시각 프로그램 편집을 위한 그래픽 토큰 편집기

박영조*, 최종명**, 유재우*
*숭실대학교 컴퓨터학과
**국립목포대학교 컴퓨터공학과
yjpark@ss.ssu.ac.kr, jmchoi@mokpo.ac.kr, cwyoo@comp.ssu.ac.kr

A Graphical Token Editor for editing the visual program

Young-Jo Park*, Jong-Myoung Choi**, Chae-Woo Yoo*
*Dept. of Computing, Soongsil Univ.
**Computer Engineering, Mokpo National Univ.

요 약

현재 GUI 인터페이스와 컴퓨터 기술의 발전으로 다양한 형태의 그래픽 모델링 도구들과 시각 프로그래밍 시스템들이 사용되고 있다. 이러한 시스템들을 개발하기 위해서는 사용자가 그래픽 요소들의 형태와 그래픽 요소들간의 연결 관계를 개발자가 직접 프로그램을 이용해서 작성하여야 하기 때문에 많은 시간과 노력을 필요로 한다. 이러한 문제를 해결하기 위해서 본 논문에서는 그래픽 요소의 형태와 그래픽 요소들간의 연결성을 직접 조작 방식을 이용해서 기술할 수 있는 토큰 편집기인 TEdi(Token Editor)와 시각 프로그래밍의 기본적인 기능을 지원할 수 있는 AVE(Abstract Visual Editor)를 소개한다. TEdi는 그래픽 요소들을 정의하기 위해서 사용되며, 토큰 단위의 각 그래픽 요소는 다른 그래픽 요소와 연결되기 위해서 연결자를 사용한다. 연결자는 그래픽 편집기에서 연결될 수 있는 그래픽 요소를 제한함으로써 시각 프로그래밍 혹은 모델링에서 구문 오류를 줄여줄 수 있다. 본 논문에서 제공하는 시스템을 사용하는 경우에 사용자는 보다 쉽게 그래픽 요소와 시각 프로그래밍 혹은 모델링 도구를 작성할 수 있을 것이다.

1. 서론

현재 다양한 시각 프로그래밍 시스템과 다이어그램을 이용한 모델링 도구들이 많이 활용되고 있다. 사용자는 이러한 시스템들의 그래픽 요소들을 이용해서 자신의 의도와 아이디어를 시각적인 형태로 표현한다. 각 그래픽 요소는 사전에 정의된 그래픽 형태, 의미 정보, 그래픽 요소들 간의 연결 정보를 가지고 있으며, 사용자는 이러한 정보들을 이용해서 모델링을 수행하거나 프로그램을 작성한다.

많은 모델링 도구 혹은 시각 프로그래밍 시스템들은 그래픽 요소들을 연결할 때 문법 구조를 고려하지 않거나 혹은 기본적인 형태의 제약 사항만을 체크하기 때문에 구문 오류가 발생할 수 있다는 단점을 가지고 있다. 이러한 단점을 극복하기 위해서 본 논문은 모델링 도구 혹은 시각 프로그래밍 시스템의 각 그래픽 요소들에 연결자라는 개념을 추가함으로써 구문 오류를 줄여줄 수 있는 AViPS(Abstract Visual Programming System) 시스템을 제안한다. 이 시스템은 각 시각 프로그래밍 언어별로 그래픽 요소를 정의할 수 있는 TEdi(Token Editor)와 생성된 그래픽 요소 활용에 필요한 기능을 제공하는 AVE(Abstract Visual Editor)로 구성된다. TEdi는 시각 프로그래밍에서 사용될 수 있는 그래픽 토큰을 직접 조작 방식을 이용해서 정의할 수 있으며, 다른 토큰들간의 연결 관계를 연결자를 이용해서 정의할 수 있다. AVE는 시각 프로그래밍 시스템의 편집기를 위한 편집, 이동, 복사, 삭제 등의 기본적인 기능들을 제공한다. 따라서 시스템 개발자는 AVE를 확장함으로써 원하는 편집기를 보다 쉽게 개발할 수 있다.

