Applying Exception Handling Patterns for User Interface Customization in Software Games Modification

Thitipong Tengtrirat and Nakornthip Prompoon

Abstract -There may be an error that usually occurs during the use of software. It may appear in different forms and may result in different effects. Developer should perform software verification and validation and make correction in order to reduce the chance of software mulfunction. In software game, end user is allowed to modify software especially interface customization. This may easily cause an error. This research presents a method for error detection and management for software game modification using exception handling patterns and game engine. The proposed method can be applied to other similar software types.

Index Terms: Game Engine, Design Pattern, Software Games Modification, User Interface Customization

I. INTRODUCTION

Software development is a continuous improvement in term of product features in order to earn the software satisfies the change of user requirements. Usually developer makes software feature extension by developing a plug-in package so it can perform according to the change of requirements. However, in software game development, there is a chace that an end user may modifiy feature of software game to meet his/her requirements during software exection. The software development that allow end user to modifiy the program is called “End User Programming” which in game industry is known as software modification or Mod.

The Mod is widely popular because it extends the software lifetime and improve software fearture from inputting innovative ideas and specific requirements which are generated by the end users. There are a supporting facilities help perform game modification such as game engine, tool, user manual and game developer social network. In some game engine, there are floders that provide set of add-on files or script files that user can modify. The modified code can be replaced the existing one automatically.

However, during software modification an error may occur in differnts ways such as software crash or some object disappeared. Therefore, an error detection and management is a challenge issue.

There are three concerns have been point out from [1]: Firstly, error management is a capability to identify fault localization and to provide a method to handel each fault occurs. Secondly, software monitoring is a capability that can help monitor the result of the program during execution. For example, it may provide a pop up message informs the status of its execution so the user can see the result once he/she has been updated game engine. Lastly, alert system is a capability informs users the potential discrepancies beween the before- and the after-modification.

Execption handling is an importation method and widely accepted among developer since it can help detect an error and manage such error in a predefined way for differen scenarios. There is a research proposed exception hadleing patterns for unsatisfied contitions occur in business processes using Little-JIL [2]. The proposed patterns can be able to apply in software programming phase as well since programmer has to use an exception handling method to help manage an error.

Thus, this research paper present a method for error detection and management for software game modification using exception handling patterns and game engine.

In this paper, there will be show background in section II and related work in Section III then, we describe our approach in Section IV and, finally we give the conclusion of this work.

II. BACKGROUND

A. Game Engine

Game engine is the design structure to assist the software development process. The focus of game engine development is that it must be simple to learn and use. Game engine development is a core of game development since it always includes special features that can be inherited to the next game [2, 3, 4]. The special feature includes functional requirements and non-functional requirements. Furthermore, the game engine has been designed in a variety of features to serve various user needs. For example, the game engine may allow user to select different types of user interface devices such as mouse, joystick or keyboard.

B. Mod or Modification

Game developers develop elements such as game rules, behavioral, character based on the game engine. The game engine allows new game rules or game components to be

Manuscript received Jan 08, 2013; revised Feb 05, 2013.

Thitipong Tengtrirat is with the Software Engineering Laboratory, Center of Excellence in Software Engineering Department of Computer Engineering, Faculty of Engineering Chulalongkorn University, Thailand; e-mail: Thitipongt.T@student.chula.ac.th

Nakornthip Prompoon is with the Software Engineering Laboratory, Center of Excellence in Software Engineering Department of Computer Engineering, Faculty of Engineering Chulalongkorn University Thailand; e-mail: Nakornthip.S@chula.ac.th

created from plug-ins [5,6] or by reusing elements. Some game engines have provided channels for ease of use, such as script languages or support tools for simple use, which can be used to create a new game, called a modification or Mod.

