확장 상태 전이 그래프에 기반을 둔 시각 병렬 프로그래밍

정 원 호† · 허 혜 정††

요 약

이해하기 쉽고, 병렬 동작을 표현할 수 있으며, 이식성이 좋은 시각 병렬 프로그래밍 환경인 ESTGVP가 설계, 구현된다. 이를 위해, 기존의 상태 전이 그래프를 동기식 혹은 비동기식 병렬 동작을 포함할 수 있도록 확장시킨 확장 상태 전이 그래프(ESTG)가 제안된다. ESTGVP는, 확장 상태 전이 그래프와 텍스트를 병용하고 있으며, 순차 및 병렬 프로그래밍 작업을 용이하게 할 수 있다는 장점을 가진다. 또한 그래프에 기반을 둔 시각 프로그래밍이기 때문에 프로그램의 제어 구조를 쉽게 이해할 수 있다. Tcl로 설계 구현되어 있으므로, 다양한 운영체제 환경에서 실행할 수 있어 높은 이식성을 가지고 있다. ESTGVP에 있어서, 주 기능은 편집, 변환, 실행으로 구성되어지며, 필요시 C언어와 Tcl 언어로 변환될 수 있으며, 실행은 Tcl을 기반으로 이루어진다.

A Visual Concurrent Programming Based on Extended State Transition Graph

Won-Ho Chung† · Hye-Jung Hur††

ABSTRACT

A visual concurrent programming environment, called ESTGVP is designed and implemented, which is easy to understand, highly portable, and can represent parallel behaviors. For our purpose, a conventional state transition graph is extended so as to enable both of synchronous and asynchronous parallel operations. We call it extended state transition graph(ESTG). ESTGVP uses the ESTG and texts for programming, and makes it easy programming sequential and parallel behaviors. Also, it is easy to understand the control structure of a program because ESTGVP is a visual programming environment based on the graph. ESTGVP is written in Tcl language and thus it is highly portable on various operating systems. It consists of three major components ; edition, transformation and execution. If necessary, ESTG can be transformed into C or Tcl language, and its execution is based on Tcl.

1. 서 론

최근, 저가의 고성능 개인용 컴퓨터의 등장으로 기존의 1차원적인 문자위주의 프로그래밍 환경이, 2차원의 그래프, 그림 혹은 아이콘 등을 기반으로 프로그래밍 작업을 용이하게 해주는, 시각 프로그래밍 환경으로 그 추세가 변화되어가고 있다. 시각언어란 가시화를 통해 학습과 작업의 용이함을 토대로 프로그래밍 작업의 효율을 높이고자 개발된 언어를 의미한다. 기존의 여타 프로그래밍 언어가 1차원적인 문자를 사용하는 반면, 시각언어는 사용자가 프로그래밍 작업을 보다 쉽게 하기 위해 2차원적인 그림이나 아이콘과 같은 도구들을 시각언어로 이용하는 프로그래밍 접근방

※ 본 연구는 '99학년도 덕성여자대학교 연구비 지원으로 이루어졌음.
† 정 회 원 : 덕성여자대학교 컴퓨터과학부 교수
†† 준 회 원 : 덕성여자대학교 대학원 전산 및 정보통신학과
논문접수 : 2000년 6월 19일, 심사완료 : 2000년 7월 31일

법이다. 이러한 시각언어는 과거 초기의 개념 정립을 거쳐[1-3], 각종 그래프 모델을 이용하는 경우와[4-8], 특정 아이콘들을 기반으로 하는 경우까지 활발히 연구되고 있다[9-12]. 가시성은 프로그램의 구조적 특성을 시각적으로 그리고 효율적으로 표현하기 위한 중요한 요건으로서, 프로그래밍 작업을 쉽게 할 수 있게 할뿐만 아니라도, 프로그램의 전체적 이해를 도와준다는 장점을 가지고 있다. 이러한 시각언어의 구성에는 크게 두 가지 접근방법이 있으며[13-14], 첫째, 일반화된 아이콘을 언어로 사용하는 방법과 둘째 기존의 그래프 도구(예를 들면, 상태전이 그래프[4, 28], 데이터 흐름 그래프[15-17], 패트리 망[7, 8] 등) 또는 새로이 제안된 그림도구를 사용하는 접근방법이다[18, 19, 28]. 그러나 전자의 경우, 의미론적 아이콘의 설계와 표현 공간의 제한이라는 문제점을 가지고 있어 특수 응용이 아닌 경우 적용하기가 쉽지 않다는 것이 단점이라고 할 수 있다. 또한 이러한 시각 언어는 Windows를 벗어나 Unix까지 그 연구를 넓히고 있으며, 사용자 인터페이스와 프로그래밍뿐만 아니라 데이터의 표현과 디버깅 그리고 성능평가를 위한 시뮬레이션에도 적용되고 있다[20-25]. Burnett는 [26]에서 이러한 시각 언어들을 각 분야 별로 일목요연하게 잘 정리하여 분류하였으며, [27]에서는 시각적 객체지향 프로그래밍을 위한 개념과 환경에 관해 정리를 하고 있다.

본 논문에서는, 먼저, 작업의 흐름을 2차원적으로 기술해 줄 수 있으며, 이해하기 쉽고 사용하기 쉬우나, 병렬 동작을 표현하지 못하는 기존의 상태 전이 그래프를 병렬성을 가지도록 확장시킨 확장 상태 전이 그래프(Extended State Transition Graph, ESTG)가 제안된다. 그리고, 이를 시각언어의 제어구조로 사용하는 시각 프로그래밍 환경인 ESTGVP(ESTG for Visual Programming)가 설계, 구현된다. ESTG는 프로그램의 일반 구조는 물론, 동기식 혹은 비동기식 병렬 구조도 표현할 수 있으며, 기존의 상태 전이 그래프에 기반을 두고 있으므로, 그 개념이 간단하다는 특징을 갖는다. 이를 프로그래밍 언어의 제어구조로 사용하면서 텍스트 처리 기능을 부가시켜 효율적 병렬 프로그래밍이 가능한 시각 프로그래밍 환경이 ESTGVP이다. 프로그램을 2차원 가시화 함으로써 기존의 문자 위주의 프로그램 개발을 좀 더 용이하게 할 수 있다. 특히, 병렬 프로그래밍인 경우 많은 어려움을 내포하고 있으나, ESTG를 사용하는 경우 손쉽게 프로그래밍에 접근할 수 있으며, 또한 병렬 동작을 쉽게 이해할 수 있다는 장점을 가진다.

