제21회 한국정보처리학회 춘계학술발표대회 논문집 제11권 제1호 (2004. 5)

인간중심의 소프트웨어 개발을 위한 구문 객체 모델과 관계 디자인

전영준*, 최용식*, 신승호*
*인천대학교 컴퓨터공학과
e-mail 0961144@incheon.ac.kr

Sentential Object Orient Model And Relationship Design For Humane Software Developement

Young-Jun John*, Yong-Sik Choi*, Seung-Ho Shin*
* Dept. of Computer Engineering, incheon University

요 약

소프트웨어 개발에서는 서비스 기간동안, 운영환경에 쓰일 부분을 관리하고 수정해야 할 사안이 발생한다. 그러나 이러한 변경은 구조적인 다른 문제점, 즉 객체와 개발자간의 효율적인 교신(communication)능력을 중요히 요구한다. 이를 위해 개발자는 시스템의 행위에 대해서 사용자와 프로그래머 모두가 이해하기 쉽고, 이후에 이러한 구조로부터 소프트웨어 아키텍처를 생성할 수 있는 OO(object orient) Model 을 사용한다. 그러므로 개발자는 이러한 환경에 대처하기 위해서 코드 기반을 보다 유연한 방법으로 구축할 필요성이 있다. 이를 위해 본 연구에서는 표기언어가 갖는 구문적 특성과 성질을 활용한 구문 객체 모델과 관계를 통해 개발자의 소프트웨어 구조의 생성을 지원하고 핵심요소의 표현의 일원화를 위한 방법을 제시한다.

1. 서론

소프트웨어 개발에는 정책(policy)결정을 통해 목적과 방향을 정하고 이러한 정책으로부터 모델을 표현하게 되는데, 이와 같이 소프트웨어 개발은 어떠한 컨셉에 대해서 프로세스 형태로 나타내며 객체간에 원활히 맞물려 돌아갈 수 있도록 관계를 설계하는 과정을 가진다. 그러나 같은 모델을 도출하기 위해서 환경과 관심분야 같은 개발자간의 개인적인 차이는 개발자간의 다른 관점, 다른 방법으로 문제에 접근하여 다양한 분석을 도출하게 된다. 따라서 분석된 관계가, 누군가에게는 기능이나 흐름으로 표현되는 분석의 불일치가 발생한다. 이러한 분석의 불일치는 프로세스를 개인마다 다르게 인식할 수 있는 추상화 된 패턴으로 분리해 내려는데 그 원인을 찾을 수 있다.

그러나 현실적으로 소프트웨어 개발 뿐만이 아니라 실생활에서 보통 사람들은 주제를 가지는 생각, 컨셉을 표현하는데 있어서 음률을 가지는 음악이나, 특정 화풍의 그림, 혹은 한글과 같은 표기문자 등을 통한 다양한 표현방법을 사용한다. 그런데 이러한 방법들에서 대개 나름대로의 구성 요소들과 정형화된 문법을 가진다는 점을 주목할 필요가 있다. 본 연구의 주요 연구 대상인 표기문자는 시간과 공간적인 시제나, 목적에 대해 형용하는 수식과 같이 안정적인 문법을 가지고 있다.

다시 말해 소프트웨어 개발상의 객체모델과 관계를 설계할 때 초기의 요구사항을 가장 잘 표현할 수 있는 가능성을 문자 스스로 가지고 있음을 인지하고 이를 활용하여 객체 자신을 표현할 때 목적과 수식을 가지도록 표현한다. 이를 위해 소프트웨어 개발상의 패턴들을 구문적인 요소로서 인식하고 각 구문간의 관계에 대해 제시한다.

본 논문의 구성은 다음과 같다. 2 장에서 관련 연구로서 일반적인 소프트웨어 명세를 기술하고 3 장에서 구문 객체를 설계하며, 4 장에서 결론을 맺는다.

본 연구는 한국과학재단 지정 인천대학교 동북아전자물류 연구센터의 지원에 의한 것입니다.

