한국산업정보학회논문지 제7권 제5호 (2002. 12)

소프트웨어 아키텍처 기술 언어의 요구 조건

(The Requirements of Software Architectural Description Language)

권 기 태*, 변 분 희**
(Ki-Tae Kwon, Boon-Hee Byun)

요 약 소프트웨어 시스템의 크기와 복잡도가 커짐에 따라서 전체적인 시스템 구조에 관한 디자인과 명세는 알고리즘의 선택과 데이터 구조의 계산보다 더욱 중요한 이슈로 나타나고 있다. 아키텍쳐 설계 단계에서는 시스템의 전체적인 제어 구조·통신·동기화·데이터 액세스를 위한 프로토콜 등을 표현하는 것을 포함한다. 본 논문에서는 소프트웨어 아키텍처를 표현하기 위해 현재 연구되고 있는 소프트웨어 아키텍처 기술 방식을 비교 분석한 후, 전통적인 아키텍쳐로부터 어떻게 소프트웨어 아키텍처를 기술할 것인지에 관한 방법을 제시하고, 이러한 과정을 통해 아키텍쳐 기술 언어(ADL)가 갖추어야 할 특성에 대하여 살펴본다. 결과적으로 우리는 기존의 접근이 왜 소프트웨어 아키텍쳐 기술에 불만족스러운지 보이고, 새로운 아키텍쳐 기술 언어의 필요성을 제시함과 동시에 소프트웨어 아키텍쳐 기술 언어가 갖추어야 할 요건을 제시하고자 한다.

Abstrat As the size and complexity of software systems increase, the design and specification of overall system structures become more significant issues than the choice of algorithms and data structures. Structural issues include the organization of a system as a composition of components; global control structures; protocols for communication, synchronization, and data access etc. In this paper we show why the existing software architectural description method is not satisfied, and suggest the necessity of the new architectural description language and the requirements of the software architectural description language.

1. 서 론

소프트웨어 시스템의 크기와 복잡도가 커짐에 따라서, 전체적인 시스템 구조에 관한 디자인과 명세는 알고리즘의 선택과 데이터 구조의 계산보다 더욱 중요한 이슈로 나타나고 있다. 아키텍쳐 설계 단계의 이슈는 시스템의 전체적인 제어 구조·통신·동기화·데이터 엑세스를 위한 프로토콜 등을 포함한다[3][4].

추상적으로 소프트웨어 아키텍처는 시스템 구성 요소 사이의 상호 작용, 구성을 설명하는 패턴, 이러한 패턴의 제약 조건 등을 기술하는 것을 포함한다. 특별한 경우의 시스템은 컴포넌트의 집합과 컴포넌트 사이의 상호작용과 관계하여 정의되기도 한다. 이러한 시스템은 좀 더 큰 시스템 디자인의 구성 요소로서 사용되어질 수 있다[5].

소프트웨어 아키텍처의 사용은 일반적으로 시스템 디자인을 표현하기 위해 사용하는 비형식적인 다이어그램과 관용어구들로 표현되어진다. 그러나, 이러한 다이어그램과 관용어구들로 표현하는 것은 애매모호한 면이 있다. 소프트웨어 아키텍처 표현에 있어 디자이너들의 공통된 직관과 과거의 경험을 통해 기술하는 방식은 일반적인 시스템 디자이너들에게 있어 정확한 개념, 툴, 즉각적인 문제 해결을 위한 시스템 구조의 표현과 선택을 위한 판단의 기준 등이 부족하므로 명확한 해답을 얻기에는 불가능하다. 본 논문은 명세적인 소프트웨어 아키텍처 표현의 문제에 기반하여 새로운 고수준 언어의 필요성을 인식하고, 전통적인 언어로부터 어떻게 소프트웨어 아키텍처 표현을 기술할 것인지에 관한 방법을 보인다. 이런 과정을 통해 아키텍처 기술 언어가 갖추어야 하는 특성에 대하여 살펴본다.

