DML(Diagram Markup Language) 시스템의 설계 및 구현

정회원 김 성 근*, 김 영 철**, 유 재 우***

A Design and Implementation of DML(Diagram Markup Language) System

Sung keun Kim*, Young chul Kim**, Chae woo Yoo*** ***Regular Members***

요 약

다이어그램은 직관성과 간결성을 갖는 장점이 있어 현재의 컴퓨팅 환경에서 많은 분야에 널리 사용되고 있다. 하지만 다이어그램 작성에 대한 표준화된 방법의 부재(不在)로 소프트웨어 상호간의 자료공유가 어렵고 다이어그램 컴포넌트와 규칙을 직접 프로그래밍 해야 하기 때문에 많은 시간과 노력을 필요로 한다. 이러한 문제점을 해결하고자 본 논문에서는 표준화된 문서규약인 XML을 이용해 다이어그램 컴포넌트의 형태와 행위를 정의하는 방법, 다이어그램의 규칙과 의미를 기술하는 방법에 대해 제안하고 제안한 XML 문서가 실행될 수 있는 다이어그램 시스템의 설계 및 구현에 관하여 논의한다.

본 다이어그램 시스템에서는 DML 그래픽 편집기를 제공하여 WYSWYG 방식으로 다이어그램 컴포넌트를 정의하고 자동으로 DML 문서를 생성할 수 있도록 함으로써 더욱 효율적으로 새로운 다이어그램을 개발할 수 있도록 하였다. 뿐만 아니라 DTD를 이용해 다이어그램에 대한 문법을 정의함으로써 문서의 구조를 정의하는 DTD에 대한 의미적 일관성을 유지하였으며 Semantic Definition XML을 이용하여 의미를 기술할 수 있도록 하였다. 또한 다이어그램 시스템 에서 다이어그램 문장에 대한 문법검사와 의미실행의 방법은 VPL(Visual Programming Language)의 여러 개념들을 이용하였다.

Key Words : DML; Diagram System; XML; DTD; VPL.

ABSTRACT

The diagram has a intuition and simplicity. So, it is widely used in various fields in current computing environment. But, because of the absence of a standard diagram specification method, we have difficulty in exchanging the diagram data between different diagram software and besides, we spend much money and time to code diagram component, rules and semantics to which diagram would be applied. So We propose a method for defining diagram component's shapes and actions, diagram's rules and semantics using XML. And We design and implement the diagram system which execute XML document specifying diagram.

In the diagram system, We can define diagram component in WISWIG manner and generate DML document automatically. So We can develop diagram system more efficiently. And by defining diagram rules using DTD,

* 가톨릭상지대학 컴퓨터정보계열 (skkim@csangji.ac.kr), ** 숭실대학교 정보미디어기술연구소 (yckim@amin.ssu.ac.kr),
*** 숭실대학교 시스템소프트웨어 연구실 (cwyoo@ssu.ac.kr)
논문번호 : KICS2004-10-231, 접수일자 : 2004년 10월 8일
※ 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.

we also achieve the consistency of DTD meaning. And We propose Semantic Definition XML for specifying diagram semantics. So, diagram sentence which drawn by users could be given semantics and executed in diagram system. In this thesis, many VPL(Visual Programming Language) concepts were adopted to implement diagram system environment.

Ⅰ. 서 론

1.1 연구 배경 및 목적

현재의 향상된 컴퓨팅 환경에서 다이어그램은 많은 분야에 널리 사용되고 있으며 점차 중요성도 높아지고 있다. 다이어그램은 표현형식을 간략화 할 수 있고 의미전달을 명확히 할 수 있는 특징이 있어 사용자의 이해를 돕고, 그룹 성원들 간의 의사소통을 원활히 하는 장점이 있다. 이 때문에 다이어그램은 단순한 문서화에서부터 UML[1]과 같은 시스템 분석 및 설계의 도구로 사용되고 있으며 프로그래밍 도구로도 사용되고 있다. 또한, 시각 프로그래밍 환경에서나 새로운 모델링을 위한 소프트웨어를 개발할 때 다이어그램은 필수적인 요소이다. 이러한 소프트웨어 시스템에서 사용되는 다이어그램들은 응용분야에 맞는 규칙과 의미를 가지고 있으며, 이러한 요구 사항들을 만족시키기 위해서 다이어그램은 단순한 GIF나 JPEG 등의 이미지가 아닌 프로그램으로 작성되어야 한다. 다이어그램을 프로그램으로 구현하는 것은 많은 시간과 노력을 필요로 하는 작업이기 때문에 자동화된 방법이 필요하다. 또한 기존 다이어그램 소프트웨어들은 서로 다른 방법으로 구현되어서 소프트웨어들 간의 다이어그램 컴포넌트 공유가 불가능하다. 이러한 문제점을 해결하고자 XML을 이용하여 그래픽 정보를 나타내는 방법들이 연구되고 있다.[2, 3]

다이어그램 컴포넌트는 정적인 형태 정보뿐만 아니라 동적인 행위 정보를 갖고 있다. 즉, 화면에서의 위치, 크기 또는 색깔 등의 정적인 정보뿐만 아니라 사용자의 속성 변경에 대한 제약 사항이나, 다이어그램 컴포넌트 자체의 의미와 같은 동적인 정보를 갖기도 한다. 그러나 기존의 XML을 이용한 그래픽 표현은 정적인 형태 정보만을 표현하기 때문에 동적인 객체의 특성을 갖는 다이어그램 컴포넌트를 표현하기에는 부족하며, 행위 속성을 정의할 수 있는 새로운 방법을 필요로 한다. 또한, 다이어그램 실행환경에서 다이어그램 컴포넌트들을 연결해서 사용할 경우 다이어그램 컴포넌트는 자신이 다른 컴포넌트와 연결될 수 있는지에 대한 정보를 가지고 있어야 한다. 즉, 잘못된 다이어그램을 그렸을 경우 사용자에게 잘못 그려진 다이어그램이라는 정보와 어느 부분이 잘못 그려진 것인지에 대한 힌트를 제공하여야 한다.