ESTGVP는 Tcl을 사용하여 설계, 구현되었으므로 이식성이 높다. ESTG와 텍스트를 병행하여 구현된 병렬 프로그램은 Tcl언어로 변환되어지고, 자체적으로 실행되며, 필요시 C언어로의 변환될 수 있다. 따라서 프로그램 설계에 도움을 줄 수 있고, 병렬 프로그램 모델을 구현해서 수행할 수도 있다. Tcl로 구현되어 있으므로, 대부분의 운영체제 환경, Windows98, Windows NT, UNIX, Macintosh에서 수행 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 제안하는 ESTGVP의 개념과 ESTG의 정의를 소개하고, 3장에서는 ESTGVP 주요 기능 중의 하나인 편집을 담당하는 모듈인 ESTGEdit의 개념과 그 특징에 대해서 기술되며, 4장에서는 ESTG의 실행을 위해 필수적인 기능인 Tcl 언어로의 변환을 담당하는 모듈인 ESTGTrans가 기술된다. 그리고 5장에서는 Tcl 언어를 기반으로 ESTG의 실행을 담당하는 모듈인 ESTGExe가 기술되며, 6장에서는, 이진탐색과 순차탐색을 병행으로 처리하는 예를 통해서 ESTGVP의 기능을 살펴보며, 마지막으로, ESTGVP에 관한 추가 연구 방향으로 결론을 맺는다.

2. 확장 상태 전이 그래프(ESTG)

ESTG는 노드(node)와 에지(edge)로 구성되며, 다음과 같이 정의되는 방향성 그래프이다.

[정의 1] 확장 상태 전이 그래프, $Gx = (V, E, L_V, L_E)$는 레이블된 방향성 그래프이다. 단

i) $V = \{v_1, v_2, \cdots, v_n\}$는 노드들의 집합으로 제어 상태를 나타내고 있으며,
ii) $E \subseteq V \times V$는 방향 에지들의 집합이며,
iii) $L_V : V \rightarrow \{0, 1, 2, 3\}$는 노드 레이블링 함수로 각 노드를 해당 타입으로 매핑하며,
iv) $L_E : E \rightarrow \{a_1, a_2, \cdots, a_k\}$는 에지 레이블링 함수로 각 에지를 해당 action으로 매핑한다. ■

ESTG의 노드들은 type-0, type-1, type-2, 그리고 type-3, 4가지 유형의 노드들로 분류된다. Type-0 노드는 수행할 action이 없는 상태의 노드를 나타내는데, 말하자면 종료된 상태를 나타내는 터미널 노드이다. Type-1 노드는 순차 수행만이 가능한 상태의 노드를 나타내면

서 동시에 수행할 action이 오직 하나밖에 없는 경우를 나타낸다. Type-2 노드는 2개 이상의 수행 가능한 action이 존재하나 그들 중 어느 하나만이 수행 가능한, 즉 서로 conflict한 상태를 나타내고 있다. Type-3 노드는 2개 이상의 수행 가능한 action들이 존재하며, 그들은 동시 병렬로 수행이 가능한 상태를 나타내고 있다.

여러 개의 type-1 노드들이 직렬로 연결되는 경우, 하나의 type-1 노드로 단순화시킬 수 있으며, type-2나 type-3 노드 바로 다음에 나오는 type-1 노드도 성격에 따라 앞의 노드 혹은 뒤의 노드에 축소되어 생략될 수 있다.

이러한 노드는 프로그램의 수행상태를 나타내는 것으로서 다음의 [정의 2], [정의 3]에 따라 <표 1>처럼 요약될 수 있다.

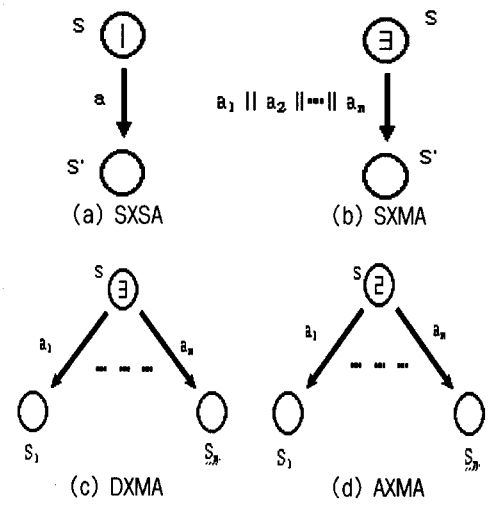
[정의 2] 어떤 상태, S에서 활성화되어있는 action의 수행형태를 $A(S) = \{A_1, A_2, \cdots, A_n\}$라고 하고, 각 A_k ($1 \le k \le N$)는 action들의 집합으로 $A_k = \{a_{k1}, a_{k2}, \cdots, a_{kn}\}$와 같이 표현된다. A(S)에 속한 각 A_k는 서로 conflict한 관계에 있으며, A_k에 속한 각 a_{ki}는 서로 병렬 관계에 있다. ■

[정의 3] S를 주어진 상태라 하고, $A(S) = \{A_1, A_2, \cdots, A_n\}$라고 할 때, S에서의 병행도 $CDR(S) = C(S)/D(S)$로 정의된다. 단, $C(S) = \sum_{k=1}^{N} |A_k|$ 이며, S에서의 총 합병행도라 하고, $D(S) = |A(S)|$로 S에서의 선택도라고 한다. ■

〈표 1〉 병행도에 따른 상태 class 분류

Class	A(S)	C(S)	D(S)	CDR(S)	비 고
⓪	{ }	0	0	0	no action
①	{{a}}	1	1	1	deterministic & sequential
②	{{a_1}, {a_2}…}	C(S)≥2	D(S)≥2	1	disjunctive & sequential
③	{{a_1, a_2, …}}	C(S)≥2	D(S)=1	≥2	synchronous & parallel

그러므로, <표 1>을 참조하면, 각 class에서 action의 수행 형태는 다음과 같이 SXSA, SXMA, DXMA 그리고 AXMA 4가지 형태로 분류될 수 있으며, 그들의 수행 형태를 ESTG를 사용하여 표현하고 있는 것이 (그림 1)이다.



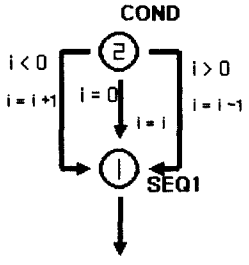
(그림 1) 각 type에 따른 4가지 수행형태를 나타내는 ESTG

- Sequential-Execution/Sing-Action(SXSA)
  S→ S'에서 $L_V(S) = 1$, $L_E(S, S') = a$
- Synchronous-Execution/Multiple-Actions(SXMA)
  S→ S'에서 $L_V(S) = 3$, $L_E(S, S') = (a_1 \| a_2 \| \cdots \| a_n)$
- Disjunctive-Execution/Multiple-Actions(DXMA)
  S→ $(S_1 | S_2 | \cdots | S_n)$에서 $L_V(S) = 2$,
  $L_E(S, S_1 | S_2 | \cdots | S_n) = a_1 | a_2 | \cdots | a_n$
- Asynchronous-Execution/Multiple-Actions(AXMA)
  S→ $(S_1 \| S_2 \| \cdots \| S_n)$에서 $L_V(S) = 3$,
  $L_E(S, S_1 \| S_2 \| \cdots \| S_n) = a_1 \| a_2 \| \cdots \| a_n$

