제33회 한국정보처리학회 춘계학술발표대회 논문집 제17권 제1호 (2010. 4)

다중속성 기반 객체의 효율적인 활용에 관한 연구

김다정*, 김윤호*, 이은서*
*안동대학교 컴퓨터공학과
e-mail:
dj_kim@hanmail.net
unokim@andong.ac.kr
eslee@andong.ac.kr

A Study on Efficency Use of Multi-attribute Based Objects

Da-Jeong Kim*, Yoon-Ho Kim*, Eun-Ser Lee*
*Dept of Computer Engineering, An-dong University

요 약

정보 활용이 유연한 현대 소비자들의 알 권리 충족을 위해 현존하는 모든 사물에 제공되어야 하는 정보가 늘어나고 있음에 따라 사물들의 객체의 속성도 또한 증가하고 있다. 다중속성 기반 객체를 관리하는 시스템에 있어 효율적인 객체관리를 위한 방법을 연구하고 그 방안으로 객체를 재구성하여 분배하는 디스패처(Dispatcher)를 제시한다. 또한 의존성에 관련한 향후 연구 방향을 제시한다.

1. 서론

현대사회는 정보처리기술, 통신기술의 급속한 진전에 따라 정보의 산업화와 함께 산업의 정보화, 사회의 정보화, 가정의 정보화를 초래하여 우리들의 생활에 근본적인 변화를 초래하고 있다. 또한 정치. 경제. 사회 등의 각 분야에서의 정보의 가치가 증대되고 그 정보는 컴퓨터에 입력. 가공. 보관. 출력되는 컴퓨터정보가 중심이 된다[1]. 또한 소프트웨어는 컴퓨터 기반 시스템이나 제품 변화에 있어 핵심 요소가 되었다[2]. 이에 따라 모든 가치는 컴퓨터 내의 객체로 표현되며 소비자들의 정보 활용 능력이 증가함에 따라 객체 내 속성도 증가하고 있다. 이런 객체에 대한 정보는 대게 데이터베이스에 저장되어 있으며 객체를 생성할 때 이런 정보들을 가져와 생성하게 된다. 이런 프로세스를 효율적으로 관리하기 위해 여러 아키텍처들이 존재하는데 MVC, 브로커 스타일, 트랜잭션 처리 스타일, 파이프 필터 스타일 등이 있다[3]. 이런 각각의 아키텍처들의 공통된 성향은 다중속성 기반 객체로 여러 가지 처리를 할 경우 객체에 모든 데이터를 담아서 처리를 한다는 것이다. MVC 모델을 예를 들어 설명한다.

그림 1, MVC 모델에서 뷰(View)는 사용자 인터페이스만을 담당한다. 또한 모델(Model)은 데이터베이스에서 데이터를 가져와 객체를 구성하고 각각의 컨트롤러(Controller)를 부르는 일을 한다[1]. 컨트롤러는 각각의 처리할 일을 처리한다. 그림 1의 모델에서 객체를 구성할 때, 컨트롤러 1에서 필요한 속성이 A1, A3뿐이고 사용하지 않는 속성이 A2, A4, A5, A6, A7, A8로써 더 많지만 모든 속성을 전달해 준다. 이는 쓸모없는 데이터의 흐름을 만드는 요인이 된다.

다른 아키텍처에서도 이런 현상은 동일하다. 다중속성 기반 객체에 대한 여러 처리를 위한 효율적인 관리는 이루어 지지 않는다. 이를 개선하고자 각각의 컨트롤러 혹은 컴포넌트에 맞는 객체를 재구성하여 배분하는 디스패처

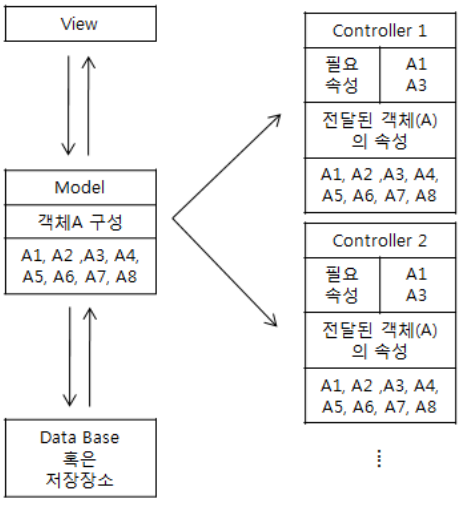

그림 1 MVC 모델의 객체 전달

(Dispatcher)를 아래 본문에 제시하고 의존성과 결합도, 응집도에 대한 향후 연구 방향을 제시한다.

