ADL 모델로부터 VRML 구현 모델을 위한 변환기 개발

김 치 수†

요 약

소프트웨어 아키텍처는 텍스트 기반 아키텍처 기술 언어(ADL)를 사용하면서 기술하게 된다. ADL의 중요한 목적은 다른 이해관계자 사이에서 대체 디자인을 통신하고, 재사용할 수 있는 구조를 찾아내고, 그리고 디자인 결정을 기록하는 것이다.

본 논문은 구조적인 관점의 3차원 표현을 위한 도구를 만듦으로써 표현 문제에 대한 해법을 제공한다. 도구는 첫째 소프트웨어 아키텍처와 아키텍처에서 관점을 기술하는 아키텍처 기술 언어(VTADL)로 구성되었고, 각 관점을 분리된 가상현실 세계로 번역하는 VTADL-to-VRML 변환기로 구성되었다.

본 논문에서는 ADL을 요구된 관점에 의거하여 효과적인 VRML 표현으로 변환하기 위한 알고리즘을 고안했다. VRML 표현은 그 전체적인 디자인에 이해를 강화하고 다양한 이해관계자 사이에 통신을 개선할 것이다.

키워드 : 아키텍처 기술 언어, 가상현실 모델링 언어, 변환기

The Development of a Translater for the VRML Implementation Model from the ADL Model

Chi-su Kim†

ABSTRACT

Software architectures may be described using text-based architecture description language(ADL). The key goals of an ADL are to communicate alternate designs between different stakeholders, to detect reusable structures, and to record design decisions.

This paper provided a solution to the representation problem by creating a tool for three-dimensional representation of architectural viewpoints. The tool consisted of an architecture description language(VTADL) to first describe the software architectures and viewpoints on the architectures; and a VTADL-to-VRML translater to translate each viewpoint into a separate virtual reality world.

The goal of the paper was to devise algorithms for translating an ADL into effective VRML representations based on the desired viewpoint. The VRML representations were intended to enhance comprehension on the overall design and to improve communications between diverse stakeholders.

Key Words : ADL, VRML, Translater

1. Introduction

소프트웨어 아키텍처는 학계, 산업계, 연구소 등 소프트웨어 관련 종사자들로부터 그 중요성이 증대되고 있는 분야이다. 많은 잡지에서 소프트웨어 아키텍처 특집을 다루기도 하고 아키텍처에 관한 워크숍과 컨퍼런스가 자주 열리고 있다. 또한 아키텍처에 관한 책과 논문 역시 중요시되고 있다[1, 2]. 소프트웨어 아키텍처는 이해관계자(stakeholder) 사이의 의사소통 도구로 사용되어질 수 있고 개발 단계 초기의 디자인에 대한 결정이라 할 수 있기 때문에 이를 통해 시스템 개발의 나머지 단계, 배치, 관리 등의 방향을 결정할 수 있다. 소프트웨어 아키텍처는 시스템의 구성과 각 컴포넌트들 간의 관계가 비교적 작은 형태의 추상화된 모델로 이루어진다. 따라서 아키텍처는 시스템 간에 교환가능하고, 재사용의 목적으로도 이용될 수 있다[3-5].

이러한 소프트웨어 아키텍처를 표현하기 위한 방법 중 대표적인 것이 아키텍처 기술 언어(ADL : Architecture Description Language)[6]이다. 지금까지 많은 아키텍처 기술 언어들이 제안되었으나, 각각의 기술 언어들은 각 적용 영역에 따라 고유한 특성을 가지면서 발전해 왔고, 이들이 사용한 아키텍처 기술은 많은 경우 정형 명세 언어에 기반하고 있기 때문에 정형기법에 관한 지식이 부족한 일반 개발자의 경우 명세에 어려움을 가질 수 있다. 또한 제안된 컴포넌트의 정적인 측면과 동적인 측면, 컴포넌트 기반 시스템, 컴포넌트간의 결합, 가변성 중 일부만 명세에 반영한다[7].