The Mod is an innovation idea occurring from the end user concept or the end user gaming experience. The requirements for game modification comes from a specific group of user. The results of mod are not a new creation, but rather only the extension of the original game. Some of them can work stand-alone but mod relies on the original system. Mods also lead to the gathering of end users in development, leading to new learning in social groups with different cultures. In some cases, the mod may be a prototype of game in the next generation. Mods have been divided into 4 categories:

1) User Interface Customization

The game players or the end users can customize the interface to meet the user requirements in these areas: a) Customize the interface to create the unique or personalized interface that meets the user requirements, b) Customize the display to improve the performance and/or the functionality of the usage, and c) Customize the interface to support the new modules when newer modules or newer functions are released that may affect the user interface.

2) Game Conversions

The game functions or game rules can be customized by the game players or the end users.

3) Machinima

The game engine can be customized the game for other functions such as to create cinematic productions or movies.

4) Hacking Closed Game Systems

The end users customize the games against the copyright software for a malicious intent towards other users or other systems, or sometimes, to create some advantages to the users that are more than what the developers originally allowed.

C. End User Software Engineer

Software users tend to have new differing idea or extended idea from the original implementation's features.

This is a specific need from the specific group of users. Thus, there is a collaboration for developemt of users who have the same specific requirements called End User Programming or End User Development [7,8].

There are simple tools that the end user programmer or developers can use to make program changes such as spreadsheet. however, this type of programming can create subsequent problems or bugs because of inadequate user's skills or the high level of program's complexity. In order to solve or prevent these problems, there is a need for a new type of software engineer which is called the end user software engineer.

The end user software engineering is based on software engineer principles but it does not cover the software life cycle, instead focusing on limiting errors from End user software development, allowing users to participate in software development success.

End user software engineering allows end users to learn and understand of errors so that they will correct errors to a certain extent. As a result, production of software will be more reliable and completely satisfactory for the end users.

D. Model View Present Pattern: MVP

MVP is a pattern developed from the Model/View/Controller Pattern (MVC) [9] because the MVC pattern allows for viewing access to the Model component. This functionality requires the view to have a logic as a component. The view also has more dependency with the user interface, that is, when the code is changed, it affects the user interface. This makes the system to be more complex, harder to test, and harder to maintain.

MVP Pattern is a model that separates control or logic from view, called the Presenter, making the view only for presentation and interface with user .The presenter is mainly responsible for receiving input data from the view to manage cccessibility between the view and model, as shown in Figure .1

The relationship between the view and presenter is similar to the Decorator Pattern. The presenter is related to the view in the components, where one view can have more than one presenter. The MVP pattern has the presenter operates and sends results to view. The non-dependency between them allows the View and presenter to be less complex, and one presenter can be reused with different views

.

III. Related Work

Ali Gokalp Peker [11] presented the process of game engine design in order to conform to design goal and design strategies. The processes was composed of 5 procedures: 1) defining features of the game engine; 2) defining a set of design goals and strategies; 3) extracting design goals from features; 4) defining design patterns and 5) integrating all defined patterns and forming the complete design of the game engine. An example of design goal, design strategis and suggested design evaluation are shown in Table I.

There is another research[11] using Little-JIL for exception handeling patterns illustration. Little-JIL is a hierarchically-scoped process language with a graphical syntax and semantics that are precisely defined by finite state machines. It can be used to explain a runtime environment that allows execution on a distributed platform.

The basic unit of Little-JIL processes is **the step**, represented graphically by **an iconic black bar** as is shown in Fig 2. [12] Little-JIL use 2 types of symbols to represent normal behavioral, called substep, and exceptional behavioral as shown in Fig 3.

1) Substep part

Little-JIL substep decomposition is represented by having substep icons connected to the left side of the parent step icon by edges. The edges are annotated with specifications of the artifacts that are passed as parameters between the parent and child steps. Each parent step specifies the execution order of its substeps using one of the four sequencing icons, shown in Figure 3. They appear in the step bar above the point where the substep edges are attached. There are four different sequencing icons: (1) Sequential which indicates that the substeps are executed in order from left to right (2) Parallel which indicates that the

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

substeps can be executed in any order. (3) A choice which allows any one of the substeps to be executed. (4) Try which indicates that the substeps are executed left to right until one succeeds.