2. 환경 설정

디스패처(Dispatcher)는 한 객체에 대해 많은 정보가 제공되어야 하며, 그러므로 인해 객체 내에 속성, 즉 클래스 혹은 객체의 데이터 변수(Data variable)를 나타내는 필드(Field)가 매우 많은 다중속성 기반 객체이어야 한다[4]. 또한 동일한 객체의 필드를 활용하여 여러 가지 기능의 컴포넌트들이 있으며 다중속성 기반 객체의 모든 필드를 사용하는 컴포넌트의 수가 많지 않을 경우 적합하다.

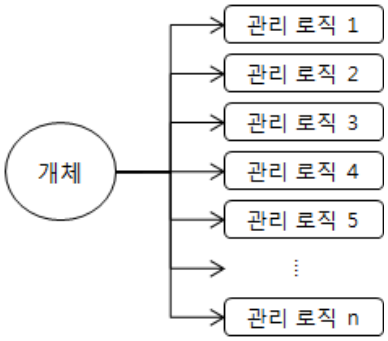

그림 2 디스패처 사용 환경

디스패처는 아키텍처를 배경으로 하여 아키텍처의 구성요소가 될 수 있고, 혹은 시스템 내의 매개 클래스로 하여 여러 클래스들을 관리할 수도 있다.

3. 디스패처(Dispatcher)

객체가 가진 속성이 A1~A8까지 8개라고 가정한다. 아래 표에 각 컴포넌트가 필요로 하는 속성을 가정한다.

컴포넌트	필요 속성
Component 1	A1, A3
Component 2	A2, A8
Component 3	A7, A8
Component 4	A4, A5, A6
Component 5	A1, A2, A4
Component 6	A5

표 1 컴포넌트와 필요 속성 가정

디스패처는 데이터베이스 혹은 서버에서 가져온 데이터들을 모두 가지고 있다. 그리고 컴포넌트 별 배분되어야 할 데이터의 종류를 표의 형태로 가진다. 이 데이터는 벡터(Vector)나 배열(Array)로 표현될 수 있다. 이 표를 배분표라 한다.

특정 컴포넌트가 호출되면 디스패처는 배분표에서 컴포넌트를 찾은 후, 필요한 속성을 확인한다. 그 후 컴포넌트에 필요한 속성으로만 이루어진 객체를 재구성한다.

사용자가 컴포넌트를 호출하면, 데이터베이스 혹은 서버에서 모든 데이터를 가져온다. 그 후 디스패처는 컴포넌트에 필요한 속성을 확인하고 객체를 재구성하여 컴포넌트에게 처리를 맡긴다.

그림 3은 디스패처 내부의 상세도이다. 디스패처 내에는 위에서 설명되어 있는 배분표 가지고 있다. 디스패처는 모든 데이터를 가지고 있다. 이 모든 데이터를 배분표이 분류하여 갖게 된다. 배분표는 컴포넌트의 이름과 그 컴포넌트에서 객체에게 요구하는 객체의 속성을 저장하고 있다.

