객체지향 개발방법의 체계적 구성
(A Systematic Construction of Object-Oriented Methods)

김 형 호 † 김 영 곤 †† 배 두 환 ††† 김 민 경 †††† 유 병 규 ††††
(Hyung Ho Kim) (Young Gon Kim) (Doo Hwan Bae) (Min Kyung Kim) (Byung Kyu Yoo)

요 약 객체지향 방법론들은 이해하기 쉽고 확장성이 우수한 모델을 제공하기 때문에 현재 크고 복잡한 소프트웨어를 개발하기 위해서 널리 사용되고 있다. 객체지향 방법론들을 개발에 적용하기 위해서는, 첫째 개발방법론으로부터 과제 특성에 맞는 개발방법을 구성해야 하며, 둘째 이로부터 개발절차를 구성하여야 한다. 그러나 현재 제시되고 있는 객체지향 방법론들에서는 개발방법과 개발절차를 구성하는 과정을 개발자들의 이해와 직관에 의존하고 있다. 이를 개선하기 위해서 본 연구에서는 변용가능한 객체지향 방법론을 제안함으로써, 개발방법 구성의 문제를 체계적으로 지원하고자 한다.

개발방법의 구성을 체계적으로 지원하기 위하여 본 연구에서는 개발방법 구성의 문제를 계획구성의 문제로 변환하여 계획구성 기법을 적용한다. 계획구성의 과정에서 개발하고자하는 소프트웨어의 구조와 특성을 기술하는 아키텍처를 입력으로 받아들여 이에 적합한 개발방법이 계획된다. 아키텍처는 소프트웨어 개발과정에서 일관되게 유지되어야 하는 결정들을 기술하기 때문에 이를 중심으로 개발방법을 구성함으로써 소프트웨어의 특성을 효율적으로 반영할 수 있다.

Abstract Object-oriented methodologies are widely used in the development of huge and complex systems since these methodologies produce the comprehensible and extensible model of systems. In order to apply an object-oriented methodology to a software development, developers should derive a method customized to a specific project from the methodology and, then, construct a development process from the method. Unfortunately, existing object-oriented methodologies lack a systematic facility for the construction of a method and a process. In this paper, we present a customizable object-oriented methodology to support the systematic derivation of a project specific method from the generic methodology.

To derive a project specific method from the methodology, we transform the problem of a derivation to a planning problem and apply planning techniques. Our planning technique uses the description of an architecture that captures the structure and characteristics of a software under development as input and constructs an appropriate method with respect to the architecture. The architecture-centered construction facilitates the effective handling for the characteristics of the software since an architecture capture the decisions that must be maintained consistently during the development.

† 비 회 원 : 한국과학기술원 전산학과
hhkim@salmosa.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과
ygkim@salmosa.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
bae@salmosa.kaist.ac.kr
†††† 비 회 원 : 한국통신 연구개발본부 연구원
5366537@multi.kotel.co.kr
5265079@multi.kotel.co.kr
논문접수 : 1999년 2월 5일
심사완료 : 2000년 2월 3일

1. 서 론

현재 크고 복잡한 소프트웨어를 개발하기 위해서 객체지향 방법론들이 널리 적용되고 있다[1-4]. 이처럼 객체지향 개발 방법론들이 각광받는 주된 이유는, 첫째 해결하여야 할 실 세계의 문제를 실 세계를 구성하는 객체들과 그들 간의 관계로 모델링함으로써 이해하기 쉽고 확장성이 우수한 모델을 제공하며, 둘째 뛰어난 재사용 기법들을 지원하는 C++과 Java와 같은 객체 지향 언어들을 충실히 지원하기 때문이다.

객체지향 방법론들을 적용하여 소프트웨어의 개발을

수행하기 위해서는 다음과 같은 작업들이 요구된다[4].

1) 개발방법론(*methodology*)으로부터 개발방법(*method*)을 구성한다.

개발방법이란 소프트웨어를 개발하기 위해서 계획된 *연속된 작업들의 집합*이다. 개발방법론은 이러한 개발방법을 구성하기 위한 연구로서 사용될 표기법, 표기법을 적용하는 기법, 개략적인 개발절차 등을 포함하고 있다. 개발방법은 개발시에 제시되는 문제들을 효율적으로 해결하기 위하여 개발방법론을 기반으로 계획되어야 한다. 이 과정에서 개발될 시스템의 특성들이 충분히 반영되어야 한다.

2) 개발방법으로부터 *개발절차(process)*를 구성한다.

개발절차는 개발방법에서 정의하고 있는 작업들이 어떠한 조직(organization)에 의해서 어떠한 순서로 수행될 것이며 그들 간의 어떠한 상호 작용이 있는가를 제시한다. 이 과정에서 수행할 조직의 구성과 개발될 시스템의 특성이 반영되어야 한다.

소프트웨어의 성공적인 개발을 위해서는 효율적인 개발방법과 절차가 필요하다. 각각의 응용 분야의 특성에 따라서 개발방법과 절차의 효율적인 구성이 다르기 때문에 개발하고자 하는 소프트웨어의 응용 분야에 맞추어 이들이 구성되어야 한다. 그러나 현재 제시되고 있는 객체지향 방법론들에서는 개발방법과 개발절차를 구성하는 과정에 대한 체계적인 지원이 없기 때문에 개발자들이 갖고 있는 객체지향 방법론에 대한 이해와 응용 분야에 대한 경험에 의존하여 개발방법과 절차가 구성되고 있다. 경험을 갖춘 개발자를 훈련시키기 위해서는 많은 시간과 비용이 소모되므로, 이러한 상황은 객체지향 방법론의 확산을 저해하는 이유가 되고 있다.

본 연구는 이러한 상황을 개선하기 위해서 *변용가능한 객체지향 방법론(customizable object-oriented methodology)*을 제안하고 이로부터 개발방법을 구성하는 체계적인 방안을 고안함으로써 작업 1을 명시적으로 지원하고자 한다. 이를 위하여 변용가능한 객체지향 방법론에서는 개발방법의 구성 문제를 *계획구성(planning)*[5]의 문제로 변환한다; 계획구성이란 주어진 목표를 획득하기 위해서 가능한 작업들의 순서를 계획하는 것을 말한다. 작업들은 작업을 수행하기 위한 조건과 수행된 다음의 결과로 구성된다. 이러한 작업들에 대해서 부과된 제약사항들을 위반하지 않으며 최종 결과가 목표가 만족될수있도록 진행될 순서를 얻고자 하는 것이다. 본 연구에서는 계획구성 문제의 목표로서 개발하고자 하는 소프트웨어의 아키텍처가 사용된다. 이 아키텍처는 개발될 소프트웨어의 특성을 응용 분야의 용어를 사용하여 기술한다. 계획구성의 작업들은 개발 과정에서 수행되어야 할 작업들을 계획구성에 사용될 수 있도록 변용가능한 형태로 기술한다. 이렇한 작업들은 소프트웨어의 특성에 상관없이 적용될 수 있는 일반적인 작업뿐 만 아니라 특정한 특성에 효율적으로 대처하기 위한 유용한 전략들도 포함한다. 아키텍처가 주어지면 이를 통하여 개발작업들을 구체화하고 그들 간의 순서를 계획함으로써 기술된 시스템의 특성을 고려하는 개발방법을 구성한다. 이처럼 계획구성과정에서 적용될 수 있는 개발작업들의 집합을 본 연구에서는 *메타 개발방법(meta method)*라 한다. 변용가능한 객체지향 방법론을 적용할 경우 개발자는 개발하고자 하는 소프트웨어의 아키텍처를 기술하고 이를 입력으로 사용함으로써 소프트웨어의 특성을 효율적으로 반영하는 개발방법을 얻을 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구의 동기를 구체적으로 설명하고 배경 지식인 개발절차 모델링과 아키텍처, 계획구성 문제에 대해서 간략히 정리한다. 3장에서는 본 연구의 주된 결과인 변용가능한 객체지향 방법론이 제시된다. 즉 구성요소인 메타 개발방법과 아키텍처 스타일이 설명되고 개발방법의 구성 방법이 제시된다. 4장에서는 변용가능한 객체지향 방법론의 적용 예제가 제시된다. 마지막으로 5장에서 결론 및 향후 연구의 방향을 제시함으로써 끝을 맺는다.