2. 소프트웨어 아키텍처

소프트웨어 아키텍처는 컴포넌트와 컴포넌트 사이의 상호작용에 있어, 구조적 의미적 차이를 명확하게 설명해 줌으로써 대형시스템을 정의하는데 이용될 수 있으

* 강릉대학교 컴퓨터공학과 교수
** 강릉대학교 컴퓨터공학과 박사과정

며, 바람직한 아키텍처는 구성 요소들이 또다른 시스템에 재사용될 수 있도록 독립적으로 정의할 수 있다. 이러한 소프트웨어 아키텍처의 주요 관심 분야는 아키텍처 스타일, 아키텍처 기술 언어, 아키텍처 지원 도구, 도메인 중심 소프트웨어 아키텍처 등의 네 분야로 정의할 수 있다.

2.1 아키텍처 스타일(Architectural Style)

아키텍처 스타일은 시스템의 전반적인 구조와 의미적 특징을 나타낸다. 그 특징을 살펴보면 파이프, 필터, 클라이언트, 서버, 파서, 데이터베이스 등과 같은 컴포넌트와 커넥터 타입을 나타내는 디자인 요소의 어휘와 디자인 규칙으로 나타낼 수 있다. 또한 시스템의 수행 상태를 분석할 수 있고, 디자인의 재사용성을 촉진시키고, 코드를 재사용할 수 있게 한다. 특징 아키텍처 스타일은 아키텍처 표현으로부터 직접적으로 프로그램 코드를 생성할 수도 있다[3][4][5][7].

2.2 아키텍처 기술 언어

형식적인 아키텍처 모델링을 가능하게 한 기법중의 하나로 ADL(Architecture description language)이 제시되었다. ADL을 사용함으로써 실무자들 사이의 의사 전달을 가능하게 하고, 지원 도구를 사용하여 많은 경우에 대해 분석할 수 있게 되었다. 대표적인 ADL로는 ACME, MetaH, Aesop, Rapide, AtTek, SADL, C2, UniCon, Darwin, Weaves, LILEANNA, Wright 등을 들 수 있다. ADL은 컴포넌트와 컴포넌트 사이의 상호작용을 표현하기 위한 도구, 표기법, 모델 등을 제공하며, 대규모 고수준의 디자인을 지원한다. 또한 사용자 정의 및 특정 응용을 위한 추상화를 지원하고, 쉽게 내용을 자동으로 코딩하는 것을 지원한다.

2.3 아키텍처 지원 도구

아키텍처 지원 도구로 Unicon과 Aesop을 들 수 있다. UniCon은 아키텍처를 표현하는 언어와 도구들의 집합으로 다양한 컴포넌트와 커넥터를 활용하여 아키텍처 디자인의 이해를 돕는다. Aesop은 아키텍처 디자인 환경을 구성하는 툴킷이다. 새로운 아키텍처 스타일을 정의할 수 있고, 아키텍처 스타일의 제약조건을 분명하게 할 수 있다. Aesop은 일반적으로 객체 지향 모델로 이 객체 지향 모델을 이용하여 아키텍처 스타일들의 특성 묘사, 그리고 아키텍처 스타일의 묘사로부터 개방형 아키텍처 디자인 환경을 생성할 수 있다[3][6][9].

2.4 도메인 중심 소프트웨어 아키텍처(DSSA)

특정 도메인 중심의 소프트웨어 아키텍처를 개발함으로써 소프트웨어 설계를 최적화할 수 있다. 일반적인 소프트웨어의 재사용은 너무 어려운 반면, 특정 도메인의 재사용은 유사한 특성을 지닌 특수한 어플리케이션 클래스에 초점을 맞춤으로써 재사용성을 실무에 적용할 수 있다.

DSSA는 특별한 형태의 작업을 위해 구체화되고, 도메인을 효과적으로 활용하기 위해 일반되고, 성공적인 어플리케이션을 만들기 위해 표준화된 구조로 구성한 소프트웨어 컴포넌트의 집합이다. DSSA는 도메인 모델, 참조 요구, ADL로 표현된 참조 아키텍처 등을 포함한다[10].

3. 소프트웨어 아키텍처 기술 방식

소프트웨어 아키텍처의 구조적, 의미적 특성을 기술해 줌으로써 실무자들 사이에 의사전달을 가능하게 하고, 지원 도구를 활용하는 방식이다.