여기서 알 수 있듯이, SXSA은 type-1 노드에서 수행 가능한 동작으로 병행도는 1이며, DXMA는 type-2 노드에서 수행 가능한 동작으로 비록 총합 병행도가 2 이상이나, 선택도가 같은 값을 가지게 되어 결국 병행도는 1이된다. 그러나, 선택도가 2이상이므로 하나의 action이 조건에 의해 선택되어 수행되는 conflict한 노드이다. Type-3 노드에서는 병렬 동작이 일어나며, 그 수행 형태는 동기식과 비동기식 2가지로 나타날 수 있다. SXMA는 동기식 병렬 동작을 나타내고 있으며 각 동작의 병렬 동작 후 다음 동작을 위해서는 각 action이 끝날 때까지 기다려야만 한다. AXMA는 비동기식 병렬 동작을 표현하고 있으며, 각 action은 다른 action에 관계없이 독립적으로 수행이 일어날 수 있다. 이처럼, ESTG는 순차 수행, 조건 수행뿐만 아니라 동기식, 비동기식 병렬 수행도 표현할 수 있음을 알 수 있다.

ESTG에 있어서 에지는 어떤 action에 혹은 조건에 의한, 노드에서 다른 노드로 상태의 변이를 나타내고 있으며, 수행을 표현하는 텍스트 문이 기술된다.

(그림 2)는 하나의 type-1 노드와 하나의 type-2 노드로 구성된 ESTG의 예이다.

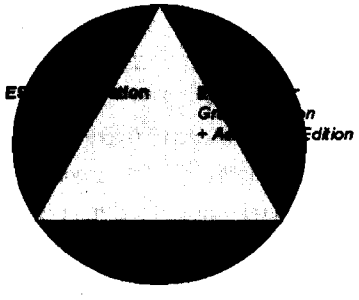


(그림 2) class-2 노드에서 action의 예

(그림 2)에서 보듯이 type-2 노드, 즉 COND에서 i의 값에 따라 해당 에지를 결정하고, 그 에지에서 해당되는 코드를 수행하고, type-1 노드, 즉 SEQ1으로 이동한다.

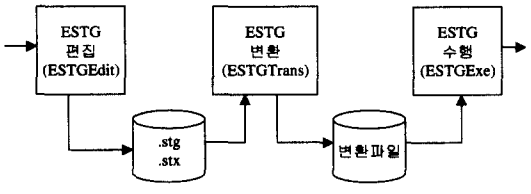
3. ESTGEdit : ESTG 프로그래밍

ESTGVP는 ESTG를 프로그램 언어의 시각화 부분으로 이용한 프로그래밍 환경으로 Tcl 언어를 기반으로 하고 있다. ESTGVP의 핵심 기능은 (그림 3)에와 같이 3가지로 분류될 수 있다. ESTG와 텍스트 코드로 프로그램을 작성하는 편집부분인 편집기(Editor)와, C 혹은 Tcl로 변환시키는 변환부분인 변환기(Translator), 직접 실행시키는 실행부분인 수행기(Executer)로 구성된다. 이들은 하나의 화면에서 메뉴선택에 의해 원하는 각 기능으로 들어갈 수 있다.

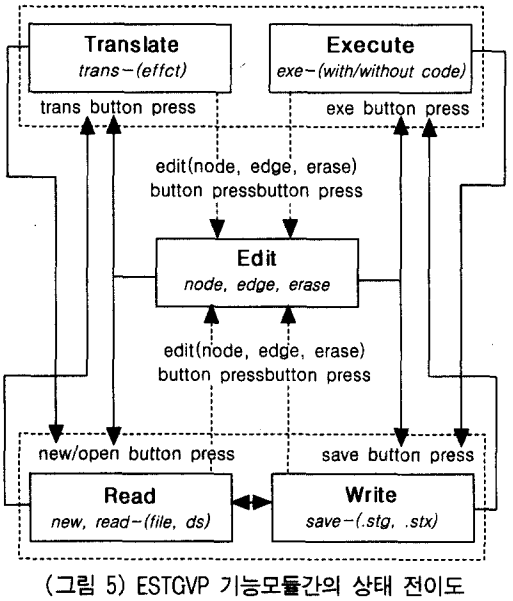


(그림 3) ESTGVP 주요 기능

사용자는 목적하는 작업의 제어 흐름을 ESTG를 사용하여 표현하여야 하는데, 이 때 사용하는 것이 ESTG 편집기이다. 그리고, 편집된 정보는 stg 확장자를 가진 그래픽 파일과 stx 확장자를 가진 텍스트 파일로 나뉘어 저장된다. 사용자가 편집한 ESTG를 분석해서 프로그램 제어구조를 파악한 후 변환기에서는 그 부분을 원하는 언어(여기서는 C 혹은 Tcl)로 변환한 후 문자와 결합시켜 프로그램을 완성하고 변환 파일에 저장한다. 실행과정은 ESTGVP에서 작성된 프로그램을 C변환 프로그램으로 완성하고, 토큰별로 인식해서 다시 한 번 ESTGVP가 인식할 수 있는 형태로 매핑 시키고, 필요한 라이브러리를 결합시킨 후 Tcl 번역기의 도움을 받아 실행시키고, 실행된 결과를 윈도우를 통해 사용자에게 보여준다. 이러한 과정을 요약하여 보여주고 있는 것이 (그림 4)이다. 그리고, 편집기, 변환기, 그리고 수행기 사이의 전이를 자세하게 외부 Event들과 연관해서 상태별로 분석하고 보여주는 것이 (그림 5)이다.