2. 연구 배경

크고 복잡한 소프트웨어들을 예정된 시간 안에 제한된 예산을 사용하여 개발하기 위해서는 명확하며 반복가능한 개발방법들이 필수적이다. 이러한 개발방법들은 응용 분야의 경험 및 유용한 전략들을 일관성있게 통합하여 수행되어야 할 작업들의 순서를 정의함으로써 개발과정의 예측성을 향상시키고 개발시간 단축과 품질의 향상을 제공한다.

그러나 개발방법이 응용분야에 대한 경험과 유용한 전략들을 통합하기 위해서는 다음의 두 문제들을 해결하여야 한다. 첫째, 구체성과 일반성 사이에서 균형을 유지하는 것이다. 개발방법은 효율적인 개발을 위해서는 응용 분야의 특성에 맞는 개발 전략을 제시하는 것이 바람직하다. 반면에 개발방법을 구성하는 노력을 절감하기 위해서는 하나의 개발방법으로 다양한 응용 분야의 개발을 지원할 수 있어야 한다. 예를 들어 분산 소프트웨어를 개발하는 경우를 보면, 객체지향 미들웨어인 CORBA[6]를 사용하여 개발할 경우 [7]에 제시된 패턴들을 통합하여 개발방법을 구성할 수 있을 것이다. 이

경우 개발방법은 CORBA에 관련된 경험 및 유용한 전략을 활용함으로써 개발과정에서 수행되어야 할 작업들을 구체적으로 제시하고 발생가능한 위험들을 명시적으로 예측하여 관리할 수 있다. 반면에 개발방법의 일반성은 저하된다. 즉 CORBA를 사용하지 않고 RPC와 같은 다른 기법을 사용할 경우에는 응용 분야의 특성이 변경되기 때문에 구성된 개발방법은 그대로 적용될 수 없다. 이 경우 개발방법을 수정하는 노력이 요구된다. 이처럼 하나의 개발방법을 통하여 구체적인 정보를 제공하면서 일반적으로 적용할 수 있도록 균형을 유지하는 일은 어려운 것이다. 가장 이상적인 경우는 개발하고자 하는 응용분야의 모든 특성을 고려하여 이에 적합한 경험과 유용한 전략들을 통합하고 개발방법을 매번 새로이 구성하여 적용하는 것이다. 즉, 개발방법을 구성하기 위하여 다양한 경험들과 전략들을 검토하여 포함되어야 할 것들을 결정하고 이들을 적절한 순서로 계획하게 된다. 그러나, 이 경우에는 개발 시마다 개발방법을 구성하는 노력이 요구되며 구성된 개발방법의 품질은 개발방법 구성자의 경험과 응용분야에 대한 이해에 의해서 좌우된다.

둘째, 응용분야의 경험과 유용한 전략을 명시적으로 기록하여 활용하는 방안을 제공하는 것이다. 현재 응용분야의 지식을 표현하기 위한 방안으로는 거시적 시각을 제공하는 아키텍쳐 스타일[1-4, 8]과 미시적 시각을 제공하는 패턴[9-11]이 대표적이라 할 수 있다. 아키텍쳐 스타일은 응용분야에서 반복되는 시스템의 전반적인 구성을 나타냄으로 거시적이라 불리운다. 반면에 "Design Patterns: Elements of Resuable Object-Oriented Software"[11]로 부터 시작된 패턴 연구는 빈번히 반복되는 문제와 해결책의 쌍을 가리키는 것으로, 특정한 문맥에서 언급되기 때문에 미시적이라 불리운다.

본 연구에서는 이러한 두가지 주요한 문제들을 해결하기 위해서 아키텍쳐를 토대로 객체지향 개발방법을 체계적으로 구성하는 방안을 제안한다. 즉 아키텍쳐를 통하여 개발하고자 하는 소프트웨어의 제약사항들과 특성들이 기술되면, 이를 반영하여 다양한 경험들과 유용한 전략들을 선택하여 일관성있게 통합함으로써 응용분야에 적합한 새로운 개발방법을 구성토록 하는 것이다. 이로서 개발하고자 하는 소프트웨어에 적합한 개발방법을 적은 노력으로 얻을 수 있다.

우선 기존의 개발절차 모델링에 대한 연구들을 간략히 정리하고 아키텍쳐와 계획구성 문제에 대해서 설명한다.

2.1 개발절차 모델링

개발절차 모델링(process modeling)은 소프트웨어 개발과정을 지원하기 위해서 개발절차들을 명시적으로 기술하고 분석을 수행하는 기술이다[12]. 현재 개발절차 모델링은 개발절차 향상(process improvement)과 연관되어 다양한 기법에 기반한 많은 도구들이 제안되고 있다. 이러한 도구들은 주로 정형적 기법에 기반하여 개발절차를 정확히 기술하고 기술된 모델에 대해서 요구되는 시간이나 자원들을 산출하는 것과 같은 분석들의 지원에 촛점을 맞추고 있다. 예를 들어 SoftPM과 같은 도구들은 MAM-net이라고 불리우는 페트리 넷에 기반한 정형적 기법을 통하여 개발절차의 기술과 분석을 제공한다[13].

이러한 도구들은 관리자들로 하여금 정확한 기술과 분석을 제공하지만 개발절차 모델링하는 과정에 대한 지원은 미약하다. 반면에 금지 모델 (proscriptive model)에 기반한 방법들은 각 개발작업들의 제약사항을 기술하고 기술된 제약사항들이 만족될 수 있도록 개발절차를 구성한다[12]. 이러한 금지 모델 방식은 개발절차 모델링 과정에 대한 지원을 제공함으로써 개발절차 모델링 도구들의 단점을 보완할 수 있을 것이다. 이러한 방식의 대표적인 예로서 GRAPPLE가 있다[14]. GRAPPLE은 본 연구에서와 같이 개발작업들을 계획구성의 문제로 기술한 첫번째 연구로서 이를 통해서 개발작업들 간의 논리적 상관관계를 자연스럽게 기술하고 이들을 토대로 개발절차를 구성할 수 있도록 했다. [14]에서는 GRAPPLE을 적용한 경험을 토대로 각 산출물들의 내부 구조, 현재 속성 그리고 외부 의존성을 명확히 파악하는 것이 중요함을 보였다.