2) Exception Handling Mechanisms

The parent step may offer exception handling facilities to its descendant steps. These facilities are defined by exception handlers connected to the parent by edges attached to the right side of the parent's step bar immediately below an 'X'. Each exception edge is annotated to identify the type of exception that it handles. There are four exception continuation icons:

(1) Completion the step to which the exception handler is attached is finished, and execution continues as specified by its parent.

(2) Continuation the step to which the exception handler is attached should proceed with its execution as though the substep that threw the exception had succeeded.

(3) Restarts the step to which the handler is attached.

(4) Rethrow the handler to propagate the triggering exception up to an enclosing scope as in a usual programming language

Barbara Staudt [2] apply exception handling patterns for managing the business operation of an error in order to work effectively. These were controlled with 3 characteristics,

1) Presenter: Manage exception by offering another presentation to replace the existing object presented in one screen.

2) Insert: Manage exception by inserting an operation that is suitable for a particular scenario. The insertion must ensure that the system still work as usual.

3) Aborting: To cancel the operation by inserting operation to manage cancel operation.

Exception handling pattern is a design pattern used to manage exception flow of processes and to controlled the operation with greater accuracy and flexibility. The exception handling pattern categories is listed in Table II [2] [12].

IV. Defined Exception Handling Method for Mod

Our research objective is to apply exception handling patterns to control errors caused by game modification operated by end user software engineer for game engine design.

From our analysis of problems occure during Mod, we can indentify the causes the problems into 2 types; 1) from user perform the modification and 2) from the design of game engine that does not provide a sufficient fuction to handle such error.

We proposed a framework help reduce the game modification error is presented in details as shown in Fig 4.

.A. Identify Candidate Customization Types of User Interface.

This step starts from the study of Mod method from various sources such as related research papers [5, 6] and then analyze the possible ways of game design customization in order to identify user interface Mod types as shown in Table III. and the feasible errors may occur according to each type as shown in Table III.

In each customization case, errors can occur in a variety of formats and may need different method to fix them. Using exception handling patterns may help resolve the error as shown in Table IV.

B. Select a Candidate Game Design Engine for Mod

According to the research work by Ali Gokalp Peker [1], we use all procedures for game design game to our research environment for game software modification. We focus on the two strategies, usability and adaptability, by adding features called extension support composed of 3 sections are integrated in an extend support module. all existing features from game engine will be migrated to the extend support module as a seperate center for game modification process.

1) Extension Management Section. User must modify game engine according to his/her needs using the existing one. 2) Addition Management section. User can add new features to create the additional module. 3) Addition Feature Management. The module can be created by user input new module in a game engine level by modifying the existing extend support module as shown in Fig 5.

From [11] that used MVP patterns for user interface presentation, we would like to perform the modification in order to comfrom to the defined strategies. Thus, another extend MVP is contructed and Bridge pattern is selected to use in order for existing MVP can calls Extend MVP once there is a modification as shown in Fig 6.

C Apply Exception Handling Pattern in Game Design Engine

Bridge pattern can be used to detect whether there is a modification occurring. If it is, then Extend MVP will activate to perform the similar assigned task as the correspond MVP as shown in Fig 7. From Fig 7, there are 4 patterns used for modification for the defined strategies. 1) MVP Pattern of the engine. It is a main function for controlling the input and output process of the system. 2) Bridge Pattern. It is use as a middle man provding the interfaces between MVP pattern and Extend MVP pattern. 3) Exception Handling. It help filter and monitor the result of modification in extend MVP. Exception Pattern is chosen for a specific case of error. 4) Extend MVP. It perfroms the function on behalf of MVP once there is a modification.

The description of the pattern characteristic at the behavioral level consists of the system functionality along with the applications to use the pattern to monitor the unwanted conditions or errors that could have been created during the interface modification or customization during mod. The guidelines to resolve the occurred errors using the analysis with Little-JIL diagram to show the structure, sample cases and functionality of pattern are shown in the Table V.

