoue and Yoshiyuki Shinkawa
*Graduate School of Science & Technology, Ryukoku University, 1-5 Seta Oe-cho Yokotani, Otsu, Shiga, Japan*

Keywords: Game software, Modeling, MDA, UML.

Abstract: Game software becomes more and more complex, according as the game platforms are improved or the requirements of game users become sophisticated, and so does the development of game software. However, there are few established development methodologies for game software development, and it decreases the productivity of the development. One of the solutions for this problem is to apply modeling technologies to it as we do to other application areas, and through modeling we can anticipate productivity improvement. This paper evaluates the applicability and suitability of MDA (Model driven Architecture) to game software development, along with establishing a UML modeling process for typical game categories.

## 1 INTRODUCTION

Since Atari released VCS (Video Computer System) in 1970's, many game platforms and games that run on them have emerged. These games are usually implemented as software, and the scale and complexity of the game software have increased year by year, according as the game platforms are improved with higher performance CPUs, higher speed graphic engines, and larger size memories, or as the requirements to the games become sophisticated.

In many software application domains which include finance systems, insurance systems, plant control systems, and embedded systems, object oriented and model driven approaches won great success in software developments. However, there are no established methodologies in game software development, including object orientation and modeling. One of the difficulties in creating a standard methodology for game software development is that each game presents its unique appearance, behavior, and functionality, or in other words, they are too different from each other.

The lack of established methodologies would decrease the productivity of game software development, and makes it difficult to reuse software assets. The introduction of the above object oriented and model driven approaches into game software development seems to improve the productivity and reusability. These approaches would improve them especially in the large scale game software development, in the following points.

- Each project member can commonly understand the system to be developed
- The resultant system can be rigorously verified to the requirements
- The system configurations can be validated in early phases of development
- The scale of development can be predicted

In this paper we discuss the applicability of object orientation and modeling technologies to game software development, along with a modeling process. As a modeling framework, we focus on MDA (Model Driven Architecture) which is one of the state-of-the-art modeling technologies, and as a object oriented modeling language, we use UML (Unified Modeling Language), which is one of the industry-standard modeling languages.

The paper is organized as follows. In section 2, we discuss the scope of modeling, in which the applicability of the above technologies is examined. Section 3 defines a classification of games, in order to view the games at more abstract level for modeling. Section 4 presents a detailed modeling process in game software development.



Inoue T. and Shinkawa Y. (2008).
APPLYING MDA TO GAME SOFTWARE DEVELOPMENT.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - ISAS*, pages 454-459
DOI: 10.5220/0001726604540459

## 2 MODEL DRIVEN GAME SOFTWARE DEVELOPMENT

MDA is a development approach which focuses on modeling of target domains or systems. MDA defines two different kinds of models, namely PIM (Platform Independent Model) and PSM (Platform Specify Model).The former represents the models that are independent to the platforms, which represent a target problem domain at higher abstract level, in order to depict and understand what it essentially is. On the other hand, the latter represents the models which show how the domain is implemented on a specific platform. PSM could automatically be transformed from PIM using some tools in MDA (Frankel, 2003). We mainly focus on the analysis and high-level design phases in game software development. Therefore, this paper deals with only PIM, that is, the models independent from platforms, in order to evaluate whether we can model game software based on MDA.

Up to now, we have been designing game software in programming-oriented style, mainly using script languages, which is relatively old-fashioned in comparison with modern model oriented approaches (Nishimori and Kuno, 2003). This old-fashioned style often decreases the productivity of game software development. If we can apply MDA to game software development, we would anticipate the improvement of productivity, since there are many advantages of modeling to the above old-fashioned development style, as stated in the previous section.

In general, there are two orthogonal aspects in software modeling. One is “static” or “structural” aspect, and the other is “dynamic” or “behavioral” aspect. Most games implemented as software show very dynamic appearances. They are composed of many game elements, namely characters including the protagonists, vehicles, weapons, animals, flowers, and so on. Each game consists of the complicated combination of the behavior of these game elements along the timeline. Therefore the dynamic aspect seems more important in game software than the static aspect, and we mainly focus on the dynamic aspect of the games in order to evaluate whether we can express them as the models in terms of MDA.