2.2 아키텍쳐

아키텍쳐란 시스템에 대한 전반적인 구성을 나타내는 최상위 모델이며, *아키텍쳐 스타일(architectural style)*이란 특정 응용 분야에 대해서 널리 적용되는 아키텍쳐의 일반적인 구성을 일컫는다[3]. 본 연구에서는 개발방법을 구성하기 위한 입력으로서 아키텍쳐를 사용한다. 아키텍쳐를 적용함으로써 다음과 같은 잇점을 얻을 수 있다.

- *아키텍쳐 중심(architecture-centered)*[2] 접근방안을 제공한다. 이는 개발과정에서의 일관성을 유지함으로써 개발이 성공적으로 수행되기 위한 필수적인 요소로 인식되고 있다.
- 특정 응용 분야의 경험을 아키텍쳐의 형태로 기술하고 사용함으로써 경험을 축적하고 재사용할 수 있다.

아키텍쳐의 구성을 포함한 자세한 사항들은 3.2에서

설명된다.

2.3 계획구성 문제

본 연구에서는 개발방법을 구성하는 과정을 계획구성 문제의 한 경우로 변환한다. 이로서 기존의 계획구성에 관한 연구들과 도구를 적용할 수 있다.

본 연구에서는 계획구성에 대한 접근방법의 하나인 *예측 후 회피 접근방법(anticipate and avoid approach)*[15]을 따른다. 예측 후 회피 접근방법은 1) 문제를 해결하기 위한 목표들을 인식하고 이를 위한 작업들을 계획하며, 이 과정에서 2) 작업을 수행하는 과정에서 발생할 수 있는 위험들을 예측하여 이들을 회피하기 위한 작업들을 추가한다.

이러한 위험 예측 및 회피 접근방법은 소프트웨어 개발의 특성을 효과적으로 반영한다. 소프트웨어를 개발하는 과정은 많은 위험들을 가지고 있다. 따라서 이러한 위험들을 인식하여 회피하는 것은 소프트웨어 개발의 성공적인 수행을 위한 필수적인 작업이다. 이처럼 개발을 수행하는 과정에서 위험 인식 및 회피를 중시하는 개발방법을 *위험 주도(risk-driven)* 개발방법이라 한다.

3. 변형가능한 객체지향 방법론

변용가능한 객체지향 방법론의 목적은 다양한 응용분야에서의 경험을 체계적으로 축적할 수 있는 토대로 제공하고 이를 기반으로 개발방법을 구성하는 체계적인 방안을 제공함에 있다. 이를 위하여 변용가능한 객체지향 방법론에서는 개발방법의 바탕이 되는 *메타 개발방법*을 구성하고 이를 적용하여 개발방법의 구성을 계획구성으로 변환하는 기법을 제공한다. 메타 개발방법은 다양한 목표를 달성하기 위한 개발작업들을 포괄하고 이들 간에 구조를 부여한다. 계획구성은 이러한 개발작업들을 적절한 순서로 선택하고 구체화함으로써 아키텍쳐에 기술된 특성을 만족시키기 위한 개발방법을 구성한다. 그림 1은 메타개발 방법을 중심으로 개발방법이 구성되는 과정을 개략적으로 나타낸다.

개발방법의 구성에서 예측 후 회피 접근방법을 지원하기 위해서 아키텍쳐의 구성요소들은 수행되어야 할 목표와 함께 발생할 수 있는 위험들을 표현할 수 있다. 개발작업들 또한 목표에 대한 해결책을 포함할 뿐만 아니라, 발생할 수 있는 위험에 대한 해결책을 제시할 수 있어야 한다.

3.1 메타 개발방법

메타 개발방법은 다양한 목표를 달성하기 위한 개발작업들을 포괄하고 이들 간의 구조를 부여한다. 개발작업들은 개발방법의 구성에서 선택적으로 사용되기 때문에 개발방법의 일반성를 해치지 않으면서 구체적인 개발작업들을 포괄 할 수 있다. 개발작업의 주요한 구성요소는 다음과 같다.

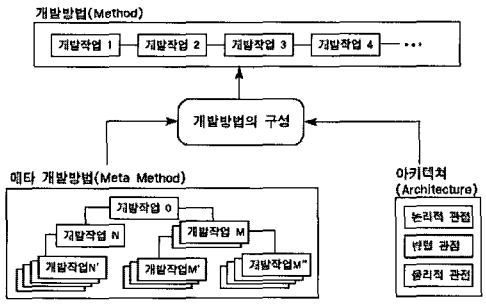

그림 1 변용가능한 객체지향 방법론을 통한 개발방법의 구성

- 개발작업이 적용가능한 문맥;
- 수행되어야 할 목표;
- 목표를 수행하기 위해서 적용될 수 있는 *기술(techniques)*들;
- 기술을 적용하고 산출물을 표현하기 위한 *표기법(notation)*;
- 수행되기위해 요구되는 산출물 및 생성되는 산출물

제시된 구성요소들은 정형적 구성요소들과 비정형적 구성요소들로 구분될 수 있다. 정형적 구성요소들은 계획구성을 위해서 명시적으로 사용되는 요소들로서 1) 적용가능한 문맥, 2) 요구되는 산출물 및 상태, 그리고 3) 생성되는 산출물 및 상태의 변화이다. 비정형적 구성요소들은 자연어 및 UML표기법[2, 16]의 형태로 제공되는 것으로 계획된 개발작업에 대한 구체적인 설명을 개발자에게 제공한다.

메타 개발방법에서 개발작업은 인자(parameter)를 갖을 수 있으며 이 경우에 계획구성 과정에서 구체적인 값이 결정된다. 예를 들어 개발작업 '서브시스템 x에 대한 클래스 모델 구축'이 계획될 경우에 계획구성과정에서 네트웍 서브시스템과 같은 특정 서브시스템으로 구체화된다. 이 경우에 개발작업은 '서브시스템 *네트웍 서브시스템*에 대한 클래스 모델 구축' 작업으로 구체화된다.

3.1.1 메타 개발방법의 구성

메타 개발방법은 계층적 구조를 갖는다. 이는 객체지향 개발방법들의 일반적인 구조를 반영하고 개발방법의 크기를 적절한 크기로 유지하기 위함이다. 상위의 개발작업들은 하위의 개발작업에 대한 문맥을 제공한다. 최

상위 구조를 구성하는 작업들은 '단계'로 불리운다. 다음은 제시되고 있는 메타 개발방법의 단계들에 대한 간략한 설명이다.

요구사항 정의 사용자의 요구사항을 도출하고 정리하여, 개발하고자 하는 시스템이 만족하여야 할 요구 사항을 정의한다.

분석 요구사항을 분석하여 시스템에 대한 논리적인 모델을 제시함으로써, 시스템이 무엇을 제공하는지 '기능'을 정의한다.