3.1 MIL

MIL은 구현된 시스템에서 사용하는 모듈 사이의 관계를 커넥터 타입으로 간결하고 정확하게 검사 가능한 형태로 표현하기 위한 도구로 제안되었다. MIL은 <그림 1>의 시스템 구조를 <그림 2>와 같은 형태로 기술한다[6][8].

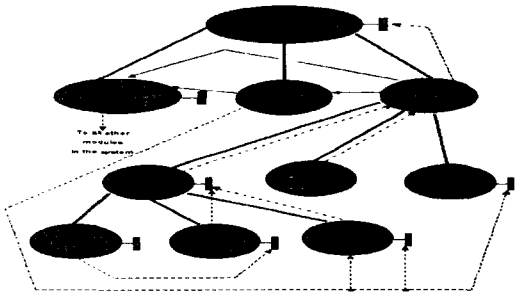

그림 1. 시스템 구조

```
system Input
  author 'Sharon Sickel'
  date 'July, 1974'
  provides Input_parser
  consists of
    root module
      originates        Input_parser
```

```
   uses derived     Parser, Post_processor
   uses nonderived  Language extensions
subsystem Scan
   must provide Scanner
subsystem Parse
   must provide Parser
   has access to Scan
subsystem Post
   must provide Post_processor
```

그림 2. 시스템 구조 기술

MIL은 소프트웨어 모듈간의 한가지 종류의 커넥터만을 지원하므로 컴포넌트 사이의 연결 재사용 패턴을 제공하지 않으므로 ADL로 활용되기에는 결점이 많다.

3.2 형식 명세 언어

특정한 시스템의 형식적인 기술은 디자인을 정확하게 문서화할 수 있고 그 시스템의 동작에 대해 추론할 수 있다. 이러한 아키텍처 스타일의 형식 기술은 그 관용법의 상이한 사용을 비교할 수 있으며, 특성을 부여할 수 있다. 소프트웨어 아키텍처에 관한 형식 이론은 아키텍처 구성의 본질을 명백히 하여 아키텍처 특성을 분석하고 기계적인 수단을 제공할 수 있다.

대표적인 소프트웨어 아키텍처의 기술을 위해 사용되는 형식 명세 언어로는 'Z 언어'가 있다. 'Z 언어'의 수학적 기초는 first-order 논리와 집합론이다. 표기법은 표준의 논리적인 연결 기호(∧,∨,⇒, 등)와 집합론의 연산 기호(∈,∪,∩, 등)를 표준의 의미론과 같이 사용한다. <그림 3>은 'Z 언어'를 이용한 소프트웨어 아키텍처의 기술 예이다.

```
- Filter -
filter_id : FILTER
in_ports, out_ports : P PORT
alphabets : PORT ⇸ P DATA
states : P FSTATE
start : FSTATE
transitions : (FSTATE × (Partial_Port_State))↔
(FSTATE × (Partial_port_State))
start∈states∧in_ports∩out_ports=∅∧dom
alphabets = in_ports∪out_ports
((c1, ps1),(c2, ps2))∈transitions⇒c1∈states∧
c2∈states∧ dom ps1=in_ports∧dom ps2 =
out_ports∧(∀p:in_ports · ran(ps1(p))⊆alphabets
(p))∧(∀p:out_ports · ran(ps2(p))⊆alphabets(p))


- Pipe -
source_filter, sink_filter : Filter
source_port, sink_port : PORT
alphabet : P DATA
source_port∈source_filter.out_ports∧sink_port
∈sink_filter.in_ports
source_filter.alphabets(source_port)=alphabet
sink_filter.alphabets(sink_port)=alphabet
- System -
filters : P Filter
pipes : P Pipe
∀ c1, c2 : filters · c1.filter_id = c2.filter_id⇔
c1=c2
∀ p : pipes · p.source_filter ∈ filters ∧
p.sink_filter ∈ filters
∀ f : filters; pt: PORT ¦ pt ∈ f.in_ports · #
{p:pipes ¦ f=p.sink_filter∧pt=p.sink_port} ≤1
∀ f : filters; pt: PORT ¦ pt ∈ f.out_ports · #
{p:pipes ¦ f=p.source_filter∧pt=p.source_port}
≤1
```

그림 3. 'Z 언어'로 표현한 Pipe & Filter 스타일

형식 명세 언어는 객체의 상태를 나타내기 어렵고, 객체들 간의 제어를 이해하기 어려우므로 일반사용자에게 친숙하지 못하다는 문제점을 여전히 가지고 있다.