We use UML as modeling tool, since UML is one of the industry standard modeling languages in object orientation (Miles and Hamilton, 2006). We first take the following five diagrams into account, that is, “use case”, “activity”, “sequence”, “state machine”, and “timing” diagrams, which represent the behavioral aspects in various ways.

## 3 THE CLASSIFICATION OF GAME SOFTWARE AND THE SCOPE OF MODELING

There have been released numerous number of games. Each of them has unique characteristics and very different from each other, in its behavior, appearance, construction of the game, and operation. This uniqueness is important to make each game competitive in the market, however it makes it difficult to view the games from common viewpoints in order to express and analyze them through modeling. From modeling viewpoints, it is important to view the games at more abstract levels. One of the approaches to abstraction is *classification* using criteria. There are several possible criteria such as *genres*, *platforms*, *prices*, *degrees of difficulty*, and *purposes*.

Among the above criteria, *genres* are suitable for abstraction to model the games, since the games in the same genre would show the similar behavior. The typical genres of video games and computer games are as follows (Laird and Lent, 2000) (Fairclough and Cunningham, 2001).

- Action games: A human player controls a character in a virtual environment. These games emphasize combat against enemies using various weapons.
- Role playing games: A human player may select a favorite character from different types of characters, such as a warrior, a magician, or a thief. The player goes on quest, trading items, fighting with monsters, or changing the capabilities.
- Adventure games: A human player solves puzzles and interacts with other characters, as he/she progresses through an unfolding adventure. These games emphasize story, plot and puzzle solving.

In addition to the above three genres, there are other game genres of *strategy games*, *god games*, *team sports games*, *board games*, *table games*, *simulation games*, *shooting games*, and so on.

Even though there are many game genres, recent market trends show the majority of games can be categorized into one of the three major genres, namely action games, role playing games, and adventure games. Therefore, this paper discusses the adaptability of modeling technologies to the games in these three genres. We abbreviate action games as ACT, role playing games as RPG, and adventure games as ADV henceforth.

In these kinds of games, a human player and game software mainly interact through a input device and display. The input device is often called a game controller. The player can take various operations using

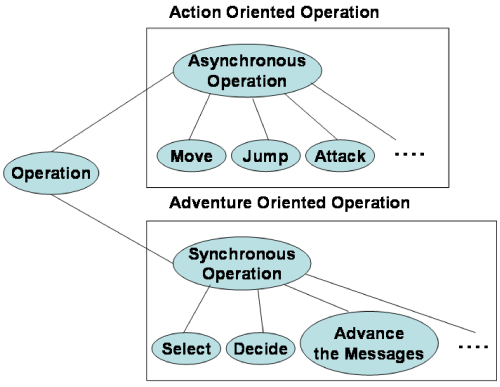


Figure 1: Classification of game operations.

the game controller, which affect the game progress. These operations can be divided into two different types. One is an asynchronous operation which the player can take at any time in a game stream, and the game proceeds without this operation. The other is a synchronous operation which is taken in order to resume a paused game stream, and is taken in the form of *selection* or *answer*. Since the former often occurs in action games, and the latter often occurs in adventure games, we refer the former as an action oriented operation, and the latter as an adventure oriented operation. Figure 1 shows the above classification of player operations and Figure 2 shows a classification of these games based on the above two operations.

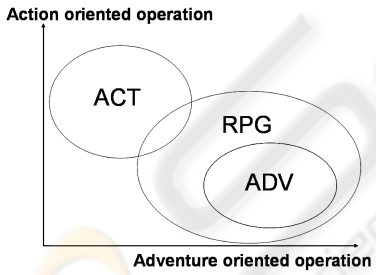


Figure 2: Classification of games.