설계 시스템의 논리적 모델에 대해서 효율적인 해결책을 고안함으로써, 시스템이 정의된 기능을 어떻게 제공할 것이 결정한다. 또한 다양한 비기능적 요구사항들이 고려된다. 이 작업은 크게 다음과 같은 두 단계로 구성된다.

1) 시스템 설계: 상위 수준의 전략이 수립된다. 시스템 설계 단계에서 결정된 사항들은 이후 시스템 전반에 대해서 폭넓은 영향을 끼치며, 일관되게 유지된다.
2) 객체 설계: 구현의 바탕을 제공하기 위해서 객체들에 대한 세부 사항들이 결정된다.

논리적으로는 '시스템 설계'와 '객체 설계'가 설계 단계를 구성하고 있으나 그 성격의 차이와 규모 때문에 실제적으로는 별도의 단계로 구분된다. 따라서 메타 개발방법은 '요구사항 정의', '분석', '시스템 설계' 그리고 '객체 설계'의 4단계로 구성된다. 이러한 구조는 객체지향 개발방법론에 제시하고 있는 일반적인 절차를 반영한다.

3.2 아키텍쳐

아키텍쳐는 구성될 시스템에 대한 상위 수준의 구성요소들과 그들의 특성 및 관계를 정의하고 있다. 효율적인 개발을 위해서 개발방법은 아키텍쳐에서 기술되고 있는 사항들로 인해서 야기되는 문제들을 해결해야할 개발작업들을 제시해야 하고, 이를 위한 기술(technique)과 표기법을 정의해야 한다.

개발방법은 아키텍쳐를 기술하고 관리함으로써 *개념적 일관성(conceptual integrity)*을 유지한다[1]. 이를 위해서 *전략적 결정(strategic decision)*으로 분류되는 결정들을 관리하여야 된다. 전략적 결정이란 한 문제에 대해서 내려진 결정들이 시스템 전반에 걸쳐 영향을 미치는 경우를 일컫는다. 이러한 결정사항들은 시스템 전체에 대해서 일관되게 결정되고 유지되어야 하며, 개발의 초기에 이루어 지는 것이 바람직하다. 전략적 결정들은 경험을 통하여 인식을 되어 아키텍쳐를 통하여 기술되게 된다. 아키텍쳐는 어떠한 전략적 결정사항들이 있는 가를 기술함으로써, 전략적 결정들이 다른 결정들과 분리하여 개발 과정에 걸쳐서 일관되게 유지될 수 있도록 한다. 개념적 일관성을 유지하는 것은 성공적인 개발을 위한 필수적 요소이기 때문에 아키텍쳐를 특징짓는 문제들을 중심으로 개발작업들을 구성하는 것은 기존의 경험을 체계화하는 것은 효율적인 전략이다. 이처럼 객체지향 개발방법론에서 아키텍쳐는 각 산출물들의 내부 구조, 현재 속성 그리고 외부 의존성을 효과적으로 나타낸다. 이러한 특성들은 [14]에서 제시된 바와 같이 효율적인 제약사항을 기술하기 위한 가장 중요한 것이다.

전략적 결정에 반해서 *전술적 결정(tactical decision)*으로 분류되는 결정 사항들은 구성요소 내에 국한되는 영향을 갖는다. 이러한 전략적 결정사항들은 전략적 결정 사항들과 상충될 수 있기 때문에 개념적 일관성을 위협하는 '위험'들로 여겨질 수 있다.

3.2.1 아키텍쳐의 구성

아키텍쳐는 각기 다른 관점을 나타내는 세가지 모델로 구성된다. 이는 Kruchten[8]이 제안한 다관점 기술을 따라 구성된 것이다. 다관점 기술은 개발과정이 진행됨에 따라서 고려하게 되는 관점들을 나누어 전개함으로써 정보들을 체계적으로 관리할 수 있도록 한다. 본 연구에서는 기술되는 세 관점은 다음과 같다.

논리적 관점(logical view) - 아키텍쳐의 패키지(package)모델을 기술한다. 패키지 모델은 아키텍쳐를 구성하는 상위 수준의 구성요소들에 대한 정적 구조를 논리적인 측면에서 제시한다. 기술된 논리적 관점은 '분석'단계의 논리적 모델 구성에 적용된다.

병렬 관점(concurrency view) - 아키텍쳐 구성요소들의 병렬성과 동기화 관점을 기술한다. 병렬 관점에서는 논리적 관점에서 제시된 정적 구조의 구성요소들이 수행시간에 갖게 될 제어 구조와 이들 간의 관계를 정의한다. 또한, 구성 요소의 *영구성(persistency)*에 대해서 기술한다. 기술된 병렬 관점은 '시스템 설계' 단계의 전술적 결정에 적용된다.

물리적 관점(physical view) - 아키텍쳐 구성요소들의 물리적인 배치를 기술한다. 즉, 시스템을 구성하는 노드들의 관계와 노드에 배치되는 구성요소들에 대해서 기술함으로써 병렬 관점에서 정의된 제어 구조적 측면을 물리적으로 구체화 한다. 기술된 물리적 관점 또한 병렬 관점과 동일하게 '시스템 설계' 단계의 전술적 결정에 적용된다.

아키텍쳐의 각 모델은 의존관계나 포함관계와 같은

구조적 특성과 구성요소들에 대한 병렬성과 영구성 같은 전략적 결정사항에 관계된 특성들을 계획구성의 제약사항으로 기술한다. 아키텍처의 구성요소 또한 개발방법과 같이 정형적 요소와 비정형적 요소로 구분된다. 비정형적 요소는 개발자의 의도를 기술하기 위해서 사용되며, 정형적 요소는 개발방법을 구성하는 과정에서 목표를 정의하기 위해서 사용된다. 아키텍처의 정형적 구성요소의 예는 4. 적용 예제에서 제시된다.

3.3 개발방법의 구성

본 연구에서는 개발방법 구성의 문제를 계획구성의 문제로 변환하고 이를 기존의 계획구성 기법을 적용함으로써 해결한다.

계획구성을 위해서는 풀고자하는 문제와 작업을 기술할 수 있어야 하며 이에 대해서 작업들의 순서를 계획하는 *계획구성자(planner)*가 있어야 한다. 본 연구에서 계획구성자는 *규칙 기반 접근방법(rule-based approach)*[17]을 따른다. 즉, 메타 개발방법의 개발작업들은 계획을 구성하기 위한 일련의 규칙들로 사용되며, 계획구성자는 이들을 아키텍처에 의해서 주어진 제약조건과 목표에 따라 해석하게 된다. 이러한 규칙 기반 접근방법은 규칙을 추가하거나 수정함이 용이하며, 이러한 장점은 개발방법을 손쉽게 확장하는 것을 가능하게 한다.

규칙 기반 접근방법을 적용하기 위해서 계획의 진행상태를 나타내기 위한 대수적 구조(algebraic structures)가 정의되며, 문제와 계획은 정의된 대수적 구조에 대한 식(formulae)로서 기술된다. 계획구성의 문제는 다음과 같이 구성된다.