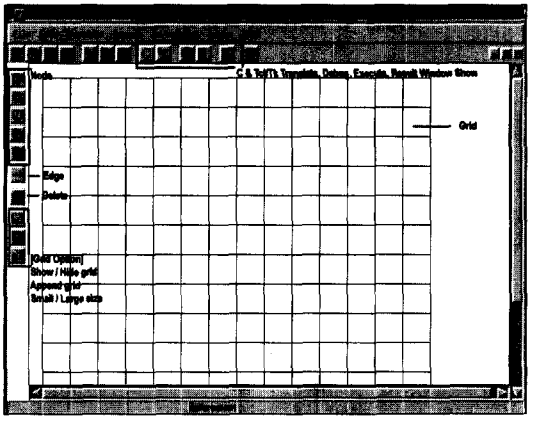


(그림 4) ESTGVP 작업의 흐름



(그림 5) ESTGVP 기능모듈간의 상태 전이도

ESTGVP는 기존의 시각 프로그래밍 환경에서 제공하는 그래프 편집기에서의 단점인 정보의 변환과 편집의 어려움을 해소하기 위해 (그림 6)에서 보여준 바와 같은 N × N 장방형 그리드 평면을 기반으로 이루어진다. 이러한 그리드 평면을 사용하면, 노드와 에지의 위치와 연결 정보를 손쉽게 유지할 수 있다는 장점을 가지게된다. ESTGVP에 있어서 프로그래밍 작업은 메뉴에서 제공해주는 각 type의 노드와 에지 아이콘을 이용하여 노드와 에지의 위치를 설정하고, 노드와 에지에 각각 조건 또는 action을 레이블링하는 2단계로 구성된다. 특히, (그림 1)에서 보여준 바와 같은 수행의 형태를 고려하여 먼저 노드들 위치를 설정하고, 노드와 노드 사이에 에지를 배치하는 것이 일반적이라 할 수 있다. 그리고 type-2 노드인 경우에 한하여, 조건과 각 조건을 만족할 경우 수행해야할 에지를 설정해 주어야 하며, 각 에지에 action을 기술하여 조건에 따라 수행할 작업을 기술한다. 이렇게 하여, 작성된 ESTG는 그래픽 파일(.stg) 부분과 텍스트 파일(.stx) 부분으로 구성된다. 그래픽 파일은 작성된 ESTG를 구성하고 있는 각 노드와 에지의 그리드 평면에서의 구별을 위한 그래픽 정보를 저장하며, 테스트 파일은 수행해야 할 action에 관한 기술 내용을 저장하고 있다.



(그림 6) ESTGVP의 초기 화면

그래픽 파일의 구조는 <표 2>와 같다. 첫 번째 정보는 설정된 그리드 평면의 크기를 나타내는 정보로 그리드를 구성하는 라인의 수(N)이다. 기본 그리드 크기는 10×10 이나, 사용자에 의해 원하는 크기로 변경 가능하다. 그 다음에 저장되는 것이 각 노드에 대한 정보들이다. 동일한 타입의 노드들은 모두 하나의 집합으로 저장되는데, 즉,

{type/ X Y 노드id 노드명 X Y 노드id 노드명 X Y 노드id 노드명……} 구조를 가지고 있다.

여기서 type은 노드의 타입을 나타내며, X Y 쌍은 노드의 그리드 상의 좌표를, 노드id는 해당 노드의 id를, 노드명은 해당 노드의 명칭을 나타내고 있다.

그러므로, 그리드 상에 나타난 특정 타입의 노드들은 해당 타입의 집합에 저장된다. <표 2>에서 S 노드라는 것을 볼 수 있다. S 노드는 type-1, type-2, type-3 모두 이에 속할 수 있으며, 처음 시작하는 노드로서 설정이 된다면 S 노드로 변경되어 저장된다 이것은 변환·실행을 하기 위해 꼭 필요로 한다.

〈표 2〉 .stg 그래픽 파일 구조

그리드 라인수
{S/ X,Y , ID, 노드의 이름 ……} - S 노드 : 시작 노드
{0/ X,Y , ID, 노드의 이름 ……} - 0노드 : 클 노드
{1/ X,Y , ID, 노드의 이름 ……} - 1노드 : 순차 노드
{2/ X,Y , ID, 노드의 이름 ……} - 2노드 : 제어 노드
{3/ X,Y , ID, 노드의 이름 ……} - 3노드 : 병렬 노드
{E/ X_s,Y_s, X_e Y_e …, ID, 에지의 이름 …} - 에지

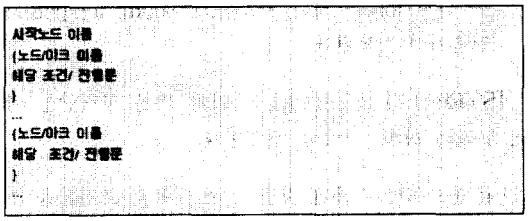
모든 에지들은 에지 집합에 저장되며, 그 집합의 구조는 노드 집합과 유사하나, 에지의 경우, 에지를 위해 사용자가 지시한 모든 좌표가 저장된다는 것이 다르다. 에지 집합은

{type-E/ X11 Y11 X12 Y12… 에지id 에지이름 X21 Y21 X22 Y22… 에지id 에지이름 …}

의 구조를 가지고 있다. 여기서 (X11 Y11 X12 Y12 ... 에지id 에지이름)이 하나의 에지를 나타내며, 모든 에지들은 노드와 마찬가지로 동일한 집합에 저장된다. 그러므로, ESTG를 나타내는 그래픽 파일은 항상 6개의 블록으로 구성된다는 것을 알 수 있다.

<표 3>은 텍스트 파일 구조를 나타내고 있으며, 먼저 ESTG의 시작노드의 명칭이 저장되고 다음부터 각 노드의 명칭과 해당 조건문(type-2 노드인 경우), 그 다음에 각 에지의 명칭에 대한 action 진행문이 기술된다. 이때 action의 기술은 C 언어 기반으로 기술된다. 여기서 텍스트 파일은 각 노드와 에지에 대한 조건 혹은 action을 블록 단위로 저장하고 있다는 것을 알 수 있다.

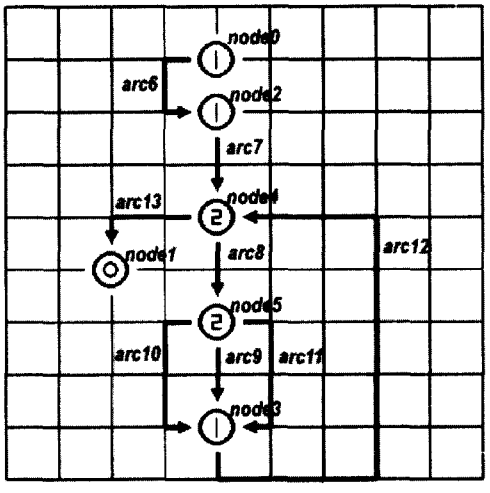
〈표 3〉 .stx 텍스트 파일 구조



그러면, 간단한 이진탐색 프로그램에 대하여, 그래픽 파일과 텍스트 파일을 살펴보기로 한다. 쉽게 접할 수 있는 간단한 이진탐색의 제어 구조를 나타낸 것이 (그림 7)이다. 이해를 돕기 위해 각 노드에는 노드 명칭을, 각 에지에는 에지 명칭을 첨가하였다. 이 ESTG에 대한 그래픽 파일인 binary.stg의 구조를 보면 다음과 같다.