3.3 프로그래밍 언어의 응용

프로그래밍 언어는 추상적인 모듈 사이의 상호작용을 정의하는 것에는 부적합하다. 따라서 전통적인 프로그래밍 언어의 디자인 한계에서 벗어나 확장을 통해 표현력을 넓혀야 한다. Implicit invocation 스타일을 기술하기 위해 Ada를 확장한 예는 <그림 4>와 같다[3][6].

Ada 문법을 활용한 응용은 두 가지 한계를 드러낸다. 첫 번째 원시 코드의 전처리를 위한 일반적인 문제이다. 두 번째 한계는 Ada 자체에서 동적 이벤트 선언과 바인딩이 제공되지 않는다는 점이다.

```
   for Package_1
       declare Event_1
         X: Integer ; Y: Package_N.My_Type;
       declare Event2
       when Event_3 => Method_1 B
   end for Package_1
   for Package_2
       declare Event_3 A, B: Integer ;
       when Event_2 => Method_4
       when Event_1 => Method_2 X
   end for Package 2
   for Package_3
       when Event_2=> Method_3
       when Event_1=> Method_4 Y
   end for Package_3

with Package_N
package Event_Manager is
       type Event is
       (Event_1, Event_2, Event_3);
  type Argument(The_Event:Event)is
       record
          case The_Event is
               when Event_1 =>
                     Event_1_X:Integer;

                    Event_1_Y:Package_N.My_Type;
               when Event_2=>
                     null;
               when Event_3=>
```

```
                    Event_3_A:Integer;
                    Event_3_B:Integer;
                when others =>
                    null;
            end case;
        end record;
    procedure Announce_Event(The_Date:Argument);
end Event_Manager
```

그림 4. 확장된 Ada를 이용한 소프트웨어 아키텍처 기술

3.4 OMT와 같은 모델링 도구의 응용

OMT는 객체지향 분석 기법의 하나로 객체 모델링, 동적 모델링, 기능 모델링을 적용하여 시스템에 요구되는 객체들을 기술한다. 또한 분석 모델링의 결과를 통합하여 새로운 소프트웨어 시스템을 생성하기도 한다[1][2].

<그림 5>는 Pipe&Filter 스타일의 소프트웨어 아키텍처를 표현한 것이고 <그림 6>은 object Modeling Technique(OMT)를 활용하여 시스템을 표현한 것이다. <그림 7>은 파이프 커넥터를 first-class 개체로 소프트웨어 아키텍처를 표현한 것이다. <그림 8>은 <그림 6>, <그림 7>에서 표현한 간단한 Pipe&Filter 아키텍처 표현을 통합한 것이다.

객체 지향 분석과 설계를 위한 그 밖의 모델링 언어로 UML을 들 수 있다. UML은 유즈 케이스다이어그램, 클래스 다이어그램, 시퀀스 다이어그램, 콜레보레이션 다이어그램등으로 구성된다. UML을 이용한 소프트웨어 아키텍처의 표현도 OMT와 유사한 방식을 따른다.