- 획득하고자하는 목표와 초기 상태

얻고자하는 목표는 요구되는 상태를 의미하는 식들의 집합으로 기술되며, 초기 상태 또한 이를 나타내는 식들의 집합으로 주어진다. 예를 들어 식 *unresolved (CORBA/CLIENT, client)*과

- 수행될 수 있는 *작업(action)*들의 집합

각 작업들은 작업이 수행되기 위한 사전조건(precondtion)과 작업이 수행한 후의 상태의 변화로 구성된다. 만약 작업의 사전조건이 만족되어 작업이 수행되면, 현재의 상태를 나타내는 식들을 변경함으로써 상태의 변화를 표현한다. 상태의 변화로는 식들의 추가, 삭제 및 변경이 있을 수 있다. 이러한 작업은 정의된 대수적 구조에 대한 연산자로 이해될 수 있다.

계획구성 과정은 계획구성을 효율적으로 진행하기 위해서 목표로부터 시작하는 후방진행(backward)을 따른다. 즉, 목표로부터 시작하여 다음의 두 동작 중 하나를 선택하여 수행하는 것을 반복함으로써 초기 상태로 진행하게 된다.

- 만족되지 않은 선행조건을 만족시키기 위한 작업을 선택한다. 만족되지 않은 선행조건은 *개방(open)*되어있다고 한다. 이를 만족시키기 위한 작업이 선택되면 이를 기록하기 위해서 선택된 작업과 이를 통해서 선행조건이 만족된 작업 사이에 *인과 연결(causal link)*가 추가된다.
- 인과 연결을 간섭하는 작업이 존재할 경우 이를 해소하기 위한 작업을 선택한다. 한 작업에 부여되는 *제약사항(constraint)*로 인해서 다른 작업 간의 인과 연결이 간섭받아 선행조건의 충족이 무효화되는 경우가 있다. 이러한 작업을 *위협(threat)*이라 한다. 만약 위협이 발견되면 작업들의 순서를 재조정하거나 새로운 하위목표를 추가하는 작업 등을 취함으로써 위협을 해소한다.

이렇게 얻어진 작업들의 순서열을 초기 상태에 차례로 적용하면 원하는 상태에 도달하게 된다.

3.3.1 계획구성 문제로의 변환

본 연구의 핵심은 객체지향 개발방법의 구성을 계획구성으로 변환하여 기존의 계획구성 기법을 적용함으로써 해결함에 있다.

다음과 같은 대응에 의해서 개발방법의 구성 문제가 계획구성으로 변환된다.

- 계획의 초기 상태는 개발하고자 하는 소프트웨어에 대한 '사용자의 요구'로서 아키텍처에 의해서 기술된다. 계획의 목적은 주어진 '사용자의 요구'를 만족하는 시스템에 대한 '설계'를 얻는 것이다.
- 메타 개발방법을 구성하고 있는 개발작업들은 계획구성 문제의 작업들에 대응한다.
- 아키텍처는 또한 개발작업들이 만족시켜야 할 제약사항들을 기술하고 있다. 아키텍처를 통해서 부과되는 제약사항들을 해소하기 위해서 다양한 개발작업들이 선택되고 구체화된다. 반면에 필요없는 개발작업들은 계획구성시 제외된다.

이러한 대응을 적용하기 위해서는 먼저 메타 개발방법과 아키텍처 간의 관계를 관찰하여야 한다. 메타 개발방법을 구성하는 4단계 즉, 요구사항 정의, 분석, 시스템 설계 및 객체 설계를 관찰하면 요구사항 정의는 아키텍처에 구애됨 없이 진행됨을 알 수 있다. 이는 요구사항 정의는 사용자 관점으로 진행되는 작업으로서 어떠한 사항들을 결정하기보다는 정의하는 과정이기 때문이다. 따라서 입력으로 주어지는 아키텍처에 의해서 재구성되

는 개발작업들은 분석 단계, 시스템 설계 단계, 그리고 객체 설계 단계의 작업이다.

앞서 언급되었듯이 메타 개발방법을 구성하는 개발작업들은 계층구조를 이루고 있다. 계획구성자는 주어진 제약사항을 만족시키며 목표를 획득하는 작업들을 구성하기 위해서 계층구조를 따라서 작업들의 순서를 계획한다. 상위의 개발작업은 또 하나의 계획구성 문제로서 여겨지며 이를 위해서 하위 개발작업들의 계획구성이 이루어진다. 이 과정에서 각 상위 개발작업들은 하위 개발작업들의 문맥을 제공하고 각 하위 개발작업들은 문맥을 토대로 구체화된다.

3.3.2 예측 후 회피 접근방법의 적용

앞서 언급되었듯이 본 연구에서는 계획구성 과정에서 예측 후 회피 접근방법을 따른다. 따라서 획득되어야 할 목표들 뿐 아니라 실패를 유발할 위험에 대해서도 명시적으로 기술되어 예측할 수 있어야 한다. 이를 위하여 *예측자(anticipator)*가 구성된다. 예측자는 사용자가 제시한 아키텍쳐를 토대로 실패를 유발할 위험들이 존재하는지 예측하여 이들을 회피할 수 있도록 목표들을 정련한다. 이렇게 정련된 목표들은 계획구성자의 입력으로 사용되어 개발방법을 구성한다. 구체적으로, 예측자는 위험을 예측하기 위해서 제시된 아키텍쳐에서 드러난 측면들을 관찰함으로써 *실패 예보자(failure predictors)*를 발견하고 이를 해소하기위한 하위목표들을 설정한다. 실패 예보자란 상황에 따라 실패의 원인이 될 수 있는 측면들을 나타내는 기술(description)이다. 예를 들어 아키텍쳐의 한 구성요소가 '병렬성'을 지닐 경우 '병렬성'으로 인해서 교착상태(deadlock)가 발생할 위험이 있음이 경험적으로 알려져 있다. 이 경우에 '병렬성'이 실패 예보자이며 이를 통해서 발생할 수 있는 실패는 교착상태가 된다. 이때 예측자는 '병렬성'에 대해서 교착상태를 발견하고 대처하는 하위목표들을 추가함으로써 발생할 수 있는 실패를 회피한다. 이로서 개발방법에는 '병렬성'을 고려하는 개발작업이 포함되게 된다. 이러한 측면들을 기술할 수 있는 다양한 어휘가 제공될 수록 더욱 정확한 예측이 가능하다. 그림2는 예측 후 회피 접근방법에 대한 개략도이다.

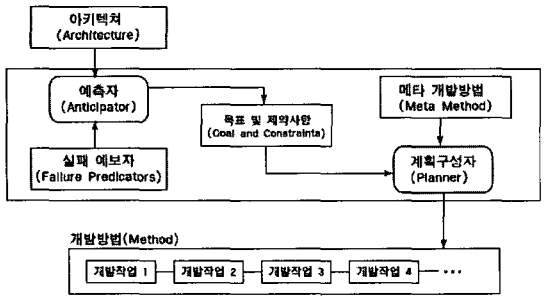

그림 2 예측 후 회피 접근 방법

3.3.3 지원 도구

제시된 방안에 따라서 개발방법을 구성하기 위해서 본 연구에서는 계획문제를 기술하기 위한 기반 언어로서 action description language(ADL)[18]을 사용한다. ADL은 계획구성에서 전통적으로 사용되던 STRIPS[17]보다 높은 표현력을 제공하지만 1차 논리(first-order logic)보다는 낮은 표현력을 갖는다. ADL은 이를 지원하거나 유사한 언어를 제공하는 다양한 계획구성자들이 연구되고 구현되어 이들을 활용할 수 있는 장점을 제공한다. 이러한 계획구성자로는 Pedestal, ucpop[19, 20, 21] 및 prodigy[22]를 들 수 있다.