As shown in Figure 2, RPG fully include ADV from these two operational viewpoints. Therefore, we deal with only ACT and RPG. In order to evaluate the adaptability of modeling technology to the above two games, namely ACT and RPG, we assume the typical *scenes* and actions in above games as shown in Table 1 and Table 2. The brief descriptions of these games are as follows (Onder, 2002)(Taylor and Baskett, 2006).

[ACT]

1. Characters horizontally move with jumping actions.

Table 1: Components of ACT.

| Scene | Component | Action |
|---|---|---|
| Battle | Player | Move |
| | | Jump |
| | | Attack |
| | | Weapon change |
| | | Change the target at which the weapon aims |
| | Enemy | Move |
| | | Jump |
| | | Attack |
| | | Change the target at which the weapon aims |
| | Bullet | Move forward |
| | Barricade | - |

Table 2: Components of RPG.

| Scene | Component | Action |
|---|---|---|
| Town | Player | Move |
| | | Talk to |
| | | Open the Menu |
| | | Inspect |
| | Construction (Houses etc) | - |
| | Fixture (Doors etc) | - |
| | Inhabitants (Merchants etc) | - |
| Prompt (Shopping, Conversation, Selection Menu) | Player | Select |
| | | decide |
| | | Advance the Messages |

2. A player may engage with enemies using weapons.
3. A game exits when a player lose all the HPs (hit points) which he/she obtained.
4. Attack items, e.g. bullets, arrows, spears or fireballs disappear after timeouts.

[RPG]

1. A player can change the atmosphere. (go to the next room etc)
2. A player can talk with inhabitants. If there merchants, he/she can trade with them.
3. A player can buy items such as weapons, protectors, tools.

# 4 APPLYING UML TO GAME SOFTWARE DEVELOPMENT

In this section, we discuss how UML is applied to ACT and RPG based on the game components and specifications stated in the previous section. Figure 3 shows the modeling process we use, which is complied with use case driven development, e.g. I. Jacobson's OOSE (Object-Oriented Software Engineering) (Jacobson, 2000).

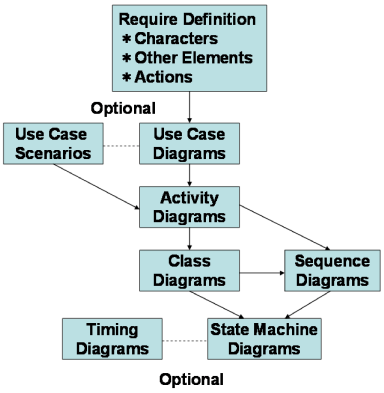


Figure 3: A modeling process for games.

In this process, use case diagrams are firstly created through requirement analysis, then each step is consecutively performed, following the directed arrows, and finally we obtain state-machine diagrams with optional timing diagrams. However, the paper mainly aims at the applicability of UML to game software development from behavioral or dynamic viewpoints, and therefore we do not discuss the step for class diagrams in Figure 3, but we assume appropriate class diagrams are derived.

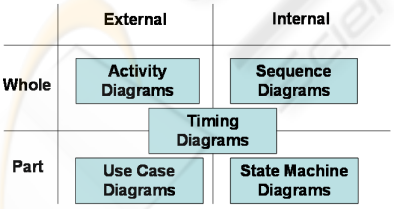


Figure 4: Modeling view points of UML diagrams.

The five kinds of UML diagrams to be created, excluding class diagrams, are classified as shown in Fig 4, from two orthogonal viewpoints, namely "part-whole" and "internal-external". The paper focuses on partial-order relationships between actions in the games and we do not deal with real time based constraints, as script design languages currently used do not. Therefore, timing diagrams are not discussed henceforth in this paper. Detailed modeling processes for ACT and RPG are presented in the following subsections.

## 4.1 Use Case Diagrams

`[ACT Modeling]`

Possible actions in ACT shown in Table 1 are aggregated into three action groups of "Player actions through input devices", "Background object actions", and "Object collisions", from player viewpoints. These three groups are represented by use case diagrams in the following way.

