UML 클래스 도해의 저작도구를 위한 MVC모델의 설계

김재훈* · 김윤호*

*안동대학교

MVC model design for an Authorization Tool of UML Class Diagram

Jae-Hoon Kim* · Yun-Ho Kim*

*Andong National University

E-mail : zoom50210@gmail.com, unokim@andong.ac.kr

요 약

본 논문에서는 UML 클래스 도해의 저작도구를 위한 MVC 모델의 설계를 제시하고자 한다. MVC 모델의 설계에서 View, Controller, Model 컴포넌트를 정의하고 각각의 컴포넌트의 역할을 서로 독립적으로 부여하고 수행하도록 설계 한다. 뷰(View)는 Gui를 나타내고, 컨트롤러(Controller)는 데이터의 입력과 출력을 담당하며, 모델(Model)은 비즈니스 로직을 처리 한다. 클래스 도해의 저작도구를 위한 MVC 모델의 설계는 각각의 컴포넌트 특징에 맞게 나눠서 독립적으로 역할을 부여하고 시스템을 유연하도록 한 것이 특징이다.

ABSTRACT

This paper suggests the design of MVC model for an authorization tool of UML class diagram. In the design of MVC model, it is designed to define view, controller and model and perform the individual role of each component. The View represents GUI and the Controller is responsible for data input and output and the Model is to handle the business logic. The MVC model design for an authorization tool of class diagram gives the role independently and tries to be flexible with system by dividing into the suitable features of each component.

키워드

UML, Class Diagram, MVC Model, 객체지향 방법론

Ⅰ. 서 론

UML[1]은 Unified Modeling Language의 약자로서 OMG(Object Management Group)에서 표준으로 채택한 통합 모델링 언어이다. 사용하는 형식과 각각의 표기법에 의미를 가지는 언어이다.

객체지향 방법론에서 시스템이 객체들로 구성된다. 시스템이 어떠한 객체들로 구성되고 그 객체들 간에 어떠한 관계가 있는지를 표현하는 다이어그램이 Class Diagram이다. 시스템의 구조를 표현하고 모델링 개념에서 가장 큰 영향을 미치기 때문에 UML에서 제공하는 많은 다이어그램 중에서도 Class Diagram은 사용빈도 수가 높다. Class Diagram[2]의 표기법은 name, attribute, operation을 기술하여 객체를 표현한 클래스의 특징을 나타낸다.

MVC(Mode-View-Controller) 모델[3]을 사용하여 컴포넌트 기반에 UML 클래스 도해의 저작도구를 위한 MVC 모델 설계를 제안한다. MV모델은 뷰-컨트롤러-모델 이렇게 세 가지로 나눠진다. 뷰(View)는 Gui를 나타내고, 컨트롤러(Controller)는 데이터의 입력과 출력을 담당하며, 모델(Model)은 비즈니스 로직을 처리 한다. 클래스 도해의 저작도구를 위한 MVC 모델의 설계는 각각의 컴포넌트 특징에 맞게 나눠서 독립적으로 역할을 부여하고 시스템을 유연하도록 하는 것이 목적이다.

Ⅱ. 본 론

2.1 클래스 저작도구를 위한 MVC 모델

UML 클래스 도해의 저작도구를 위한 MVC 모델의 구조는 View, Controller, Model 컴포넌트로 구성하였다. 컴포넌트의 정의는 같은 성격의 역할을 수행하는 클래스들의 집합을 뜻하는데, View에 관련된 클래스들, Controller에 관련된 클래스들, Model에 관련된 클래스들을 묶어 놓았기 때문에 각각의 컴포넌트로 지칭하여 사용하였다. 아래의 그림 1은 UML 클래스 도해의 저작도구를 위한 MVC 모델의 구성도를 나타내었다.