In order to illustrate our concept, an example of user satisfaction case is elaborated in detail using pattern format as shown in table VI. In addition, by applying with Little-Jill, the result of how exception handeling will be performed for each substep is shown in Fig. 8.

D. Pattern Implementation Evaluation

Tools are developed using open source sofware based on the proposed design patterns, in order to test whether the implementation according to the design patterns can perform according to the pattern objective. There are 4

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

steps for pattern implementation evaluation: 1) Define the objective of testing. The activities and procedures defined within the framework are covers all substeps and executed correctly. 2) Design and specify the evaluation criteria. Test cases are generated according to the control error. 3) Test the selected game. The selected game is tested by adding a Mod and pattern based on the design test case. 4) Check the test results. The test results are evaluated based on evaluation criteria using a checklist covers all feasible scenarios. If the test result does not meet the expect result, the proposed pattern are needed to redesign. An example of evaluation details is shown in Table VII.

V. Conclusions and Future Work

This paper presents an approach for the application of exception handling patterns in the design of the game engine in order to support the improvement of game software, based on the principles of End user software engineering. The paper focuses on filtering errors and presenting approaches to fix errors, in order to assist the end users to be able to perform the development successfully. The proposed design patern can be used as a reference pattern for development by game developers. In the future, apart from user satisfaction, other proposed patterns will be further designed in details and tested whether they can operate according to the pattern objective. Our proposed framework can be applied in other exception handlings such as synchronization between two parallel operations from different machine performed by different game players..

REFERENCE

[1] Burnett, M., Cook, C., Rothermel, G., ”End-User Software Engineering”. Communications of the ACM 47(9) 2004, 53–58

[2] Barbara Staudt Lerner and Stefan Christov, "Exception Handling Patterns for Process Modeling", IEEE Transactions on Software Engineering, Vol. 36, No. 2, March/April 2010, p.162-183

[3] Stephen Tang and Martin Hanneghan, Game Content Model: Stephen Tang and Martin Hanneghan "An Ontology for Documenting Serious Game Design", 2011 Developments in E-systems Engineering, p. 431-431

[4] Seung Hun Lee , Gum Hee Lee , Hyun Hoon , Doo Heon Song and Sung Yul Rhew, "An Empirical Model of the Game Software Development Processes", Proceedings of the 4th International Conference on Software Engineering Research, Management and Applications (SERA’06)

[5] Magy Seif El-Nasr and Brian K Smith,"Learning Through Game Modding" .ACM Computers in Entertainment, Vol. 4, No. 1, January 2006, p.1-20

[6] Walt Scacchi, "Modding as a Basis for Developing Game Systems". 1st International Workshop on Games and Software Engineering, GAS 2011, p. 5-8

[7] M. Burnett. “What is end-user software engineering and why does it matter?”, In 2nd International Symposium on End-User Development 2009, p 15–28

[8] Andrew J.Ko.”, The State of the art in end-user software engineering”. ACM Computer Survey 2011(21)

[9] Model View Presenter (MVP) VS Model View Controller (MVC). Available:http://blog.vuscode.com/malovicn/archive/2007/12/18/model-view-presenter-mvp-vs-model-view-controller-mvc.aspx . [Last Accessed: September 03, 2012].

[10] Martin Hunter, “The MVPC Software Design Pattern”, Tidying the House 2006, p 1-5

[11] Ali Gokalp Peker and Tolga Can,"A Design Goal and Design Pattern Based Approach for Development of Game Engines for Mobile Platforms". The 16th International Conference on Computer Games, 2011

[12] Barbara Staudt Lerner,Stefan Christov, Alexander Wise,Leon J. Osterweil, ”Exception Handling Patterns for Processes”. WEH '08, Proceedings of the 4th International Workshop on Exception Handling, p 55-61

Table I Example of design goal, strategies and evaluation [11]

Design Goal	Design strategy	Evaluation
(1) Usability	Can be defined as readability and intuitiveness of the game engine and the game implemented on it. It helps reduce learning curve and increase productivity.	Can user predict the expected results after entering any action? Are there any diagram or tool used to simplify the presentation?
(2) Efficiency	Should use underlying platform efficient, in terms of power, memory and performance. It can make the design more complex.	Are there any tool used help measure power consumption of reference implementation? Does the design concern memory efficiency and consumption?
(3) Portability	Design goal for multi-platform support such as strategies that assist in the adaptation of programs to target environment.	Are there a comparison evaluation between the typical platform and the different target platforms?
(4) Adaptability	Ensured in different levels of game engine design or different configuration by using abstraction or in runtime level by using a script language.	Does the design using the appropriate design pattern such as using the adapter pattern for input controller design?