본 논문의 2장에서는 시각 프로그래밍 언어와 편

집기에 대해서 기술하고, 3장과 4장에서는 시스템 설계와 실험에 대해서 설명한다. 마지막으로 5장에서는 결론과 향후 연구에 대해서 논한다.

2. 관련연구

시각 프로그래밍 언어는 사용자가 시각적인 요소를 사용해서 프로그래밍을 할 수 있는 언어이다[1]. 시각 프로그래밍 언어는 텍스트 기반 언어들에 비해서 생산성이 높고, 배우기 쉬운 장점을 갖는다[2].

텍스트 편집기가 모든 텍스트 기반의 프로그래밍 언어를 만족시키는 반면에, 모든 시각 프로그래밍 언어를 만족하는 범용적인 시각 프로그래밍 편집기는 다음과 같은 이유 때문에 개발하기 어렵다. 첫째, 시각적으로 나타나는 그래픽 원시 요소들의 집합만으로 범용적인 편집기에서 모든 시각 프로그래밍 언어에 대한 프로그래밍은 한계가 있다. 간단한 시각 프로그래밍은 가능하지만, 통합된 요소가 요구되는 경우에는 시각 프로그래밍이 불가능하다. 둘째, 편집기에서 제공하는 원시 요소들의 집합이 너무 포괄적이거나, 집합 내에서 프로그램 안에는 포함되지 말아야 하는 그래픽 요소가 포함될 수 있다. 셋째, 일반적인 그래픽 편집기는 의미 정보의 기술 및 활용 방법을 지원하지 않는다.

범용적인 시각 프로그래밍 편집기의 장점은 매번 새로운 시각 프로그래밍 언어에 대해서 새로운 편집기가 요구되지 않는다. 이러한 편집기의 예로는 Palette[3]가 있다. 이 편집기의 특징은 시각 프로그래밍 편집기를 만들기 위해서 일반적인 그래픽 편집기를 생성하고 각 시각 프로그래밍 언어에 알맞도록 변화시킬 수 있는 점에 있다.[3].

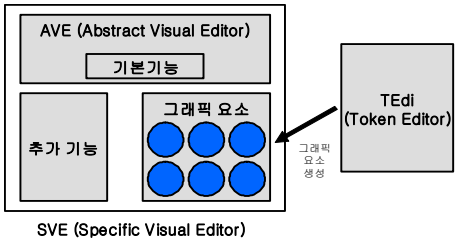
시각 프로그래밍의 다른 형태로는 시각 구문지향 편집기 생성시스템[4]이 있다. 이 시스템은 시각 프로그래밍 언어에 대해서 그들의 명세에 알맞은 객체지향적이고, 구문지향적인 편집기를 생성해주는 시스템이다. 이 시스템은 크게 두 부분으로 구성되며, 명세언어와 생성 시스템으로 나누어진다. 명세언어는 시각 프로그래밍 언어에 대해서 구문과 의미를 정의한다. 생성 시스템은 명세언어를 입력으로 받아 구문지향 편집기를 생성한다[4].

3. 설계

3.1 전체 시스템 설계

AViPS 시스템은 시각 프로그래밍 언어의 특성을 부여하기 위한 그래픽 요소를 생성하는 TEdi와 AVE로 구성된다. TEdi는 특화된 그래픽 요소 및 연결자를 정의하고 생성할 수 있다. AVE는 TEdi에서 생성된 그래픽 요소를 나타내기 위한 편집기의 기본 기능을 제공한다. AVE는 그래픽 요소를 올바르게 표현하는 기능과 각 그래픽 요소간의 연결 관계를 표현하는 기능을 지원한다.

사용자는 AViPS를 이용해서 AVE를 상속받은 각 시각 프로그래밍 언어에 특화된 SVE(Specific Visual Editor)를 생성할 수 있다. 따라서 SVE는 특화된 그래픽 요소를 사용할 수 있고, 그래픽 요소들간의 연결성을 표현할 수 있다.

