DOI: 10.3745/KIPSTD.2009.16D.6.881

물품관리를 위한 VMDC(View, Model, Dispatcher, Controller) 아키텍처

김 다 정[†] · 이 은 서[††]

요 약

본 연구에서는 소프트웨어 기반으로 물품관리를 하기 위한 VMDC(View, Model, Dispatcher, Controller) 아키텍처를 제안한다. 물품을 인력이 아닌 소프트웨어로 관리하기 위하여 여러 아키텍처들이 존재한다. MVC와 기존 아키텍처의 경우, 공통된 객체의 전달로 불필요한 데이터가 이동되는 문제점이 발생한다. 따라서 이와 같은 문제점을 해결하기 위하여 VMDC(View, Model, Dispatcher, Controller) 아키텍처를 제시하고자 한다. VMDC(View, Model, Dispatcher, Controller) 아키텍처의 Dispatcher(사령부)는 각 Controller(컨트롤러)가 필요로 하는 데이터를 파악하고 그것을 기반으로 하여 객체를 재구성함으로서 효율적인 데이터 이동을 하도록 한다. 또한 VMDC(View, Model, Dispatcher, Controller) 아키텍처를 이용하여 개발된 식품관리 프로그램으로 사용사례를 보이고, 효율성을 제시 후 향후 연구방향 또한 제시한다.

키워드 : 모델링, 소프트웨어 아키텍처, VMDC

The VMDC(View, Model, Dispatcher, Controller) Architecture for Products Management

Kim Da Jeong[†] · Lee Eun Ser[††]

ABSTRACT

This research introduces the architecture of managing products based software. There are many of the architectures for managing products using software instead of manpower. In case of MVC and existing architectures, The architectures transfer redundant data so the architectures cause a problem that unnecessary data moved. This research presents VMDC(View, Model, Dispatcher, Controller) architecture to solve the problem. Dispatcher of VMDC grasps necessary data and reconstructs objects to efficient transferring data. This research shows usecase that designed VMDC(View, Model, Dispatcher, Controller) and demonstrate efficiency of VMDC(View, Model, Dispatcher, Controller) together. after demonstration this research present with next research.

Keywords : Modeling, Software Architecture, VMDC(View, Model, Dispatcher, Controller)

1. 서 론

물품의 종류가 방대한 현재, 과거 물품의 관리와는 달리 인력이 아닌 소프트웨어로 물품의 관리를 하고 있다. 이에 소프트웨어들의 수요는 늘어나고 있고, 소프트웨어들을 효과적으로 설계할 수 있는 아키텍처들도 필요하게 되었다.

기존에 존재하는 아키텍처 중 계층구조 스타일은 각 서브시스템이 하나의 계층이 되어 하위 층이 제공하는 서비스를 상위층의 서브시스템이 사용하도록 구성되는 것이다.

클라이언트 서버 스타일은 말 그대로 적어도 하나의 서버가 있어서 커넥션을 기다리고 처리해 주며 적어도 하나의 클라이언트가 서버와 연결하려고 시도하는 구조이다. 브로커 스타일은 여러 노드에 투명하도록 소프트웨어 시스템을 분산 시키는 것이며, 트랜잭션 처리 스타일은 입력을 하나씩 읽어 처리한다. 입력은 시스템에 저장되어 있는 데이터를 조작하는 명령들, 즉 트랜잭션이며 그 트랜잭션을 어디서 처리하는지를 결정하는 사령 컴포넌트가 필요한 아키텍처이다. 파이프 필터 스타일은 데이터 변환에 사용되는 아키텍처이다. 데이터 스트림이 프로세스에 차례로 전달되어 각 프로세스가 데이터를 일정한 형태로 변형시킨다[1].

MVC스타일은 모델(Model), 뷰(View), 컨트롤러(Controller)

※ 이 논문은 2008학년도 안동대학교 학술연구조성비에 의하여 연구되었음.
† 준 회 원 : 안동대학교 컴퓨터공학과 학부생
†† 종신회원 : 안동대학교 컴퓨터공학과 조교수
논문접수 : 2009년 7월 6일
수 정 일 : 1차 2009년 7월 29일, 2차 2009년 9월 28일, 3차 2009년 10월 14일
심사완료 : 2009년 10월 17일

와 같이 각 계층 간의 결합력이 최소화 되도록 어플리케이션 영역을 나눈 것으로 Web-Tier Application을 포함한 소프트웨어 설계에 사용되는 개발 방법론에 활용 된다.

모델들의 특성에 의하여 본 연구에서 제시하고자 하는 물품관리의 경우 사용자 인터페이스를 담당하는 View가 존재하며, Model이 사용자인터페이스인 View와 Controller사이를 중재함으로써 모든 클래스를 역할별로 구분 짓는 MVC 모델이 적합하다. 그러나 MVC 모델로 물품관리 아키텍처를 사용할 경우에는 모델이 데이터베이스에서 자료를 가져오게 되면 쓸모가 없는 데이터가 전달된다는 문제점이 보였다. 따라서 본 연구에서는 이와 같은 문제점을 해결할 수 있는 새로운 아키텍처를 제시한다.