2. 관련 연구

2.1 소프트웨어 명세

일반적인 어플리케이션 형태는 자료 (data) 로부터 특정한 행위 (behave) 를 통해서 의미 있는 정보를 추출해 사용한다. 이러한 행위에는 최소한의 결합을 통해 독립된 알고리즘이나, 자료에 대한 검색, 접근을 해내기 위한 접근자를 예로 들 수 있다. 또한 이러한 행위들은 부산물로 행위의 결과나 과정을 나타내는 일련의 사건 (event) 을 발생시킬 수 있다. 게다가 자료의 I/O 를 위해 특정장치 (device) 와 통신이 가능하다. 이러한 전반적인 프로세스의 흐름은 어플리케이션 (application) 을 통해서 통제 할 수 있다. 그리고 사용자와의 인터페이스를 위해 버튼이나, 메뉴와 같은 외관적인 요소를 제공하기도 한다. 게다가 특정 Framework 형태의 솔루션과 통신 할 수 있어야 한다.

2.2 소프트웨어 계층

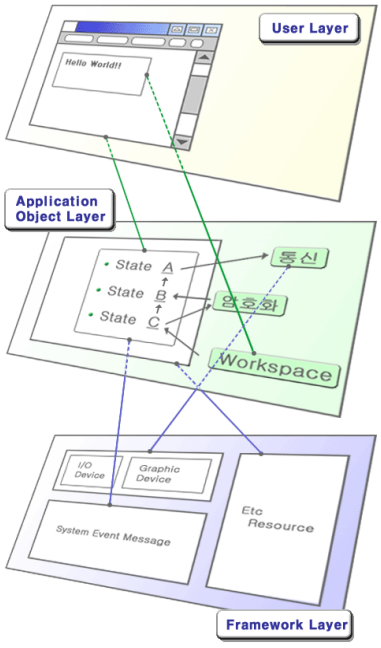


그림 1 소프트웨어 계층

소프트웨어를 구성적인 측면에서 바라보면 첫째 Frame 층은 시스템에 특화 되어 있는 자원에 대한 접근을 지원하고 둘째 User 층은 버튼을 누르고 메뉴를 선택하고 공란에 텍스트를 입력하는 등의 어플리케이션의 질의에 의사를 표현한다. 마지막으로 사용자와 시스템 자원을 연결하는 객체 층은 어플리케이션의 프로세스를 제어하고 사용자와 질의에 응답하여 새로운 질의를 생성하며 시스템자원을 사용하는 층이다. 대개의 소프트웨어가 프로세스를 생성할 때 어플리케이션 중심적으로 생성과, 구조, 행위중심적으로 이루어지는 것에서 사용자의 목적을 명확히 구체화 하기위해 어플리케이션 중심적이 아닌 사용자 즉 인간 중심적인 객체 설계방법이 요구 된다.

예를 들어 -아름다운 감정을 유발하는 '그림' 이라는 컨셉에 대해서 프레임이 '모나리자' 와 같은 구현적 접근과 패턴이 물감, 화풍, 캔버스 같은 요소적 접근을 제시 한다. 그러나 이러한 접근은 컨셉이 외곡되거나 제한될 수 있으므로 초기의 구문적, 언어적 형태 그대로 설계 될 수 있는가에 초점을 둔다. 어플리케이션이라는 추상적형 형태를 표현하는데 다시 추상적인 어플리케이션 중심의 패턴 단위보다는 개발자라는 인간에게 적합한 구문체계에 상응하는 형태로 재 접근이 필요하다.

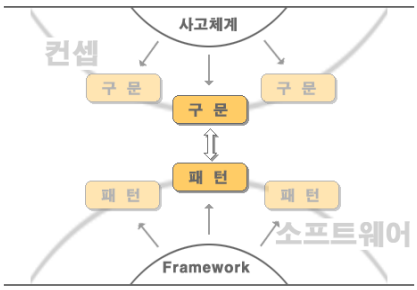


그림 2 컨셉과 소프트웨어간의 표현 관계

2.3 필요조건을 만족하는 상세설명

이를 위해 본 연구에서는 소프트웨어 개발 지원을 위해, 개발에 필요한 객체들을 직관적이고 독립적인 형태로 표현하기 위한 구분방법과 목적, 그리고 이들 간의 관계를 최적의 형태로 설계를 제시한다.

적절한 객체를 식별하고 울바른 범주의 구문과 구문간의 인터페이스를 정의해야 한다. 구문 문법에 대한 설계는 지금 당장 가지고 있는 문제를 해결 할 수 있어야 하지만 나중에 생길 수 있는 문제나 추가된 요구 사항들도 수용할 수 있도록 일반적이고 포괄적으로 되어 있어야 한다., 즉 재설계를 하지 않아도 다시 사용할 수 있어야 하고, 아니면 가능한 최소한의 수정을 통해서 다시 사용할 수 있어야 한다.