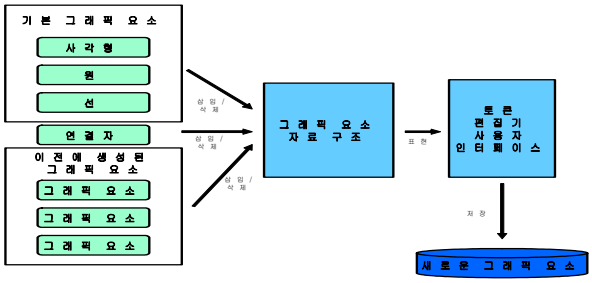


(그림 1) AViPS 구조

3.2 TEdi(Token Editor) 설계

AViPS는 각 시각 프로그래밍 언어에 특화된 그래픽 요소를 지원하기 위한 TEdi를 포함한다. TEdi는 모든 시각 프로그래밍 언어에 포함될 수 있는 기본적인 그래픽 요소를 갖고 있다. 또한, TEdi는 기본적인 그래픽 요소를 결합한 새로운 그래픽 요소를 생성할 수 있다.

TEdi는 그래픽 요소 내에 사용할 연결자를 정의하고, 연결이 허용되는 그래픽 요소의 형태를 연결자에 등록한다. 각 그래픽 요소는 연결자를 이용해서 다른 그래픽 요소들과 연결 된다.



(그림 2) TEdi 구조

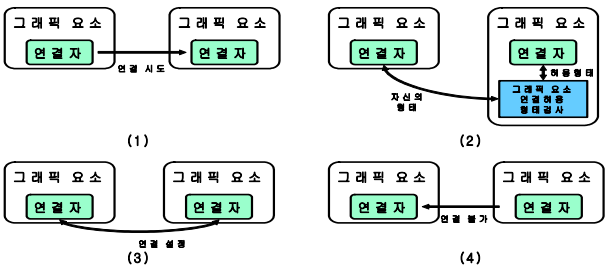
3.3 연결자 설계

많은 모델링 도구 혹은 시각 프로그래밍 시스템들이 그래픽 요소들을 연결할 때 문법 구조를 고려하

지 않거나 기본적인 제약사항만은 검사하기 때문에 구문 오류가 발생할 수 있는 단점을 가진다. 따라서, 본 논문에서 제안한 추상적인 시각 프로그래밍 시스템은 그래픽 요소 간의 관계를 설정하기 위해서 연결자를 사용한다. 연결자는 TEdi에서 생성할 때 연결 가능한 그래픽 요소 형태를 등록한다. 등록된 그래픽 요소 형태는 SVE에서 그래픽 요소간의 관계를 설정할 때 연결 허용 형태 검사를 통해서 구문 오류를 줄이는 데 사용한다.

그래픽 요소는 다중의 연결자를 가지고 다른 그래픽 요소와 연결된다. 연결 과정을 살펴보면 그림 3와 같은 절차로 그래픽 요소 간의 연결이 이루어진다. 그림 3의 (1)에서 그래픽 요소는 다른 그래픽 요소와 연결하기 위해서 연결 요청을 한다. 그림 3의 (2)에서 요청받은 그래픽 요소는 요청된 그래픽 요소 형태에 대한 그래픽 요소 연결 허용 형태 검사를 수행한다. 이 검사는 연결을 요청한 그래픽 요소의 형태와 다른 그래픽 요소의 연결자의 허용목록을 이용해서 연결이 허용 되는지 여부를 검사한다.

그림 3의 (3)과 (4)는 연결 설정과 연결 불가를 나타낸다. 그림 3의 (3)의 연결 설정은 SVE에서 두 그래픽 요소간의 연결 관계가 있음을 나타낸다. 그림 3의 (4)에서 연결 요청을 받은 그래픽 요소는 연결자에 요청한 그래픽 요소의 형태가 등록되어 있지 않기 때문에 연결 불가 메시지를 요청한 그래픽 요소에 전송한다.



(그림 3) 그래픽 요소 간의 연결

SVE에서는 그림 4의 (2)에서 나타낸 바와 같이 그래픽 요소 허용 형태 검사를 이용해서 연결 검사를 수행하며, 연결 검사 알고리즘은 표 1에서와 나타낸 형태로 진행한다.