현재 ADL의 부분집합을 지원하는 계획구성자인 ucpop를 기반으로 개발방법 구성을 위한 지원 도구의 원형이 개발되었다. 지원 도구에서 ucpop는 ADL로 기술된 작업들의 순서를 하위목표에 대해서 계획을 구성한다. 지원 도구의 계획구성자는 메타 개발방법의 계층구조를 따라서 목표를 분할하여 하위목표들을 설정하고 ucpop를 사용하여 각각에 대한 계획을 구성하고자 시도한다. 앞서 언급되었듯이 메타 개발방법을 구성하는 개발작업들은 계획구성 문제의 작업들의 집합에 대응하여 ADL로 기술되게 된다.

```
(define (operator construct-class-model)
   :parameters (?cls-tbl ?act-dgm ?cls-dgm)
   :precondition (and (class-table ?cls-tbl)
                      (activity-diagram ?act-dgm)
                      (make-class-diagram ?cls-dgm))
   :effect (and (class-diagram ?cls-dgm)
                (derived-from ?cls-dgm ?cls-tbl)
                (derived-from ?cls-dgm ?act-dgm)))
```

그림 3 개발작업에 대한 ADL의 예

그림 3은 ADL로 기술된 개발작업의 예[1]로서 클래스 모델 구축 작업을 기술하고 있다. 이를 간략히 살펴보면 다음과 같다. operator는 개발작업의 이름을 정의한다. 즉, construct-class-model은 기술되고 있는 개발작업의 이름이 된다. 앞서 언급되었듯이 개발작업은 대수적

1) 이 ADL기술은 설명을 위해서 간략화되었다.

구조 위에서 연산자(operator)로 여겨진다. :parameter는 개발작업이 적용되는 인수들을 정의한다. 계획이 구성될 때 인수들에게 특정한 값들이 주어지게 된다. :precondition은 개발작업이 적용되기 위한 사전조건을 나타내며 :effect는 개발작업이 적용되었을 경우의 상태 변화를 기술한다. 이들을 종합해서 관찰하면 개발작업 construct-class-model은 클래스 테이블(class table)과 액티비티 다이어그램(activity diagram)을 받아들여 이로부터 클래스 다이어그램(class diagram)를 생성함을 알 수 있다.

4. 적용 예제

본 연구의 적용 예제로서 MODE(Multimedia Object-Oriented DEvelopment) 방법론을 변용가능한 객체지향 방법론으로 재구성하는 작업을 진행하였다. MODE 방법론은 멀티미디어 및 병렬처리에 속하는 응용분야의 소프트웨어 개발을 지원하기 위한 방법론으로서 멀티미디어 및 병렬처리에 대한 다양한 경험 및 유용한 전략들을 포함하고 있다. 이처럼 특정한 응용분야에 특화된 개발방법론을 적용하는 경우에는 개발절차들을 적절히 취사선택하여 개발방법을 구성하게 된다. 개발하고자 하는 시스템의 특성에 개발절차들이 적절치 않거나 필요없는 결과를 제공할 수도 있기 때문이다. 즉, 다중의 제어흐름이 없는 시스템을 개발하는 과정에서 '병렬성'의 분석과 설계에 대한 개발절차는 프로젝트 계획에 포함되지 않는 것이 바람직하다. MODE방법론을 변용가능한 객체지향 방법론으로 재구성함으로써 개발하고자 하는 시스템의 특성에 맞추어 개발절차들을 선택하고 일관된 순서로 구성하는 과정을 지원한다. 이러한 지원은 다양한 경우에 대해 특화된 개발절차들을 개발방법론에 쉽게 포함할 수 있도록 한다. 개발방법론의 일관성을 해치지 않으면서 자동적으로 개발절차들을 선택할 수 있기 때문이다. 다음은 각각 MODE방법론을 변용가능한 객체지향 방법론으로 재구성하는 작업과 이를 토대로 과제의 특성을 반영하는 개발방법을 구성하는 과정을 간략히 기술한다.

4.1 MODE방법론의 재구성

MODE방법론을 변용가능한 객체지향 방법론으로 재구성하기 위해서 다음과 같은 작업들이 수행되었다.

- 아키텍처에서 응용분야의 특성들을 특징짓기 위해 사용되는 '어휘'들을 정의한다.

정의된 어휘들은 아키텍처를 구성하는 단위인 팩키지에 대한 제약사항과 속성들을 기술한다. MODE방법론을 위한 어휘들의 예는 표 1과 같다.

표 1 제약 사항 및 속성 기술 어휘

어휘	설명
dependent(A, B)	팩키지 A가 팩키지 B에 의존함을 기술한다.
contain(A, B)	팩키지 A가 팩키지 B에 포함됨을 기술한다.
concurrent	팩키지가 '병렬성'을 가짐을 기술한다. 즉, 다중의 제어흐름을 제공함으로써 동시에 여러 작업을 수행할 수 있다.
persistent	팩키지가 '지속성'을 가짐을 기술한다. 즉, 객체들을 지속적인 매체에 기술하고 읽는 기능을 제공한다.
CORBA/client	팩키지가 CORBA를 통해서 클라이언트(client)의 역할을 수행함을 기술한다.
CORBA/server	팩키지가 CORBA를 통해서 서버(server)의 역할을 수행함을 기술한다.

표 2 메타 개발방법의 구성

요구사항 정의	요구사항을 정의하는 '유즈 케이스 모델'과 사용되는 단어들을 정의하기 위한 '데이타 사전'을 작성한다. '초기 유즈 케이스 모델 기술,' '유즈 케이스 모델 개선,' 그리고 '데이타 사전 준비'의 작업들로 구성된다.
분석	유즈 케이스 모델을 실제화(realization)하는 '액티비티 모델'을 구성하고 이를 지원하기 위한 '클래스 모델'을 구성한다. '액티비티 모델의 구성,' '클래스 모델의 구성,' 그리고 '액티비티 모델의 수정'의 작업들로 구성된다.
시스템 설계	시스템에 대한 전략적 결정들이 이루어지고 시스템의 행위를 나타내는 '순차 모델'과 '상태 모델'이 구성되고 이를 반영하기 위해서 '클래스 모델'이 개선된다. '병렬성 설계,' '지속성 설계,' '네트웍 인터페이스설계,' '시스템 행위 설계' 및 '검증'의 작업들로 구성된다.
객체 설계	지금까지 구성된 모델들을 기반으로 구현의 바탕을 제공하기 위해서 객체들에 대한 세부 사항들이 결정된다. '접근 경로 최적화,' '연관관계 설계,' '클래스 모델과 상태모델의 통합,' 그리고 '알고리즘 및 자료 구조 설계'의 작업들로 구성된다.