본 논문에서는 기존 ADL 언어의 단점을 보완하고자 VTADL

※ 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10555-0)지원으로 수행되었음.
† 종신회원 : 공주대학교 컴퓨터공학부 교수(공학연구원)
논문접수 : 2004년 12월 13일, 심사완료 : 2005년 12월 5일

(Visually Translatable Architecture Description Language)을 새롭게 정의하였으며, 정의된 VTADL을 VRML (Virtual Reality Modeling Language)로 변환해 주는 알고리즘을 개발하였다. VRML로의 표현은 전체적인 설계에 대한 이해가 쉽고 다양한 이해관계자들 사이에 의사소통을 더욱 원활하게 해줄 것이다. 따라서 VRML 표현은 하나 이상의 관점을 시각화하고 웹 브라우저에서 토글(toggle)로 이해관계자의 멀티 관점을 이해하는데 도움을 준다. 또한 VRML에서 하이퍼링크를 사용하여 이해관계자들은 확실한 설계 원리를 추적하거나 아키텍처 요소들의 정당성과 결함을 검증할 수 있다. 더 중요한 것은 ADL이 가상 현실에서 시각화함은 물론 장래의 프로젝트에서 재사용 할 수 있는 패턴의 저장소로도 적합할 것이다.

2. 관련 연구

2.1 소프트웨어 아키텍처 기술 언어

지금까지 연구되어진 대표적인 ADL은 크게 4가지 유형으로 나눌 수 있다. 다음 <표 1>은 대표적인 ADL의 4가지 유형별 특징과 언어들을 소개한다[8].

다양한 아키텍처 기술 언어들은 각기 자신이 적용될 영역에 알맞게 설계되었기 때문에, 각각이 명세 하는 부분과 검증할 수 있는 부분 또한 각기 다르다. 그럼에도 불구하고 아키텍처 기술 언어들은 대부분 공통적으로 컴포넌트 기술, 커넥터 기술, 구성(configuration) 기술을 포함한다.

또한 각각의 아키텍처 기술 언어들은 검증의 과정을 수행하기 위하여 각기 다른 정형적인 언어에 기반을 두고 있다. 아키텍처 기술 언어의 기반에 따라서 언어가 할 수 있는 검증의 종류도 각기 달라진다. 이처럼 각각의 아키텍처 기술 언어들은 자신이 기반을 둔 정형 기법에 따라 제한된 분석만을 지원한다. 또한 코드 생성의 경우 아키텍처 기술 언어가 제공한다 할지라도 대부분 기본적인 단계 코드 생성만을 지원한다.

〈표 1〉 ADL의 유형에 의한 특징과 언어

유 형	특 징	언 어
시맨틱 모델 기반 ADL	· 아키텍처의 구조적 또는 행위적 성질들을 정형 시맨틱 모델에 의거하여 엄밀하게 기술하는 표현 수단을 제공	Wright, Rapide, Darwin
스타일 기반 ADL	· 특정한 아키텍처 스타일을 따르는 아키텍처의 기술을 지원하거나, 아키텍처 스타일을 사용자가 정의할 수 있는 표현 수단을 제공	Unicon, Aesop, C2SADL
도메인 기반 ADL	· 특정 응용 도메인에 속한 아키텍처들의 기술에 적합한 언어로서 해당 도메인에서 중요한 아키텍처 측면들을 잘 기술 · 실시간 또는 내장 시스템들의 아키텍처들을 기술하는데 적합	ROOM, MetaH
ADL 인터체인지	· 다수의 ADL들을 병용하여 사용할 수 있도록 서로 다른 ADL로 표현된 아키텍처 기술들의 상호 변환과 공유를 지원	Acme

지금까지 다양한 아키텍처 기술 언어들에 대해 간략하게 비교해 보았지만, 각각의 기술 언어들이 서로 다른 특정한 영역에 국한되어 발전해 왔기 때문에, 일반적인 아키텍처를 기술함에 있어 광범위하게 사용되기에는 미흡한 점이 있을 뿐만 아니라 정형 기법에 관한 지식이 부족한 일반 개발자의 경우 명세에 어려움을 가질 수 있다.