Table II Categories of exception handling patterns [2]

Category	Pattern name	Intent
1. Presenter: Trying other alternatives		Use alternative operation
	Ordered Alternatives	The same functionality as a normal operation.
	Unordered Alternatives	The different functionality as a normal operation.
2. Insert: Inserting behavior		Insert and check operation
	Immediate Fixing	Adding steps to fix the process.
	Deferred Fixing	Adding steps to fix the process action must be taken to record the error and possibly provide partial fixing
	Retry	Adding steps to fix the process and return to the conditions operation.
	Exception-Driven Rework	Check and insert modified behavior. It will continue when success all required conditions.
3. Aborting: Cancelling behavior		Insert for checking operation
	Reject	To ignore when the conditions operation failed.
	Compensate	To reverse action back to the previous step if the condition fails.

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Table III Show Category purpose and Example of user interface customize

User Interface Customization Category	Customized Case	Examples
1. Customize unique interface	Satisfaction	Customization aesthetic. Example: change the image
	Specific applications	For exposure to that work. Example : adds event voice , change color for apparent
2. Customizing the display	Function	Customize for operation with new display or work function.
	Performance	Reducing the activity of the user. Example: Merge button
3. Customize interface cause of expansion modules	Adapt	Add the new interface or add the new display.
	Maintenance	When you add function but they are affected with the interface. Customized to work properly.

Table IV Structures, examples and pattern usage for satisfaction modifications

Pattern	Purpose (To handle errors that occur in the case).	Guidelines for resolving errors that occur.
User satisfaction	Customization for uniqueness	The mod can be used to display another display screen that is unique or different from the original model.
Specific applications	Customization for specific display	Upon error, cancel the customized form, and use the default to enable continued operation.
Function	Inserting additional operations	Cancelling the operation or adjusting the order of operation to return to normal operation (which may result in cancelling the customization or partial use of customization)
Performance	Re-order operations	Filtering operations that work properly, and perform only those operations to return to normal operation.
Adaptation	Adding new functionality	Cancel the customized operation and display of the system. Report the error to the user.
Maintenance	Editing a specific function	Report the error to the user.

Table V Troubleshooting and applied exception handling pattern

Modification type	Troubleshooting	Applied pattern
Filtering errors	Display alternative	Present
	Insert Behavior	Insert
Display errors to the user	Cancel the operation	Aborting