2. 기반 연구

2.1 MVC와 트랜잭션 처리 아키텍처

MVC 아키텍처는 모델, 뷰, 컨트롤러의 줄임말로써 사용자 인터페이스를 시스템의 다른 부분과 분리하여 결합도를 낮추기 위한 아키텍처 스타일이다. 모델은 응용 프로그램의 내부 상태, 즉, 비즈니스 자료나 비즈니스 로직과 같은 것을 나타낸 것으로 어플리케이션의 데이터를 표현하고, 데이터에 접근을 제어하는 작업을 수행한다. 뷰는 모델의 콘텐츠를 어떻게 보여주는가 하는 규정으로, 클라이언트에 출력되는 사용자 인터페이스와 관련이 있다. 컨트롤러는 어플리케이션의 동작을 정의한다. 사용자의 입력(input)과 어플리케이션 프로그램 간의 상호작용 처리와 데이터 저장과 조회 등과 같은 모델과의 상호작용도 처리한다[2-5].물품관리를 위한 소프트웨어를 사용함에 있어서 사용자 들은 간단하게 사용자 인터페이스 창에서 모든 조작을 할 것이고 사용자들은 시스템의 내부를 알 필요가 전혀 없는 상황에서 MVC 아키텍처는 적합한 아키텍처이다[1].

〈표 1〉 MVC와 트랜잭션 아키텍처의 비교

	트랜잭션 처리	MVC
분할 정복	트랜잭션 처리기는 잘 분리된 단위이다.	세 가지 컴포넌트가 독립적으로 설계될 수 있음
응집도 증진	트랜잭션 처리기는 원래 응집된 단위. 기능적, 순차적, 절차적 응집은 있으나 교환적 응집은 없음	뷰와 제어부분이 단일 UI계층에 같이 있다면 계층 응집도가 큼
결합력 감소	처리기에서 사령 컴포넌트를 분리함으로서 결합력을 낮춤. 트랜잭션 처리기 사이의 결합력 제어가 중요	세 컴포넌트 사이의 통신 채널이 최소화 됨
융통성	새로운 트랜잭션 처리기를 추가하기 쉽다.	뷰, 제어를 변경하여 UI를 쉽게 갱신할 수 있음
테스트 가능성		애플리케이션을 UI와 분리하여 테스트 가능
방어적 설계	각 트랜잭션 처리기와 사령 컴포넌트에 assertion 체크 추가	

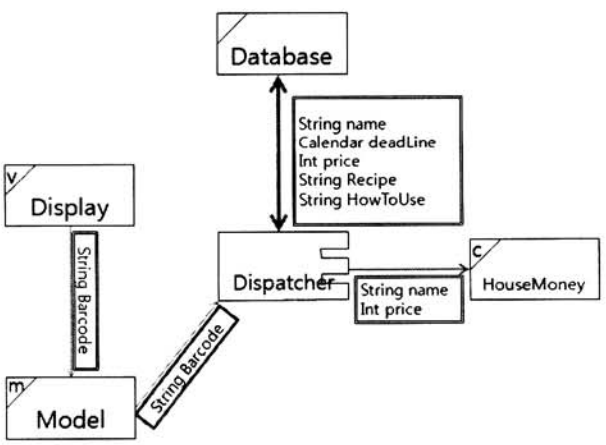
MVC 모델의 핵심은 비즈니스 로직을 처리하는 모델과 사용자에게 데이터를 받고 비즈니스 로직이 처리한 결과 화면을 보여주는 뷰가 분리되어 있으며, 어플리케이션의 흐름 제어나 사용자의 처리 요청은 컨트롤러에 집중되어 있다는 것이다. 뷰와 모델이 분리되어 있기 때문에 모델의 내부 로직이 변경되어도 뷰는 영향을 받지 않는다는 장점이 있다[4, 6-7].

트랜잭션 처리 아키텍처는 입력을 하나씩 받아서 처리하는 아키텍처이다. 여기서 입력 이란 시스템에 저장되어 있는 데이터를 조작 하는 명령, 자료나 제어 시그널 등이 어떠한 행위를 유발시키는 것이 트랜잭션이다[8]. 트랜잭션 아키텍처는 물품관리 소프트웨어에 적당한 아키텍처가 아니다. 하지만 관리를 위한 데이터를 하나씩 처리한다는 부분에 있어서 물품관리 소프트웨어에 일부 필요한 아키텍처라 볼 수 있다.

3장에서 이 두 아키텍처의 장점을 조합하고 새로운 요소를 추가하여 물품관리 소프트웨어에 효율적으로 사용 될 수 있는 최적의 새로운 아키텍처를 제시하고자 한다[1].

2.2 Data Move Class Diagram

UML 표현 중에 데이터의 이동을 나타내기 위한 다이어그램에는 시퀀스 다이어그램이 있다. 시퀀스 다이어그램은 시간에 따른 메시지의 흐름을 보는 다이어그램이다. 시퀀스 다이어그램에서 메소드와 함께 제시되는 파라미터 값을 보고 데이터를 볼 수 있다. 여기서 객체의 이동, 더 자세히 인수들의 이동을 한 눈에 볼 수 있는 다이어그램의 부족함을 느꼈으며 3.3에서 새로운 다이어그램을 제시한다.



(그림 1) Data Move Class Diagram

3. 본 론

본 장에서는 물품 관리를 위한 아키텍처의 필요성을 밝히고, 새로운 아키텍처를 제시한다. 또, 이 아키텍처를 이용하여 물품관리를 위한 소프트웨어를 개발했을 때의 효율성을 제시하여 이 아키텍처의 필요성을 확립한다.