본 논문에서는 위와 같은 단점을 보완하여 폭넓게 사용할 수 있는 아키텍처 기술 언어인 VTADL을 정의하였다. VTADL은 인터페이스 동작과 같은 성질들은 생략하고 기초적인 아키텍처 성질(구성, 기본적인 컴포넌트, 정보 교환 등)을 전달하기 위해 작성되었고[9], 정의된 요소들을 시각적으로 표현하기 위해서 위치(상, 하, 좌, 우, 전, 후) 정보를 추가하였다. 즉, 하나 이상의 소프트웨어 아키텍처에 다른 관점들을 표현하기 위한 기능, 선택된 요소에 대한 하이퍼링크 기능, 이미 정의된 아키텍처 요소들을 선택적으로 나타내거나 숨기는 기능과 다양한 아키텍처들을 나타내는 기능을 추가하였다.

또한 아키텍처를 표현하기한 모델 중 4+1 관점[10]과는 별개로 3개의 주요한 관점(시스템 관련 관점, 그룹 협력 관점과 조직 관점)을 사용하였고 각 관점들 사이의 의미 손실을 줄이기 위하여 표현된 아키텍처 구조에 요구 또는 조직이나 다른 문서들로 연결되는 하이퍼링크 기능을 추가한다[11, 12].

3. VRML구현 모델로의 변환과정

본 논문에서 제시한 변환 과정은 다음과 같다. 첫 번째 단계에서는 단순하지만 시각화가 가능한 효과적인 아키텍처 기술 언어를 정의하였다. 그리고 두 번째 단계에서는 ADL 소스에서 궁극적으로 정규식을 VRML로 시각화될 변환기를 설계 하였다.

3.1 아키텍처 기술 언어 정의

첫 번째 단계는 아키텍처 기술을 위해 ADL의 기본적 요구들을 만족하는 일반적인 아키텍처 기술 언어를 정의한다. 본질적으로, ADL은 소프트웨어 구조를 전체적으로 대략 기술한다. 그러나, 언어는 컴포넌트, 컴포넌트 사이의 커넥션, 그리고 토폴로지의 구성을 기술해야 하며, 컴포넌트(역할, 관련된 프로세스, 인터페이스, 등등)나 커넥터(관계형, 무게, 방향성, 연결성 정의, 등등)의 일반 성질도 언어로 정의되어야만 한다.

또한 언어는 아키텍처 스타일이 표현 가능해야 한다. 여기서 아키텍처 스타일이란 토폴로지와 인스턴스화에 관한 제약들로 컴포넌트와 커넥터가 사용된 방법을 정의하는데, 대표적인 스타일의 예는 파이프라인, main-program-and-subroutine, 객체 지향, 계층화와 상태 기반을 둔 스타일이다. VTADL이 다른 몇 개의 스타일들을 고려하지만 본 논문에서는 program call-and-return과 계층화 스타일 두 가지만 VRML에서 시각화한다.

그리고 ADL은 특정한 도메인에 국한되어서는 안된다. 즉 언어는 일반적인 성질을 요구하며, 비전문가가 비교적 이해

하기 쉬워야 하고 시각화된 분명한 구조들을 포함해야 한다. 예를 들면 만약 계층적 구조이면 언어는 한 컴포넌트가 다른 컴포넌트의 자식인지 아닌지 분명히 진술되어야 한다.

따라서 ADL은 동적 행동을 모델링하는 상태-기준 표기법보다는 소프트웨어 아키텍처의 정적인 표현에 집중한다.