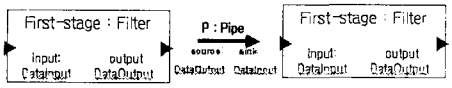

그림 5. Pipe&Filter 스타일

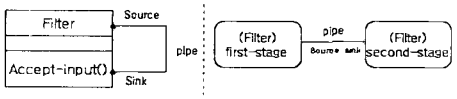

그림 6. Pipe&Filte시스템 아키텍처의 간략한 OMT 표현

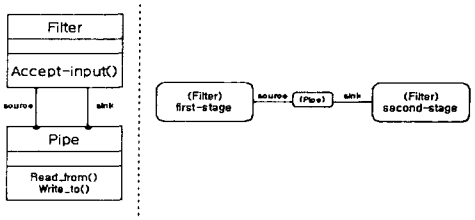

그림 7. Pipe를 first-class 개체로 나타낸 OMT 표현

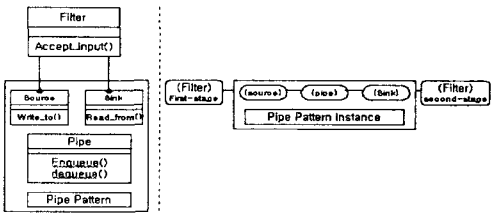

그림 8. 최종적인 OMT 기반의 소프트웨어 아키텍처 표현

3.5 순수한 ADL

순수한 ADL은 개념적 아키텍처와 first-class 개체로써 커넥터를 명확히 다루는 것에 초점을 두어 아키텍처를 표현한다. Wright와 Rapide와 같은 ADL은 고수준의 추상성으로 컴포넌트와 커넥터를 모델화한다. 그리고 아키텍처 표현과 구현 사이에 특별한 관계를 묘사하지 않으므로써 구현에 독립적이다. Weaves, UniCon, and MetaH와 같은 몇몇 ADL은 구현에서 고 수준의 아키텍처 일치성을 요구한다. 이러한 언어에서 모델화된 컴포넌트는 직접적으로 구현과 관계되어지고, 모듈 내부 명세는 아키텍처 표현과 구별된다.

이외에도 시스템 아키텍처의 특성에 따라 ACME, Aesop, C2, Darwin, SADL등의 ADL중 적합한 ADL을 선택하여 아키텍처를 표현할 수 있다. <그림 9>는 Wright를 이용하여 'Pipe&Filter' 아키텍처 스타일을 표현한 예이다[3][5][6][9][10].

```
Style pipe-and-filter
 Interface Type DataInput=(read AE→ (data?x→
    DataInput  []  end-of-data→close→✓)) [ ]
(close→✓)
 Interface Type DataOutput=write→ DataOutput
[]close→✓

Connector Pipe
  Role Source = DataOutput
  Role Sink = DataInput
  Glue = Buf < >
    where
```

```
Buf < > = Source.write?x→Buf <x> []
          Source.close → Closed < >
Buf_s <x> = Source.write?y→Buf <y> s <x>
          [] Source.close → Closed_s <x>
          [] Sink.read → Sink.data!x →
                         Buf_s
          [] Sink.close → Killed
Closed_s <x> = Sink.read → Sink.data!x→Closed_s
          [] Sink.close → ✓
Closed < > = Sink.read→Sink.end-of-data→
          Sink.close→✓
Killed = Source.write→Killed [] Source.close→
         ✓

Constraints
∀ c : Connectors · Type(c)=pipe
∀ c : Components · Filter(c)
      where
       Filter(c:Component)=∀p    :    Ports(c) ·
Type(p)
                          =DataInput∨Type(p)
                          =DataOutput
End Style
```

그림 9. Wright를 이용한 소프트웨어 아키텍쳐 기술의 예

4. 아키텍쳐 기술 언어의 요구조건

아키텍쳐 기술언어는 대규모, 고 수준 설계를 다루며 이러한 설계는 특정 구현으로 적용할 수 있어야 한다. 또한 사용자 정의나 응용에 특정한 추상화를 지원해야 하며, 아키텍쳐 패러다임의 선택을 지원해야 한다. 시스템은 언어와 환경 사이의 밀접한 상호작용으로 이루어진다. 즉 언어는 정확한 표현을 가지기 위해 필요한 반면, 환경은 그러한 표현이 유용하고 재사용할 수 있게 하는데 필요하다.

언어상의 논점에 초점을 맞추어 보면 이상적인 아키텍쳐 기술 언어가 제공해야 할 특성은 다음과 같다.