3.1 물품관리 소프트웨어를 위한 아키텍처

기존의 아키텍처 중 물품관리에 적합한 아키텍처로는 MVC 모델임을 제시했다. 그러나 MVC 모델을 물품관리 소프트웨어를 위해 채택하기에는 문제점이 보인다. 하여 아래에 새로운 VMDC(뷰(View), 모델(Model), 사령부(Dispatcher), 컨트롤러(Controller))모델을 제시한다.

3.1.1 환경 설정

VMDC 아키텍처는 물품관리를 위한 아키텍처이다. 관리가 되어야 하고 각각의 고유의 데이터가 있으며 관리시에 각각 구분될 수 있는 데이터를 가지는 물품에 사용된다는 것을 전제로 한다. 그리고 그 물품들의 정보를 한 곳에 모아야 한다. 그것이 데이터베이스가 될 수도 있고, 웹 서버가 될 수도 있다. 물품관리를 위해 만들어진 소프트웨어는 이 물품의 정보에 쉽게 접근할 수 있어야 하고 사용할 수 있어야 한다.

물품관리 시스템은 대량의 물품을 관리하는 물품 창고뿐만이 아니라 가정의 관리까지도 자동으로 되도록 할 것이다. 물품에 표시되어 나오는 식별자(일반적으로 바코드)에 관리에 최적이 될 많은 정보들이 들어있어야만 이 시스템이 활성화 될 수 있다.

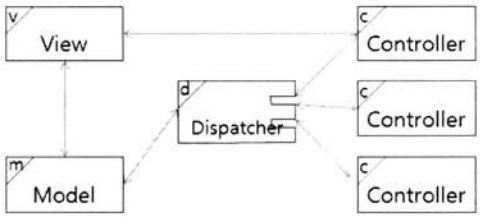
하드웨어 환경은 물품의 식별자를 읽을 수 있는 하드웨어가 갖추어져 있고, VMDC를 적용한 소프트웨어가 수행될 수 있는 환경이다.

(그림 2) 환경설정

3.1.2 VMDC 아키텍처

VMDC아키텍처는 뷰(View), 모델(Model), 사령부(Dispatcher), 컨트롤러(Controller)의 앞 글자를 딴 이름으로, MVC 모델과 사령부라는 차별화를 둔 모델이다.

모든 사용자의 인터페이스는 뷰에서만 관리한다. 사용자가 입력하거나 사용자가 결과를 확인하는 부분을 담당한다. 이 부분에 있어서 사용자가 해야 할 일은 최소로 하고 시스템이 대부분의 관리를 해야 할 것이다. 또 사용 대상자가 큰 물류창고의 전문적인 관리자뿐만 아니라 가정의 사용자



(그림 3) VMDC 아키텍처

까지 사용하도록 직관적인 그래픽 사용자 인터페이스를 가지도록 설계해야 할 것이다.

모델에서는 뷰창을 제어하는 부분을 담당한다. 뷰에서 사용자가 입력하는 데이터를 임시 저장하고 간단한 처리를 하는 부분을 맡는다. 기존의 MVC 아키텍처에서는 모델이 뷰와 컨트롤러 사이에서 처리해야 할 메시지가 다량이었으나 본 아키텍처에서는 뷰에서 받아온 물품의 정보들을 사령부로 전달하는 역할만 주로하게 된다.

데이터 사령부가 모델의 역할을 분담한다. 데이터 사령부는 트랜잭션 처리 아키텍처에서의 사령부의 역할과 흡사하지만 그 세부 개념은 다르다. 트랜잭션 처리 아키텍처에서 사령부가 하는 일은 사령부에 들어온 트랜잭션을 받아 그것을 처리해 줄 적절한 컴포넌트에 배분시켜 주는 일이다. VMDC 아키텍처에 있어서 사령부는 트랜잭션 사령부가 아닌 객체를 재구성 하는 곳으로써 데이터베이스에서 물품의 고유 식별자로 찾아온 모든 정보들을 이용하여 필요한 데이터와 메소드로 객체를 재구성 한다. 재구성이 필요한 이유는 객체지향에서의 객체의 성질 때문이다. 객체는 자신의 속성과 자신이 해야 할 행동 즉, 메소드를 모두 가지고 있다. 각각의 컨트롤러는 하는 일이 다 다르고 필요로 하는 속성이나 메소드가 모두 다르다. 하지만 객체는 모든 속성과 모든 메소드를 가지고 있는 물품 객체 하나이다. 처리하는 일이 모두 다른 컨트롤러에 일률적인 객체가 제공됨으로써 비효율적인 데이터 이동이 이루어진다. 이에 사령부를 둠으로써 하나의 물품 객체에서 각각의 컨트롤러에 맞는 맞춤형 물품 객체가 되고 쓸모없는 데이터의 이동을 줄일 수 있게 된다.

컨트롤러는 기존의 MVC 모델과 같이 사용자 인터페이스를 제외한 모든 처리를 해주는 컴포넌트 부분이다. 데이터 사령부에서 받아온 정보들을 이용하여 각 컴포넌트에게 주어진 일들에 맞게 수행한다.

