□ 특별기고 □

개념 기술 언어

경북대학교 김상욱*·이춘희**

목 차

Ⅰ. 서 론
Ⅱ. 개념 기술 언어의 개요
2.1 개념 기술언어 시스템
2.2 개념 기술언어의 문법
2.3 시각언어
2.4 시맨틱 네트워크
Ⅳ. 개념 기술 언어의 번역기술
3.1 개념 기술 언어 구조 분석기
3.2 개념 인식기
3.3 개념 추론기
3.4 시각 언어 생성기
Ⅴ. 전 망

Ⅰ. 서 론

이글은 개념 기술 언어(CDL: Concept Description Language)에 대하여 설명한다. 개념 기술 언어는 프로그래밍의 한 방법으로 프로그램을 개발하는 과정에서 인간의 지식을 "개념 구조"로 번역하도록 개념을 기술하는 것이다[2, 3, 4, 5, 12, 20, 23, 24].

기존의 프로그래밍 언어인 C, Pascal, C++와 같은 프로그래밍 언어에서 사용되는 변수, 자료 구조, 문자 등과 같이 기계 구조 중심의 프로그래밍 기법에서 사용하는 방법으로는 개념을 표현하기가 자연스럽지 못하고 부적절하다[2]. 그러나 개념 기술 언어는 개념을 표현하기가 편리할 뿐만 아니라, 지식의 표현이나 프로그래밍도 가능하다. 이와 같은 개념 기술 언어의 프로그래밍 과정이란 컴퓨터에게 개념과 개념 사이의 관계를 지정하여 주는 과정을 의미하므로, 개념 기술 언어에서의 개념 기술은 여러 프로그램들을 생성하는데 사용되어진다[2, 3, 5, 12]

개념을 기술하거나 명세하는데 사용되는 도구나 유사한 언어는 논문[11, 13, 17, 18, 24]에 잘 설명되어 있다.

이 글에서 설명하는 개념 기술 언어는 1990년 "멀티미디어 언어"를 개발하는 과정에서 시작되었는데, 객체 중심 시각 언어의 컴파일러와 이에 관한 여러 도구를 만들면서 개발되어졌다[22, 23, 24]. 여기에서 설명되는 개념 기술 언어는 프로그램을 어떻게 작성하는가 보다는 큰 프로그램의 개발을 위하여 개념 기술 언어로 어떻게 기술하는가와 개념 기술 언어로 된 작은 프로그램이 어떻게 컴파일러에 의하여 번역되어 객체 중심 시각언어로 변환되는가를 설명한다[1,6,8,9, 16,19,24]. 이 글은 개념 기술 언어를 설명하지만 이와같은 개념 기술 언어는 지식 컴파일러 이기도 하다[20, 21].

개념 기술 언어의 목표에서 가장 중요한 목표는 바로 개념을 개념 기술 언어로 명확하게 정의, 표현하는 것인데 지식의 명세를 자연어에 가깝게 표현한다. 또한, 정보의 형식과 상호 관계를 자세하게 구현하지 않아도 처리 가능하여야 하며, 동시에 여러 개념을 구현할 수 있어야 한다.

* 정회원
** 준회원

사용되어질 목적 기계와 목적 언어가 개념 기술 언어와는 독립적이어야 한다.

이 글에서 설명하는 개념 기술 언어의 특징은 큰 대형 프로그램이나 대화형 인터페이스를 쉽게 작성할 수 있으며, 여러 응용 분야에 적용할 수 있다[4, 11, 22, 24]. 여러 프로그래밍 언어나 컴퓨터에 독립적으로 적용 가능하며, 여러 자료 구조, 객체, 지식의 표현이 가능하다. 문법이나 어휘는 자연어에 가까우며, 사용자에 의하여 표현된 개념을 지식 네트워크에서 유사성 정도를 추론하여 의미를 분석한다. 의미가 분석되면 사용자의 개념에 해당하는 객체 중심 시각 언어의 프로그램이 생성되어진다. 또한, C++ 코드를 생성하거나 직접 실행이 가능하다[22, 24].

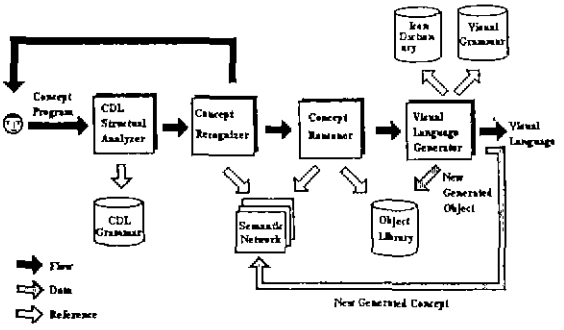
이 글의 제 2장에서는 개념 기술 언어의 전체적인 개요를 설명하고, 제 3장에서는 표현된 개념 기술 언어가 번역되어 객체 중심 시각 언어로 변환 생성되어지는 과정을 설명하고 끝으로 전망을 살펴본다.