본 논문에서는 이러한 요구사항에 맞도록 XML을 이용하여 다이어그램 컴포넌트를 정의하는 방법과 다이어그램 유효성을 검사하는 방법, 의미에 맞게 실행할 수 있는 방법에 대해 소개하고 다이어그램 시스템을 설계 및 구현하고자 한다.

Ⅱ. 관련 연구

SVG[2]는 XML을 기반으로 그림정보를 나타내는 VML[3]이 연구되었지만 제한된 벡터 그래픽만을 표현할 수 있는 단점이 있었다. 이 때문에 W3C를 중심으로 벡터 그래픽뿐만 아니라 래스터 그래픽을 포함하는 새로운 XML 응용프로그램을 연구하게 되었다. SVG(Scalable Vector Graphics)는 세 가지 그래픽 객체 즉 shapes, images 그리고 text로 구성되며 객체들은 그룹으로 묶을 수 있고 스타일을 지정할 수도 있다. 이외에 간단한 움직임을 표현하는 것뿐만 아니라 3차원 모핑(Morphing) 효과까지 그래픽 표현을 위한 많은 기능들을 정의함으로써 웹 개발자에게 동적인 웹 페이지 개발에 대한 가능성을 심어주게 되었다.

XGMML[12]는 그래프 표현을 위한 GML에 기반한 XML 응용프로그램인 XGMML(Extensible Graph Markup and Modeling Language)은 원소가 그래프의 노드(node)와 에지(edge)를 나타낸다. XGMML은 서로 다른 그래프 저작도구나 탐색 도구들 간의 그래프 교환을 목적으로 WWWPAL시스템에서 웹 사이트를 표현하기 위해 개발되었다.

DiaGen(Diagram Editor Generator)[10]은 다이어그램 컴포넌트를 정의하는 방법, 리듀싱 규칙, 문법 및 의미를 기술하여 자동으로 다이어그램 편집기를 생성하며 다이어그램 해석 과정은 어휘분석단계(Scanning Step), 리듀싱 단계(Reducing Step), 구분 분석 단계(Parsing Step), 의미 해석 단계(Semantic Analysis Step) 순이다.

이 과정은 우리가 익숙히 알고 있는 텍스트 언어(text language)의 해석 과정과 유사함을 알 수

있다. 물론 본 논문이 제안하는 방법과의 유사점 또한 있으나 DiaGen은 하나의 다이어그램 정의를 위해 여러 개의 복잡한 hypergraph 문법을 사용함으로써 이해와 구현하기 어렵고 확장성이 낮은 단점이 있다.

Ⅲ. 이론적 배경

속성문법(Attribute Grammar)[11]은 의미 규칙을 추가한 문맥 자유 문법(context-free grammar)으로 속성은 다음 두 가지 타입으로 구분한다. 첫째는 상속 속성(inherited attribute)으로 파스 트리에서 부모노드에서 자식노드로 속성 값이 전달되는 것을 말하며 둘째는 합성 속성(synthesized attribute)으로 자식노드에서 부모노드로 속성 값이 전달되는 것을 말한다. 속성문법은 본 논문에서 제안하는 다이어그램의 정의에서 두 가지 방식으로 사용된다. 첫째는 다이어그램 컴포넌트의 정의에서 DML 문서에 직접 속성문법을 기술하는 방법과 둘째는 다이어그램 문장의 정의에서 의미 행위를 실제 문법기호와 분리하여 외부 파일인 Semantic Definition XML에 기술하는 방법이다.

hypergraph[9]은 유한한 개수의 node와 hyperedge를 갖는 방향성 그래프의 일반화된 자료구조이다. 각 hyperedge는 이름과 노드와 연결될 수 있는 여러 개의 촉수(tentacle)를 갖고 각 hyperedge의 촉수는 node를 통해 서로 연결된다.

그림 1은 하나의 원에 세 개의 화살표가 연결된 형태에 대해 Hypergraph Model을 적용한 예제이다.

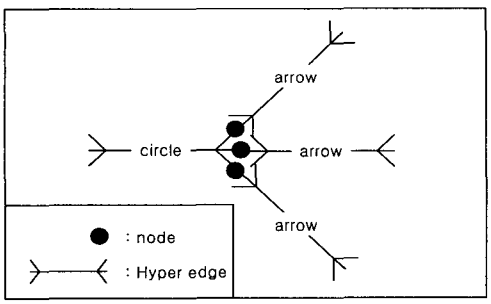

그림 1. Hypergraph Model

Ⅳ. 다이어그램의 정의

다이어그램의 정의는 다이어그램 컴포넌트와 다이어그램 문장의 정의방법으로 나눈다.

4.1 다이어그램 컴포넌트의 정의

본 논문에서는 다이어그램 컴포넌트의 정의를 위해 DML(Diagram Markup Language)을 제안한다.