3. 구문 객체

3.1 구문 객체 설계 시 고려사항

구문 객체는 언어상 단어의 품사와 같이 객체를 문법상의 의미 직능·형태에 따른 분류이다. 구문 객체는 기존 환경 내에서 반복적으로 일어나는 문제들을 설명하고 그 문제들에 대한 해법의 핵심을 설명할 수 있어야 한다. 보통 객체설계는 는 다음 세 가지 요소를 포함하여야 한다.

1.이름 : 한 두 단어로 설계문제와 해법으로 설계의도를 서술한다.

2.문제 : 언제 구문을 사용하는가를 서술하며 해결할 문제와 그 배경을 설명한다.

3.해법 : 구문객체를 구성하는 요소들과 그 요소를 간의 관계, 책임 그리고 협력 관계를 서술한다. 구문 객체설계는 문제에 대한 추상적인 설명을 제공한다.

객체 일반화에 대한 시각은 어떤 것이 객체이고 어떤 것이 아닌가를 분석하는 데 영향을 준다. 어떤 사람에게는 객체인 것이 다른 사람에게는 단순한 기능 수준 일 수 있다, 본 연구에서는 어느 정도 수준의 추상화를 갖는 객체 설계를 제시한다.

본 연구에서 객체는 “특정한 상황에서 일반적 설계 문제를 해결하기 위해 상호 교류하는 수정 가능한 객체와 클래스들” 에 대한 설명이다.

3.2 기본 구문 분류

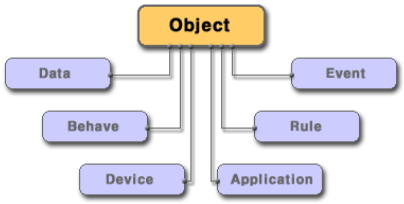


그림 3 기본 구문 분류

3.2.1 OBJECT 구문 객체

객체간은 상속이나 합성과 같은 방법을 사용해 서로 통신을 하게 되는데 이러한 통신을 위해 서로간의 별도의 인터페이스를 두는 것은 이후의 확정성을 고려해볼 때 지나치게 많은 인터페이스를 양산 하게 될 우려가 있다. 그러므로 이를 해결 하기 위해서는 다른 객체간의 인터페이스를 위해 상위의 단일하고 추상적인 통로가 필요하며 Object 구문객체는 이를 지원하여야 한다. 또한 Object 구문객체는 객체의 생성방법과 할당 제거의 명시적인 투명성을 위한 안전한 방법을 제공하기위해 다음과 같은 패턴상의 성질을 만족하여야 한다.

*Factory:*구체적인 클래스를 지정하지 않고 관련성을 갖는 객체들의 집합을 생성하거나 서로 독립적인 객체들의 집합을 생성할수 있는 인터페이스를 제공한다.

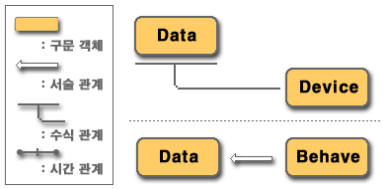
*Builder:*복합 객체의 생성 과정과 표현 방법을 분리함으로써 동일한 생성공정이 서로 다른 표현을 만들 수 있게 한다.

*Factory method:*객체를 생성하는 인터페이스를 정의하지만, 인스턴스를 만들 클래스의 결정은 서브 클래스가 한다. 클래스의 인스턴스를 만드는 시점을 서브 클래스로 미룬다.

*Composite:*부분-전체 계층을 나타내기 위해 복합 객체를 트리 구조로 만든다. 컴포짓 패턴은 클라이언트가 개별적 객체와 복합객체 모두를 동일하게 다루도록 한다. Decorator 객체에 동적으로 책임을 추가할 수 있게 한다. 기능의 유연한 확장을 위해 상속 대신 사용할 수 있는 방법이다.

3.2.2. DATA 구문 객체

data 구문 객체는 구문 객체들을 중심으로 목적과 대상에 해당되며 behave 의 수식을 받아 행위를 표현한다. 이는 어플리케이션 상에서 사용자들이 데이터저장매체의 복잡한 구조나 전문적인 지식이 없더라도 손쉽게 정보에 접근해서 원하는 분석을 수행할 수 있어야 한다. 이를 위해 자료로부터 일련의 정보를 추출하고 자료에 대한 접근과 검색, 저장과 출력에 을 지원해야 하며, 실제 Device 와 무관한 추상적인 Data 에 대한 접근을 뜻한다.



