소프트웨어 개발을 위한 통합 점진 시스템 구축

한정란
협성대학교 경영정보학부

Building Integrated Increment System for Software Development

Junglan Han
Management Information Science, Hyupsung University

요 약

본 논문에서는 소프트웨어의 재사용성과 생산성을 향상시키기 위해, 에디팅, 디버깅, 점진 번역 및 실행을 하나의 통합 환경으로 구성하여 보다 친근하고 편리하게 사용할 수 있는 사용자 인터페이스를 제공하는 소프트웨어 개발을 위한 통합 점진 시스템을 구축하고자 한다. 객체 지향 언어인 IMPLO(IMPerative Language with Object) 언어를 EBNF 표기법으로 정의하고 이 언어에 대한 점진 해석기를 구현하고 에디터와 디버거를 가진 통합 점진 시스템을 구축한다. 본 통합 점진 시스템은 SUN 1000에서 Lex와 Yacc을 이용해서 C 언어로 구현하고 사용자 인터페이스는 X로 구현하였다.

1. 서론

소프트웨어가 대형화되고 고도화되어 감에 따라 소프트웨어의 유지 보수와 재사용이 점점 중요한 과제로 부각되고 있다. 소프트웨어 개발 과정에 관련된 모든 작업들을 보다 능률적으로 처리하도록 지원하는 도구에 대한 연구가 활발하게 진행되고 있다.

응용 소프트웨어나 유틸리티를 개발하려고 할 때 unix 계열의 환경에서 프로그램 개발자는 다양한 시스템 유틸리티인 에디터, 컴파일러, 링커 및 로더 등을 별개의 단계로 반복적으로 사용하면서 완성된 프로그램의 결과를 출력하는 과정을 통해 소프트웨어를 비효율적으로 생산하고 소프트웨어를 개발하는데 상당한 시간을 소요하고 있다.

프로그램 개발 환경에서 프로그래밍 생산성과 효율성을 향상시키기 위해 개발 단계에서 필요한 에디팅, 컴파일링, 디버깅, 및 실행을 하나의 통합 환경으로 구축하여 보다 친근하고 편리하게 사용할 수 있는 사용자 인터페이스를 제공하는 통합 시스템이 필요하다. 이러한 소프트웨어 개발 환경에서 개별적인 작업을 하나의 인터페이스로 통합한 프로그램 개발 시스템이라는 통합 도구가 필요하고 이 환경에서 중요한 도구가 되는 것이 바로 점진 번역기이다. 점진 번역기는 프로그래머가 원시 프로그램을 변경할 때 작동되는 것으로 전체 프로그램을 다시 번역하지 않고 필요한 부분만을 다시 번역하는 방법으로 전체를 번역한 것과 같은 결과를 얻을 수 있다.

본 논문에서는 소프트웨어의 생산성과 실행 효율성을 향상시키기 위해 점진 번역기를 해석방법을 사용하여 구현한 점진 해석기를 기반으로 하여 객체 지향 언어인 IMPLO(IMPerative Language with Object) 언어에 대한 소프트웨어 개발 통합 시스템을 구축하고자 한다. 이 통합 시스템은 프로그램 개발 단계에서 수행되는 에디팅, 인터프리팅(해석), 디버깅 및 실행을 하나의 사용자 인터페이스로 구현한 프로그램 개발 지원 시스템이다.

2. 관련 연구

소프트웨어 개발 환경에서 통합 점진 시스템의 핵심 도구인 점진 번역기를 구현하는 방법에는 해석 방식을 사용하는 것과 컴파일 방식을 사용하는 방법이 있는데 대부분의 경우 컴파일 기법을 사용해서 점진 번역기를 구현하고 있다.

점진 컴파일러를 구현하는 가장 간단한 방법은 언어에서 컴파일할 수 있는 최소의 단위를 결정하여 소스 코드가 변경되었을 때 그러한 가장 작은 단위를 다시 컴파일 하는 방법이 있다.

컴파일 방법으로 구현된 점진 시스템들 중에서 ALOE 에디터[7]나 Gandalf 프로젝트[8]에서는 언어의 추상 구문과 관련하여 작용 루틴을 작성하여 점진 컴파일러를 구현하였다.