II. 개념 기술 언어의 개요

개념 기술 언어는 개념 또는 개념과 개념 사이의 관계에 의하여 구성된 또 다른 개념의 집합으로 구성되어지며, 개념 프로그래밍은 기계 중심의 프로그래밍이 아니라, 인간의 개념으로 프로그래밍하게 된다.

2.1 개념 기술 언어 시스템

개념으로부터 시각 언어를 자동 생성하기 위한 개념 기술 언어 시스템은 그림 1과 같이, 개념 기술 언어 구조 분석기, 개념 인식기, 개념 추론기, 시각 언어 생성기로 구성된다[24].



(그림 1) 개념 기술 언어 시스템 구조

개념 기술 언어를 이용하여 개념 프로그래밍하면, 개념 기술 언어 구조 분석기는 개념 기술 언어의 문법을 참조하여 개념 기술 언어를 구조 분석한다. 개념 인식기는 개념의 의미를 나타내고 있는 시맨틱 네트워크를 참조하여 구조 분석된 개념 프로그램을 시스템이 가지고 있는 범위 내의 지식으로 인식할 수 있는지를 결정한다[12]. 개념 추론기는 시맨틱 네트워크와 객체 라이브러리를 참조하여 개념 프로그램의 의미를 분석하고, 라이브러리에 존재하는 객체의 인자에 맞도록 개념 프로그램에서의 인자를 추론하여 객체 사이의 실행 순서를 결정한다.

2.2 개념 기술 언어의 문법

개념 기술 언어는 특정 범위 내에서의 지식을 자연 언어 형태로 자연스럽게 표현하므로써 인간 중심적인 프로그래밍이 가능하도록 한다.

개념 기술 문법G는 Context-Free Grammar로서 다음과 같이 구성된다.

G_c=(N, T, P, S)
- N: 비 단말 기호의 유한 집합
- T: 단말 기호의 유한 집합
- P: 생성 규칙
- S: 시작 기호

여기에서 생성 규칙 P는 다음과 같다.

1) S→〈Concept〉 '.'S|〈Concept〉'.'
2) 〈Concept〉→〈assertion〉
3) 〈assertion〉→〈command〉
4) 〈command〉→〈verb〉〈attribute〉〈method〉|〈compute〉〈preposition〉〈variable〉
5) 〈compute〉→〈math-function〉〈number-parameter〉
6) 〈verb〉→ε|〈alphabet〉〈verb〉
7) 〈attribute〉→ε|〈noun〉|〈adjective〉
8) 〈method〉→ε|〈expression〉〈method〉|

〈prepositon〉〈method〉|〈variable〉〈method〉

9) 〈math_function〉→add|delete|div|max|min|mod|multiply|sprt

10) 〈prepositon〉→into |to|by|from|at

11) 〈noun〉→ε|〈alphabet〉〈noun〉

12) 〈adjective〉→ε|〈alphabet〉〈adjective〉

13) 〈character〉→'"'〈alphabet〉'"'

14) 〈sequence〉→ε|'"'〈alphanum〉〈sequence〉'"'

...

35) 〈alphabet〉→'a'|…|'z'|…|'A'|…|'Z'

36) 〈numeric〉→'0'|…|'9'

37) 〈operator〉→'+'|'−'|'*'|'/'|'**'

위 개념 기술 언어의 문법은 개념을 나타내는 여러 문장에서 단순 문장으로 제한하여 표현하였는데, 개념 기술 언어의 문법 G에 의하여 다음과 같은 개념 기술 문장이 유도 되어질 수 있다.

puts {10, 5} into S.
pushes S.
pop.

여기에서, "puts {10, 5} into S."는 변수 S에 정수값 {10, 5}를 할당하라는 개념이고, "pushes S."는 S의 데이타를 삽입하라는 개념이며, "pop."은 삽입된 데이타를 삭제하라는 개념이다.

2.3 시각 언어

사용자에 의하여 기술되어진 개념 기술 언어로부터 자동 생성되는 시각 언어는 컴퓨터 화면에 시각적으로 표현되므로써 개념을 자연스럽게 나타낼 수 있다[6, 7, 9, 10, 14, 15, 19].

생성되어지는 시각 언어는 아이콘 사전에 정의되어 있는 아이콘으로 표현되는데, 이 시각 프로그램은 객체 중심 시각 언어로 자동 생성되어진다[24].