이러한 자료들은 서로 관계를 가질 수 있으며, 일련의 연산자를 통해 관계에 대해 집합연산을 하고 결과로 다른 관계나 정보를 도출해 낼 수 있어야 한다. 또한 자료들은 정확하고 일관성을 유지할 수 있어야 하며, 검색과 같은 데이터 연산자를 지원 하기위해 다음과 같은 패턴상의 성질을 만족하여야 한다.

Iterator: 내부 표현 방법을 노출하지 않고 객체의 원소를 순차적으로 접근할 수 있는 방법을 제공한다.

Flyweight: 작은 크기의 객체들이 여러 개 있는 경우, 객체를 효과적으로 사용하는 방법으로 객체를 공유하게 한다.

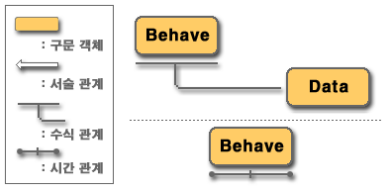
Value Set: 어플리케이션은 동시작업(Concurrent) 프로그램에 포함될 코드 중에 setup 또는 기타의 사유로 변경이 가능함에도 불구하고 source coed 에서 상수 값으로 지정하여 해당 값의 변동 시 유지보수의 어려움을 증대 시키는 요인이 되므로 이를 해결하기 위해 Value Set 을 이용한 기본값을 지정하고 이를 통해 자료를 검증한다.

3.2.3. DEVICE 구문 객체

device 구문객체는 data 의 성질을 형용하는 수식으로 어플리케이션상 사용자나 시스템과 인터페이스 할 수 있는 기본 장치들인 마우스, 키보드나 그래픽장치 등에대한 접근을 제공한다.

3.2.4 BEHAVE 구문 객체

behave 구문 객체는 data 의 행위에 대한 서술을 뜻하며 일반적인 OO(object orient) Model 에서 Method 와 같은 객체의 행위를 표현하기 위해 객체가 제공할 기능/ 행위 간의 합성을 지원한다.



이러한 행위로 합성등에 낮은 결합 도를 가져야 하며 특정한 알고리즘과 제어 흐름을 기술하고, 행위 구문 객체는 하나의 작업을 수행하기 위해 객체 집합의 협력을 지원 하기위해 다음과 같은 패턴상의 성질을 만족하여야 한다.

State: 객체의 내부 상태에 따라 행위를 변경할 수 있게 한다. 이렇게 하면 객체는 마치 클래스를 바꾸는 것처럼 보인다.

Strategy: 알고리즘이 존재하는 경우 각각의 알고리즘을 별도의 클래스로 캡슐화하고 클라이언트에 영향을 주지 않도록 독립적으로 알고리즘을 다양하게 변경할 수 있게 한다.

Visitor: 객체 구조의 요소들에 수행할 오퍼레이션을

표현한 패턴이다. 오퍼레이션이 처리할 요소의 클래스를 변경하지 않고도 새로운 오퍼레이션을 정의할 수 있다.

3.2.5 APPLICATION: 구문 객체

APPLICATION 구문 객체는 생성된 구문간의 관계가 문장을 이루게 되어 각 문장간의 관계를 단락형태로 관리하기 위한 객체이며 이는 일반적인 어플리케이션에서 사용자에게 메뉴나 Window 와 같은 UI 를 제공하기위해 에플리케이션은 사용자와의 인터페이스를 위해 시각적인 메뉴바 나 버튼 과 같은 컴포넌트 구조를 가질 수 있어야 하고 각 구조들간의 메시지를 전달하며 에플리케이션의 생성과 종료를 제어하기 위한 방법을 제공 하기위해 다음과 같은 패턴상의 성질을 만족하여야 한다.

Chain of responsibility: 요청을 처리할 수 있는 기회를 하나 이상의 객체에게 부여 함으로써 요청하는 객체와 처리하는 객체 사이의 결합 도를 없애려는 것이다. 요청을 해결할 객체를 만날 때까지 객체 고리를 따라서 요청을 전달한다.

Command: 요청을 객체로 캡슐화 하여 서로 다른 요청으로 클라이언트 파라미터화 하고, 요청을 저장하거나 기록을 남겨 오퍼레이션의 취소도 가능하게 한다.