본 논문에서 "아키텍처 구조 기술 언어"가 ADL의 디자인에서 모델로 사용되었기 때문에 기존의 아키텍처 기술 언어를 ASDL이라 부른다. ASDL은 언어 디자인에 기본적인 특징인 행동의 명세들을 숨기고 중요한 정적인 구조적 성질들을 강조한다. 비록 ASDL이 원래 레거시 시스템(legacy systems)에서 아키텍처 복구로 쓰여졌지만 여러 도메인에서 일반적인 사용도 가능하다.

본 논문은 기존의 ASDL을 기초로 하고, 새로운 기능을 추가한 시각화된 새로운 아키텍처 기술 언어인 VTADL (Visually Translatable Architectural Description Language)을 개발한다. 개발될 VTADL의 표현식들은 LR(왼쪽에서 오른쪽) 파싱에서 문법적인 생성 규칙들부터 재귀적으로 생성된 정식 언어이며 VTADL은 LR 파서에 의해 VRML로 변환된다.

3.1.1 아키텍처 리스트

VTADL 소스 파일은 아키텍처 리스트와 뷰 리스트의 파트로 이루어져 있고 뷰는 아키텍처 리스트에서 하나 이상의 아키텍처를 사용하지만 반드시 정의된 아키텍처만 사용한다.

```
Architecture <Arch-Name>
type <Style>
{
   ComponentList
   {
      Component <Name-1>;
        ⋮
      Component <Name-n>;
   }
   ConnectionList
   {
      Connection <ConnName-1>;
        ⋮
      Connection <ConnName-n>;
   }
}
```

(그림 1) 아키텍처 정의 템플레이트

```
Component <Comp-Name>
ComponentType <Type-of-Component>;
Properties:
   CompRole: <Role-of-Component>;
   ChildOf: <Component-Parent>;
   Layer: <Layer-Name>;
   Process: <Process-Name>;
      InterfaceList:
      Interface <Relative-Position> <Interface-Name-1>;
      { InterfaceRole: <Role-Selection>; }
            ⋮
      Interface <Relative-Position> <Interface-Name-n>;
      { InterfaceRole: <Role-Selection>; }
```

(그림 2) VTADL의 컴포넌트 속성들

```
ConnectionList
{
   Connector <Connect-Name-1>
      ConnectType <Connect-Type> <Connect-Direction>;
      Connect(<From-Interface-Name>, <To-Interface-Name>);
        ⋮
   Connector <Connect-Name-n>
      ConnectType <Connect-Type> <Connect-Direction>;
      Connect(<From-Interface-Name>, <To-Interface-Name>);
}
```

(그림 3) VTADL의 커넥션 리스트

(그림 1)은 실제적인 아키텍처를 정의한 것이다.

(그림 1)에서 아키텍처 이름은 유일한 이름이 할당되고, 아키텍처의 스타일은 완전한 구조로 할당되었다. 본 논문에서는 program call-and-return이나 계층화 스타일을 위해 Program Component만 사용한다.

(그림 2)는 VTADL의 컴포넌트 속성들을 정의한 것이다.

한편 컴포넌트가 컴포넌트리스트에서 정의되면 커넥션리스트는 이전에 정의된 컴포넌트 사이에서 커넥션들을 정의하기 위해 사용된다. 따라서 커넥션은 컴포넌트 속성에서 인터페이스 이름을 사용하면서 정의되기 때문에 커넥션리스트는 항상 컴포넌트리스트와 함께 한다. (그림 3)은 VTADL의 커넥션 리스트를 위한 템플레이트를 보여 준다.

(그림 3)에서 보듯이 커넥션은 **Connector** 예약어를 사용한다. 커넥터 타입은 DataFlow, ControlFlow나 Associates의 하나로 지정하고 커넥터의 방향성을 Unidirect나 Bidirect로 지정한다.