표 1. DML의 DTD

```
<!ELEMENT diagram (model, connections?)>
<!ATTLIST diagram name NMTOKEN #REQUIRED
                  coordx CDATA #REQUIRED
                  coordy CDATA #REQUIRED
                  namespace NMTOKEN #REQUIRED>
<!ELEMENT model (appearance, semantics?)>
<!ELEMENT appearance (shape+,
                      constraints?, appearance*)>
<!ELEMENT constraints (inheritedconstraints?,
                       synthesizedconstraints?)>
<!ELEMENT connections (connection)+>
<!ELEMENT shape EMPTY>
<!ELEMENT semantics EMPTY>
<!ATTLIST semantics
                  name NMTOKEN #REQUIRED>
<!ELEMENT inheritedconstraints (#PCDATA)>
<!ELEMENT synthesizedconstraints (#PCDATA)>
<!ELEMENT connection EMPTY>
<!ATTLIST connection type NMTOKEN #REQUIRED
                  value CDATA  #REQUIRED>
```

즉, DML은 다이어그램 컴포넌트의 형태정보(shape), 의미정보(semantics), 제약사항(constraints) 그리고 연결정보(connections)를 위한 원소로 구성된다.

4.1.1 형태정보 정의

다이어그램 컴포넌트의 형태정보는 컴포넌트를 구성하는 그래픽 객체의 종류, 위치, 크기, 색 등의 속성정보를 말한다.

표 2. 컴포넌트의 형태 속성

속 성	설 명
type	그래픽 객체의 종류를 지정
x	컴포넌트의 초기 X 좌표값
y	컴포넌트의 초기 Y 좌표값
endx	1차원 컴포넌트의 최종 X 좌표값
endy	1차원 컴포넌트의 최종 Y 좌표값
width	2차원 컴포넌트들의 폭 값
height	2차원 컴포넌트들의 높이 값
linecolor	컴포넌트 선의 색 값
fillcolor	컴포넌트 공간의 색 값

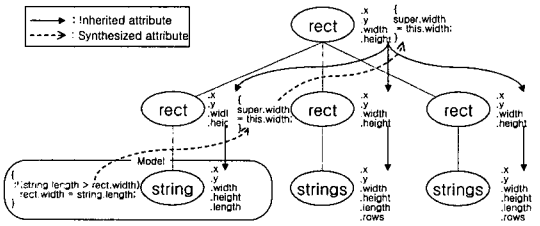

그림 2. 클래스 컴포넌트 의존그래프

4.1.2 제약사항 정의

다이어그램 컴포넌트는 2차원 공간에서 여러 컴포넌트들의 집합으로 구성되어 있으며 공간에 대한 집합적 관계성을 부여함으로써 계층적 구조로 표현이 가능하다. 이렇게 생성된 트리 자료구조에서 제약사항들을 추출하여 의존그래프(dependency graph)를 생성함으로써 각 노드간의 영향 범위와 순서를 찾아내어 관리할 수 있다.

의존그래프의 각 노드는 속성(x, y, width, height 등)과 행위({...})를 갖는 객체이며 전체 노드가 서로 유기적으로 묶여 하나의 다이어그램 컴포넌트를 구성한다. 상위 노드의 변경이 하위 노드의 속성 정보에 영향을 주는 것이 상속 속성으로 표현되었으며, 하위 노드의 속성 변화에 대해 상위 노드로 영향을 주는 것이 합성 속성으로 표현된다.

속성 문법의 타입에 따라 두 개의 독립된 DAG (Directed Acyclic Graph)로 볼 수 있고 각 DAG의 위상 정렬(topological sort)에 의해 각 노드의 영향 범위에 따른 정렬이 가능해져 어떤 한 노드의 변형에 의한 다른 노드의 변형에 대한 정의가 가능해진다. 또한 정렬된 노드 정보를 이용하여 다이어그램 컴포넌트를 list, switch-case 또는 table-driven 방식으로 관리할 수 있다.

4.1.3 의미정보 정의

다이어그램 컴포넌트는 자신이 갖는 의미를 갖고 있다. 예를 들어 데이터베이스 연결, 다이얼로그 프레임, 클래스 생성 등 사용자가 컴포넌트를 이용해 표현하고자 하는 의미를 갖고 있다. 이를 의미 원소를 이용하여 정의한다.

4.1.4 연결정보 정의

다이어그램 컴포넌트는 다른 컴포넌트와의 연결영역을 갖는다. 이 연결영역은 점(point) 또는 영역(area)으로 구분되며 다이어그램 스캐너가 다른 컴포넌트와의 관계성을 파악할 때 사용한다. 즉, 두 컴포넌트 사이의 연결영역 포함관계를 파악해 inside, outside, touch 등의 관계성을 파악한다.

4.2 다이어그램 문장의 정의

다이어그램을 그리기 위해서 해당 다이어그램에 포함되는 다이어그램 컴포넌트들을 사용한다. 다이어그램 컴포넌트는 자신이 다른 컴포넌트와 연결될 수 있는지에 대한 정보를 갖고 있어 만약 잘못된 연결을 할 경우 오류메시지를 화면에 출력한다. 또한 올바른 다이어그램이 그려지면 의미에 맞게 실행된다. 예를 들어 UML 다이어그램을 그리기 위해 package, class, interface, association 등의 다이어그램 컴포넌트를 사용한다. 그런데, class와 package 컴포넌트를 association 컴포넌트를 이용하여 연결할 경우 오류메시지를 출력하고 올바른 클래스 다이어그램을 그린 경우 이를 Java, C++ 등의 언어로 변환시켜주는 것과 같다.