Observer: 객체사이에 일대 다의 종속성을 정의하고 한 객체의 상태가 변하면 종속된 다른 객체에 통보가 가고 자동으로 일어나게 한다.

Proxy: 다른 객체로의 접근을 통제하기 위해서 다른 객체의 대리자 또는 다른 객체로의 정보 보유자를 공유한다.

3.2.6. EVENT 구문 객체

행위(Behave)는 행위의 결과나 상황, 과정을 나타내는 일련의 사건(event)을 발생시킬 수 있다. 이를 다른 구문 객체에게 통보하거나 전달 받기 위한, 혹은 트리거로서 사용될 수 있는 일반적인 메시지들에 대해 제어하기 위한 방법을 제공한다.

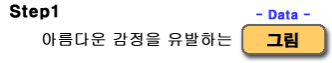
3.2.7. RULE 구문 객체

Application 구문 객체의 접근에 대한 권한을 제어하는, 사용자 PROFILE 과 같이 어플리케이션이 실행하는 방식에 영향을 주는 변경 가능한 옵션 집합이다. 사용자가 책임에 로그 온 하거나 변경할 때 사용자의 프로파일에 있는 각 옵션에 대해 값을 생성할수 있으며 사용자가 자신의 요구에 맞게 에플리케이션의 실행을 변경할 수 있어야 한다. 에플리케이션의 해당 기능과 자료에 접근할 수 있도록 허용하는 어플리케이션 권한 레벨이다.

각 사용자는 최소한 하나의 책임을 가지며, 몇 명의 사용자가 같은 책임을 공유할 수 있다. 시스템 관리자는, 사용자에게 어블리케이션과 함께 제공되는, responsibility 하나를 지정하거나 사용자를 위해 새로운 책임을 생성시킬 수 있다.

3.3 구문 문법 적용 예

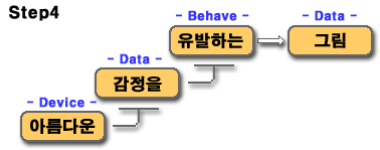
하단의 각 단계들은 개발 컨셉을 구문 개체를 기반으로 구성한 관계로 적용한 일부 예이다.



첫 번째 단계는 '그림' 객체가 가져야 할 행위와 수식이 표현된 초기 컨셉을 나타낸다.

두 번째 단계는 '그림' 객체는 '유발하는' 행위를 서술로서 관계함을 나타낸다.

세 번째 단계는 '유발하는' 행위는 구문 문법상 행위에 대한 목적에 해당하는 '감정을' 객체의 수식을 받음을 나타낸다.



네 번째 단계는 '감정을' 이라는 목적은 '아름다운' 이라는 구문을 통해 수식을 받는다. 이와 같이 구문관계를 통해 컨셉을 개발상의 관계로 표현할 수 있다.

4. 결 론

인간 중심적인 개발은 인간의 추상적인 정신구조에 적합하게 소프트웨어 개발상의 객체 구조와 관계를 표현하는데 그 중심을 둔다. 이를 위해 본 연구에서는 표기문자가 시간과 공간적인 시제나, 목적에 대해 형용하는 수식과 같은 안정적인 문법에 기반한 구문 객체간의 관계를 제시하였다. 이는 소프트웨어를 개발하는 동안에는 특정한 부분을 제외하면 일반적인 상황에서 재활용할 수 있는 부분이 존재하는데 이와 같이 입증된 기술을 구문형태로 표현해두면 새로운 개발자들은 이러한 구문을 문제해결을 위한 어휘로서 더 자주 유용하게 사용할 수 있다. 그러나 기존의 패턴분석에 대응한 구문중심의 분석과 구문상의 관계 대한 상세 연구가 추가적으로 요구되며 향후 구문객체를 기반으로 한 저작도구 개발을 통해 인간 중심적인 소프트웨어 개발과, 설계 연구등에 활용할 수 있을 것으로 판단한다.

참고문헌

[1]Roger S. "Software Engineering, A Practitoner's Approach"
[2]Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides , "Design Pattern"
[3]Steve McConnell, "Professional Software Development"
[4]Jef Raskin, "Humane Interface"
[5]카이호 히로유키, "인터페이스란 무엇인가"
[6]Leon Starr, "Executable UML"
[7]김방한, "현대 언어학의 원류"
[8]옴베르토 에코, "기호학 이론"