실제 커넥션은 **Connect** 예약어를 사용하며 Connect절의 괄호에서 From-Interface- Name은 커넥터가 시작된 인터페이스 이름이고 To-Interface-Name은 최종 수신지 인터페이스 이름으로 지정된다.

3.1.2 뷰 리스트

뷰 리스트는 VTADL 파일에서 ViewList로 정의하고 하나 이상의 뷰들을 정의한다.

다음 (그림 4)는 VTADL 파일에서 VRML로 변환되었을 때 시각적으로 나타나는 부분을 정의한 뷰 리스트 템플레이트를 보여 준다.

```
ViewList
{
   ViewMain
   {
      { UsingArch <Arch-Name-1>; }
        ⋮
      { UsingArch <Arch-Name-n>; }
   }
   View <View-Name-1>
   {
     . . .
```

(그림 4) VTADL의 뷰 리스트 템플레이트

3.1.3 VTADL의 BNF정의

(그림 5)는 VTADL을 위한 BNF를 일부 나타낸다.

```
<vtadl>          ::= <archmain> <viewpart>
<archmain>       ::=NULL | <archmain> <archdef>
<archdef>        ::= <architecture> <archname>
                     <architect_style>
<leftmark> <partslist> <connslist> <rightmark>
<architecture>   ::= ARCHITECTURE
<leftmark>       ::= LEFTBRACKET
<rightmark>      ::= RIGHTBRACKET
<architect_style>   ::= ARCHSTYLE <stylechoice>
<stylechoice>    ::= PROGRAM | OBJECT | PIPELINE |
                     LAYER
<archname>       ::= ID
<partslist>      ::= COMPLIST <leftmark> <compbreakdown>
                     <rightmark>
<connslist>      ::= CONNLIST <leftmark> <connbreakdown>
                     <rightmark>
<compbreakdown>  ::= NULL | <compbreakdown>
                     <middlemark>
<middlemark>     ::= COMPONENT <compname> COMPTYPE
                     <comptype> <semimark> <proplist> <intlist>
<proplist>       ::= PROPERTIESCOLON <propdetails>
<propdetails>    ::= <component_role> <child_of> <layer_no>
                     <process_def>
<component_role>   ::= COMPROLE COLON <child_selection>
                     <semimar>
```

(그림 5) VTADL의 BNF

3.2 기하학적인 모델 설계

기하학적인 모델은 후에 비주얼 매체를 통하여 보여 질 수 있도록 그래픽적인 객체의 중간 표현이라 할 수 있다. 이 기하학적인 모델은 어레이, 큐, 메트릭과 같은 자료구조로 이루어진다. 따라서 이 과정에서는 VTADL을 VRML로의 변환 과정을 설명한다. 여기서 (그림 6)은 컴파일러의 재분할(subdivisions)로 변환 과정을 분해한 것인데 텍스트 기본 소스 파일이 그래픽의 언어로 변환된 중간 기하 모델은 VRML 객체들의 원시 언어에서 정의된 아키텍처 구조의 사상을 포함하며 1개의 VRML 파일은 VTADL 소스 파일에서 정의된 각 뷰를 위해 생성하게 된다.

VTADL 소스 파일 → 어휘 분석기 → 토큰 → 파서 →
기하학 모델러 → VRML 생성기 → VRML 뷰 → 시각화된 뷰

(그림 6) 기본적인 VTADL을 VRML로의 편집 과정

3.2.1 아키텍처 리스트

VTADL 소스 파일의 첫 번째는 아키텍처 리스트이다. 아키텍처 리스트는 한 개의 아키텍처를 표현하는 각 노드와 함께 선형적 링크 리스트로 표현되고 VTADL에 기술된 아키텍처는 아키텍처 노드에 저장된다. 아키텍처가 VTADL 파일의 아키텍처 리스트에 나타난 순서대로 아키텍처 노드는 링크 리스트에 삽입된다.