이와 같은 요구사항을 만족시키기 위해 다음 두 가지의 정의를 필요로 한다. 첫째 다이어그램의 문법을 정의하는 방법이며 둘째 다이어그램이 의미에 맞게 실행될 수 있는 방법을 정의하는 것이다.

4.2.1 다이어그램 문법 정의

다이어그램 문장의 유효성 판단을 위해 표준화된 문서규약인 XML의 구조적 정의를 위한 DTD를 이용한다. 본 논문에서는 DTD의 문법기호를 넌 터미널 원소, 터미널 원소, 객체에 대한 링크구현을 위한 context symbol 터미널 원소로 구분한다.

다음은 본 논문의 적용 예제로서 상태도에 대한 DTD 정의이다.

상태도 DTD에서 STD, SS, S, TR은 넌 터미널 원소이고 이외의 원소는 터미널 원소이다. 터미널 원소 중 state와 endstate는 context symbol로서 Semantic Definition XML에 정의되어 있다.

표 3. 상태도 DTD

```
<!ELEMENT STD (SS, S*)>
<!ELEMENT SS (startstate, TR)>
<!ELEMENT S ((state|endstate), TR*)>
<!ELEMENT TR (leave, transition, arrive,
(state|endstate))>
<!ELEMENT startstate EMPTY>
<!ELEMENT state EMPTY>
<!ELEMENT endstate EMPTY>
<!ELEMENT transition EMPTY>
<!ELEMENT leave EMPTY>
<!ELEMENT arrive EMPTY>
```

본 논문에서 제안하는 시스템은 다이어그램 문법을 extended LL(1) 문법으로 작성한다. 이것은 DTD가 Top-Down의 성격과 EBNF 형식으로 나타낼 수 있는 장점을 갖고 있어 구현이 쉽고 효율적인 파싱이 가능하기 때문이다.[15]

4.2.2 의미 규칙의 정의

DTD만으로는 의미정보를 추가하기에 어려움이 있다. 따라서 별도의 XML 문서를 이용해 속성문법을 적용하는 방법으로 의미정의(Semantic Definition) XML을 제안한다. 이렇게 문법과 의미정보의 분리는 XML 문서를 다양한 방식으로 처리할 수 있도록 한다.

표 4에서 문서의 최상위 원소는 semantics 원소이고, semantics 원소는 definition, rules, programs라는 자식 원소를 갖는다. definition 원소는 다이어그램 시스템에서 필요로 하는 정보를 정의하며 usercode, lookahead, relations, contextsymbols 원소를 포함한다. usercode와 programs 원소는 다이어그램 시스템의 실행모듈에서 실행할 인터프리터 언어인 Jython 코드를 기술한다. lookahead 원소는 다이어그램 파서가 파싱 중 새로운 lookahead의 요청 시 다이어그램 스캐너가 현재의 SRG 노드에서 인식할 수 있는 lookahead의 종류를 기술한다. relations 원소는 다이어그램 문장에 적용될 관계성에 대해 방향성을 고려한 정의를 하고 다이어그램 스캐너가 수행할 관계성 계산 수식(expression)을 Jython 코드로 기술한다. rules 원소에는 다이어그램 DTD의 각 원소에 대한 의미규칙들을 Jython 코드로 기술한다. 이는 프로그래밍 언어에서 넌 터미널에 의미규칙을 부여하는 방법과 유사하다.

표 4. Semantic Definition XML의 DTD

```
<!ELEMENT semantics
        (definition?, rules?, programs?)>
<!ATTLIST semantics namespace
                    NMTOKEN #REQUIRED>
<!ELEMENT definition (usercode?, lookaheads?,
relations?, contextsymbols?)>
<!ELEMENT usercode (#PCDATA)>
<!ELEMENT lookaheads (lookahead)*>
<!ELEMENT lookahead EMPTY>
<!ATTLIST lookahead source CDATA #REQUIRED
            target CDATA #REQUIRED>
<!ELEMENT relations (relation)*>
<!ELEMENT relation (type)+>
<!ATTLIST relation source CDATA #REQUIRED
            target CDATA #REQUIRED>
<!ELEMENT type (expression)>
<!ATTLIST type name NMTOKEN #REQUIRED>
<!ELEMENT expression (#PCDATA)>
<!ELEMENT contextsymbols EMPTY>
<!ATTLIST contextsymbols name
                    CDATA #REQUIRED>
<!ELEMENT rules (rule)*>
<!ELEMENT rule (pre?, static?, post?)>
<!ATTLIST rule name NMTOKEN #REQUIRED>
<!ELEMENT pre (#PCDATA)>
<!ELEMENT static (#PCDATA)>
<!ELEMENT post (#PCDATA)>
<!ELEMENT programs (#PCDATA)>
```

V. 다이어그램 시스템의 설계 및 구현

다음은 다이어그램 시스템에 대한 구조도이다. 시스템의 확장성을 고려하여 모듈단위로 구분하였고 서로 분리하여 설계 및 구현하였다.