4.1 아키텍쳐 기술 언어의 필요 특성

(1) 결합
독립적인 컴포넌트의 결합과 연결(connection)로 시스템을 표현하는 것이 가능해야 한다.
(2) 추상화
소프트웨어 아키텍쳐 내의 컴포넌트와 컴포넌트들간의 상호작용은 시스템에서의 추상적인 역할을 분명하고 명시적으로 기술할 수 있어야 한다.
(3) 재사용성
서로 다른 아키텍쳐 표현에서 컴포넌트, 커넥터 그리고 아키텍쳐 패턴이 재사용될 수 있어야 한다.
(4) 구성
아키텍쳐 표현은 구조화되는 요소들과 독립적으로 시스템 구조의 표현을 지역화해야 한다. 또한 동적 재구성을 지원해야 한다.
(5) 이질성
다중의 이질적인 아키텍쳐 기술을 결합하는 것이 가능해야 한다.
(6) 분석
아키텍쳐 기술의 풍부하고 다양한 분석을 수행할 수 있어야 한다.
(7) 사용의 용이성
사용자들이 소프트웨어 아키텍쳐를 표현함에 있어 쉽고 편리하게 사용할 수 있어야 한다.

4.2 아키텍쳐 기술 방식의 비교

MIL은 전체적인 시스템 구성을 표현하기에는 적합하나 컴포넌트간의 재사용 패턴을 제공하지 않으므로 결합성 · 재사용성이 떨어지며, 그에 따른 추상화 · 이질성 · 분석 등을 나타내기 어렵다.

형식 명세 언어는 디자인을 정확하게 문서화할 수 있고 그 시스템의 동작에 대해 추론할 수 있도록 결합성 · 추상화 · 재사용성 · 구성 · 이질성 · 분석 등을 나타내기에는 적합하나 수학적인 기초를 바탕으로 표현되어지므로 일반 사용자들이 쉽게 활용하고, 재사용하기에는 어렵다는 문제점을 지니고 있다.

프로그래밍 언어 응용은 기존 언어를 활용하여 아키텍쳐를 기술함으로써 사용이 용이하며 대략적인 결합 · 추상화 · 재사용성 · 분석을 나타낼 수는 있으나, 정확하게 기술할 수 없으므로 인해 오류가 발생될 수 있으며, 시스템 전반에 관한 구성 및 이질성을 표현하기에는 부적합하다.

OMT 방식은 추상화 · 재사용성을 나타내기에는 적합하고 사용 용이성도 좋은 반면, 결합 · 구성 · 이질성 · 분석 등의 기술이 부족하여 소프트웨어 아키텍쳐에서 표현해야 하는 복잡한 프로토콜의 컴포넌트 상호작용을 표현하고, 컴포넌트 인터페이스를 표현하기에는 부적합하다.

순수한 ADL은 소프트웨어 아키텍쳐 기술 언어가 갖추어야 할 결합 · 추상화 · 재사용성 · 구성 · 이질성 · 분석 등을 만족시킨다. 사용의 용이성 면에서 조금 부족한 점이 있으나, 이 점은 사용자에게 친숙하도록 계속적으로 연구되어지고 있다. 결론적으로 순수한 ADL이 소프트웨어 아키텍쳐를 표현하기에 가장 적합하다. 아키텍쳐 기술 방식에 따른 특성을 비교한 것은 <표 1>과 같다.

<표 1> 아키텍처 기술 방식의 비교

요구조건 / 기술방식	결합	추상화	재사용성	구성	이질성	분석	사용 용이성
MIL	×	×	×	○	×	×	×
형식명세언어	○	○	△	○	○	○	×
프로그래밍 언어 응용	△	△	△	×	×	△	○
OMT	△	○	○	△	△	△	○
순수한 ADL	○	○	○	○	○	○	△

5. 결 론

소프트웨어 시스템의 크기와 복잡도가 커짐에 따라 시스템의 전체적인 제어 구조·통신·동기화·데이터 엑세스를 위한 프로토콜 등을 나타내는 아키텍처 기술이 중요한 이슈로 대두되고 있다.