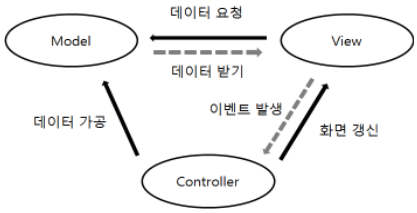

그림 1. UML 클래스 도해의 저작도구를 위한 MVC 모델의 구성도

View 컴포넌트는 Model 컴포넌트가 가진 데이터를 시각적인 요소로 표현하기 위해서 GUI에 관련된 것이며, Model 컴포넌트의 정보를 알고 있지만 Model 컴포넌트는 View 컴포넌트에서 데이터가 어떻게 표현되는지 몰라도 된다. 만약 데이터를 입력하여 이벤트가 발생하면 Controller 컴포넌트에게 넘겨준다. Model 컴포넌트에서 데이터 정보를 얻어오고, Controller 컴포넌트는 View 컴포넌트와 Model 컴포넌트 사이에서 데이터 또는 이벤트가 발생하는 것을 컨트롤하며, View 컴포넌트로부터 입력은 받은 데이터를 파악하여 해당 데이터를 Model 컴포넌트의 어떤 모델에게 전달할지를 결정하고. 데이터를 가공할 Model 컴포넌트의 해당 모델로 전달한다. 필요에 따라서 View 컴포넌트의 상태 변경을 요청 할 수도 있다. Model 컴포넌트는 데이터의 상태를 저장하거나 비즈니스로직을 처리하는 역할을 하며, Controller 컴포넌트와 View 컴포넌트를 직접 control하지 않으며 독립적으로 존재한다.

UML 클래스의 입력된 데이터 정보를 표현하기 위해서는 View 컴포넌트가 필요하고, View 컴포넌트에서 발생한 데이터 또는 이벤트를 처리하기 위해서는 controller 컴포넌트가 필요하다. 그리고 입력된 데이터를 저장하고 가공하기 위해서는 Model 컴포넌트가 필요하다. 그림 4와 같이 입력된 데이터를 저장하는 과정을 나타낸 것이다.

Step 1. View에서 데이터 입력
Step 2. 이벤트가 발생되면 Controller에게 이벤트 넘기기
Step 3. 모델이 입력된 데이터를 가공하기
Step 4. 모델이 변경되었으므로 업데이트하기 위한 View에게 통보하기
Step 5. View에서 다시 그리기

그림 2. 입력된 데이터를 저장하는 과정

2.1.1 View

View 컴포넌트를 크게 두 부분으로 표현할 수 있다. 첫 번째는 DiagramPanel이 다이어그램을 출력하기 위한 것이고, 두 번째는 CodePanel이 다이어그램을 CodeGeneration하여 코드로 변환되어 출력하기 위한 클래스이다. ToolBarPanel은 ToolBar와 CodeGeneration을 묶어서 관리하는 클래스이며, ToolBar는 다이어그램을 작성하기 위한 메뉴를 포함하고 있으며, CodeGenerationBar는 다이어그램을 코드로 작성하기 위한 메뉴를 포함하고 있다. GDrawStorage는 GNode와 GEdge를 저장하기 위한 클래스이며 GNode는 클래스를 그리기 위한 인터페이스를 정의하고 있다. GEdge는 관계를 표현하기 위한 인터페이스를 정의한다. GClass는 GNode인터페이스의 실체화이면서 클래스의 속성(name, attribute, method) 값을 출력하는 클래스이다. GRelationship는 GEdge 인터페이스의 실체화이고, ArrowHead와 LineStyle을 사용하여 관계를 출력을 수행한다. PropertyDialog는 클래스의 속성 값(클래스의 이름, 클래스의 속성, 클래스의 오퍼레이션)을 입력하기 위한 GUI를 나타낸다. Frame은 View 컴포넌트의 모든 클래스의 관리를 담당하고 있다. View 컴포넌트의 구성도는 그림 3과 같으며, View 기능을 제공하기 위해서 설계한 클래스 역할은 표 1과 같다.

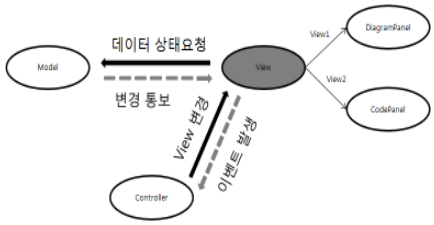

그림 3. View 구성도

표1. View 컴포넌트의 클래스 요약

클래스 이름	클래스 역할
Frame	GUI에 관련된 클래스를 묶어서 관리
DiagramPanel	다이어그램이 그려지는 영역
CodePanel	다이어그램을 코드로 변환되어 나타내는 영역
ToolBarPanel	다이어그램과 코드 메뉴를 관리하여 나타내는 영역
ToolBar	다이어그램을 작성하기 위한 메뉴
CodeGenerationBar	다이어그램을 코드로 작성하기 위한 메뉴
GDrawStorage	GNode와 GEdge를 저장하는 저장소