[binary.stg]

10 {S/ 1 4 0 node0} {0/ 5 2 1 node1} {1/ 2 4 2 node2 8 4 3 node3} {2/ 4 4 4 node4 6 4 5 node5} {3/} {E/ 1 4 1 3 2 3 2 4 6 arc6 2 4 3 4 4 4 7 arc7 4 4 5 4 6 4 8 arc8 6 4 7 4 8 4 9 arc9 6 4 6 3 8 3 8 4 10 arc10 6 4 6 5 8 5 8 4 11 arc11 8 4 9 4 9 7 4 7 4 4 12 arc12 4 4 4 3 4 2 5 2 13 arc13} ■



(그림 7) 노드명칭과 에지명칭이 첨부된 간단한 이진탐색을 나타내는 ESTG

그리드 평면의 크기는 10이므로 10×10 그리드 평면을 나타내고 있으며, {1/ 2 4 2 node2 8 4 3 node3}는 type-1 노드로서 2개의 노드가 있으며, 하나는 좌표 (2, 4)에 위치하고 노드id는 2이며 노드의 명칭은 *node2*이다. 다른 하나는 좌표 (8, 4)에 위치하고 노드id는 3이고 노드의 명칭은 *node3*이다. 즉, 이 ESTG는 하나의 시작노드와, 하나의 type-0 노드, 2개의 type-1 노드, 2개의 type-2 노드들과 arc6부터 arc13까지의 에지 이름을 가진 8개의 에지들로 구성된다는 것을 알 수 있다. 예를 들어, (1 4 1 3 2 3 2 4 6 arc6)가 나타내는 에지는 그리드 상의 좌표 (1, 4)에서 시작하여 (1, 3)과 (2, 3)을 거쳐 (2, 4)까지의 경로로 이루어지고 있음을 표현하고 있다. 그러므로, 첫 번째 좌표와 마지막 좌표는 해당 에지의 시작 노드의 좌표와 동일하고 마지막 좌표는 해당 에지의 목적지 노드인 것이다.

텍스트 파일은 ESTGEdit이 제공하는 action 창을 통해 각 type-2 노드에는 조건문이 각 에지에는 해당 action의 진행문이 기술되는데, 다음의 텍스트 파일처럼 기술되었다고 가정한다. 앞에서도 언급하였듯이 텍스트는 일반적으로 에지에 기술되며, 노드인 경우에는 type-2 노드의 경우에만 조건을 기술하기 위해 기술된다.

[binary.stx]

node0
{node1 exit}
{node4 if {(low <= high)} next arc8 if {(low > high)} next arc13}
{node5 if {(x == S[mid])} next arc10 if {(x < S[mid])} next arc9 if {(x > S[mid])} next arc11}
{arc6 int S[20] = {3,4,6,7,10,12,15,17,28,38,39,40,42,44,56,77,78,79,80,82} ;
int low, high, mid, x, location;
}
{arc7 low = 1;
high = 20;
..... / 이하 생략 ■

그래픽 파일에서도 나타나 있듯이 시작노드의 명칭은 *node0*이고, *{node4 if {(low <= high)} next arc8 if {(low > high)} next arc13}*는 *node2*라는 이름의 type-2 노드이며, 노드와 에지의 구조와 조건변수들을 통해서 이와 같은 조건/진행문이 저장된다.

4. ESTGTrans : 프로그램 변환

ESTGEdit을 통해 작성된 ESTG를 C 또는 Tcl 언

어로 변환하는 과정은 해당 그래픽 파일의 정보 분석, 텍스트 정보와의 매핑, 그리고 토큰별로 분석해서 C 혹은 Tcl 명령어로 변환하는 작업 이렇게 크게 3부분으로 구성된다. 먼저 그래픽 파일의 정보 분석은 ESTG 노드와 에지의 연결 상태를 분석해서 프로그램의 흐름 구조를 파악하는 단계이다. 예를 들면 type-2 노드는 조건문으로, type-3 노드는 병렬처리 문으로, 그리고 ESTG에서의 사이클은 반복문으로 대치된다. 그 다음으로, 텍스트 정보와의 매핑은 전 단계에서 파악된 제어 구조와 더불어 각 노드 혹은 에지의 id에 해당하는 텍스트 정보를 매핑 한다. 이렇게 변형한 후 마지막으로 각각을 토큰 별로 분류해서 해당 언어의 명령어로 변환하는 작업을 거쳐 C 또는 Tcl 언어로 변환된다. 이 과정에서 일반적인 프로그래밍 언어의 파서에서 취하고 있는 방식과 유사하게 한번에 하나의 토큰을 왼쪽에서 오른쪽 방향으로 읽으면서 처리하는 하향식 방식을 취하고 있다. 이를 세분해서 정리하면 다음과 같은 4단계로 표현할 수 있다.

[ESTGTrans 과정]

[STEP-1] 그래픽 정보 분석과정

[STEP-1.1] 시작노드로부터 시작하여 다음에 오는 노드 혹은 에지들의 흐름을 따라 id를 *history buffer*에 기록한다. 단,

(1) 현재 노드가 type-2 노드인 경우, 노드id 다음에 해당되는 조건문과 그 조건을 만족한 경우의 노드 혹은 에지들의 흐름을 따라 id를 depth-first 방식으로 *history buffer*에 기록한다. 이때, 현재 노드id와 동일한 값의 노드id가 *history buffer*에 이미 기록되어 있고, 기록되어 있는 노드가 type-2일 경우에는 그 노드id가 앞에 while을 설정하고, 현재 노드id 대신에 endwhile을 *history buffer*에 기록한다. Type-2 노드가 아닐 경우에는 goto[노드id]라는 형식으로 *history buffer*에 기록한다.

(2) 현재 노드가 type-3 노드인 경우, type-2 노드의 경우와 유사하다. Type-3 노드에서 나가는 에지가 2개인 경우 처음 에지에 forkp[id]로 기록하며, 다른 에지에는 child[id]로 설정한다. 이와 같은 원리로 에지가 3개 이상인 경우에는 child[id] 에지에 다시 forkp[id]를 기록하게 되며, 그 다음 에지에는 child[id]로 기록하는 것을 에지가 없을 동안 반복한다. 즉, 3개 이상의 에지가 있을 경우 child에서만 다시 fork를 하기 때문에, child[id], forkp[id]가 연속해서 기록된다.

[STEP-1.2] [STEP-1.1]을 거쳐 기본 골격만을 가진 *history buffer* 구성을 완성한다.

(1) if 문으로만 기록된 것을 if 문 내부의 조건문을 조사하여 if ~ elsief ~ else문으로 변형한다.

(2) while문은 do-while(조건) 혹은 while-do(조건)문으로의 변환을 실행한다.