정의된 어휘들 중에서 dependent와 contain은 UML 표기법을 따라서 팩키지 간의 화살표와 팩키지 간의 포함관계로 각각 표현된다. 나머지 어휘들을 UML의 스테

레오 타입을 통해서 기술된다.

• 메타 개발방법을 구성하는 개발작업들을 정의한다.

개발작업들은 특정 목적을 획득하기 위해서 구성된 것으로 ADL표기법을 따라서 수행되기 위한 사전조건과 입력 산출물 그리고 출력 산출물을 통하여 특징지워진다. 이러한 개발작업들은 산출물에 대해서 다양한 제약사항을 부과하거나 해소함으로써 상태를 변경한다. MODE 방법론을 위한 메타 개발방법의 구성은 표 2와 같다.

4.2 개발방법의 구성

개발방법을 구성하는 과정은 메타 개발방법의 계층구조를 따라서 아키텍처로부터 주어진 제약사항들과 목표들을 계획 구성자를 사용하여 계획함으로써 진행된다. 다음은 분산 개인정보 시스템에 대한 아키텍처를 나타내는 UML 다이어그램으로 각각의 패키지는 시스템을 구성하는 서브시스템에 대응하며 각각의 서브시스템에 대한 속성과 제약사항들은 패키지 간의 관계와 스테레오 타입을 사용하여 기술되고 있다.

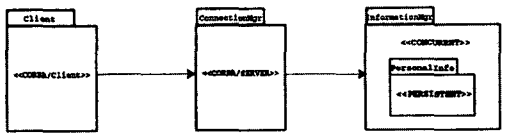

그림 4 분산 개인정보 시스템의 아키텍처

주어진 아키텍처는 개발하고자 하는 분산 개인정보 시스템이 4개의 서브시스템으로 구성되어 있음을 기술하고 각 서브시스템에 대한 제약사항들을 나타내고 있다. 예를 들어 서브시스템 *Client*는 스테레오 타입 《CORBA/CLIENT》을 통하여 CORBA를 통하여 서비스를 제공받음을 알 수 있다.

아키텍처는 다음과 같은 식들의 집합으로 변환되어 계획구성의 초기 상태(initial state)를 구성한다.

$$
\begin{aligned}
&\textit{Initial State}\\
&= \begin{cases}
\textit{dependent}(\textit{Client},\ \textit{ConnectionMgr})\\
\textit{dependent}(\textit{ConnectionMgr},\ \textit{InformatinMgr})\\
\textit{contain}(\textit{InformatinMgr},\ \textit{PersonalInfo})\\
\\
\textit{constrain}(\textit{PERSISTENT},\ \textit{PersonalInfo})\\
\textit{unresolved}(\textit{PERSISTENT},\ \textit{PersonalInfo})\\
\ldots\\
\textit{unresolved}(\textit{CORBA/Client},\ \textit{Client})
\end{cases}
\end{aligned}
$$

그림 5 계획구성의 초기 상태

초기 상태를 구성하는 식들은 아키텍처로부터 얻어진 소프트웨어의 구성요소들과 이들에 대한 구조적 속성, 제약사항들을 나타낸다. *dependent*와 *contain*은 서브시스템 간의 구조적 속성을 기술하며 제약사항들은 *constrain*를 통해서 기술된다. 예를 들어 *constrain(PERSISTENT, PersonalInfo)*은 서브시스템 *PersonalInfo*에 제약사항 *PERSISTENT*가 부과됨을 나타낸다. 또한 *unresolved*를 통해서 부과된 제약사항들이 아직 존재함을 나타낸다.

주어진 초기 상태에 대해서 계획구성자에게 요구되는 목표는 다음과 같다.

$$
\begin{aligned}
\textit{Goal} = \forall s | \textit{subsystem}(s) \cdot\\
(\textit{getAllModels}(s) \wedge \forall c | \textit{constrain}(c, s) \cdot \textit{resolved}(c, s))
\end{aligned}
$$

위의 목표는 모든 서브시스템 S에 대해서 모든 모델들을 생성해야 하며 부과된 모든 제약사항들을 해소해야 함을 기술한다. 이러한 목표는 모든 계획구성에서 동일하게 적용될 수 있다.

이처럼 초기 상태와 목표가 주어지면 이에 대해서 계획구성자는 개발방법들의 순서를 계획하게 된다. 이는 요구되는 개발작업들을 선택하여 구체화하고 이들 간의 수행 순서를 정의함으로써 이루어진다. 개발작업에 대한 구체화가 이루어지면, 개발방법의 계층구조에 따라서 개발작업을 구성하는 세부작업들과 세부작업을 구성하는 지침들에 대한 계획이 구성되게 된다.

계획구성의 과정을 통하여 적절치 않은 개발작업들은 프로젝트 계획에서 제거되게 된다. 예를 들어 서브시스템 *Client*의 경우 병렬성을 지원하지 않는다. 즉, 스테레오 타입 《CONCURRENT》에 속하지 않다. 이 경우 서브시스템 *Client*의 시스템 설계과정에 개발작업 '병렬성 설계'는 포함되지 않는다. 반면에 스테레오 타입 《CORBA/CLIENT》에 속하기 때문에 개발작업 '네트웍 인터페이스 설계'는 서브시스템 *Client*에 대해서 구체화되어 프로젝트 계획에 포함되게 된다.

다음은 개발작업의 예로서 제약사항 *CONCURRENT*를 해소하기 위한 작업 '병렬성 설계(DesignConcurrency)'(그림 6)를 나타낸다. 이 작업은 제약사항 *CONCURRENT*에 대해서 *unresolved*를 제거하고 *resolved*를 설정함으로써 제약사항을 해소됨을 나타낸다. 또한, 병렬성을 설계하는 과정에서 발생할 수 있는 위험인 교착 상태를 새로운 제약사항으로 부과하고 있다. 이후에 새로이 부과된 제약사항 (*DEADLOCKS*)을 해소하기 위해서 개발작업 '교착상태 검증 (DeadlocksChecking)'이 구체화되어 개발방법에 포함된다.

DesignConcurrency(s)

precondition : *subsystem(s) possible(s)*
Effects : ...
¬*unresolved(CONCURRENT, s)*
resolved(CONCURRENT, s)
unresolved(DEADLOCKS, s)

그림 6 개발 작업 병렬성 설계(DeisngConcurrency)

이 작업이 수행되기 위한 조건의 하나인 *possible*의 정의는 다음과 같다.

possible(subsystem) =
$\forall$ *other* | *depdend(subsystem, other)* ·
($\forall$ *c* | *constrain(c, other)* · *resolved(c, other)*)
$\forall$ *other* | *contain(subsystem, other)* ·
($\forall$ *c* | *constrain(c, other)* · *resolved(c, other)*)

술어 *possible*은 서브시스템에 대한 작업을 수행하기 위해서는 의존하고 있는 서브시스템이나 포함하고 있는 서브시스템에 대해서 모든 제약사항이 해소되어야 함을 나타낸다. 이 술어를 통하여 구조적 제약사항이 강제된다.