GNode	클래스를 그리기 위한 인터페이스 제공
GClass	GNode 인터페이스의 실체화이며 클래스를 그리는 역할
GEdge	관계를 그리기 위한 인터페이스 제공
GRelationship	GEdge 인터페이스의 실체화이며, LineStyle과 ArrowHead을 사용해서 하나의 관계를 그리는 역할
LineStyle	Relationship을 표현하는 선 종류(실선, 점선)를 출력(display)
ArrowHead	화살표 머리 모양 종류(삼각형, 다이어몬드, 꺽쇠)를 출력(display)
Property Dialog	클래스의 속성(name, attribute, operation) 값을 입력하도록 하는 GUI

2.2.1 Controller

View 컴포넌트에 발생하는 이벤트를 Controller 컴포넌트의 인터페이스인 ControllerBroker에게 넘겨줘서 모든 요청을 받도록 설계했다. 그리고 View 컴포넌트의 특정 클래스 마다 각각의 컨트롤러를 만들어서 발생한 이벤트에 유형에 따른 컨트롤러에게 요청을 위임한다. 각각의 컨트롤러에게 이벤트 처리를 분산 시켜서 조금 더 유연하게 발생된 이벤트를 처리하도록 설계했다. 그래서 입력을 받고 이벤트가 발생하면 이벤트를 처리하여 Model 컴포넌트의 해당 모델에게 입력받은 데이터를 넘겨주고, View 컴포넌트와 Model 컴포넌트를 분리하기 위한 목적을 가지고도 있다. 다이어그램의 유효성을 검증하는 역할도 한다. 그리고 필요에 따라 입력된 데이터 값을 Model 컴포넌트의 해당 모델에 전달되는 데이터와 관련이 없을 경우(데이터를 가공하지 않고 입력된 데이터 값을 화면에 표현하는 경우) Controller 컴포넌트에서 View 컴포넌트에게 직접 상태를 변경 할 수도 있다. Controller 컴포넌트의 구성도는 그림 4와 같으며, Controller 기능을 제공하기 위해서 설계한 클래스 역할은 표2와 같다.

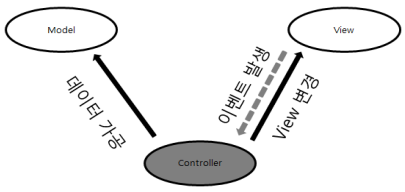

그림 4. Controller 구성도

표2. Controller 컴포넌트의 클래스 요약

클래스 이름	클래스 역할
ControllerBroker	Controller 컴포넌트의 facade로써, 데이터 입력 또는 발생된 이벤트를 처리하는 인터페이스 제공
DiagramPanel Manager	다이어그램을 그리기 위해서 DiagramPanel에서 발생하는 이벤트 수행
PropertyDialog	클래스의 이름, 클래스의 속성, 클래스의 오퍼레이션의 입력된 데이터를 처리하기 위해서 PropertyDialog에서 발생하는 이벤트 수행
CodeGeneration BarManager	다이어그램을 코드로 변환시키기 위해서 codeGenerationBar에서 발생하는 이벤트 수행
FileManager	클래스의 정보와 관계의 정보를 파일로 저장하기 위해서 파일에 관련된 이벤트 수행
GDrawStorage Manager	다이어그램을 그리기 위한 GNode와 GEdge를 저장하고 있는 GDraw Storage에 관련된 이벤트 수행
DiagramInspection	다이어그램의 검사를 수행하는 인터페이스 정의
DiagramInspection Manager	DiagramInspectiond의 실체화이며, 다이어그램 검사를 수행

ControllerBroker는 View 컴포넌트로부터 데이터 또는 이벤트를 전달 받도록 하는 인터페이스를 제공하고 있다. DiagramPanelManager는 DiagramPanel에서 발생하는 이벤트를 수행하기 위해서 DiagramPanel 클래스를 가지며, 다이어그램을 표현하기 위한 이벤트를 처리하는 controller이다. PropertyDialogManager는 PropertyDialog에서 발생하는 이벤트를 처리하기 위해서 PropertyDialog를 가진다. 입력된 클래스의 속성 값을 저장하기 위해서 Model 컴포넌트로 전달하는 controller이다. CodeGenerationBarManager는 codeGenerationBar에서 발생하는 이벤트를 수행하고, 다이어그램을 코드로 변환시키는 controller이다. FileManager는 클래스의 정보와 관계의 정보를 파일로 저장하기 위해서 파일에 관련된 이벤트를 수행하는 controller이다. GDrawStorageManager는 GDrawStorage를 가지고, GDrawStorage에 관련된 이벤트를 처리하는 controller이다. DiagramInspection인터페이스는 다이어그램의 유효성 검사를 하기 위해서 인테페이스를 정의하였고, DiagramInspectionManager 클래스는 DiagramInspection인터페이스의 실체화를 나타내고 있다.