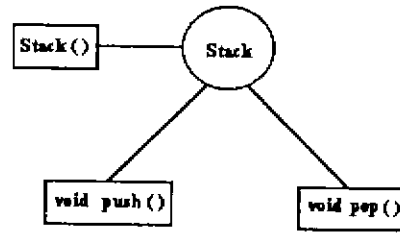
아이콘 사전에 정의되어 있는 아이콘,

{○, ◎, ▭, −, text, →}

들이 관계 연산자,

아이콘이름	물리적이미지	개념
circle	○	[CIRCLE : cx,cy,r]
doublecircle	◎	[DOUBLECIRCLE : cx,cy,r]
text	abc	[TEXT : sx,sy,ex,ey]
trapezium	▱	[TRAPEZIUM : sx,sy,ex,ey]
line	——	[LINE : sx,sy,ex,ey]
arrow	→	[ARROW : sx,sy,ex,ey]

(그림 2) 아이콘 사전



(그림 3) 스택 연산을 위한 객체 중심 시각 프로그램

{connect, contain, overlap}

에 의하여 결합되어 또 다른 아이콘을 생성하게 되고, 이러한 결합이 필요할 때까지 계속 진행되어 최종적으로 시각 프로그램을 생성하게 된다[24]. 예를 들면, 다음과 같은 과정을 거친다.

contain (○, text)에 의하여 object 아이콘이 생성된다.
connect (message_name, argument)에 의하여 message 아이콘이 생성된다.
overlap (object, message)에 의하여 시각 프로그램이 생성된다.

시각 문장을 형성하는 기본 아이콘은 아이콘 사전에 그림 2와 같이 정의되어 있다.

스택에 데이타를 삽입하고 삭제하는 시각 프로그램은 그림 3과 같다.

2.4 시맨틱 네트워크

시맨틱 네트워크는 개념의 의미와 그 개념과 유사한 의미의 개념을 정의하므로써 개념 기술 언어를 융통성 있게 하여 준다[12].

개념의 의미를 나타내는 시맨틱 네트워크는 개념을 나타내는 노드와 개념과 개념사이의 관계를 나타내는 관계 노드로 구성되는데, 시맨틱

네트워크는 객체 라이브러리에 있는 객체를 대상으로 구성된다. 즉, 클래스와 멤버 함수에 대하여 시맨틱 네트워크가 구성된다.

시맨틱 네트워크에서 개념의 의미는 다음과 같은 개념 그래프로 표현된다.

[CONCEPT1]→(RELATION)→[CONCEPT 2]

위의 개념 그래프는 [CONCEPT1],[CONCEPT 2]라는 개념과 개념 사이의 관련성을 나타내는 관계인 (RELATION)으로 구성되는데, 개념 사이를 연결하는 관계는 하나 또는 하나 이상의 링크로 연결되어 있다[12].

시맨틱 네트워크에서 개념과 개념사이를 연결하는 관계는 Is, present_Tense, Past_Tense, Synonymy, Synonymy_Degree, Attribute, Class의 7개로 구성되는데 각각의 특성은 다음과 같다.

Is: 개념이 Class, method, Assigenmet 혹은 라이브러리에 존재하지 않는지를 정의하며, [Concept]→(Is)→[Class or method or Assignment, Not_Exist]와 같이 쓰여진다.

Present_Tense: 개념의 현재형을 정의하며, [Concept]→(Present_Tense)→[Present_Concept]과 같이 쓰여진다.

Past_Tense: 개념의 과거형을 정의하며, [Concept]→(Past_Tense)→[Past_Concept]과 같이 쓰여진다.

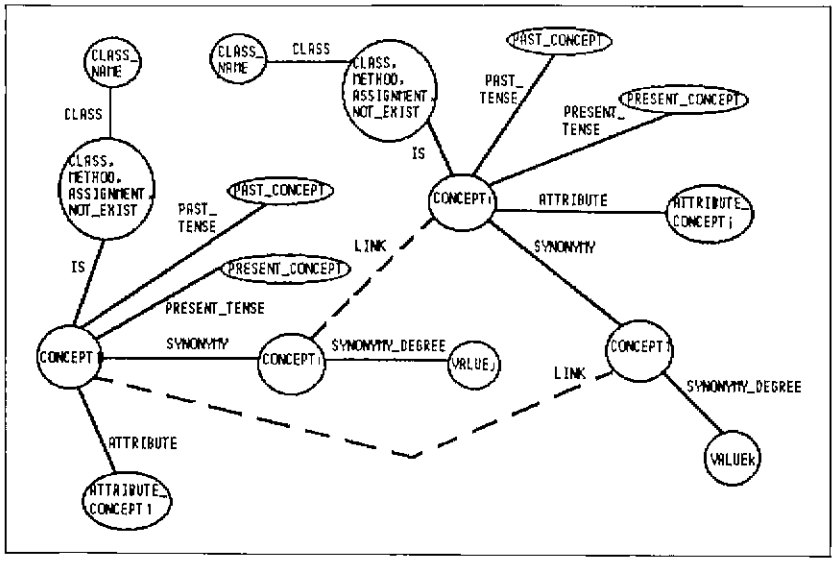
Synonymy: 사용자의 개념과 유사한 의미를 가지고 있는 유사 개념을 정의하며, [Concept]→(Synonymy)→[Synonymy_Concept]과 같이 쓰여진다.

Synonymy_Degree: 개념과 유사 개념 사이의 유사성 정도를 정의하며, [Synonymy_Concept]→(Synonymy_Degree)→[Value]와 같이 쓰여진다.