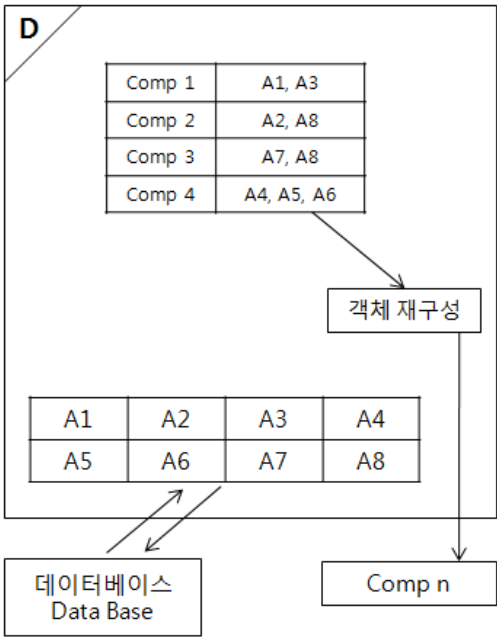

그림 3 디스패처 상세도

이때 배분표의 형태는 벡터나 배열의 형태가 되며 이 배분표를 이용해 객체를 재구성한다. 그림 3에서 디스패처의 배분표에 Component 1 ~ Component 6까지의 컴포넌트가 저장되어있고 각각의 컴포넌트가 필요로 하는 속성이 각각 저장되어 있다. 사용자가 Component 5에서 처리해야 하는 명령을 했다고 가정하자. 디스패처는 이 처리들에 필요한 모든 데이터를 가지고 있으며 Component 5이 필요로 하는 A1, A2, A4만을 가지고 객체를 재구성 한다. 속성이 A1~A8이었던 이 객체는 이제 세 개의 속성만을 가진 객체가 된다. 이 작아진 객체는 Component 5에게로 전달되어 처리된다.

4. 향후 연구 방향

향후 연구에서는 기존의 프로그램과 디스패처를 적용한 프로그램을 비교 및 분석을 한 후, 디스패처의 사용이 효율성이 있음을 밝히고 이를 수치적으로 표현하는 연구를 한다.

또 디스패처 사용의 문제점은 디스패처와 컨트롤러와의 의존성이다. 디스패처가 데이터의 낭비를 막기 위해 컨트롤러에게 맞는 객체를 재구성하여 보내어 준다. 이를 위해서 디스패처는 컨트롤러에 정보를 가지고 있어야 하며, 컨트롤러가 바뀌면 디스패처의 재구성 방법도 바뀌어야 한다. 이것은 디스패처와 컨트롤러가 의존성이 높다는 것을 말한다. 객체지향에 있어서 모듈 사이의 상호 교류가 많고 서로의 의존이 많을수록 모듈들 사이의 결합도는 높아진다고 말한다. 결합도가 높으면 한 모듈의 변화가 다른 모듈에도 영향을 주는 파문효과가 일어나게 된다[5][6]. 디스패처와 컨트롤러와의 의존성이 높은 것은 바람직하지 못한 상황이므로, 이를 향후 연구하여 의존성을 줄여 결합도를 낮추면서 응집력을 높일 수 있는 더욱 바람직한 방향으로 향후 연구할 것이다.

5. 결 론

본 연구에서는 다중속성 기반 객체 관리의 문제점을 찾고 이를 개선하기 위한 디스패처(Dispatcher)를 제시하였다. 디스패처는 기존의 방식과 차별화하여 다중속성 기반 객체의 여러 처리에 적합하도록 한 방법으로써 이 연구의 핵심은 디스패처이었다. 이에 관한 자세한 설명은 3절에서 하고 있다.

디스패처는 데이터베이스나 다른 저장 장소에서 데이터를 모두 가져오고, 디스패처 내의 배분표를 이용하여 분류한 뒤 객체를 재구성 하는 방식으로 각 컴포넌트에 전달되는 데이터의 이동을 통제, 관리한다. 이로써 불필요한 데이터의 이동을 막고 필요한 데이터의 이동만 이루어지게 함으로써 효율적인 객체의 사용이 이루어지도록 한다.

종류	디스패처 사용	기 존
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
이동 데이터 수	적음	많음

표 2 디스패처 사용과 기존 방식의 비교

또한, 향후 연구에서 디스패처 사용의 효율성을 수치적으로 입증하고 효율적인 아키텍처를 개발한다. 또한 의존성의 문제를 해결한 방안을 찾고 향후 개선된 디스패처를 연구한다.

참고문헌

[1] 김배원 “정보사회와 소비자 알 권리” 한국소비자원
[2] Roser S.Pressman, “Software Engineering :A practitioner’s approach”, Mc Groaw Hill
[3] 최은만 “객체지향 소프트웨어 공학” 사이텍미디어
[4] Amold, Gosling, Holmes "The java programming language" 4th Ed. addison-Wesley
[5] 윤 청 “공적인 소프트웨어 개발 방법론” 생능출판사
[6] Cay Horstmann "Object-Oriented Design&Patterns" San Jose State University