2.3.1 Model

Model 컴포넌트는 Controller 컴포넌트로부터 전달 받은 데이터를 비즈니스 로직에 따라 가공을 하고 저장한다. Model 컴포넌트에서 제공하는 기능은 데이터 저장, 다이어그램의 정보를 코드로 변환 시키는 것이다. 비즈니스 로직에 따라 가공된 데이터는 저장되고, View 컴포넌트에게 변경되었다고 통지하고 통지를 받은 View 컴포넌트는 GUI화면 상태를 변경한다. Model 컴포넌트의 구성도는 그림 5와 같고, Model 기능을 제공하기 위해서 클래스 역할은 표3과 같다.

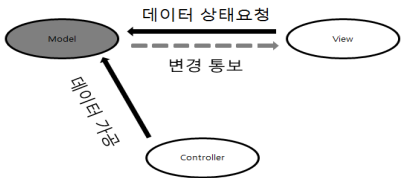

그림 5. Model 구성도

표3. Model 컴포넌트의 클래스 요약

클래스이름	클래스역할
ModelBroker	Model 컴포넌트의 fa□ade로써, 데이터의 저장과 다이어그램의 정보를 코드로 변환 시키는 인터페이스 제공
InforamtionStorage	클래스와 관계와 파일의 정보를 저장하는 저장소
ClassInforamtion	클래스의 정보를 저장하기 위한 클래스의 정보 정의
Attribte	클래스의 attribute의 정보를 정의
Operation	클래스의 operation의 정보를 정의
OperationParameter	operation의 파라미터의 정보를 정의
Relationship Information	관계의 정보를 저장하기 위한 관계의 정보 정의
GenerationFile	클래스와 관계의 정보를 파일로 저장하기 위한 파일 정보 정의
Generation	다이어그램을 코드로 변환하기 위한 인터페이스 정의
CodeGeneration	Generation 인터페이스의 실체화이며, 다이어그램을 코드로 변환시키기는 것을 수행

ModelBroker는 Model 컴포넌트로 전달되는 모든 요청을 받기위한 인터페이스를 제공한다. InforamtionStorage는 DiagramPanel에 표현되는 클래스, 관계의 데이터를 저장하고, CodePanel에 다이어그램의 정보를 코드로 표현하기 위해서 클래스와 관계의 정보를 파일로 저장하여 관리한다. ClassInforamtion는 클래스의 정보를 저장하기 위해서 클래스에 관련된 정보를 정의하고 Attribute는 클래스의 attribute 정보를 정의하며, Operation은 클래스의 operation 정보를 정의한다. 그리고 OperationParameter는 operation의 파라미터 정보를 정의하고 있다. RelationshipInformation은 관계(Relationship)의 정보를 저장하기 위해서 관계에 관련된 정보를 정의한다. GenerationFile은 클래스와 관계의 정보를 저장하기 위한 파일에 관련된 정보를 정의하고 있다. Generation은 다이어그램을 코드로 변환하기 위한 인터페이스 정의를 하며, CodeGeneration이 실체화이다. InformationStorage에 저장되어 있는 파일의 정보를 읽어서 코드로 변환 시키는 역할을 한다.

Ⅲ. 결 론

본 논문에서는 UML 클래스 도해의 저작도구를 위한 MVC 모델의 설계를 제안했다. 같은 특징의 업무를 수행하는 클래스들의 집합인 View, Controller, Model의 컴포넌트를 분류하여 각각의 컴포넌트에 대해 역할을 서로 독립적으로 부여하고 수행하도록 설계 하였다. 향후 과제로는 본 논문에서 제시한 MVC 모델의 설계 기반에 클래스 도해의 저작도구에 관련된 정보 구축과 처리 설계에 대한 연구가 요구된다.

참고문헌

[1] Grady Booch, Jim Rumbaugh, and Ivar jacobson, The Unified Modeling Language User Guide Second Edition, Addison Wesley,

[2] Martin Fowler, UML Distilled Third Edition, Addison Wesley, 2004

[3] Grady Booch, Design Patterns: Elements of Reusable Object-Oriented Software Addison Wesley, 1994