Attribute: 사용자의 개념에 필요한 속성을 정의하며, [Concept]→(Attribute)→[Attribute_Concept]과 같이 쓰여진다.

Class: 개념이 Method인 경우 Method가 속한 Class를 정의하며, [Method]→(Class)→[Class_Name]과 같이 쓰여진다.

시맨틱 네트워크는 개념의 의미를 정의하므로써, 개념으로부터 프로그램이 가능하게 하여 준다. 관계 노드 Present_Tense, Past_Tense는 개념의 현재형과 과거형을 정의하고 있으며, Synonymy는 개념과 유사한 의미의 개념을 정의하므로써, 사용자에 의하여 정의된 개념과 똑같



(그림 4) 시맨틱 네트워크

은 개념이 시맨틱 네트워크에 존재하지 않을 때는 Synonymy_Degree에 의하여 연결된 유사도가 가장 높은 개념으로 대치하므로써, 개념 기술 언어를 융통성있게 하여 준다. 관계 노드 Attribute는 개념의 속성을 정의하므로써 개념의 의미를 더 정확하게 정의하게 하고, 관계 노드 Is는 개념이 객체 라이브러리에 어떤 형태로 존재하는지를 정의하고 있으며, 개념이 멤버 함수인 경우는 Class관계에 의하여 개념이 속한 클래스를 정의하여 주므로써, 시맨틱 네트워크에 있는 개념과 라이브러리에 있는 객체를 연겨시켜 주는 기능을 한다.

그림 4의 시맨틱 네트워크에서 CONCEPTI의 CONCEPTi는 SYNONYMY 관계에 의하여 연결되어 있다. LINK는 관계 노드가 아니라, 시맨틱 네트워크에서 CONCEPTi의 의미를 정의하고 있는 위치를 가리키는 포인터이다. CONCEPTi와 연결된 LINK를 따라 가면 시맨틱 네트워크에서 CONCEPTi에 대하여 정의되어 있는 의미를 즉시 알 수 있다.

III. 개념 기술 언어의 번역 기술

개념 기술 언어를 위한 번역기의 주요 모듈은 개념 기술 언어 구조 분석기, 개념 인식기, 개념 추론기, 시각 언어 생성기가 있는데, 전체적인 번역 절차는 그림 5와 같다.

```
CDL-TRANSLATOR의 번역 절차:
  입력: CDL 문장
  출력: 객체 중심 시각 언어
개념 기술 언어 구조 분석기
  {
    개념 문법에 의한 개념 기술 문장 구조의
    분석.
  }
개념 인식기
  {
    입력된 개념에 대하여 시맨틱 네트워크
    검색.
    만약, 같은 개념이 존재하지 않는다면 시
    맨틱네트워크에서 가장 유사도가 높은
    유사개념으로 대치.
    지식 베이스에 인식된 개념 저장.
  }
    인식되지 않을 경우는 기술언어 구조 분
    석부터 반복함
개념 추론기
  {
    할당문인 경우 변수형 추론.
    객체 라이브러리에서 개념과 일치하는
    객체 검색.
    객체의 실행 순서 추론.
    객체에 인자형과 일치하도록 개념의 인
    자형 조정.
    지식 베이스에 추론된 개념의 정보 저장.
  }
시각 언어 생성기
  {
    아이콘 사전과 시각 문법을 참조하면서
    시각 언어 생성.
    컴퓨터 화면에서 위치 결정.
    객체 중심 시각 프로그램 생성.
  }
```

(그림 5) 개념 기술 언어의 번역 절차

3.1 개념 기술 언어 구조 분석기

개념 프로그램이 입력되면 개념 기술 언어 구조 분석기는 입력된 개념 프로그램을 각각의 개념 단위로 분리한다. 그리고, 정의된 개념 문법 G를 참조하면서 개념 문장이 문법에 맞는가를 분석한다.

예를 들어, "puts {10, 5} into S."가 입력되었을 때, 개념 기술 언어 구조 분석기는 개념 문장을 각 토큰 단위로 분리하여 개념 문법에 맞는지를 구조 분석한다. 문법에 맞는 문장인 경우 CORRECT 메시지를 보내고, 문법에 맞지 않는 문장인 경우는 SYNTAX ERROR를 발생한다.

입력: puts {10, 5} into S.
동작: 'puts' '{' '10', ',', '5'} 'into' 'S',

∴
→개념 문법을 참조하여 구조분석.
출력:CORRECT 또는 SYNTAX ERROR

3.2 개념 인식기

구조 분석된 개념 문장이 입력되면 개념 인식기는 개념의 의미를 나타내고 있는 시맨틱 네트워크를 참조하여 시스템이 가지고 있는 지식으로 인식할 수 있는가를 결정한다. 시스템에서 사용자의 개념과 같은 개념을 시맨틱 네트워크에서 찾을 수 없을 때는 시맨틱 네트워크에 존재하는 유사한 의미의 개념을 검색하여 유사도가 가장 높은 개념을 이용한다. 만약, 사용자의 개념이 시맨틱 네트워크에 존재하지 않거나, 유사한 개념도 존재하지 않을 때는 그와 같은 개념을 인식할 수 없다고 판단하고, 개념 기술 언어로 새로 작성하게 한다.