일반적인 시스템 설계자들에게 정확한 개념·분·추상적인 문제 해결을 위한 시스템 구조의 표현과 선택을 위한 판단 기준 등을 나타내는 아키텍처 기술을 만족시키는 기법으로 제시된 것이 순수한 아키텍처 기술 언어(ADL)이다. 순수한 아키텍처 기술 언어는 결합·추상화·재사용성·구성·이질성·분석·사용의 용이성 등을 만족시켜 소프트웨어 아키텍처를 정확하게 기술하므로써 소프트웨어 유지 보수를 예측하고 재사용성, 수정을 통한 확장성을 용이하게 한다. 지금까지 개발된 아키텍처 기술 언어는 일반 사용자에게 친숙하지 못하다는 문제점과 다양한 아키텍처 특성에 따른 고유한 기술 언어를 선택하여 아키텍처를 기술하여야 한다는 본질적인 문제점을 안고 있다. 본 논문을 통해서 우리는 아키텍처 기술 언어가 갖추어야 할 요구 조건을 제시하였으며, 향후 계속적인 연구를 통해 제시한 요구 조건을 만족하면서, 시스템 설계자가 쉽게 활용할 수 있으며 다양한 아키텍처 특성을 동시에 만족할 수 있는 새로운 아키텍처 기술 언어가 개발되어야 할 것이다.

참 고 문 헌

[1] 윤 청, "성공적인 소프트웨어 개발 방법론", 생능출판사 pp.320~367, 1999.

[2] 천유식, "소프트웨어 개발 방법론", 대청정보시스템(주) pp.151~184, 1996.

[3] M. Shaw and D. Garlan, "Software Architecture", Prentice-Hall, 1996.

[4] M. Garlan and D. Shaw, "An Introduction to Software Architecture", Advandes in Software Engineering and Knowledge Engineering, Volume I, edited by V. Ambriola and G. Tortora, World Scientific Publishing Company, New Jersey, pp.1~16, 1993.

[5] C. Gacek, A. Abd-Allah, B. Clark and B. Boehm, "On the Definition of Software System Architecture", Technical Report USC/CSE-95-TR-500 April. 1995.

[6] N. Medvidovic and R. N. Taylor, "A classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, Vol 26, No. 1, 2000.

[7] D. E. Perry and A. L. Wolf, "Foundation for the Study of Software Architecture", ACM SIGSOFT Software Engineering Notes vol 17 no 4 pp. 40~52, October 1992.

[8] F. Deremer, H. H. Kron, "Programming-in -the-large Versus Programming-in-the-small", IEEE Transactions on Software Engineering, Vol 5, No. 5, pp.321~327, 1976.

[9] N. Medvidovic and R. N. Taylor, "A Framework for Classifying and Comparing Architecture Description Languages", Proceeding of the Sixth European Software Engineering Conference together with Fifth ACM SIGSOFT Symposium on the Foundation of Software Engineering, pp. 60~76, 1997.

[10] N. Medvidovic and D. S. Rosenblum, "Domains of Concern in Software Architectures and Architecture Description Languages", Proceedings of the USENIC Conference on Domain-Specific Languages, pp. 199~212, 1997.

[11] J. E. Robbins, N. Medvidovic, D. F. Redmiles and D. S. Rosenblum, "Intergrating Architecture Description Languages with a Standard Design Method", Second EDCS Cross Cluster Meeting,

[12] M. Shaw and D. Garlan, "Characteristics of Higher-level Language for Software Architecture", CMU-CS-94-210, December 1994.

[13] R. T. Monroe, D. Kompanek, R. Melton, and D. Garlan, "Stylized Architecture, Design Patterns, and Objects", IEEE Software, September 1996.

권 기 태 (Ki-Tae Kwon)

1986년 2월 서울대학교 졸업(이학사)
1988년 2월 서울대학교 대학원 졸업 (이학석사)
1993년 2월 서울대학교 대학원 졸업 (이학박사)
1990년 9월~현재 강릉대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어공학, 데이터베이스

변 분 희 (Boon-Hee Byun)

1996년 2월 강릉대학교 졸업(이학사)
2000년 2월 강릉대학교 교육대학원 졸업(전산교육석사)
2002년 3월~ 현재 강릉대학교 컴퓨터공학과 박사과정 중
관심분야 : 소프트웨어공학