5.1 시스템 구조

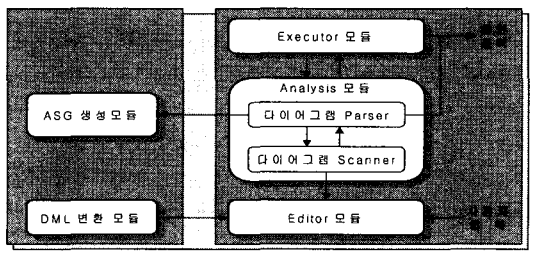

그림 3. 다이어그램 시스템 구조도

전체 다이어그램 시스템은 위의 그림과 같은 모듈 구조를 갖는다. 다음은 각 모듈에 대한 정의와 자료구조를 중심으로 설명한다.

5.1.1 DML 변환 모듈

DML은 다이어그램 컴포넌트를 정의하는 XML로 다이어그램 시스템에서 다이어그램 컴포넌트 객체로 변환되어야 한다. 이러한 역할을 DML 변환 모듈이 수행한다.

DML 그래픽 편집기는 사용자가 여러 개의 기본 도형을 이용해 다이어그램 컴포넌트를 정의하면 집합관계 트리 생성기를 통해 집합관계를 파악하여 트리를 생성한다. 이 트리를 순회하여 앞에서 정의한 DML 문서로 변환한다. 생성된 DML문서는 XSLT[13] 또는 DOM[14] 파서를 통해 Java, C++ 또는 C#등의 언어로 변환된다. 변환된 다이어그램 컴포넌트 객체는 다이어그램 시스템의 편집모듈에 등록되어 사용된다.

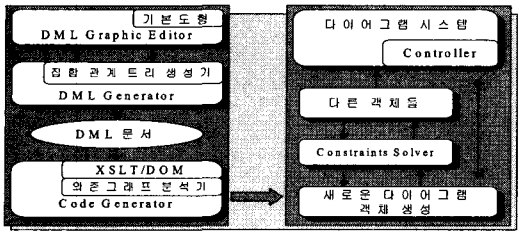

그림 4. DML 변환모듈 구조도

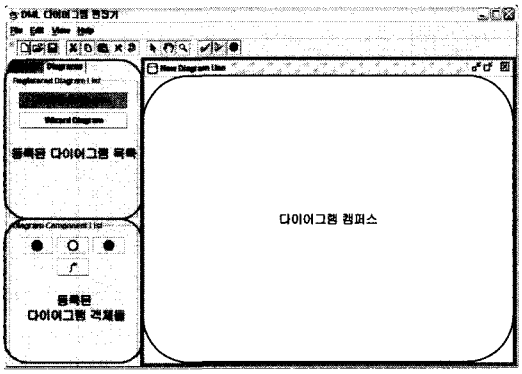

그림 5. 다이어그램 시스템 화면

5.1.2 편집모듈

그림 5는 다이어그램 시스템 편집기의 편집모듈로 다음 다섯 가지 주요 패널로 구성되어 있다. 즉, 등록된 다이어그램 목록, 해당 다이어그램의 다이어그램 컴포넌트 목록, 다이어그램 캠퍼스, 편집 툴 그리고 실행 툴로 구성된다.

5.1.3 ASG 생성모듈

ASG(Attribute Syntax Graph)는 다이어그램의 DTD를 기반으로 생성되며 DTD가 갖는 의미적 결함을 의미정의 XML을 이용해 보완한 구문그래프(syntax graph)로서 다이어그램 파서가 다이어그램 문장의 유효성 판단을 위한 자료구조로 활용한다. 다음 그림 6은 입력과 출력을 고려하여 그려진 ASG 생성모듈 구조도이다.

다이어그램 DTD는 wutka사의 DTDParser 1.19에서 파싱되어 추상 구문 트리를 생성하고 이 추상 구문 트리는 Linked Forest 변환기를 이용하여 Linked Forest 자료구조로 변환된다. ASG 생성기는 Linked Forest와 Semantic Definition XML의 정보를 이용해 ASG를 생성한다. 노드의 종류는 터미널노드로 ELEMENT, PCDATA, ANY, EMPTY 노드가 있고 넌 터미널 노드로 SEQ(','), OR('|'), OPT('?'), PLUS('+'), STAR('*') 노드로 구성되며

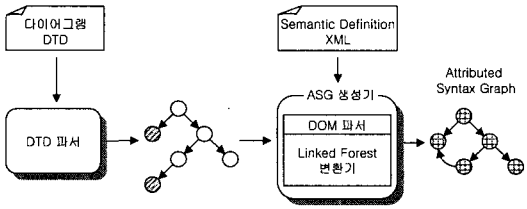

그림 6. ASG 생성모듈

의미 행위(semantic action)는 Semantic Definition XML의 해당 규칙(rule)에 정의된 의미정보를 추출하여 설정한다.

5.1.4 분석모듈

다이어그램 스캐너는 편집모듈에서 읽은 다이어그램 컴포넌트에 대한 속성정보와 Semantic Definition XML의 관계성에 대한 정보를 바탕으로 SRG(Spatial Relationship Graph) 자료구조를 생성하고 다이어그램 파서는 다이어그램 문장에 대해 문법을 검사하고 잘못된 문장인 경우 오류 메시지를 보내고 옳은 문장인 경우 Derived DAG를 생성한다.

5.1.5 실행모듈

실행모듈은 분석모듈에서 생성된 Derived DAG를 순회하며 노드에 부여된 semantic action을 실행한다. 다음 그림은 분석모듈과 실행모듈의 관계를 보여준다.