예를 들어, 개념 프로그램 "puts {10, 5} into S."가 입력되었을 때, 개념 인식기는 먼저 시맨틱 네트워크에서 "puts"를 검색한 후, "puts"가 "put"의 현재형임과 변수를 할당하는 기능임을 인식한다.

입력: first_concept←"puts"
동작: Search_Semantic_Network: [put]
→(Present_Tense)→[puts]
CONCEPT←"put"
출력: Knowledge_Base←"put"
RECOGNIZE

또한 개념 프로그램 "pushes S."가 입력되었을 때, "pushes"는 "push"의 현재형임과 Stack 클래스의 Method임을 인식한다. 인식된 개념은 Knowledge_Base에 저장하고 인식되었다는 "RECOGNIZE"라는 메세지를 개념 추론기에게 전하여 준다.

입력: first_concept←"pushes"
동작: Search_Semantic_Network: [push]
→(present_tense)→(pushes)
CONCEPT—"put"
Is_RELATION: [push]→(Is)→[METHOD]→[THEME]→[Stack]
출력: Knowledge_Base←"push"
RECOGNIZE

3.3 개념 추론기

개념 인식기에 의하여 의미가 인식된 개념에 대하여 속성이 정의되어 있는 경우 개념과 속성을 조합하여 문장 전체의 의미를 추론한다. 만약, 변수에 값을 할당하는 문장인 경우는 먼저 변수형을 결정한 후 값을 할당한다. 또한, 개념에 대한 라이브러리의 객체를 검색하여 그 객체가 필요로 하는 정보와 일치하도록 개념을 조정한다. 그리고, 객체의 멤버 함수들에 대하여 위상 정렬방법을 이용하여 실행 순서를 결정한다.

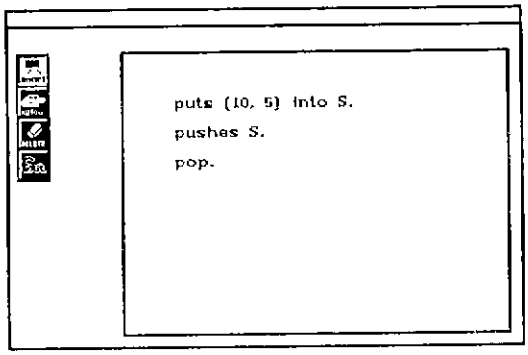
예를 들어, 입력된 개념문장 : "puts {10, 5} into S."에서 개념 인식기가 "puts"의 의미를 인식한 후에 개념 추론기는 "{10, 5} into S."에서 {10, 5}의 값을 저장할 수 있는 S에 대한 변수형을 추론하여 결정한 후 S에 {10, 5}를 할당하고, 이를 Knowledge_Base에 저장한다.

입력: Syntax←"{10, 5}."
동작: CONCEPT←"put"
S←{10, 5}
Knowledge_Base←S

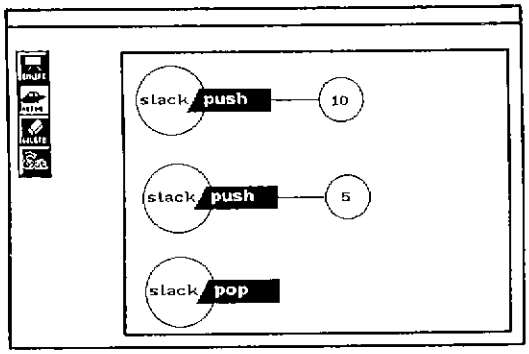
예를 들어, 객체 라이브러리에 다음과 같은 객체가 존재한다고 가정한다.

```
class Stack{
        Stack() {…};
        push(int) {…};
        pop() {…};
    }
```

개념 문장 "pushes S."에서 개념 인식기가 "pushes"의 의미를 인식한 후에 개념 추론기는 객체 라이브러리에서 Stack 클래스를 검색한다. Stack 클래스의 Construct Method부터 실행 순서를 추론한다. Stack 클래스에서 실행 순서는 Stack, push, pop의 각 멤버 함수가 서로 종속되지 않고 독립적으로 실행되므로 멤버 함수 push에 대한 인자형을 검색하여, 개념 문장의



(그림 6) 입력된 개념 프로그램



(그림 7) 생성된 객체 중심 시각 프로그램

인자형과 일치하도록 조정한다. 이 때, 멤버 함수 push의 인자형은 정수형이므로, {10, 5}를 10, 5로 각각 분리한 후 Knowledge_Base에 저장한다.