3.2 기존 아키텍처와의 비교

기존 아키텍처와 VMDC 아키텍처를 비교 한다. 여기서 기존 아키텍처란 VMDC 아키텍처를 착안할 때 동기가 되었던 트랜잭션 아키텍처와 MVC 아키텍처이다. 그러나 트랜잭션 아키텍처는 사령부의 기능이라는 동기만 제공했을 뿐, 그 기능이 VMDC 아키텍처와 많이 다르기 때문에 비교하지 않겠다. 트랜잭션 아키텍처는 시스템에 저장되어 있는 데이터를 조작하는 명령들, 즉 트랜잭션을 처리하는 아키텍처로서 일반 객체나 클래스를 처리하는 MVC 아키텍처나 VMDC

〈표 2〉 기존 아키텍처와의 비교 표

	MVC	VMDC
객체의 재구성	특별히 객체를 재구성 하는 부분은 없음	사령부 부분에서 컨트롤러로 넘어가는 자료들을 효율성 있게 재구성 함
모델의 역할	View를 제어하고 View의 내용을 임시로 저장함. View와 Controller와의 연결 담당	MVC와 같이 Mod -el의 역할을 하지만 Controller와 직접 연관하지 않고 Dispatcher로 넘겨주는 역할을 함
컨트롤러로의 데이터 이동	객체의 모든 데이터가 전달 됨	필요한 데이터만 재구성되어 전달 됨

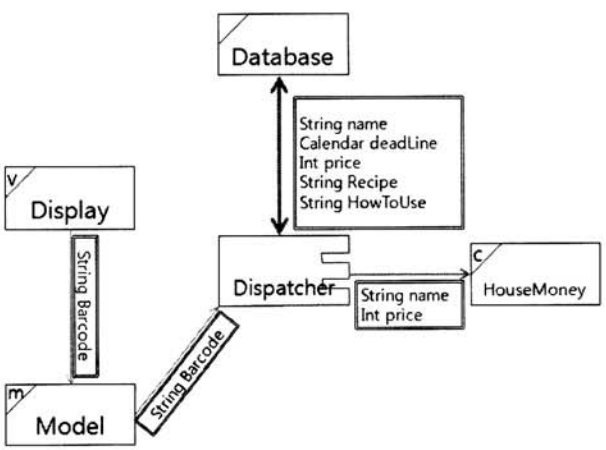
아키텍처와는 비교가 어렵다. 그러므로 MVC 아키텍처와 VMDC 아키텍처의 비교를 중점으로 한다.

MVC 아키텍처와 VMDC 아키텍처의 가장 중요한 차이점은 객체의 재구성이다. 3.1에서 설명 했듯이 MVC 아키텍처에서는 찾아 볼 수 없는 객체의 재구성이 VMDC 아키텍처의 사령부에 추가되었다. 이로써 데이터 이동을 제어하여 쓸모없는 이동을 줄이고 필요한 데이터만 이동 할 수 있도록 한다. 또 모델의 역할의 차이점이다. MVC 아키텍처에서는 하는 모델의 역할과 VMDC 아키텍처에서 하는 모델의 역할은 거의 동일하다. 하지만 MVC는 모델은 컨트롤러에 직접 연관되어 있는 반면 VMDC는 사령부에게 데이터를 넘겨주게 된다.

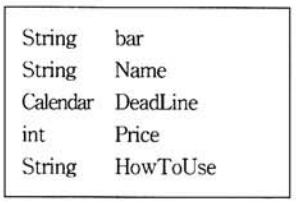
3.3 Data Move Diagram

VMDC 아키텍처를 설계하는 동안 데이터의 이동을 한눈에 볼 수 있는 다이어그램의 부족을 느꼈으며 이에 Data Move Diagram을 제시한다.

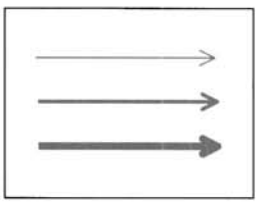
Data Move Diagram은 한 눈에 객체가 가지고 가는 데이터들을 볼 수 있도록 하기 위하여 제안하였다. 컴포넌트 간의 관계를 표현하면서 그 동시에 관계 사이에서 전달되는 객체 혹은 객체가 가진 속성 값들을 표현한다. 컴포넌트 간의 관계를 알아야 하는 동시에 생성자나 특정 메소드에 전



(그림 4) VMDC Data Move Diagram



(그림 5) 이동 데이터 보이기



(그림 6) 선 표현

달되어 지는 데이터의 종류와 그 양을 알아야 할 경우도 발생한다. Data Move Diagram은 클래스 다이어그램과 같이 컴포넌트 간의 관계를 나타내면서 그 동시에 이중선으로 된 상자 안에 그 데이터들을 표시해 줌으로써 관계와 종류를 모두 보일 수 있는 다이어그램이 된다.

Data Move Diagram은 연결 선 위에 객체나 객체의 이동을 표현한다. 객체나 객체의 속성이 많을 경우 아래와 같이 상자를 만든 후, 그 안에 내용들을 표현해 준다.

본 다이어그램의 주요 역할은 데이터의 이동을 직관적으로 보이기 위해서이다. 그래서 데이터, 즉, 객체 또는 객체의 속성들의 양을 한 눈에 볼 수 있도록 아래와 같이 연결선의 굵기를 이용하여 나타낸다.

데이터 내용이 가장 많은 객체를 기준 수치 n, 어느 객체로 이동하는 데이터의 수를 m이라 할 때, $m \le n*\frac{1}{3}$일 경우 가장 가는 선, $m \le n*\frac{2}{3}$일 경우 가운데 굵기의 선, $m > n*\frac{2}{3}$일 경우 가장 굵은 선으로 표현한다.