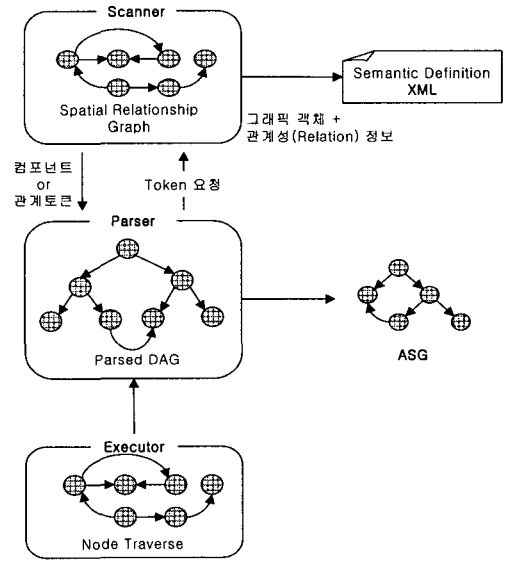

그림 7. 다이어그램 분석 및 실행과정

5.2 다이어그램 스캐너

분석모듈에 속한 다이어그램 스캐너는 사용자가 편집기 모듈을 통해 입력한 다이어그램 문장에 대한 SRG를 생성하고 다이어그램 파서가 토큰을 요청할 경우 lookahead 토큰을 반환한다.

5.2.1 스캐너 설계

다이어그램 스캐너는 getRelationType 메소드를 호출하여 두 다이어그램 컴포넌트 사이의 관계성을 파악하고 buildSRG 메소드를 호출하여 SRG 자료구조를 생성한다. 생성된 SRG 자료구조에서 getNextNode 메소드를 호출하여 현재의 노드에 연결된 다음 lookahead 노드를 다이어그램 파서에게 반환한다.

5.2.2 관계성 처리

다이어그램의 종류에 따라 관계성(relationship)의 종류도 다르기 때문에 다이어그램 스캐너는 관계성에 대한 정의를 하지 않고 단지 관계성을 파악 할 수 있는 방법에 대한 정의만을 하고 있다. 즉, 다이어그램 스캐너는 해당 다이어그램의 Semantic Definition XML의 relation 원소의 정보를 이용하여 관계성의 이름(name)과 관계성 계산 수식(expression)을 알아내고 source와 target 속성을 이용해 방향성에 대한 처리를 한다. 수식은 Jython 코드로 기술되어 바로 시스템에 적용가능하며 관계성의 추가 및 삭제가 손쉽게 이루어진다.

5.2.3 SRG 생성

SRG는 Hypergraph Model을 적용한 자료구조로서 SRG의 노드는 다이어그램 컴포넌트인 경우 COMPONENT, 관계성 노드인 경우 RELATION의 종류를 갖고 실제 다이어그램 컴포넌트 객체에 대한 링크정보를 갖는다. 그리고 연결된 다른 노드들에 대한 링크정보를 갖는다.

다음 표 5는 전체 다이어그램 컴포넌트에 대한 관계성을 파악하여 SRG 자료구조를 생성하기 위한 알고리즘이다.

5.3 다이어그램 파서

다이어그램 파서는 사용자가 입력한 다이어그램 문장에 대해 문법적 오류를 검사한다.

5.3.1 파서 설계

그림 8은 다이어그램 파서와 ASG 생성모듈에 대한 설계도이다.

표 5. SRG 생성 알고리즘

```
입력: 다이어그램 컴포넌트 집합
출력: SRG(Spatial Relationship Graph)
함수:
전체 다이어그램 컴포넌트 객체의 SRG노드를 생성한다.
for( 다이어그램 컴포넌트 집합의 각 모델 α에 대해 ) {
  for( 모델 α의 연결영역에 연결된 각 모델 β에 대해 ) {
    if( α와 β가 아직 관계성을 파악하지 않은 경우 ) {
      α와 β가 갖는 관계성의 타입과 수식을 파악한다.
      if( 관계성 수식을 계산하여 true인 경우 )
      {
        관계성에 대한 SRG노드를 생성한다.
        관계성 노드와 α와 β노드를 서로 연결한다.
      }
    }
  }
}
```

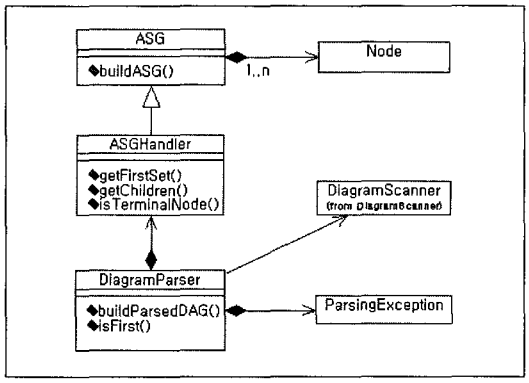

그림 8. 다이어그램 파서 설계

다이어그램 파서는 ASG 생성모듈(ASGHandler 객체)을 생성하여 ASG 자료구조를 생성하고 DiagramParser객체의 buildParsedDAG 메소드를 호출하여 Parsed DAG를 생성한다.

5.3.2 다이어그램 문법 검사

다이어그램 파서는 ASG를 기반으로 Recursive Descent Parsing(또는 Syntax Graph-based Parsing)을 수행한다.

다이어그램 DTD는 extended LL(1) 문법으로 작성되어 있기 때문에 STAR(*), PLUS(+), OPT(?)에 대해 다음과 같은 방법으로 처리한다.