입력: Syntax←"S."
동작: CONCEPT←"push"
CLASS←"Stack"
METHOD→"push"
Construct_Method←"Stack"
Priority of Method← —Stack
—push
—pop
METHOD's parameter type←int
Adjust CONCEPT's parameter←10, 5

3.4 시각 언어 생성기

개념 기술 언어 구조 분석기, 개념 인식기, 개념 추론기에 의하여 의미 분석된 개념을 정의된 시각 문법과 아이콘 사전을 참조하여 시각 언어를 생성한다. 예를 들어, 사용자가 개념 기술 언어를 이용하여 스택에 {10, 5}를 삽입한 후, 스택에서 하나의 데이타를 삭제하는 개념 프로그램은 그림 6과 같다.

위와 같은 개념 프로그램의 의미가 분석된 후, 시각 언어 생성기에 의하여 그림 7과 같은 시각 프로그램이 자동 생성된다.

개념 기술 언어로 개념을 정의하면 그 개념의 관계를 이해하기 쉬운 객체 중심 시각 언어로 자동 생성할 수 있는데, 이는 매우 복잡한 추론 과정을 거치게 된다. 자동 생성되어진 객체 중심 시각 프로그램은 C++ 코드를 생성하거나 직접 실행하게 된다.

IV. 전 망

지금까지 객체 중심 시각 시스템에서 사용되어질 개념 기술 언어에 관하여 설명하였으나 아직 많은 부분이 연구되어져야 한다. 이 장에서는 응용 분야와 전망에 대하여 설명한다.

■ 프로그래밍 언어

개념 기술 언어는 일종의 프로그래밍 언어이므로 기존의 프로그래밍 언어와 다르게 개념 중심으로 프로그래밍하는 프로그래밍 언어의 새로운 분야로 발전하였으며 다음과 같은 특성을 가진다.

(1) 영역 수준의 개념을 지원하고 독립성을 가진다.
(2) 개념이 번역되면서 새로운 개념은 학습되어진다.
(3) 프로그래밍에서 개념의 재사용이 가능하다.

개념 기술 언어가 실행되기 위하여 필요한 지식은 개념 기술 언어로 직접 프로그래밍 하거나, 이미 만들어진 개념이나 객체를 재사용하여 프로그래밍 할 수 있다.

■ 번역에서의 문제

번역에서는 보다 많은 문제와 연구 이슈가 있는데, 주요 논점은 개념 기술 언어의 번역시에 컴파일의 정확한 정의와 그 의미를 정의할 수 있어야 하며, 입력 개념의 구조와 목적 개념의 정확한 구조를 정확히 알아야 한다. 개념 기술 언어로 과연 개념을 얼마나 정확히 기술할 수 있을지도 의문이다. 번역 과정은 입력되는 개념에 중심을 두는지, 목적 개념에 중심을 두는지도 주요 문제이다. 실행시에 제공되는 서비스도 연구되어야 한다.

■ 자연어

개념 기술 언어는 자연어와 매우 흡사하다. 개념 기술 언어에서 사용하는 개념은 백과 사전이나 단어 사전에서 사용하는 개념이나 의미가 비슷하다. 또한, 개념 기술 언어에서 표현하는 구문 구조는 자연어의 구문 구조와 비슷하므로 개념 기술 언어는 자연어를 대신하여 사용하는 자연어와 컴퓨터 사이의 중간 형태의 언어 표현이라고 할 수 있다. 이와 같이 자연어의 중간 형태로 표현 가능하다는 것은 다른 중간 형태 표현 언어들에 비하여 매우 중요한 장점이 된다. 결국, 프로그래밍 언어로 사용되기 위하여 지식을 가지고 있는 제한된 자연어를 사용한다는 것도 가능하다는 것이다.

■ 다른 언어의 매개 언어

개념 기술 언어 프로그램은 인간의 개념에 의하여만 구성되어지고 있다. 이러한 관점에서, 다른 프로그래밍 언어 보다 월등히 자연어에 가깝고, 지식을 표현하는데 효과적이다. 그러므로 사전의 내용을 구성하거나 기술하기가 쉽고, 개념 기술 언어에 또다른 문장을 추가하기가 쉽다. 그러므로, 개념 기술 언어는 직접 프로그래밍 언어로 사용할 수도 있는 매개 언어가 될 수 있도록 연구되어야 한다.

■ 지식 표현

지식을 어떻게 표현하는가는 매우 중요한 문제이지만, 한 방법은 개념 그래프를 사용할 수 있다. 이때 개념과 개념 사이의 관계를 기술할 수 있는데, 이러한 기술 방법에 의하여 지식을 표현하는 언어로 효과적으로 사용할 수 있을 것이다. 즉, 개념 기술 언어는 개념이나 지식을 문장의 형식으로 표현 가능하다. 개념 기술 언어로 지식을 표현하면 시스템은 지식을 어떻게 사용하는지를 자동적으로 알게 된다. 그러므로 이미 만들어진 지식을 재사용함으로써 더욱 큰 시스템의 지식을 구축할 수 있다. 결국, 중복된 지식을 사용하지 않고 필요한 지식만 사용할 수 있다. 개념 기술 언어로 표현되는 지식은 자연어로 표현된 지식과 유사하기 때문에 인간과 인간의 상호 작용에 의한 개념과 지식을 전달하는 수준과 비슷하게 된다. 그러므로, 이미 정의된 개념과 관계를 사용하여 쉽게 지식을 기술할 수 있다.