1. Player actions through input devices
   Each player operation through input devices is expressed as a use case. These use cases characterize the game software from operational viewpoints, since the game requires rapid player reactions through the devices.
2. Background object actions
   Each action performed by objects except the player, e.g. a background object movement or an enemy attack, is expressed as a use case. A player takes an appropriate action through the devices, reacting these object actions.
3. Object collisions
   Each event that occurs by an object collision is expressed as a use case. Such events include explosions, vanishment, or splits of the objects. In ACT, a player could be affected by these events, e.g. he obtains score points when the bullets hit enemies. There are enormous numbers of object collisions to be considered, however most collisions do not affect the player, for example, even if a collision between the player and a wall occurs, it only affects his movement, and is expressed as a use case in "background object actions". We only deal with the collisions that affect the player seriously.

`[RPG Modeling]`

RPG include much more adventure based operations than action based ones, therefore we mainly make effort to model the adventure based operations. As for action based operations, we can follow the same process for ACT. The use case modeling process for adventure based operations is as follows.

1. An adventure based operation can be represented as a series of decisions which are made by selecting from option lists. Each option in the list is regarded as a use case, and it composes a decision tree, the nodes of which are use cases. Such a tree

structure of use cases clearly describes the relationships between use cases, and makes the use case diagrams understandable.

2. The option lists that are dealt with in the above step are limited to those with a fixed number of predefined options, in order to make the modeling simplified. The lists with a variable number of options, or variable kinds of options are dealt with in use case scenarios.

## 4.2 Activity Diagram

`[ACT Modeling]`

Numerous object collisions could occur in ACT, and it seems important to model these collisions in activity modeling for ACT. In order to handle these collisions easily, we assume a collision monitoring component is included in ACT software (Rocker, 2002). The activity modeling process for ACT is as follows.

1. ACT usually include very complicated game flows according to player actions and object collisions. Therefore, it is not practical to include all the game flows from the beginning to the end of ACT in a single activity diagram. Instead, we take a time slice out of the whole duration of the game, and build an iterative activity model with possible concurrent actions, which are consolidated into a collision detection mechanism. The collision detection is performed at the end of each time slice in order to reduce complexity. The game may exit from any times slices discussed above, based on exit conditions. In order to simplify the activity diagrams, the actions are hierarchically decomposed, and so are the activity diagrams.
2. Complete all the activity diagrams at every levels of the above action hierarchy. In above each time slice, the game player may or may not take an action through an input devices, therefore two different situations could occur according to the player's choice. This choice expressed as a decision node of UML activity diagrams. If there are multiple objects other than the player controlled character in the time slice, they are expressed as an input collection of an expansion region. Using expansion regions could simplify the complicated multiple object behavior, which may be depicted with similar actions of multiple objects.
3. If the above collision detection mechanism detects collisions, complete the activity diagrams which represent the resultant effects caused by the collisions. These diagrams can be built in a similar way to the diagrams discussed in item 1 and 2.

`[RPG Modeling]`

In RPG, activity modeling is performed based on the combination of action oriented operations and adventure oriented operations. The modeling procedure is as follows.

1. Action oriented operations in RPG can be represented similarly to ACT. However, RPG allows a game player to change action scenes and adventure scenes bidirectionally, and all the action scenes must be resumed from the interruptions. In order to resume the scenes, all the states at the interaction points must be stored in any way. For this purpose, we use *history pseudo-states* in UML state machine diagrams to store the states. Even though it is not a standard notation of a UML activity diagram, we expand UML activity diagrams using these pseudo-states to express the resumption of the scenes.
2. Adventure oriented operations are decomposed based on the selections in adventure scenes, and as a result, activity diagrams for the operations are expressed hierarchical activity diagrams. This notation can simplify each activity diagram, in comparison to a huge activity diagram which includes all the selection.

## 4.3 Sequence Diagram

In our approach, a sequence diagram is basically created for each activity diagram.