Table VI Example of User Satisfaction Case

Objective	Mod can adapt other methods in order for interface customization can operate properly by choosing the alternate options in many levels and cancelling the previous steps.	
Application.	Use exception handling pattern when the system cannot operate, to choose a similar operation. For operations related to other steps, when cancelling the operation, the other steps should be undone to allow the job followed appropriately. For example, when changing Pictures and setting new picture position, if the alignment is correct, but the image is too large,The system must resize the image and re-align with the image.	
Structure	Step Name	Substep and Description
	Normal flow	Use Default: Use default system instead of mod.
		Use Extension: Activate the extension.
		Checker: Make decisions on the use of extensions.
	Ordered Alternative	Extension Checker: Examines the operation under the condition of the extension.
		Ordered Alternative: The choice of other extensions which behave in the same format as the source. The behavior of the alternative is to run indefinitely until the defined condition is detected. In case of the happening of the unsatisfied condition, the system will use the default to replace the mod and report the error message.
	Unordered Alternative	Extension Checker: Examines the operation under the condition of the extension.
		Unordered Alternative: The choice of other extensions which behave in differnt format from the source. The behavior of the alternative is to run indefinitely until the defined condition is detected. In case of the happening of the unsatisfied condition, the system will use the default to replace the mod and report the error message.
	Compensate	Extension Checker: Examines the operation under the condition of the extension.
		Use Default: Use default system instead of mod.
		Compensate: Modify operations to use the default system instead of the mod according to the relevant conditions.
Example (Shown in Figure 9**)**	Customizating the interface by changing the background of the game menu, using a file with a GIF extension. First, the system will store the default configuration of the game without customization as the default value and store the customization (mod), before determining whether there has been modification. Second, determine if the mod is allowed to operate by checking the file extension from the given priority of conditione.g., JPEG and GIF. For example, Check extension file set priorities. If Mod finds files under the specified conditions then the Mod is allowed to operate. If the Mod is not under the specified conditions, then the system will use default values without the Mod. After doing this step, the system will check the next condition, the size of the image, based on the priorities defined. If the mod does not pass the condition, the system is returned back to the default value. Repeat until the conditions specified for image authentication are completed, and replace the stored default values with the modified values.	

Table VII: Design Pattern Evaluation for User Satisfaction Case.

Operation	Form of customization	Errors expected to occur	Display when using patterns
Customizating background image.	Change background image.	Black Screen or image not displayed	o Display correctly. o Use the default before customizeation. o Cancel the operation of the system and report the results to the user.
		Display distortion	
		System crashes	
Customizatign Cursor	Change Cursor image.	Cursor loss	
		Cursor distortion	
		System crashes	

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

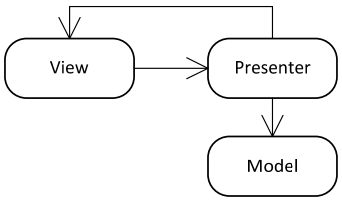

Fig 1. MVP Pattern [10]

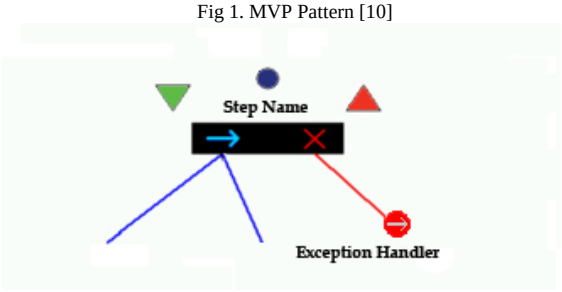

Fig 2. Little-JIL Syntax [12]

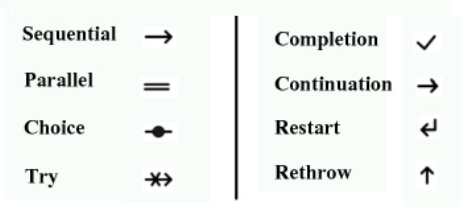

Fig 3. Little-JIL Sequencing Icons and Exception Continuation Icons [12]

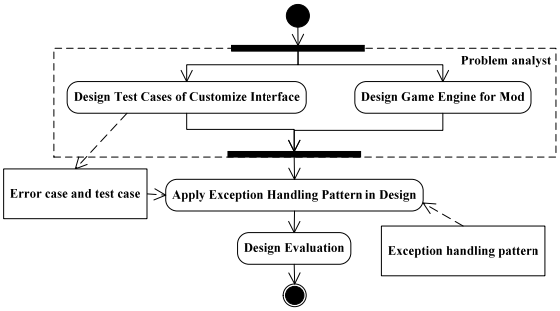

Fig 4. The proposed framework of applying exception handleing for a software game modification performs by end user software engineer