■ 지식 베이스의 관리

개념 기술 언어 시스템은 개념 프로그램에서 사용할 모든 지식을 재사용할 수 있어야 한다. 그러나, 다른 사용자에 의하여 사용되어질 때는 일치하지 않을 수도 있기 때문에 재사용되어질 지식이나 객체를 효과적으로 관리할 수 있도록 하여야 한다. 즉, 객체나 지식이 일치하는가를 검사하고 지식 베이스를 효율적으로 재구성하는 방법이 연구되어야 한다.

■ 멀티미디어

기존 멀티미디어는 객체 중심 프로그래밍 언어의 추상화 자료형을 사용하여 멀티미디어인 여러 객체를 다루었다. 이때, 멀티미디어는 컴퓨터 내부에서는 여러 객체의 유기적인 결합에 의하여 관리되어지고 있다. 그러나 그래픽 객체를 취급하는데 한계가 있으며, 응용 프로그램을 분리하기 힘들고, 편집 기능이 약하다. 중요한 문제는 컴파일러가 없어서 시맨틱을 파악하지 못한다. 존재하는 객체가 무엇인지를 사용자가 알고 있어야만 사용할 수 있고, 사용된 함수도 의미를 파악하는 것이 아니라 매칭에 의하여 비교한다. 즉, 멀티미디어를 위한 편집과 프로그래밍

은 매우 제한적이었다. 그러나, 개념 기술 언어는 자연어에 가깝기 때문에 의미 분석을 통하고 객체를 관리하여 효과적으로 멀티미디어 객체를 기술할 수 있도록 프로그래밍할 수 있다.

■ 시각 언어

시각 언어는 인간의 지식과 개념을 효과적으로 나타낼 수 있으므로, 시각 언어로 개념의 구조를 변환하므로써 개념을 이해하기 쉽게할 뿐 아니라, 정확성을 나타내게 한다. 시각 언어로 번역하는 문제는 매우 복잡하면서도 많은 추론 과정을 거쳐야 하기 때문에 다양한 연구가 필요하다.

■ 화면 표현

컴퓨터 화면에 시각 언어가 표현되어질 때, 제한된 크기의 화면에 시각 프로그램을 효과적으로 표현할 수 있어야 한다. 또한 생성된 시각 프로그램이 정확하여야 하며, 처리 절차가 정확하여야 한다. 입력된 개념 프로그램의 개념이 정확하게 시각 언어의 의미로 전달되어야 한다. 이 외에도 개념의 애매성을 효과적으로 표현할 수 있도록 연구되어야 한다.

■ 개념 기술 언어에 의한 상호 작용

개념 기술 언어로 작성된 프로그램은 개념을 포함하고 있기 때문에 개념 기술 언어를 사용하는 객체 중심 시각 시스템은 이 개념 기술 언어를 통하여 사용자와 대화하므로써, 정보를 표현한다. 그러므로 개념 언어를 변환하는 기술이 더욱 절실히 필요하게 된다. 개념 기술 언어를 통한 인간과 컴퓨터의 상호 작용은 지식을 서로 교환하는 도구가 되므로, 이러한 시스템에서는 개념 기술 언어를 통한 훌륭한 대화형 지식 시스템이 되도록 하여야 한다.

■ 개념의 한글 표현

개념을 표현하는 방법은 다양하지만 개념을 표현할 때는 어떤 지식과 함께 문화도 고려되지 않으면 정확한 개념이 전달되지 않을 수도 있다. 개념을 문화적인 특성에 알맞게 기술한다는 것은 매우 어려운 문제이지만 문화적인 특성에 알맞게 기술되어야 한다. 그러므로, 우리 문화에 알맞은 개념의 기술은 한글로 개념을 기술하는 방법을 연구하여야 할 것이다.

컴퓨터를 사용하는데에 있어서 사용자와 친숙한 언어를 사용하는 것은 매우 중요한 일이다. 자연어를 사용하는 것은 매우 제한적이기 때문에 특정 범위와 영역에서는 자연어와 유사한 개념 기술 언어를 사용하여 프로그래밍 환경을 변화시키면, 인간과 컴퓨터의 상호 작용을 원활히 하는 것이다. 사용자는 컴퓨터 중심의 개념과 프로그래밍 방법을 배우는데 힘을 기울일 필요없이 사용자의 개념과 사고의 방법으로 프로그래밍 환경을 변화하도록 하여야 한다. 그러므로 차세대 컴퓨터의 프로그래밍 환경으로서 효과적인 개념 기술 언어와 컴파일러 기술을 연구하여야 한다.