선 굵기를 다르게 표현하여 한눈에 그 양을 가늠 할 수 있게 함으로써 가독성을 높인다.

4. 사례 연구

VMDC를 이용하여 설계한 물품관리 시스템을 제시한다. 또 기존의 MVC 모델과의 코드길이, 복잡도에 관한 비교를 통해 VMDC 모델의 효율성을 밝힌다.

4.1 식품관리 시스템

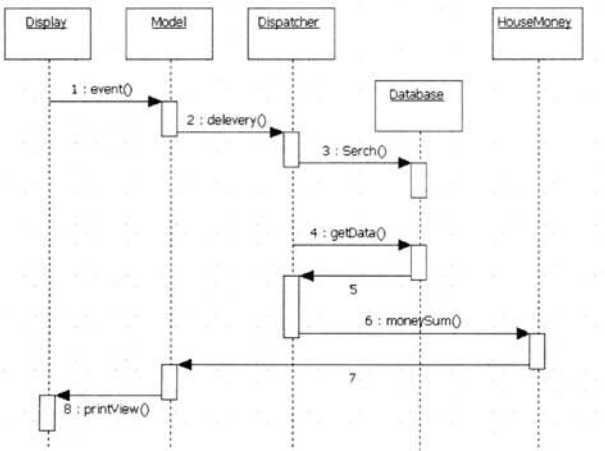
물품관리 아키텍처를 이용하여 식품관리 시스템을 설계하려고 한다. 아래에 간단한 클래스 설명을 하였다.

식품관리 시스템 중 한 예로 가계부를 작성하는 클래스를 나타내었다. 식품관리 시스템 일부(바코드 정보를 이용하여, 데이터베이스에서 모든 정보를 가져 온 후, 가계부를 쓰는 부분)의 시퀀스 다이어그램은 아래와 같다.

Display에서 사용자는 이벤트를 발생시킨다. 식품관리 시스템에서는 바코드 정보를 가지고 Model객체로 이동하는데, 여기서 Model 객체는 Display의 관리를 하고 바코드 정보를 Dispatcher객체로 넘겨준다. 물론 원래의 MVC 아키텍처에서 Model의 뷰를 관리하고 뷰에서 발생되는 정보를 임시로 저장하는 일도 수행한다. 그러면 사령부는 바코드 정보를

〈표 3〉 식품관리 시스템의 클래스 설명

클래스명	설 명
Display	View의 역할을 하는 클래스로서사용자 인터페이스만을 표현한다. 화면의 디자인을 맡는 클래스이다.
Model	사용자 인터페스를 관리하고 Dis- play에서의 바코드 정보를 Dispa- tcher에게 전달한다.
Dispatcher	데이터베이스에서 해당 바코드의 데이터를 찾고 각각의 클래스에게 필요한 정보만으로 객체를 재구성 하여 배분해 준다.
Database	각 물품들의 관리되는데 필요한 정보들을 담아둔 데이터베이스이다.
HouseMoney	가계부를 쓰는 클래스로서 이 것 외에도 다른 클래스가 있다. 컨트롤러의 역할이다.



(그림 7) 가계부의 시퀀스 다이어그램

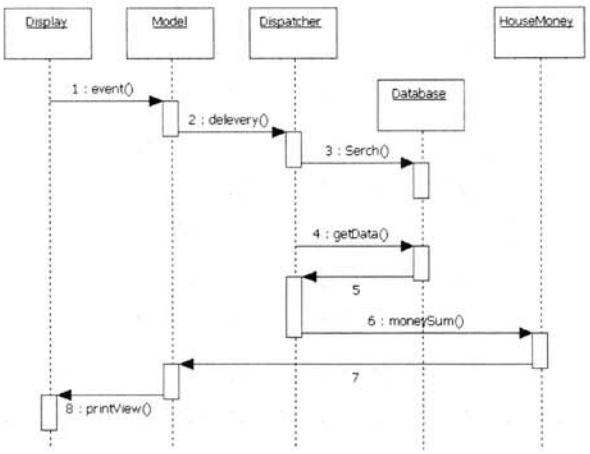
데이터베이스나 웹서버로 보내어 검색을 하고 그 결과 값, 즉, 식품의 모든 정보(식품명, 가격, 유통기한, 관리방법, 유의사항, 레시피정보 등)를 받아온다. 그 후 각 컨트롤러에게 갈 객체들을 선별하고 재구성한다. 여기서 재구성이란, 데이터베이스에서 넘어온 모든 데이터를 HouseMoney 객체에서 모두 사용되지는 않기 때문에 Dispatcher에서 HouseMoney에 전달된 객체에 name, price만을 실어 보내는 일을 일컫는다.

4.2 MVC와 VMDC의 사례 차이점 비교

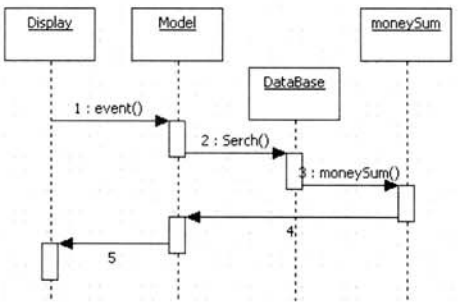
아래에 기존의 MVC 모델과 VMDC 모델의 사례 비교를 한다.