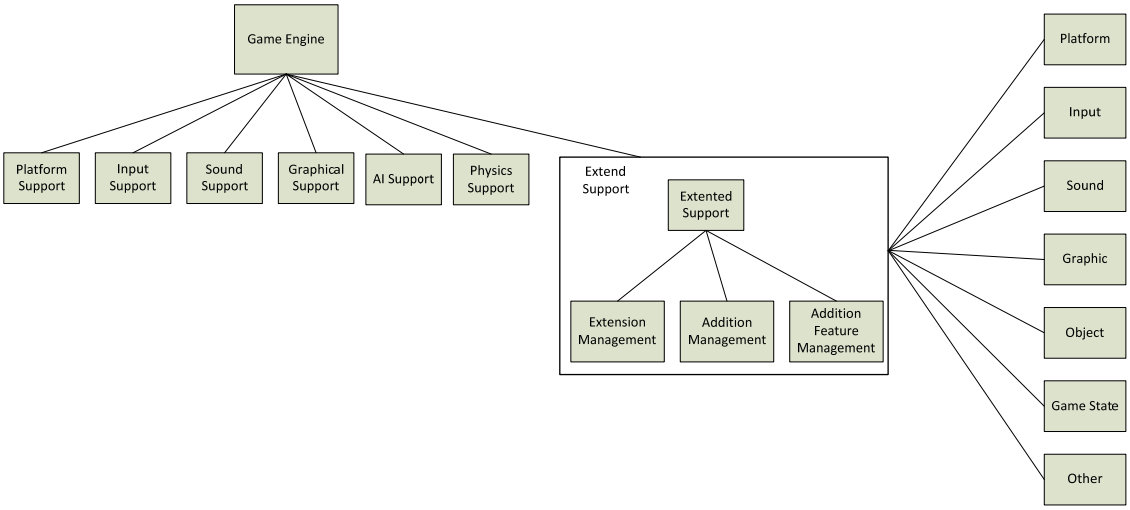

Fig 5 feature component of game engine

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

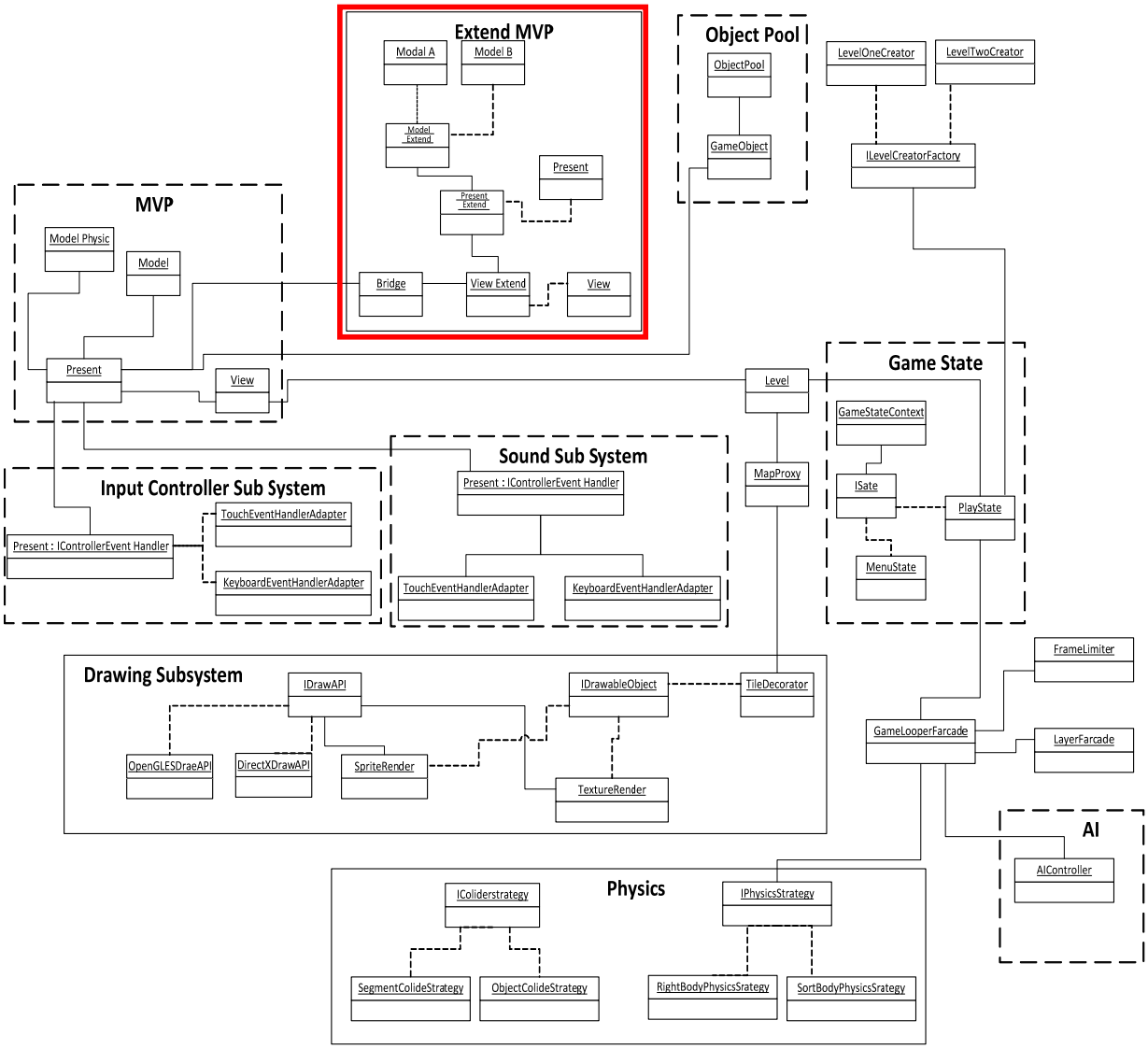

Fig 6 Class Diagram Show Engine Structure

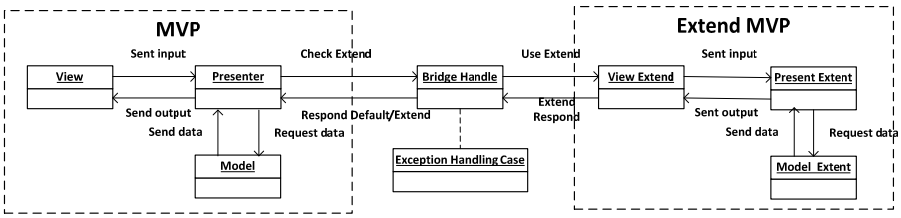