주어진 아키텍처에 의하면 서브시스템 *InformationMgr*은 병렬성을 갖는다. 따라서 계획구성자는 서브시스템 *InformationMgr*의 제약사항 *CONCURRENT*을 해소하기 위해서 개발작업 DesignConcurrency를 *InformationMgr*에 대해서 구체화한 DesignConcurrency(*InformationMgr*)를 계획에 포함하게 된다. 이렇게 구체화된 개발작업은 메타 개발방법의 계층구조를 따라서 세부작업들과 지침에 대한 계획구성을 위한 입력으로 사용될 뿐 아니라 교착상태의 예처럼 또 다른 개발작업에 대해서 영향을 미치게 된다.

5. 결론 및 향후 연구 방향

본 연구에서는 객체지향 개발방법론의 적용을 지원하기 위해서 변용가능한 객체지향 방법론을 제시하였다. 제시된 방법론에서는 개발하고자하는 소프트웨어의 구조와 특성을 기술하는 아키텍처를 입력으로 받아들여 이에 적합한 개발방법을 계획한다. 이를 위하여 개발방법 구성의 문제를 계획구성의 문제로 변환하는 기법을 제안하였다. 또한, 계획구성 과정에서 적용될 작업들을 기술하기 위해서 개발작업 기술방법인 메타 개발방법을 정의하였다. 이러한 변용가능한 객체지향 방법론을 적용함으로써 기존의 개발자의 이해와 직관에 의존하던 개발방법의 구성을 체계적으로 지원하고 구성을 위한 노력을 감소시킬 수 있을 것이다. 이러한 장점들은 응용분야의 특성에 적합한 개발방법의 구성 및 적용을 촉진하여 객체지향 개발방법론의 확산을 도울것이다. 이후 지원 도구를 완성함으로써 MODE 방법론을 변용가능한 방법론으로 구축, 이를 실 문제에 적용할 수 있을 것이다.

향후 연구 방향으로 개발방법 구성의 다음 단계인 개발절차 구성과의 연계가 연구되어야 할 것이다. 개발절차의 구성을 위해서는 개발을 수행할 조직과 개발될 시스템의 규모 및 특성이 고려되어야 함으로 보다 다양한 요소들이 계획구성을 위해서 요구된다. 예를 들어 개발작업을 위해서 요구되는 자원과 시간, 특히 개발자의 할당과 같은 상호 배타적인 자원의 스케쥴링이 수행되어야 한다.

참 고 문 헌

[1] G. Booch, *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings Publishing Company, Inc., 1994.

[2] H. Eriksson and M. Penker, *UML Toolkit*, Addison-Wesley, 1998.

[3] J. Rumbaugh, M. Blaha, W. Premerlani, R. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[4] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.

[5] P. H. Winston, "Chapter 15. Planning," *Artificial Intelligence*, Addison-Wesley, 1992.

[6] J. Siegel, *CORBA Fundamentals and Programming*, John Wiley & Sons, Inc., 1996.

[7] T. J. Mowbray and R. C. Malveau, *CORBA Design Patterns*, John Wiley & Sons, Inc., 1997.

[8] P. Kruchten, "The 4+1 View Model of Architecture," in *IEEE Softeware*, Nov., 1995.

[9] W. J. Brown, R. C. Malveau, and H. W. McCormick III, and T. J. Mowbray, *AntiPatterns: Refactoring Software, Architectures and Projects in Crisis*, John Wiley & Sons, Inc., 1998.

[10] M. Folwer, *Analysis Patterns: Reusable Object Models*, Addison Wesley, 1997.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[12] B. Curtis, M. I. Keller, and J. Over, "Process Modeling," In *Comm. of the ACM*, Sept., 1992.

[13] S. Y. Min, H. D. Lee, and D. H. Bae, "SoftPM: A Software Project Management System Reconciling Formalism with Easiness," *Information and Software Technology*, to be published in 1999.

[14] K. E. Huff and V. R. Lessor, "A plan-based intelligent assistant that supports the software development process," In *Proc. of the Third Software Engineering Symposium on Practical Software Development Environments*, 1989.
[15] C. K. Riesbeck and R. C. Schank, *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, Inc., 1989.
[16] Rational Corp, *UML Notation Guide, UML Notation Semantics and UML Sumnary*, http://www.rational.com, 1998
[17] R. Fikes and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," In *Artificial Intelligence*, vol. 2., 1971.
[18] E. P. D. Pednault, "ADL: Exploring the middle ground between strips and the situation calculus," In *Proceedings Knowledge Representation Conference*, 1989.
[19] D. McDermott, "Regression planning," In *International Journal of Intelligent Systems*, vol. 6, 1991.
[20] A. Barrett, K. Golden, S. Penberthy, and D. Weld. *UCPOP User's Manual(version 2.0)*, Technical Report 93-09-06, 1994.
[21] J. S. Penberthy and D. S. Weld, "UCPOP: A Sound, Complete, Partial Order Planner for ADL," In *proceedings of 3rd International Conference on Knowledge Representation and Reasoning*, Oct., 1992.
[22] J. G. Carbonell, C. A. Knoblock, and S. Minton, "Prodigy: An integrated architecture for planning and learning," In *Architectures for Intelligence*, Erlbaum, 1990.

김 형 호

1992년 ~ 1996년 서강대학교 전자계산학과(학사). 1996년 ~ 1998년 한국과학기술원 전산학과(석사). 1998년 ~ 현재 한국과학기술원 전산학과 박사과정 재학중. 관심분야는 객체지향 개발기법, 컴포넌트 기반 개발기법 및 정형 명세와 분석.

김 영 곤

정보과학회 논문지 : 소프트웨어 및 응용 제 27 권 제 1 호 참조

배 두 환

정보과학회 논문지 : 소프트웨어 및 응용 제 27 권 제 1 호 참조

김 민 경

1985년 ~ 1988년 경신고등학교 졸업. 1990년 ~ 1994년 동국대학교 전자계산학과 졸업(학사). 1994년 1996년 포항공과대학교 전자계산학과 졸업(석사). 1996년 ~ 현재 한국통신 연구개발본부 멀티미디어연구소 번호안내서비스연구실 전임연구원. 관심분야는 소프트웨어 재사용, 객체지향, CTI, 인터넷 응용기술.

유 병 규

1974년 ~ 1977년 성남고등학교 졸업. 1977년 ~ 1981년 서울대학교 산업공학과 졸업(학사). 1981년 ~ 1983년 한국과학기술원 산업공학과 졸업(석사). 1987년 ~ 1991년 한국과학기술원 산업공학과 졸업(박사). 1997년 ~ 현재 한국통신 연구개발본부 멀티미디어연구소 인터넷연구팀장. 1992년 ~ 1997년 한국통신 연구개발본부 멀티미디어연구소 하이텔연구실장. 1991년 ~ 1992년 한국통신 연구개발원 데이타베이스 연구실 전임연구원. 1987년 ~ 1991년 한국과학기술원 산업공학과 박사과정 교육파견. 1983년 ~ 1987년 한국통신 연구개발원 정보통신연구실 전임연구원. 관심분야는 인터넷 응용기술, 정보검색기술, VoIP, 시스템최적화.