Dispacther객체는 데이터베이스에서 가져온 많은 양의 데이터중 HouseMoney에 필요한 name과 price만을 뽑아 재구성 한 뒤 HouseMoney로 전달한다. 아래는 같은 내용을 MVC모델로 설계한 시퀀스 다이어그램이다.

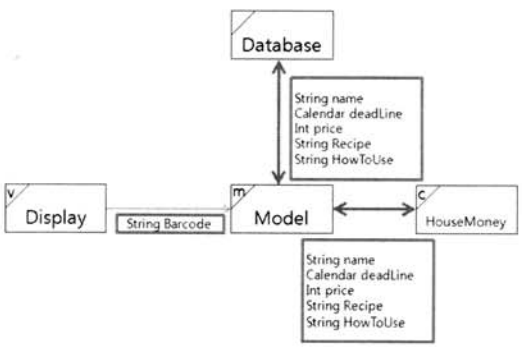
본 시퀀스 다이어그램을 봤을 때, MVC모델이 VMDC모델보다 더 간단해 보인다. 아래의 Data Move Class Diagram에서 보이겠지만 MVC의 DataBase객체에서 HouseMoney로 넘어가는 데이터의 양이 VMDC모델의 Dispactcher에서



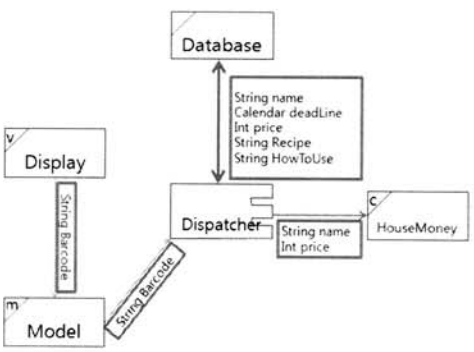
(그림 8) VMDC로 설계한 식품관리시스템



(그림 9) MVC로 설계한 식품관리시스템



(그림 10) MVC Data Move Class Diagram



(그림 11) VMDC Data Move Class Diagram

HouseMoney로 가는 데이터의 이동이 훨씬 더 많다. 아래의 다이어그램은 2.2에서 설명하고 있다[11-12].

위의 그림은 MVC 아키텍처로 가계부를 설계한 다이어그램이다. HouseMoney는 가계부를 쓰는 클래스로서 사실상 식품의 이름과 가격만 있으면 처리가 가능한 클래스이다. 하지만 MVC 아키텍처는 데이터베이스에서 모든 정보를 받아 만들어진 일률적인 객체를 보내는 방법 외에는 없다.

VMDC 모델에서는 Dispatcher가 모든 정보를 받아 moneySum에게 Name과 Price만을 전달해 준다. 필요하지 않은 정보는 전달하지 않음으로써 데이터의 낭비를 줄인다.

4.3 VMDC 아키텍처의 효율성

VMDC 아키텍처는 기존의 아키텍처에 비해 데이터 이동이 적고, 효율적인 데이터 이동으로 인해 가독성도 높아지는 이점이 있다.

4.3.1 코드상의 효율성

아래에 MVC 모델을 구현한 코드와 VMDC 모델을 구현한 코드를 제시한다.

(그림 12, 13)에 걸쳐 보듯이 코드상의 차이점은 없다. 하지만 객체내의 속성 값의 차이는 없다. 가계부(House Money)는 식품의 이름(name)과 가격(price)만을 사용한다. 그렇기

```
import java.util.*;
public class HouseMoney_NonDispatcher {
     ArrayList<String> arName =
          new ArrayList<String>();
     ArrayList<String> arPrice =
          new ArrayList<String>();
     int sum;
     public HouseMoney_NonDispatcher(){
         sum = 0;
     }
     public int CalcMoney(Objects obj1){
         if(arName.size()==arPrice.size()){
         arName.add(obj1.name);
         String str =
             Integer.toString(obj1.price);
         arPrice.add(str);
         sum = sum + obj1.price;
         return 1;
         }
         else{
             return 0;
         }
         obj1.
     }
}
```

△ bar String - Objects
△ howToUse String - Objects
△ name String - Objects
△ nCal Calendar - Objects
△ price int - Objects
△ recipe String - Objects
Press 'Ctrl+Space' to show Template

(그림 12) MVC 모델의 구현물 (가계부)

```
import java.util.*;
public class HouseMoney {
    ArrayList<String> arName =
        new ArrayList<String>();
    ArrayList<String> arPrice =
        new ArrayList<String>();
    int sum;
    public HouseMoney(){
        sum = 0;
    }
    public int CalcMoney(Objects obj){
        if(arName.size()==arPrice.size()){
        arName.add(obj.name);
        String str =
            Integer.toString(obj.price);
        arPrice.add(str);
        sum = sum + obj.price;
        return 1;
        }
        else{
            return 0;
        }
        obj.
    }
}
```

△ name String - Objects
△ price int - Objects
Press 'Ctrl+Space' to show Template

(그림 13) VMDC 모델의 구현물 (가계부)