Fig 7 Class Diagram Show Dataflow of Extend Structure in Game Engine

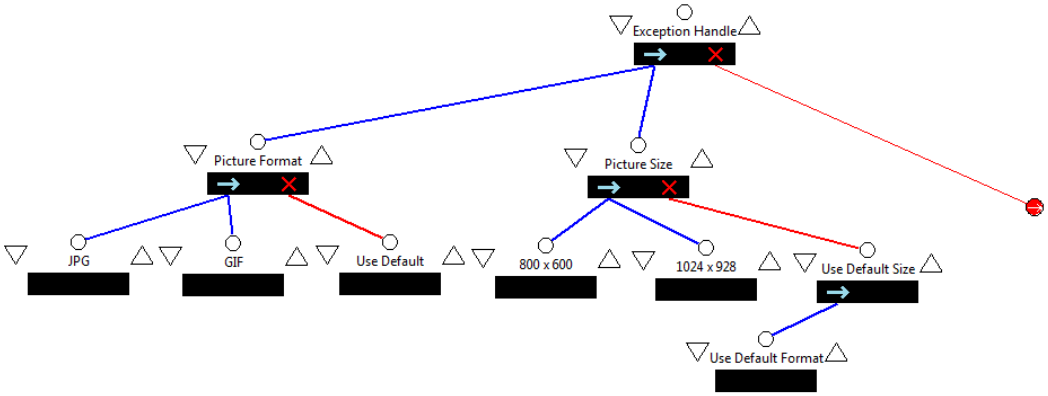

Fig 8 Little-Jill Diagram Showing Mod Management from Exception Handling Pattern for Game Mod for Satisfaction

ISBN: 978-988-19251-8-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)