표 6. OP 노드 처리방법

C*의 경우	while(lookahead가 C의 FIRST인 동안) {
	...
	}

C+의 경우	do { ... } while(lookahead가 C의 FIRST인 동안)

C? 의 경우	if(lookahead가 C의 FIRST이면) {
	...
	}

5.3.3 Parsed DAG 생성

노드의 종류는 다이어그램 컴포넌트와 관계성을 TERMINAL, 넌 터미널 원소를 나타내는 NONTERMINAL 그리고 문법규칙을 적용하기 위해 존재하는 터미널 노드인 CONTEXTSYMBOL로 구성된다. 그리고 사용자 입력 파라미터 값과 의미정보의 참조를 위해 SRG와 ASG노드에 대한 링크정보를 저장하고 있다.

그림 9는 Parsed DAG의 노드구조에 대한 전체 참조그림이다.

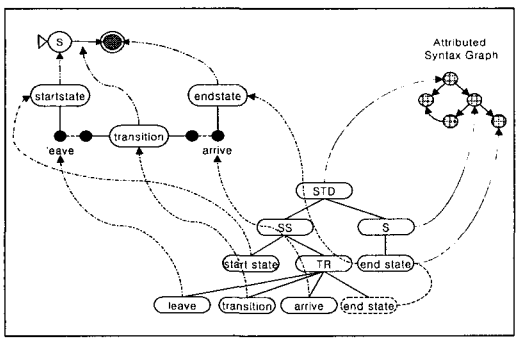

그림 9. Parsed DAG의 참조구조

5.4 다이어그램 실행기

다이어그램 실행기는 실행모듈에 해당하며 파서를 통해 생성된 Parsed DAG를 순회하여 각 노드에 부여된 의미정보를 수행한다.

5.4.1 실행기 설계

실행기는 쓰레드(thread)로 구현되며 Parsed DAG의 각 노드에 부여된 pre_condition, main, 그리고 post_condition을 수행한다.

실행기는 사용자와의 비동기적인(asynchronous) 인터페이스를 처리하기위해 WaitObject 객체를 가지고 있다.

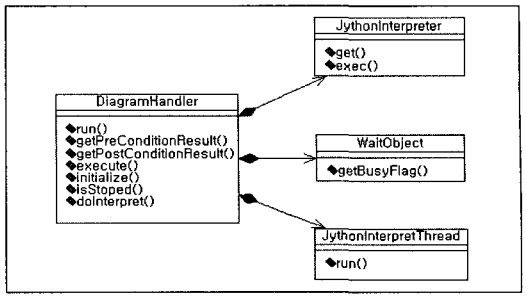

그림 10. 다이어그램 실행기 설계

5.4.2 Parsed DAG의 순회 및 의미해석 방법

다이어그램 파서를 통해 생성된 Parsed DAG는 의미해석을 위한 기반 자료구조로 사용된다.

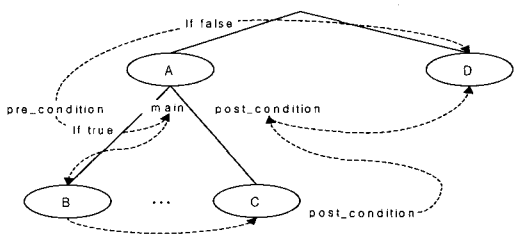

그림 11. Parsed DAG 순회 방법

그림 11과 같이 실행기는 Parsed DAG를 순회하면서 각 노드의 pre_condition, main, post_condition에 기술된 Jython코드를 실행한다.

실행기는 A 노드의 pre_condition을 수행하여 결과 값이 참이면 A 노드의 main을 수행하고 자식 노드를 순회하고 만약 거짓이라면 A 노드의 이웃 노드인 D 노드를 순회한다.

이러한 노드 순회방법은 LL(1) 문법의 속성(attribute)을 계산하기 위한 깊이우선 평가(depth-first evaluation order)와 유사하다.[16]

이외에 실행기는 Jython 인터프리터 환경과의 인터페이스를 위해 몇 가지 변수를 공유한다.

표 7. 실행기와 Jython 환경과의 인터페이스 변수

parameter	Edite 모듈을 통한 사용자 입력 값을 Jython 영역으로 넘겨준다.
precondition	boolean 값으로 true 또는 false
stop	실행기와 Jython 영역 간 실행중지를 표시
lock	사용자와의 비동기적인 인터페이스를 위한 Locking 기법을 위한 변수로서 true 또는 false의 boolean 값이다.

5.4.3 context symbol 처리방법

Context symbol은 문법적용을 위한 터미널로서 생성규칙의 LAST에 위치하며 생성규칙의 FIRST로 존재하는 자신노드를 링크한다. 이 문법기호는 실행기가 다음에 실행할 노드를 선택할 때 다음 노드의 위치정보를 나타내는 역할을 한다.

다이어그램 실행기는 컨텍스트 심벌의 정보를 스택을 이용해 관리한다. 즉, 실행 중 컨텍스트 심벌이 나타나면 스택에 삽입한 후 다음 노드를 선택할 때 스택에서 삭제하고 제어를 해당 컨텍스트 심벌의 부모노드로 이동시킨 후 순회를 계속한다.