아키텍처 노드는 아키텍처의 이름을 식별하고 스타일, 컴포넌트, 커넥터와 토폴로지를 정의한 정보를 포함하며 아키텍처의 컴포넌트 개수와 커넥터 개수도 아키텍처 노드에 포함된다. (그림 7)은 아키텍처 링크 리스트에서 노드를 설명한 것이다.

Arch-List Node

Style
Arch_ID[25]
Arch_CompID[25][25]
Arch_CompPosition[25][4]

Arch_ConnID[25][25]
Arch_ConnPosition[24][4]
Topology[25][25]

No_Comps
No_Conns

archlink: Pointer to next node.

(그림 7) 아키텍처 링크 리스트에서 노드

3.2.2 뷰 리스트

VTADL 파일의 두 번째는 뷰 리스트이며 뷰 리스트는 메인 뷰("ViewMain")를 포함하고, 하나 이상의 사용자 정의 뷰들을 포함할 수 있다. 뷰 리스트는 직렬 링크 리스트로서 실행된다. 링크 리스트의 각 노드는 한 개의 뷰를 표현하며 링크 리스트의 첫 번째 노드들은 필수의 메인 노드들이 표현되고 그 다음에 사용자 정의 뷰 노드들이 표현된다. 링크 리스트의 뷰들은 VTADL 소스 파일에 나타났던 순서대로 삽입된다.

뷰는 파일의 처음에 정의된 arch-list에서 하나 이상의 아키텍처를 사용할 수 있으며 single view의 모든 속성들은 뷰 리스트의 노드에 포함시킨다. (그림 8)은 뷰 리스트 노드의 속성들을 기술한다.

View-List Node

View_ID[25]

No_Archs
Arch_ID[25][25]

All_Comps[25]
All_Conns[25]
View_Comps[25][25]
View_Conns[25][25]

Hyperlinks_Comps[625][25]
Hyperlinks_Conns[625][25]

Viewlink: pointer to next View node.

(그림 8) 뷰 리스트 노드 정의

3.3 사례 연구

사례 연구는 VTADL을 VRML로 변환하는 시각화 도구의

실행가능성을 보여주기 위해 실시한다. 사례 연구는 첫 번째로 사례 연구의 배경을 상세하게 기록하고 아키텍처의 VTADL 해석을 기술하고 VTADL 소스 파일의 VRML 시각화를 제공한다.

(1) 사례 연구 이름 : 모바일 로봇 아키텍처에 관한 관점들

(2) 목적의 개요와 사례 연구의 배경

이 사례연구는 모바일 로봇을 제어하기 위해 소프트웨어 아키텍처 모델링하는 문제를 3가지 다른 솔루션으로 재표현한다. 각 솔루션은 VTADL로 표현하게 되고, 그 다음에 VRML 뷰들로 변환된다.

모바일 로봇의 아키텍처는 로봇이 환경으로부터 자극들에 성공적으로 응답할 수 있도록 설계되어야 하며, 불확실한 정보와 시스템에 대한 위험과 미래의 요구 사항들에 대해서 유연성 있게 설계되어야 한다. Shaw와 Garlan[2]이 기술한 것처럼 솔루션 1은 제어 루프 아키텍처를 사용했다. 환경 정보가 센서 컴포넌트에 공급되며 센서 컴포넌트는 제어기에 데이터를 공급한다. 센서 데이터에 의하여 제어기는 작동기 컴포넌트로 모바일 로봇의 동작을 명령할 것이고 또한 작동기 컴포넌트는 모바일 로봇의 동작을 명령할 것이다. 솔루션 1의 장점은 간결함에 있으며 환경으로부터 로봇의 상호작용의 핵심이 달성되었다. 그러나 제어 루프의 단점으로 복소수 분해를 요구하는 모델 태스크들은 실패했다.