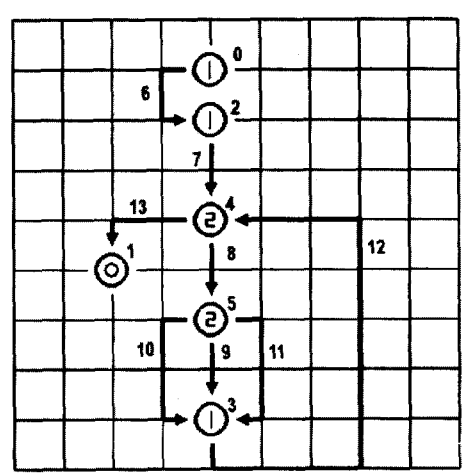
(3) goto[id]로 기록된 것을 해당 id가 처음 나온 곳으로부터 goto[id]가 기록된 부분에서 가장 가까운 endwhile를 현재 goto[id]가 나온 위치로 옮기고 goto[id]를 삭제한다.

[STEP-2] 텍스트 정보와의 매핑을 통한 C언어 구조로의 변환

*history buffer*에 기록된 id에 해당하는 텍스트 문을 텍스트 파일(.stx)로부터 탐색하여, *history buffer*의 해당 id 위치에 매핑시키며, *history buffer*에 기록된 조건문, 반복 구조, 병행 구조와 결합시킨다. 이 과정이 완료되면, 일반적인 C 언어 구조로의 변환이 일어난다.

[STEP-3] Tcl 언어구조로의 변환

[STEP-2]까지의 과정을 통해 변환된 C 언어 형태를 토큰별로 분석해서 그에 해당되는 Tcl 언어로 변화하는 작업을 거쳐서 Tcl로의 변환작업을 완성한다. (그림 8)은 3장의 예로 살펴본 이진탐색 ESTG인 (그림 7)에



(그림 8) id와 함께 표시된 이진탐색 ESTG

id를 표시한 그림이다. Id는 각 노드, 에지들을 구별하기 위해 ESTGVP에서 자동적으로 부여되는 유일한 번호이다. 이 예를 변환 알고리즘에 적용하면 다음과 같은 결과를 얻을 수 있다.

[STEP-1.1]

0 6 2 7 while 4 {if (low <= high)} 8 5 {if (x == S[mid])} 10 3 12 endwhile {endif (x == S[mid])} {if (x < S[mid])} 9 goto3 {endif (x < S[mid])} {if (x > S[mid])} 11 goto3 {endif (x > S[mid])} {endif (low <= high)} {if (low > high)} 13 1 {endif (low > high)} ■

[STEP-1.2]

0 6 2 7 while 4 {if (low <= high)} 8 5 {if (x == S[mid])} 10 {endif (x == S[mid])} {if (x < S[mid])} 9 3 12 endwhile {endif (x < S[mid])} {if (x > S[mid])} 11 goto3 {endif (x > S[mid])} {endif (low <= high)} {if (low > high)} 13 1 {endif (low > high)}

0 6 2 7 while 4 {if (low <= high)} 8 5 {if (x == S[mid])} 10 {endif (x == S[mid])} {if (x < S[mid])} 9 {endif (x < S[mid])} {if (x > S[mid])} 11 3 12 endwhile {endif (x > S[mid])} {endif (low <= high)} {if (low > high)} 13 1 {endif (low > high)}

0 6 2 7 {whiledo (low <= high)} 4 8 5 {if (x == S[mid])} 10 {endif (x == S[mid])} {if (x < S[mid])} 9 {endif (x < S[mid])} {if (x > S[mid])} 11 3 12 {endif (x > S[mid])} {endwhile (low <= high)} {if (low > high)} 13 1 {endif (low > high)}

0 6 2 7 {whiledo (low <= high)} 4 8 5 {if (x == S[mid])} 10 {endif (x == S[mid])} {elseif (x < S[mid])} 9 {endif (x < S[mid])} {elseif (x > S[mid])} 11 {endif (x > S[mid])} 3 12 {endwhile (low <= high)} 13 1 ■

[STEP-2]

```
main () {
int S[20] = {3,4,6,7,10,12,15,17,28,38,39,40,42,44,56,77,78,
79,80,82} ;
int low, high, mid, x, location;
 low = 1;
 high = 20;
 location = -1;
 printf ("Input Search Data : ");
 scanf ("%d",&x);
 while (low <= high) {
  mid = (low + high) / 2
  if (x == S[mid]) {
   location = mid;
   break;
  }
  else if (x < S[mid]) {
   high = mid - 1;
  }
  else if (x > S[mid]) {
   high = mid + 1;
  }
 }
 printf ("Data position : %d", location);
 exit
} ■
```

[STEP-3] Appendix-A 참조

[STEP-1.1]에서 [STEP-1.2]까지는 ESTGVP의 그래픽 정보와 노드의 조건정보의 분석을 통해서 이루어지며, [STEP-2]는 [STEP-1]까지의 결과와 텍스트 정보를 연결하여 매핑 시키는 것을 통해서 변환이 이루어진다. C언어로의 변환은 [STEP-2]에서 나타나므로, [STEP-3]에서는 Tcl 언어로의 변환된 결과를 보여주고 있다(Appendix-A 참조).

5. ESTGExe : 프로그램 실행

ESTG의 실행은 Tcl 언어를 기반으로 하여 수행된다. 따라서 ESTGTrans에 의해 Tcl로 변환되어 임시파일(temp.dll)에 기록된다. 이 임시파일은 ESTGVP에서 제공되는 입력 library 파일과 결합되어 수행 가능한 Tcl 프로그램(exe.dll)이 생성되며, Tcl/Tk에서 수행된다. 4장에서 예로 들은 이진탐색 ESTG는 실행을 위해 다음과 같이 변경되어 "exe.dll"로 저장이 된다. ESTGTrans에 의해 변환된 Tcl 프로그램과 ESTGExe에 의해 생성된 실행 파일(exe.dll)은 입력 library와의 결합만을 제외하면 동일한 내용으로 구성되고 있다.

[exe.dll]