Ⅵ. 결론 및 향후 연구방향

본 논문에서는 현재 다이어그램 시스템을 만들기 위해 많은 시간과 노력을 들이는 문제와 서로 다른 시스템간의 자료공유의 어려움을 해결하고자 표준화된 문서형식인 XML을 적용해 다이어그램 컴포넌트를 나타내고 올바른 다이어그램 문장을 작성할 수 있도록 도와주는 시스템에 대해 제안하였다.

본 논문의 시스템을 사용하는 과정은 다음과 같다. 개발자는 새로운 다이어그램 개발 환경을 만들기 위해 DML 그래픽 편집기로 새로운 다이어그램 컴포넌트를 정의하고 편집기는 DML 문서를 자동으로 생성한다. 자동으로 생성된 DML 문서는 소스코드 변환기를 통해 Java 또는 C# 등의 언어로 변환되고 다이어그램 시스템에 등록되어 사용 가능하게 된다. 다이어그램 문장에 대한 문법검사는 XML 문서의 구조 정의를 위한 DTD를 이용해 기술하고 다이어그램 시스템은 DTD를 통해 다이어그램 문장의 옳고 틀림에 대한 판단을 하여 사용자에게 메시지를 보여준다. 올바른 문장의 경우 Parsed DAG를 순회하며 각 노드의 semantic action을 수행한다.

본 논문의 다이어그램 시스템에서는 VPL(Visual Programming Language) 이론을 다이어그램 문장 해석 부분에서 이용하였다. 예를 들어 visual sentence를 SRG로 나타내고 이를 기반으로 문법검사를 실행하여 ASG를 구성하는 것 등을 들 수 있다. 하지만 기존 이론이 갖는 복잡함과 다이어그램 컴포넌트 모듈화의 부재는 구현에 어려움을 주었고 사용자의 편의성 또한 고려하지 않은 것이었다. 예를 들어, 상태도의 transition의 표현 시 기존의 visualizing 방법에서는 transition을 구성하는 화살표와 label은 서로 독립된 것으로 간주하였지만 본 논문의 시스템에서는 transition을 화살표와 label로 구성된 하나의 객체로 정의한다. 이렇게 함으로써 다이어그램 컴포넌트에 대한 모듈화가 가능해져 문법 검사와 구현이 쉬워진다.

향후 연구과제로는 제약사항을 직접 기술하는 현재의 불편함을 개선하여 자동화된 방법을 제공하고 여러 개의 다이어그램 컴포넌트를 사용하면서 생기는 복잡한 화면을 자동으로 layout을 지정하는 방법과 추상화(abstraction)시키는 방법에 대해 연구하고자 한다.

참 고 문 헌

[1] Booch, G., Rumbaugh, J. and Jacobson, I., "The Unified Modeling Language User Guide", *Addison-Wesley*, 1999.

[2] Scalable Vector Graphics, http://www.w3.org/Graphics/SVG

[3] Vector Markup Language, http://www.w3.org/TR/NOTE-VML, 1998.

[4] Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/2000/REC-xml-20001006

[5] Simple Object Access Protocol (SOAP) 1.1, 2000. http://www.w3.org/TR/SOAP

[6] XML Query, http://www.w3.org/XML/Query

[7] Chemical Markup Language, http://www.xml-cml.org

[8] Mathematical Markup Language, http://www.w3.org/Math

[9] Mark Minas, "Concept and realization of a diagram editor generator based on hypergraph transformation", *Journal of Science of ComputerProgramming(SCP)*, 2001.

[10] Mark Minas and Oliver Köth, Generating Diagram Editors with DiaGen, *Proc. of the Int'l Workshop with Industrial Relevance*, pp. 433-440, Sep., 1999.

[11] Jukka Paakki, "Attribute Grammar Paradigm -- A High-level Methodology in Language Implementation", *ACM Computing Surveys*, Vol. 27. No. 2, pp. 197-255, 1995.

[12] Extensible Graph Markup and Modeling Language, http://www.cs.rpi.edu/~puninj/XGMML

[13] XSL Transformations, http://www.w3.org/TR/xslt
[14] Document Object Model, http://www.w3.org/DOM
[15] Robert W, "Sebesta, Concepts of programming language", *Addison-Wesley*, p. 122-205, 1999.

김 성 근(Sung keun Kim) 정회원

1993년 숭실대학교 전자계산학과 학사
1995년 숭실대학교 대학원 전자계산학과 석사
1995년~현재 숭실대학교 대학원 컴퓨터학과 박사과정
1998년~현재 가톨릭상지대학 컴퓨터정보계열 조교수
<관심분야> 프로그래밍 언어, XML, 프로그래밍 환경, 에이전트 프로그래밍 언어

김 영 철(Young chul Kim) 정회원

1990년 한남대학교 전자계산학과 학사
1996년 숭실대학교 대학원 전자계산학과 석사
2003년 숭실대학교 대학원 컴퓨터학과 박사
2003년~현재 숭실대정보미디어기술연구소 전임연구원, 명지전문대학 겸임조교수
<관심분야> 프로그래밍 언어, 컴파일러, XML

유 재 우(Chae woo Yoo) 정회원

1976년 숭실대학교 전자계산학과 학사
1985년 한국과학기술원 전산학과 박사
1986년~1987년 코넬대학교 객원교수
1996년~1997년 피츠버그대학교 객원교수
1999년~2000년 한국정보과학회 프로그래밍 언어 연구회 위원장
1983년~현재 숭실대학교 컴퓨터학부 교수
<관심분야> 프로그래밍 언어, 컴파일러