솔루션 2는 레이어 아키텍처를 사용했다. 잘 조직된 컴포넌트는 로봇의 오퍼레이션을 통합하기 위해 필요하며 솔루션 2의 단점은 계층화 모델이 의미하는 것처럼 정보가 종종 인접한 레이어 사이에서 정확히 전달되지 못한다는 것이다.

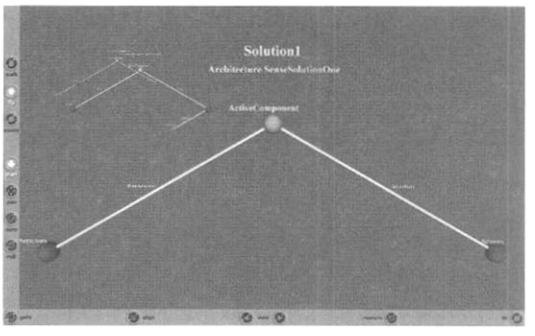
솔루션 3은 TCA(Task-Control Architecture)라는 암시적 호출 아키텍처를 사용했다. "암시적 호출"이란 호출 프로세스들이 서로 상호작용하지 않는 제약과 프로세스가 이벤트 발생에 의한 호출되는 것을 의미하며 TCA는 테스크 트리라는 태스크들의 계층을 인스턴트화 한다. 또한 TCA는 로봇 상태들과 환경을 변경하는 것에 대응하여 테스크 트리를 변경할 수 있는데 이 모델의 장점은 모듈 방식으로 모델의 기능이 명확히 분리된다는 것이지만 아키텍처가 로봇이 환경에서 불확실한 조건들을 처리하는 방법은 모델링되지 않았다는 단점이 있다.

솔루션 4는 블랙 보드 아키텍처를 사용했지만 이 아키텍처는 블랙 보드 아키텍처 스타일이 변환기에서 구현되지 않았기 때문에 모델링하지 않았다.

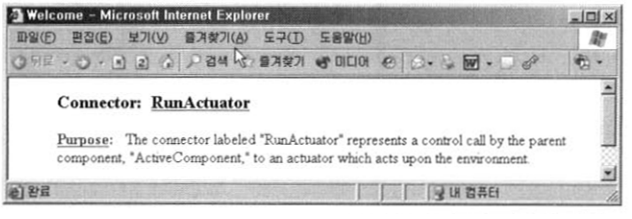
3개의 솔루션들(솔루션 1, 2, 3)은 시각화 이전에 VTADL를 사용하면서 표현되었으며 VTADL 표현은 본 논문에서 개발된 변환기를 사용하여 하나 이상의 VRML 파일들로 변환하였다.

(3) VRML 이미지 화면

(그림 9)의 "Solution 1"은 SenseSolutionOne과 ActSolutionOne 2개의 아키텍처가 있다. 이 2개의 변환은 SenseSolutionOne으로 만들어진다. 아키텍처 SenseSolutionOne은 모바일 로봇에서 작동기와 센서를 제어하는 일반적인 활동 컴포넌트를



(그림 9) SenseSolutionOne 아키텍처



(그림 10) RunActuator에 하이퍼링크된 RunActuator.html 파일

표현하는 call-and-return 아키텍처로 아래와 같이 그려지게 되고 ActSolutionOne과 함께 다른 변환은 배경에 보이게 된다. 화면에서 숨겨진 다른 아키텍처 요소, 3개의 컴포넌트와 2개의 커넥터를 시각화했다.

루트 ActiveComponent에서 소스 VTADL 파일 "MobileRobot.txt"로 하이퍼링크 된다.

커넥터 RunActuator에 링크된 HTML 파일 "RunActuator.html"은 (그림 10)에서 화면과 같이 제공된다.