때문에 MVC 모델의 구현물에서 바코드(bar), 관리법(howToUse), 유통기한(nCal), 레시피(recipe)는 쓸모없는 데이터의 이동임이 밝혀진다. 그에 반해 VMDC 모델의 구현물은 사령부(Dispatcher)가 가계부에 필요한 이름(name), 가격(price)만을 새로운 객체로 구성하여 보냄으로써 데이터 이동이 효율적으로 이루어졌다. 또한 코드의 가독성을 높인다. 사령부를 가지지 않는 obj1의 속성을 봤을 때 관계가 없

```
import java.util.*;
public class RefDispatcher {
    String bar, name, recipe, howToUse;
    int price;
    Calendar deadLine;
    HouseMoney hm;

   // 식별자를 이용하여 해당 물품의 데이터를 검색
   // 모든 데이터를 DB로 부터 받아 각각의 변수에 담아 놓았다고 가정

    public RefDispatcher(){
        hm = new HouseMoney();
    }

    Objects objA
     = new Objects
      (bar, name, price, deadLine, recipe, howToUse);

    public void RecordMoney(){
      Objects objHM =
        new Objects(objA.name, objA.price);
        hm.CalcMoney(objHM);
    }
}
```

(그림 14) 사령부(Dispatcher)의 구현물

```
import java.util.*;
public class Objects {
    String bar, name, recipe, howToUse;
    int price;
    Calendar nCal;

    public Objects(String bar, String name,
            int price,  Calendar deadLine,
            String recipe, String howToUse){
        this.bar = bar;
        this.name = name;
        this.price = price;
        this.nCal = deadLine;
        this.recipe = recipe;
        this.howToUse = howToUse;
    }

    public Objects(String name, int price){
        this.name = name;
        this.price = price;
    }
}
```

(그림 15) 물품의 객체 구현

〈표 4〉 사례연구에 쓰인 클래스의 이름과 기능

클래스 명	기 능
HouseMoney_Nondispatcher	MVC 모델을 구현한 코드로써, 사령부(Dispatcher)를 거치지 않는 클래스
HouseMoney	VMDC 모델을 구현한 코드로써, 사령부(Dispatcher)를 거치는 클래스
RefDispatcher	VMDC 모델의 사령부(Dispatcher) 클래스
Objects	식품이 들어올 때 마다 생성되는 물품의 객체를 생성하는 클래스

는 데이터가 있음으로써 혼란을 줄 수 있다. 하지만 사령부를 통해 재구성된 객체의 속성을 보게 되었을 때 필요한 속성만을 보이기 때문에 혼란스러움을 줄일 수 있다.

(그림 15)에 사령부의 코드와 물품(Objects)의 코드를 제시한다.

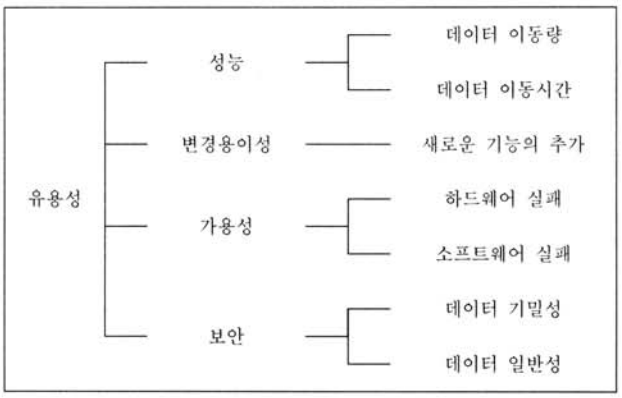
4.3.2 VMDC아키텍처의 평가

아키텍처 평가의 평가대상 품질 속성은 성능, 신뢰성, 가용성, 보안, 변경용이성, 이식용이성, 기능성, 가변성, 부분성,

〈표 5〉 VMDC 아키텍처의 평가

	평가 내용
변경용이성	컨트롤러가 변경되었을 경우, Dispatcher부분만 수정하여 변경을 할 수 있다.
이식용이성	어떠한 소프트웨어서라도 Dispatcher에 그 컨트롤러가 시작될 때의 데이터를 정확히 제시하게 되면 이식될 수 있다.
부분성	초기에 Dispatcher부분에 기본적인 선언과 컨트롤러만으로 시스템을 구현한 후, Dispatcher에 여러 컨트롤러의 재구성 방법을 제시해 나가면서 점진 적으로 시스템을 완성해 나갈 수 있다.
가변성	변경용이성의 우수성으로 인해 시스템의 확장이 용이하다.

〈표 6〉 VMDC 아키텍처의 유틸리티 트리



개념적 무결성 등이 있다[13]. 아래 표에 VMDC 아키텍처 평가 결과를 제시한다.

품질속성들 중 VMDC 아키텍처가 강점을 나타내는 평가 내용들을 표로 나타내었으며, 유용성을 유틸리티 트리로서 제시한다[13].

유틸리티 트리를 보아 본 VMDC 아키텍처는 성능면에서 이동하는 데이터의 양을 재구성 되자 않은 객체의 데이터에서 재구성 된 객체의 필요한 데이터의 양으로 줄이면서 이동시간에서 유용함을 보였다. 변경용이성 면에서 사령부에 재구성 정보를 제공함으로써 새로운 컨트롤러에 객체를 재구성 해 전송할 수 있다.