점진 컴파일러를 구현하기 위해 속성 문법을 사용했을 경우 문법에서 자동적으로 유도되는 속성들간의 종속 정보를 사용해서 주어진 속성 문법에서 최소한으로 필요한 속성이 다시 평가되어 변경된 부분에 영향받는 부분만을 다시 컴파일 하게 된다. 이 방법으로 구현한 대표적 시스템으로 POE 에디터[9]가 있다.

해석 방법을 사용하는 점진 해석기에서는 속성 문법을 사용하여 속성들간에 종속성을 표현하고 변화가 생겼을 때 변화된 속성에 종속하는 속성들을 점진 속성 평가 방법을 사용하여 찾아내게 된다. 이 방법으로 구현한 대표적 시스템으로 Cornell Program Synthesizer[6]가 있다.

본 연구에서는 기존의 시스템과 다르게 소프트웨어의 재사용성을 향상시키고 개발자의 편의를 도모하기 위해 프로그램 개발자가 프로그램을 수정할 경우, 수정된 부분에 대한 점진 스캐닝과 점진 파싱을 수행한 후 이 부분에 영향받는 부분들을 동적 의미 분석 방법을 통해 다시 해석하여 실행하게 된다. 기존의 연구와 다르게 동적 의미 구조를 잘 표현한 새로운 작용식(action equation)[1, 3, 4, 5]과 속성간의 종속성을 표현한 종속 차트(*dependency chart*)[1, 2, 4, 5]를 통하여 수정된 명령문에 영향받는 변수들의 속성을 다시 계산하게 된다.

3. 점진 해석 방법

통합 소프트웨어 개발 환경에서 중요한 도구가 되는 것이 점진 번역기이고 이러한 번역기는 프로그래머가 원시 프로그램을 변경할 때 작동되는 것으로 전체 프로그램을 다시 번역하지 않고 필요한 부분만을 다시 번역하는 방법이다. 본 논문에서는 점진 번역을 해석(interpretation) 방법을 사용하여 구현하였다. 해석기는 초보자가 프로그램을 빠르게 개발할 수 있고 사용하기 쉬운 장점이 있다.

본 논문의 핵심 도구인 점진 해석기를 구현하려면 실제로 언어를 구현하는데 필요한 동적 의미 구조를 표현하는 새로운 작용 식(action equation)[1,3,4,5]을 사용하여 동적 의미 분석 방법으로 실행하게 된다. 기존의 연구와 달리 수정된 부분에 대해 변화 파급(change propagation) 과정이 복잡하게 수행되는 점을 개선하기 위해, 속성들간의 종속성을 나타내는 종속 차트(dependency chart)[1,2,4,5]를 만들어 수정된 부분과 그 부분에 영향받는 부분을 찾아내어 이 부분들만을 다시 실행하게 된다.

3.1 IMPLO 언어

어떤 언어에 대한 번역기를 구현하기 위해서는 언어의 동적인 의미 구조를 잘 표현하는 것이 중요하다. 따라서 본 논문에서는 속성 문법을 확장하고 변형하여 정적이고 동적인 의미 구조를 표현하는 작용식(action equation)을 사용함으로써 점진 해석 방법을 수행하고자 한다. 본 점진 해석 방법을 실제의 언어로 구현하여 효율성을 평가하기 위해 (그림 1)에 IMPLO 언어를 EBNF 표기법으로 각 명령문의 구문을 정의한 다음 이 언어에 대한 점진 해석기를 구현한다.

```
<program>::=<module_list><main_program>
<main_program>::=program <prog_name><decl_part><main_body>
<main_body>::=begin <statement_list> end
<module_list>::=<module>{<module>}
<module>::=<sub_program> | <class_module>
<class_module>::=class<class_name>[<derived_class>](<formal_list>)<class_body>
<derived_class>::= : parent <class_name>
<class_body>::=begin <class_stmt_list> end
<class_stmt_list>::=<class_stmt>{<class_stmt>}
<class_stmt>::=[<modifier>]<statement>
<sub_program>::=<sub_heading><sub_body>
<sub_body>::=begin <sub_statement_list> end
<sub_heading>::=<sub_keyword><sub_name>(<formal_list>)
<sub_keyword>::=procerure | function
<decl_parts>::=<declaration>{<declaration>}
<declaration>::=<class_id><identifier>
<modifier>::=public: | private:
<statement_list>::=<statement>{<statement>}
<statement>::=<in_statement>
              |<out_statement>
              |<ass_statement>
              |<if_statement>
              |<for_statement>
              |<while_statement>
              |<call_statement>
```

(그림 1) IMPLO 언어

IMPLO 언어는 명령형 언어이면서 객체(object)를 다룰 수 있는 언어로 참고문헌 [1]에 자세히 기술되어 있다. 일반적인 명령형 언어의 명령문들인 배정문, if 문, while 문, for 문 및 입출력문을 다루고 있으며, 언어에서 호출할 수 있는 부 프로그램으로는 프로시저(procedure)와 함수(function)가 있다. 부 프로그램에서 매개변수(parameter)를 호출하기 위해 값 호출(call-by-value) 과 참조 호출(call-by-reference) 방법이 사용된다.