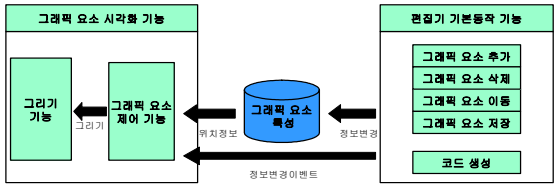
<표 1> 그래픽 요소 허용 형태 검사 알고리즘

```
graphical element A, B;
A.requestConnection(B);
if B.hasType(A)
   then B.setConnection(A);
   else  B.sendMessage(A, "Not Connectable");
```

3.4 AVE(Abstract Visual Editor) 설계

AVE는 각 시각 프로그래밍에서 사용하는 공통적인 기능을 제공하는 편집기이다. 이 편집기는 시각 프로그래밍에 필요한 그래픽 요소를 표현하고, 그래픽 요소 간의 설정된 연결을 표현하는 기능을 갖는다. 또한, 그래픽 요소를 관리와 제어를 위한 기능을 갖고 있다.

그림 4와 같이 AVE는 그래픽 요소 시각화 기능과 편집기 기본 동작 기능으로 구성된다. 그래픽 요소 시각화 기능은 그래픽 요소를 사용자 환경에 시각적으로 나타내기 위한 기능이다. 편집기 기본 동작 기능은 SVE에서 그래픽 요소를 추가하거나 삭제하는 등의 동작과 관련된 기능이다.

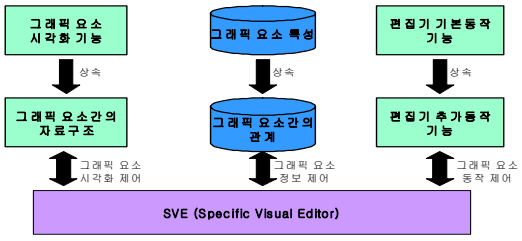


(그림 4) AVE 구조

3.5 SVE(Specific Visual Editor) 설계

그림 5에서 나타낸 바와 같이 SVE는 AVE를 상속받아 시각 프로그래밍에 필요한 기본적인 기능을 제공한다. 또한, 각 시각 프로그래밍 고유한 특성을 표현하기 위해서 추가적인 기능을 제공한다.

위에서 정의한 그래픽 요소의 연결자는 SVE에서 그래픽 요소 간의 연결 관계를 표현할 때 사용된다. 사용자가 표현된 그래픽 요소들 중에서 연결을 시도하는 경우에 SVE는 연결 허용 여부를 검사하고, 허용된 연결만을 SVE에 표현한다.



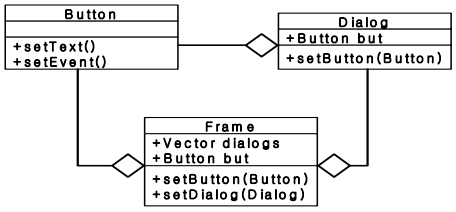
(그림 5) SVE 구조

4. 실험

본 논문에서 제안한 AViPS를 이용해서 시각 프로그래밍 시스템을 구축하는 방법을 알아보도록 하자. 작성할 예제는 버튼을 포함한 다이얼로그를 호출할 수 있는 윈도우 프레임을 생성하는 편집기를

생성하는 것이다.

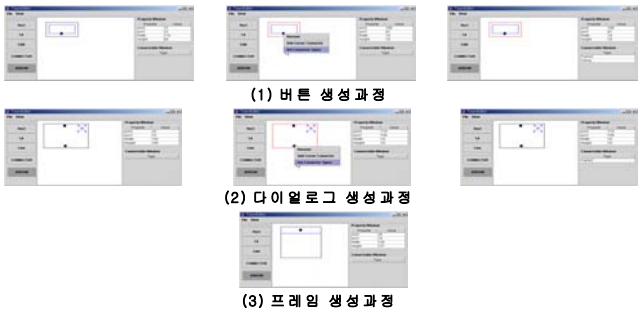
그림 6에서 나타낸 바와 같이 예제를 표현하기 위해서 “버튼”, “다이얼로그”, “프레임” 등 총 3종류의 그래픽 요소 클래스를 가진다. “버튼”은 “다이얼로그”와 “프레임”에 포함될 수 있다. “다이얼로그”는 “프레임”에서 호출할 수 있다. “프레임”은 호출할 “다이얼로그”와 “버튼”을 나타낼 수 있다. 위의 예제는 “다이얼로그” 간에는 상호 연결 관계가 없다고 가정한다.