5. 향후 연구 방향

VMDC의 문제점은 사령부와 컨트롤러와의 의존성이다. 사령부가 데이터의 낭비를 막기 위해 컨트롤러에게 맞는 객체를 재구성하여 보내어 준다. 이를 위해서 사령부는 컨트롤러에 정보를 가지고 있어야 하며, 컨트롤러가 바뀌면 사령부의 재구성 방법도 바뀌어야 한다. 이것은 사령부와 컨트롤러가 의존성이 높다는 것을 말한다. 모듈 사이의 상호교류가 많고 서로의 의존이 많을수록 모듈들 사이의 결합도는 높아진다. 결합도가 높으면 한 모듈의 변화가 다른 모듈에도 영향을 주는 파문효과가 일어나게 된다[8-9]. 사령부와 컨트롤러와의 의존성이 높아 파문효과가 일어나는 것은 바람직하지 못한 상황이므로, 이를 향후 연구하여 의존성을 줄여 결합도를 낮추면서 응집력을 높일 수 있는 더욱 바람직한 방향으로 향후 연구할 것이다.

6. 결 론

이 논문에서는 현 물품관리 아키텍처의 문제점을 찾고 이를 개선하기 위한 VMDC(View, Model, Dispatcher, Controller) 아키텍처를 제시하였다. VMDC는 기존의 MVC 모델과 차별화하여 물품관리에 적합하도록 한 아키텍처로써 이 모델의 핵심은 사령부이었다. 이에 관한 자세한 설명은 3.1.2절에서 하고 있다.

〈표 7〉 VMDC와 기존 아키텍처의 비교

종류	VMDC	기존 아키텍처
객체의 구분	각 컨트롤러 마다 객체가 가지는 속성이 차이가 있음	하나의 공통된 객체를 사용 함
데이터 효율성	쓰이는 속성과 데이터만 객체 안에 속해 있음	쓰이지 않는 속성과 데이터가 객체에 포함되어 있음
가독성	객체를 보았을 때 필요 없는 데이터는 없다고 간주할 수 있음	필요 없는 데이터도 포함되어 있음으로 이를 따져 봐야 함
사령부와의 결합도	높음	
이동 데이터 수	적음	많음

뷰가 사용자 인터페이스 부분을 담당한다. 모델은 이런 뷰를 관리하고 뷰에서 받아온 정보를 임시로 저장하는 기능을 한다.

컨트롤러는 각자 자신에게 주어진 책임을 해결한다.

사령부는 각 컨트롤러에 전달되는 데이터의 이동을 객체를 재구성함으로써 통제, 관리한다. 본 연구에서 데이터의 이동을 표현하는 Data Move Diagram을 제시하였으며, Data Move Diagram은 객체가 가지고 가는 속성, 즉, 데이터를 중점으로 보는 다이어그램으로써 데이터의 양을 화살표 굵기로 표현하고 그 속성들은 상자 안에 표현해 준다.

VMDC 아키텍처는 각 컨트롤러에 전달되는 데이터를 정리하고 낭비되는 데이터가 없도록 재구성하는 기능이 있다. 이 사령부를 배치하여 코드 길이상의 차이는 없지만, 데이터이동을 효율적으로 하고, 이로 인해 가독성을 높이는 효과를 볼 수 있다.

VMDC 아키텍처의 문제점은 사령부가 각 컨트롤러와 너무 많은 의존관계를 가지고 있어서 컨트롤러가 변경되면 사령부에 영향이 가게 되는 것인데 이는 향후에 연구 하고자 한다.

참 고 문 헌

[1] 최은만, "(객체지향) 소프트웨어공학", 사이텍미디어, 2005

[2] Rich Green, "Design Enterprise Applications with the J2EE Platform, Second Edition:, Sun Microsystems, 2002

[3] James W.Cooper, "Java Design Patterns", Addison-Wesley, 2002

[4] 유주현, "MVC 기반의 웹 컴포넌트 컨테이너 시스템에 관한 연구", 순천대학교 대학원 이학석사학위논문, 2003

[5] 김철민, "Object Pool 패턴을 이용한 WIPI기반 MVC모델 개선에 관한 연구", 전남대학교 대학원 석사학위논문, 2005

[6] Andrew Hessey and David Carington, "Comparing Two User-Interface Architecture : MVC and PAC", Software Verification Research Center, 1996

[7] 이선숙 박문화, "MVC 모델을 적용한 자동 문제 출제 시스템", 성신여자대학교 컴퓨터정보학부, 2006

[8] 윤 청, "공적인 소프트웨어 개발 방법론", 생능출판사, 1999

[9] Cay Horstmann, "Object-Oriented Design & Patterns", San Jose State University

[10] 조셉 슈멀러, "초보자를 위한 UML 객체지향설계(3판)", 정보문화사, 2004

[11] McLaughlin, O'Reilly , "Head First Object-Oriented Analysis and Design", 한빛미디어, 2007

[12] Paul Clements, Rick Kazman, Mark Klein, "Evaluating Software Architectures", Addison Wesley

김 다 정

e-mail : dj_km@hanmail.net

2006년~현 재 안동대학교 컴퓨터공학과 학부생

관심분야 : Software Engineering, Software Architecture

이 은 서

e-mail : eslee@andong.ac.kr

2001년~현 재 ISO/IEC 15504 국제 심사원

2004년 중앙대학교 컴퓨터공학과(박사)

2004년~현 재 임베디드 산업협회 전문 위원

2004년~현 재 한국정보통신기술협회 위원

2005년~2007년 숭실대학교 정보미디어기술연구소 연구교수

2008년 3월~ 현 재 안동대학교 컴퓨터공학과 조교수

관심분야 : CBD, Formal method, Quality model, SPI(Defect Analysis)