`[ACT Modeling]`

A sequence diagram of an ACT is composed of a single loop fragment, since each activity diagram for a time slice or referenced diagrams from it is expressed as a loop. The creation rules for sequence diagrams are as follows.

1. The behavior of each object that occurs in the above loop fragment is depicted in a *par fragment* resides in the *loop fragment*, followed by an *opt fragment* which deal with the collisions of the objects. This structure reflects the above mentioned collision detection mechanism.
2. If there are *ref fragments* in the above diagram, break it down into another sequence diagram.
3. Similarly, each *ref fragment* in the *opt fragment* for object collisions is broken down into another sequence diagram.

`[RPG Modeling]`

In RPG, additional modeling efforts are needed for adventure oriented operations.

1. Depict an *opt fragment* for scene switching, succeeded by the *per fragments* that reside in the above discussed *loop fragment*. This fragment includes *ref fragments* which represent scene switching operations.
2. The rest of the *loop fragments* are identical to ACT.
3. Depict the object behavior that is represented by the *ref fragments* in item 1 in the form of sequence diagrams.

### 4.4 State Machine Diagram

State machine diagrams for ACT and RPG are expressed as the hierarchies of composite states in our approach.

`[ACT Modeling]`

1. At the top of the hierarchy of state machine diagram, crate a state machine diagram represent the whole game. This state machine diagram includes other diagrams, each of which represents more detailed level game operations.
2. If there are multiple state transitions within an object, these transitions are depicted by the regions in a composite state.

`[RPG Modeling]`

In RPG, in addition to the hierarchical descriptions for action oriented operations, adventure oriented operations are also described hierarchically. The following shows the basic rules for modeling.

1. Action oriented operations are modeled in the same way to ACT.
2. As for adventure oriented operations, possible series of player's decisions for selection items are hierarchically expressed as composite state machines. Each player's choice represented as a *choice pseudo-state*.

## 5 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a model based approach to developing game software. We adopted MDA (Model Driven Architecture) as a modeling framework and UML as a notational tool. Among various game categories, we selected three game categories, that is, ACT (ACTion games), ADV (ADVenture games) and RPG (Role Playing Games), for modeling, since these categories are seemed to dominate today's game marketplace.

We characterize the games from two orthogonal game player operations, namely adventure oriented operations and action oriented operations. From these view points, RPG include ADV, therefore the target categories of our modeling are limited to ACT and RPG. We proposed a systematic way to model the above games for each UML diagram type and game player operation type. These UML diagrams include use case diagrams, activity diagrams, sequence diagrams and state machine diagrams. By applying our approach to modeling typical game situations, we conclude the modeling approach to ACT and RPG to be feasible. Our modeling approach makes the development of game software identical to other traditional software development like business software, in which modeling technologies are effectively used.

In order to deal with real time constraint of the games, timing diagrams would be needed. In addition, static diagrams like class diagrams, deployment diagrams or component diagrams are also needed to determine the game software structure.

## REFERENCES

Fairclough, C, F. M. N. B. and Cunningham, P. (2001). Research directions for ai in computer games. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science PP.333-344*.

Frankel, D. (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley.

Jacobson, I. e. (2000). *Object-Oriented Software Engineering*. ADDISON-WESLEY.

Laird, J. and Lent, M. (2000). Human-level ai's killer application:interactive computer games. In *AAAI/IAAI PP.1171-1178*.

Miles, R. and Hamilton, K. (2006). *Learning UML 2.0*. O'REILLY.

Nishimori, T. and Kuno, Y. (2003). Action game-oriented programming language. In *ISSN Vol.44th PP.35-46*.

Onder, B. e. (2002). *GAME DESIGN PRESPECTIVES*. Charles River Media.

Rocker, R. (2002). *Engineering and Computer Games*. Addison-Wesley.

Taylor, M.J, G. D. and Baskett, M. (2006). Computer gameflow design. In *ACM Computers in Entertainment Vol.4,No.1 PP.1-9*.