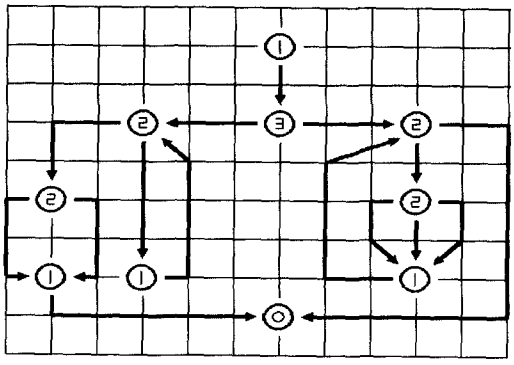
```
global scan_flag scan_start val_exe
set scan_flag false
#frame .tx
#text .tx.text -state disabled
proc scanf { } {
global scan_flag scan_start
  .translate.text configure -state normal
  set scan_start [.translate.text index insert]
  .translate.text configure -state normal
  set scan_flag true
  focus .translate.text
  vwait scan_flag
  return -1
}

bind .translate.text <Key> {
global scan_flag scan_start val_exe
   if { %k == 13 || %k == 32 } {
      if { $scan_flag  == "true" } then {
         .translate.text insert end "\n"
         set val_exe [.translate.text get $scan_start
         [.translat e.text index insert]]
         .translate.text tag add scan $scan_start
         [.translate. text index insert]
         .translate.text tag configure scan -foreground blue
         set val_exe [string trimright [string trimleft
         $val_e xe]]
         if {$val_exe != "" } then {
            set scan_flag false
            .translate.text configure -state disabled
         }
      }
   }
}
#pack .translate.text
#pack .translate
# status node0

# proces arc6
#declartion :  int  S
set S { 3 4 6 7 10 12 15 17 28 38 39 40 42 48 56 75 78
79 80 82 }
#declartion :  int  low high mid x location

# status node2
# proces arc7
set low  1
set high  20
set location  1

.translate.text configure -state normal
.translate.text insert end "Data : $S \n "
.translate.text configure -state disabled
.translate.text configure -state normal
.translate.text insert end "Input Search Data : \n "
.translate.text configure -state disabled
scanf
set x $val_exe

# status node4
while { $low <= $high } {

# proces arc8
set mid [expr  ( $low + $high ) / 2  ]

# status node5
if {  $x ==   [lindex $S $mid ]   } {
# proces arc10
set location  $mid
break
}   elseif { $x <  [lindex $S $mid ]   } {
# proces arc9
set high [expr  $mid -1  ]
}   elseif { $x >  [lindex $S $mid ]   } {
# proces arc11
set low [expr  $mid + 1  ]
}
# status node3
# proces arc12
}
# proces arc13

.translate.text configure -state normal
.translate.text insert end "Data position : $location \n "
.translate.text configure -state disabled

# status node1
#return ■
```

이렇게 변경되어 임시 저장된 파일은 Tcl/Tk의 번역기의 도움을 받아 자체 수행된다.

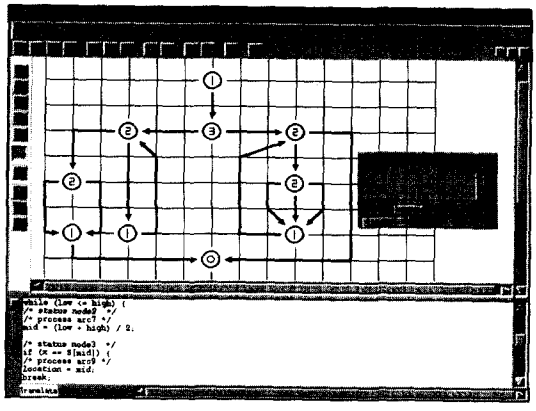
6. 순차탐색과 이진 탐색의 병행 수행에 적용

본 장에서는 순차탐색과 이진탐색의 병행적 수행을 ESTGVP에서 구현하는 예를 보여주기로 한다. 순차탐색과 이진 탐색의 병행적 수행을 ESTGEdit에서 노드와 에지를 사용하여 편집한 모델이 (그림 9)이다. 각 노드 혹은 에지에 기록되는 조건 혹은 action은 C 언어를 기반으로 기술하며, (그림 10)에 보여준 action 창을 통해 입력되어 텍스트 파일로 저장된다.

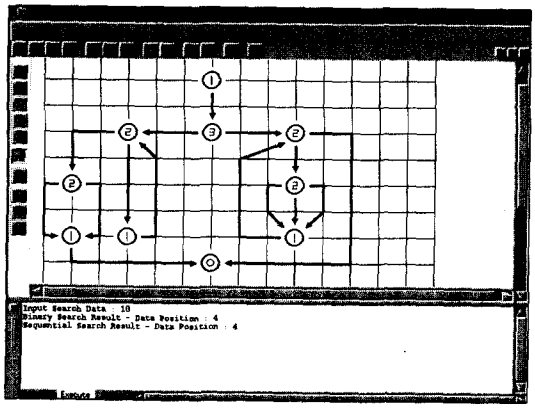
ESTGTrans에 의한 C언어와 Tcl 언어로의 변환과 Tcl/Tk에서의 실행은 (그림 10)과 (그림 11)에서 보여준 하단의 결과 창에서 이루어진다. (그림 10)는 C언어로 변환했을 경우의 변환결과이고(해당결과는 Appendix-B 참조), (그림 11)은 (그림 10)를 실행했을 경우 실행되는 상태이다. 입력을 받아서 순차탐색과 이진탐색을 병행적으로 실행하고 결과 값을 출력한다.



(그림 9) 순차탐색·이진탐색을 병행적으로 수행하는 ESTG



(그림 10) C언어로 변환



(그림 11) 순차·이진탐색의 병행적 실행

7. 결론 및 향후 연구

프로그래밍에 시각성을 첨부한 시각 프로그래밍은 프로그램 작업을 보다 쉽고 간단하게 해주며 프로그램의 이해도를 높여준다. 그러므로 그래픽 도구를 언어의 일부로 사용하는 시각언어는 의미 있는 언어라 할 수 있다.

본 논문에서 제안한 ESTGVP, 병렬 시각 프로그래밍 시스템은 확장 상태 그래프(ESTG)를 그래픽 도구로 사용해서 시각언어에 접근한 방식으로 그래프와 문자 인터페이스를 함께 제공해서 사용하기 쉽고, 그래프를 통해서 구조를 이해하기 쉬우며, 어렵다고 알려진 병렬성을 쉽게 표현하는데 도움을 주며, 이식성이 좋다. 또한 ESTG는 비동기식 혹은 동기식 병렬성과 완벽한 제어구조를 가지고 있으며 이해하기가 용이하여 손쉽게 사용할 수 있다. 그러나 그 자체에 자료의 흐름을 나타내지는 못한다는 단점을 가진다.

향후 연구과제로는 토큰 메카니즘을 적용한 시각적 실행 및 병렬 혹은 분산 target machine의 구축과 병렬언어로의 확장 등을 들 수 있으며, 그 실행의 효율성을 검증해 볼 것이다.