이 사례 연구는 모바일 로봇의 3가지 솔루션을 VTADL로 표현하고 뷰들을 VRML로 변환하였다. 이해관계자들의 필요에 따라 각 컴포넌트와 커넥터를 선택적으로 숨기게 하거나 나타낼 수 있었고 하나의 아키텍처를 여러 가지의 뷰들로 표현하여 관점의 불일치를 해소하였으며 하이퍼링크 기능으로 관련된 문서파일 링크시켜 이해관계자들의 이해를 도울 수 있음도 확인하였다.

4. 결론 및 향후 연구 과제

본 논문에서는 아직 일부분이지만 call-and-return 스타일과 계층화 스타일을 시각화로 활용할 수 있는 VTADL을 정의하였고, 변환기를 통하여 VRML파일로 변환하여 다른 관점에서 여러 가지 뷰들을 보였다.

그 결과로 복잡한 구조를 3차원 시각화 표현하고, 전체적인 구조와 일부를 가린 부분적인 구조를 살펴봄으로서 이해관계자들의 이해를 크게 향상시켰다. 하이퍼링크를 통하여 ADL의 소스문서까지 링크할 수 있는 기능은 이해관계자로 하여금 시스템의 구조를 빠르게 파악할 수 있게 해 주었다.

개발된 변환기는 제한된 컴포넌트와 커넥터를 사용한 프로토타입으로 개발된 것이다. 그러므로 향후 연구과제로는 제한한 컴포넌트와 커넥터의 사용을 확장하고 잘 정제되고

세련된 버전으로 개발될 필요가 있고 현재는 정적인 아키텍처만을 구현하였지만 차후에는 동적인 아키텍처까지도 표현할 수 있는 변환기를 개발할 예정이다.

참 고 문 헌

[1] P. Clements, L. Northrop, "Software Architecture: An Executive Overview", CMU/SEI-96-TR-003 ADA305470.

[2] M. Shaw, D. Garlan, "Software Architecture : Perspectives on an Emerging Discipline", Prentice-Hall, 1996.

[3] L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice", Addison-Wesley, 1998.

[4] I. Jacobson, G. Booch, "James R umbaugh, The Unified Software Development Process", Addison Wesley, 1999.

[5] http://www.sei.cmu.edu/architecture/definitions.html

[6] N. Medvidovic, R. N. Taylor, "A classification and comparison framework for software architecture description languages", IEEE Transactions on Software Engineering, Vol.26, No.1, January, 2000.

[7] D. Garlan, R. Allen, J. Ockerbloom, "Architectural Mismatch or Why it's hard to build systems out of existing parts", Proceedings Of the 17th ICSE, IEEE Software, Vol.12, No.6, pp.17-26, April, 1995.

[8] 노성환, 신동익, 전태웅, "아키텍처 기반의 컴포넌트 조립을 위한 ADL 지원 환경", 소프트웨어공학기술 논문지, 제2권, 소프트웨어공학기술 논문지. 2002.

[9] W. Eixelsberger, & H. Gall, "Describing Software Architectures by System Structure and Properties", In Proceedings of the 22nd Computer Software and Applications Conference, COMSAC '98 (pp.106-11). Los Alamitos: IEEE Computer Society Press, 1998.

[10] P. Knuchten, "The 4+1 View Model of Architecture", IEEE Software, Vol.12, No.6, pp.43-50, Nov., 1995.

[11] C. Hofmeister, R. Nord, D. Soni, "Applied Software Architecture", Addison-Wesley, 2000.

[12] J. Rumvaugh, I. Jacobson, G. Booch, "The Unified Mideling Language Reference Manual", Addison-Wesley, 2000.

김 치 수

e-mail : cskim@kongju.ac.kr

1984년 중앙대학교 전자계산학과(학사)

1986년 중앙대학교 일반대학원 전자계산학과(석사)

1990년 중앙대학교 일반대학원 컴퓨터공학과(공학박사)

1990년~1992년 공주교육대학교 전임강사

1992년~현재 공주대학교 컴퓨터공학부 교수

관심분야 : 객체지향 분석 및 설계, CBD 방법론