(그림 6) 그래픽 요소 클래스 다이어그램

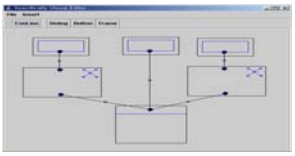
사용자는 실험예제를 표현하기 위해서 필요한 3종류의 그래픽 요소를 TEdi를 이용해서 생성한다. “버튼”의 연결자는 “다이얼로그”와 “프레임을 연결 허용 형태로 갖는다. “다이얼로그”는 2개의 연결자를 갖는다. 하나는 “프레임”형태를 가지고 “프레임” 그래픽 요소와 연결된다. 다른 하나의 연결자는 “버튼”과 연결 요청하는데 사용하기 때문에 연결 허용 형태를 갖지 않는다. “프레임”은 “버튼”과 “다이얼로그”에게 연결 요청만 하기 때문에 연결 허용 형태를 갖지 않는다.

그림 7의 (1)은 TEdi를 이용해서 “버튼”을 생성하는 과정이다. 첫째로 “버튼”을 표현하기 위해 기본적인 그래픽 요소를 활용해서 그림으로 표현한다. 둘째, 연결자에 연결을 허용할 형태를 정의한다. 최종적으로 생성된 “버튼”을 저장한다. 그림 7의 (2)는 그림 7의 (1)과 유사한 과정을 통해서 “다이얼로그”를 생성하는 과정을 나타낸다. 그림 7의 (3)은 연결자에서 연결을 허용할 형태를 가지고 있지 않기 때문에 그림으로 표현한 결과를 저장하는 과정을 나타낸다.



(그림 7) 그래픽 요소 생성 과정

실험예제를 SVE를 이용해서 표현하면 그림 8와 같은 형태를 갖는다. 각 “다이얼로그”는 하나의 “버튼”을 포함하며, “프레임”은 두개의 “다이얼로그”를 호출할 수 있다. 또한, 하나의 “버튼”을 포함하고 있다. “다이얼로그”간에는 연결을 설정할 수 없다.



(그림 8) 그래픽 요소간의 연결 관계 표현

5. 결론 및 향후과제

현재 다양한 시각 프로그래밍 시스템과 다이어그램을 이용한 모델링 도구가 활용되고 있다. 이러한 도구들은 의미를 가진 그래픽 요소들로 프로그램을 작성하거나 모델링에 사용한다. 각각의 그래픽 요소는 독립적인 의미와 더불어 상호 연결되어 구문을 형성하거나 추가적인 의미를 가진다. 하지만, 기존의 도구들은 연결의 의미와 표현은 가능하지만, 두 요소 간의 연결의 타당성 검사가 부족하다.

본 논문에서 제안한 시스템은 언어별로 특화된 그래픽 요소를 정의하고, 연결자를 설정하며, 그래픽 요소간의 연결 타당성을 검사한다. 따라서 본 논문에서 제안한 시스템의 장점은 다음과 같다.

- 각 언어별로 연결자를 정의하고 타당성 검사를 통한 프로그램의 구문 오류 축소
- 각 언어별로 특화된 그래픽 요소 정의와 그래픽 요소간의 연결 관계를 통한 언어의 고유성 지원

향후에는 다양한 시각 프로그래밍 언어에서 정의한 그래픽 요소를 연결하여 의미를 표현하는 시스템에 대해서 연구할 예정이다.

참고문헌

[1] N. C. Shu, "Visual Programming: Perspective and approaches", IBM Systems Journal, Vol. 38, 1999.

[2] M. Boshernitsan, "Visual Programming Languages : A Survey", 1997.

[3] Eric J.Golin, "Palette: An Extensible Visual Editor", Proc. of ACM/SIGAPP symposium on Applied computing, 1992.

[4] Frahangiz Arefi, "Automatically Generating Visual Syntax-Directed Editors", Vol. 33, CACM, p349~360, 1990.