부록 A : Tcl로 변환된 이진 탐색 ESTG 예제

```
# proces arc6
#declartion :  int  S
set S {3 4 6 7 10 12 15 17 28 38 39 40 42 48 56 75 78 79 80 82}
#declartion :  int  low high mid x location

# status node2
# proces arc7
set low  1
set high  20
set location  1

.translate.text configure -state normal
.translate.text insert end "Data : $S \n "
.translate.text configure -state disabled
.translate.text configure -state normal
.translate.text insert end "Input Search Data : \n "
.translate.text configure -state disabled
scanf
set x $val_exe

# status node4
while { $low <= $high } {

# proces arc8
set mid [expr  ( $low + $high ) / 2  ]

# status node5
if {  $x ==   [lindex $S $mid ]   } {
# proces arc10
```

```
set location  $mid
break
}   elseif { $x <  [lindex $S $mid ]   } {
# proces arc9
set high [expr  $mid -1  ]
}   elseif { $x >  [lindex $S $mid ]   } {
# proces arc11
set low [expr  $mid + 1  ]
}
# status node3
# proces arc12
}
# proces arc13

.translate.text configure -state normal
.translate.text insert end "Data position : $location \n "
.translate.text configure -state disabled

# status node1
#return ■
```

부록 B : C로 변환된 순차탐색과 이진탐색 병행 프로그램

```
#include <stdio.h>

main ( ) {

int S[20] = {3,4,6,7,10,12,15,17,28,38,39,40,42,44,56,77,78,79,89,82};
int low,high, mid,location, seq_point, x;
int pid;

low = 1;
seq_point = 0;
high = 20;
location = -1;
printf ("Input Search Data: " );
scanf ("%d",&x);
pid = fork();
if (pid == 0) {
  while (x != S[seq_point] && seq_point < 20) {
     seq_point = seq_point + 1;
  }
  if (seq_point == 20)
       seq_point = -1;
  printf("Sequential Search Result - Data Position : %d\n", seq
  _point);
  exit(1);
} else if (pid > 0) {
  while (low <= high) {
       mid = (low + high)/2;
       if (x == S[mid]) {
          location = mid;
          break;
       } else if (x < S[mid]) {
          high = mid -1;
       } else if (x > S[mid]) {
          low = mid +1;
       }
  }
  printf("Binary Search Result - Data Position : %d\n",
  location);
  exit(1);
} else {

  printf("Error \n");
  exit(1);
}
} ■
```

참 고 문 헌

[1] S. S. Yau and P. Grabow, "A Model for Representing Programs Using Hierarchical Graphs," IEEE Trans. on SE, Vol.SE-7, No.6, Nov. 1981.

[2] S. P. Reiss, "An Object-Oriented Framework for Graphical Programming," Tech. Rep. No.CS-86-17, Dept. of Comp. Sci., Brown Univ., March 1986.

[3] G. P. Brown and C. F. Herot, P. Souza, "Program Visualization : Graphical Support for Software Development," IEEE Computer, Aug. 1985.

[4] R. J. K. Jacob, "A state Transition Diagram Language for Visual Programming," IEEE Computer, Aug. 1985.

[5] E. P. Glinert and S. L. Tanimoto, "Pict : An Interactive Graphical Programming Environment," IEEE Computer, Vol.17, No.11, Nov. 1984.

[6] M. Moriconi and D. F. Hare, "Visualizing Progem Designs Through PegaSys," IEEE Computer, Aug. 1985.

[7] Won-Ho Chung, "A Petri Net Approach to Graphical Programming," Proc. of 1996 ITC-CSCC, Vol.2, July 1996.

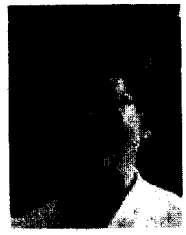
[8] M. Usher and D, Jackson, "A Concurrent Visual Language Based on Petri Nets," Proceedings of VL'98, 1-4 September, 1998, Nova Scotia, Canada, IEEE Computer Society Press.

[9] G. M. Karam, "An Icon-Based Design Method for Prolog," IEEE Computer, Jan. 1988.

[10] J. Johnson, T. L. Roberts, et al, "The Xerox Star : A Restospective," IEEE Computer, Sep. 1989.

[11] M. Ackroyd and D. Daum, "Graphical Notation for object-oriented design and programming," Journal

of OOP, Jan. 1991.
[12] A. Repeonning, "Agentsheets : A Medium for Creating Domain-Oriented Visual Languages," IEEE Computer, Mar. 1995.
[13] G. Raeder, "A Survey of Current Graphic Programming Techniques," IEEE Computer, aug 1985.
[14] Shi-Huo Chang, "Visual Language : A Tutorial and Survey," IEEE Computer, Jan. 1987.
[15] J. C. Browne, M. Azam and S. Sobek, "CODE : A Unified Approach to Parallel Programming," IEEE Software, 1989.
[16] J. C Brown and S. I Hyder, "Visual Programming and Debugging for Parallel Computing," IEEE Computer, Spring 1995.
[17] E. Ghittori, M. Mosconi, M. Porta "Designing new Programming Constructs in a Data Flow VL," Proceedings of VL'98, 1-4 September, 1998, Nova Scotia, Canada, IEEE Computer Society Press.
[18] N. C. Shu, "Formal : A Forms-Oriented, Visual-Directed Application Development System," IEEE Computer, Aug. 1985.
[19] Takayuki Dan Kimura, et al, "Form/Formula A Visual Programming Paradigm for User-Definable User Interfaces," IEEE Computer, Mar. 1995.
[20] A. L. Ambler and M. M. Burnett, "Influence of Visual Technology on the Evolution of Language Environments," IEEE Computer, Oct. 1989.
[21] Takao Shimomura and Sadahiro Isoda, "Linked-List Visualization for Debugging," IEEE, May. 1991.
[22] R. Hesketh, "Perly-UNIX with Buttons," Software-Practice and Experience, Vol.21(11), Nov. 1991.
[23] M. Bhattacharyya, et al, "A Visual Process Connector for Unix," IEEE Computer, July 1988.
[24] B. Melamed and R. J. T. Morris, "Visual Simulation : The Performance Analysis Workstation," IEEE Computer, Aug. 1985.
[25] Shi-Kuo Chang, G. Costagliola, G. Pacini, M. Tucci, G. Tortora, Bing Yu and Jung-Sheng Yu, "Visual Language System For User Interfaces," IEEE Trans. on SE, Vol.12 No.2 Mar. 1995.
[26] M. Burnett and M. J. Baker, "A Classification System for Visual Programming Languages," Tech. Report93-60-14, OSU, June 1994.
[27] M. Burnett et al, "Visual Object-Oriented Programming," Manning Pub., 1995.
[28] 허혜정, 정원호 "확장 상태 전이 그래프에 기초한 시각 병렬 프로그래밍", 한국 정보처리학회 춘계 학술대회 논문집, 1999.



정 원 호

e-mail : whchung@center.duksung.ac.kr
1979년 서울대학교 전자공학과 (학사)
1981년 한국과학기술원 전기 및 전자공학과(석사)
1989년 한국과학기술원 전기 및 전자공학과(박사)
1979년~1982년 대한전선(주)
1983년~1984년 대우통신(주)
1993년 IBM T. J. Watson 연구소 방문 연구원
1989년~현재 덕성여자대학교 컴퓨터과학부 교수
관심분야 : 분산 및 병렬처리, 시각프로그래밍, 모빌 컴퓨팅 등

허 혜 정

e-mail : hjhur@center.duksung.ac.kr
1999년 덕성여자대학교 전산학과 (학사)
1999년~현재 덕성여자대학교 . 대학원 전산 및 정보 통신학과 석사과정
관심분야 : 시각프로그래밍, 알고리즘, 그래프드로잉 등