참 고 문 헌

1. Anthony I. Wasserman, "Extending State Transition Diagrams for the Specification of Human-Computer Interaction," IEEE Transactions on Software Engineering, Vol. 11, No. 8, 1985, pp. 699~713.
2. Ashok K. Goel, "Knowledge Compilation: A Symposium," IEEE Expert, Vol. 6, No. 2, Apr. 1991. pp. 71~73.
3. B. Chandrasekaran, "Models versus Rules, Deep versus Compiled, Content versus Form: Some Distinctions in Knowledge Systems Research," IEEE Expert, Vol. 6, No. 2, Apr. 1991, pp. 75~79
4. Cheng-xiang Zhai, "Preliminary ideas of a Conceptual Programming Language," ACM SIGPLAN Notices, Vol. 25, No. 12, Dec. 1990, pp. 93~100.
5. Chris Tong, "The Nature and Significance of Knowledge Compilation," IEEE Expert, Vol. 6, No. 2, Apr. 1991, pp. 88~91
6. Claudia Crimi, Angela Guercio, Giuliano Pacini, Genoveffa Tortora, Maurizio Tucci, "Automating

Visual Language Generation," IEEE Transactions on Software Engineering, Vol. 16, No. 10, Oct. 1990, pp. 1122~1135.
7. Danny Goodman, The Complete HyperCard Handbook, Bantam Books, 1988, pp. 875.
8. Ephraim P. Glinert, Jakob Gonczarowski, "A (Formal) Model for (iconic) Programming Environments," Visual Programming Environments: Applications and Issues, pp. 295~502.
9. Eric J. Golin, Steven P. Reiss, "The Specification of Visual Languages Syntax," Journal of Visual Languages and Computing, 1990, pp. 141~157.
10. Genoveffa Tortora, Paolo Leoncini, "A Model for the Specification and Interpretation of Visual Languages," IEEE Proceedings Workshop on Visual Languages, 1988, pp. 52~60
11. James Martin, Fourth-Generation Languages, Prentice-Hall, 1985, pp. 420.
12. John F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley Publishing Company, 1983, pp. 481.
13. John R. Nestor, Joseph M. Newcomer, Paola Giannini, Donald L. Stone, IDL: The Language and Its Implementation. Prentice Hall, 1990, pp. 560.
14. Masahito Hirakawa, Minoru Tanaka, Tadao Ichikawa, "An Iconic Programming System, Hi-Visual," IEEE Transactions on Software Engineering, Vol. 16, No. 10, Oct. 1990, pp. 1178~1184.
15. P. Reisner, "Formal Grammar and Human Factors Design of an Interactive Graphics System," IEEE Transactions on Software Engineering, Vol. 7, No. 2, Mar. 1981, pp. 229~240.
16. Robert J. K. Jacob, "Using Formal Specifications in the Design of a Human-Computer Interface," Communications of the ACM, Vol. 26, No. 4, Apr. 1983, pp. 259~264.
17. Ramin Yasdi, "Learning Classification Rules from Database in the Context of Knowledge Acquisition and Representation," IEEE Transactions on Knowledge and Data Engineering, Vol. 3, No. 3, Sep. 1991, pp. 293~306.
18. Rolf Bahlke, Gregor Snelting, "The PSG System: from Formal Language Definitions to Interactive Programming Environments," ACM Transactions on Programming Languages and System, Vol. 8, No. 4, Oct. 1986, pp. 547~576.
19. Shi-Kuo Chang, "A Visual Language Compiler," IEEE Transactions on Software Engineering, Vol. 15, No. 5, May. 1989, pp. 506~524.
20. Tomas G. Dietterich, "Bridging the Gap Between Specification and Implementation," IEEE Expert, Vol. 6, No. 2, Apr. 1991, pp. 80~82.
21. Tom Bylander, "A Simple Model of Knowledge Compilation," IEEE Expert, Vol. 6, No. 2, Apr. 1991, pp. 73~74.
22. 김상욱, 이미선, 박지은, "멀티미디어 언어의 개발", 학술발표논문집, 한국정보과학회, 18권 1호, 1991, 4, pp. 169~172.
23. 김상욱, 이춘희, "무엇이 객체 중심 프로그래밍 환경인가?", '92 한국인지과학회 춘계학술대회논문집, 1992, 5, pp. 171~183.
24. 김상욱, 이춘희, "개념으로부터 객체 중심 시각 프로그램의 자동 생성", '93 한국인지과학회 춘계학술대회논문집, 1993, 5, pp. 163~174

김 상 욱

1979 경북대학교에서 전자계산학으로 학사학위 취득
1981 서울대학교에서 전자계산학으로 석사학위 취득
1989 서울대학교에서 전자계산학으로 박사학위 취득
1988 ~현재 경북대학교 전자계산학과 부교수로 재직중
관심분야 : 컴퓨터 언어, 객체 중심 컴퓨팅, 시각 언어 멀티어와 지식처리

이 춘 희

1992 경북대학교에서 전자계산학으로 이학사 학위 취득.
1992 ~현재 경북대학교 전자계산학과 석사 과정
관심분야 : 컴퓨터 언어, 시각 및 멀티미디어 언어, 지식 처리, 컴파일러