객체(object)를 표현하기 위해 사용자가 정의한 자료형이 클래스이다. 새로운 클래스를 정의할 때는 이 객체를 나타내는데 필요한 자료 부분과 이 객체에 적용할 수 있는 연산 부분을 정의해야 한다. IMPLO 언어에서는 자료를 사용하기 위해 그 자료를 선언할 필요가 없고 배정문으로 바로 사용하면 된다. 메인에서 사용되는 동일한 이름의 자료를 참조하기 위해 '::'를 사용한다. 예를 들면 **main::x** 는 메인 프로그램에서 사용되는 변수 x를 의미한다.

IMPLO 언어의 클래스는 공용(public) 파트와 전용(private) 파트로 구성되어진다. 전용 부분은 사용자가 이 자료를 직접 이용할 수 없고 클래스 내에 선언된 멤버 함수를 통해 사용할 수 있다. 공용 파트의 자료를 참조하기 위해서는 **class_name.id와** 같이 작성한다. 예를 들면 **date** 라는 객체의 변수 **x** 를 참조하기 위해 **date.x** 라고 표현한다.

3.2 작용 식

어떤 언어에 대한 번역기를 구현하기 위해서는 언어의 동적인 의미 구조를 잘 표현하는 것이 중요하다. 따라서 본 연구에서는 속성 문법을 확장하고 변형하여 정적이고 동적인 의미 구조를 표현하는 작용 식을 제시함으로써 본 통합 시스템의 핵심이 되는 점진 해석기를 구축하려고 한다.

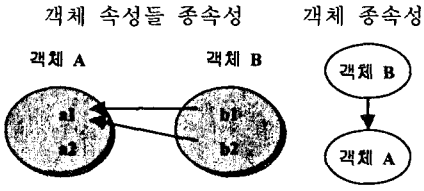
작용 식에는 Execute equation, Evaluate equation, Eval_rel equation, Eval_par equation, Wait_in equation, 및 Eval_out equation 이 있다. 작용식에 대한 내용은 참고문헌 [1, 3]에 자세히 기술되어 있다. 작용 식 중에서 클래스 처리를 위한 Execute equation은 다음과 같고 나머지 식은 참고문헌 [3]에

자세히 기술되어 있다.

<class 문의 Execute equation>
· Execute [**class** <class_name>]→
  class_name.env ← class_name.name
  make_table(class_name.name, class_name.env)
  class_name.addr ← current_point
· Execute [**class** <class_name><derived_class>]→
  class_name.env ← class_name.name
  class_name.penv ← derived_class.name
  make_table(class_name.name, class_name.env)
  class_name.addr ← current_point
· Execute [**class** <class_id><identifier>]→
  identifier.env ← class_id.name
  make_table(identifier.name, identifier.env)

IMPLO 언어에서는 객체들 간에 종속성을 표시하여야 하고 이를 위해 다음과 같은 방법으로 종속성을 나타낸다. A와 B는 객체이고 A의 속성으로 a1, a2가 있고 B의 속성으로 b1, b2이 있고 각 속성들을 다른 객체에서 사용할 수 있는 "public" 특성을 갖는다고 가정할 때 **A.a1 = B.b1 + B.b2 명령문**에서 객체 A의 속성 a1은 객체 B의 속성 b1과 b2에 종속하게 된다[1]. 즉 객체 B의 속성인 b1과 b2의 값이 변경될 때 객체 A의 속성 a1의 값도 변하게 된다.

객체들 간의 속성들의 값이 변할 때 속성 평가를 효율적으로 수행하기 위해 객체들간의 종속성을 나타내어야 한다. 객체가 가진 속성들에서 이러한 종속성이 존재할 때 객체들간에 종속성을 (그림 2)에서처럼 표현한다.



(그림 2) 객체 종속성

4. 통합 점진 시스템

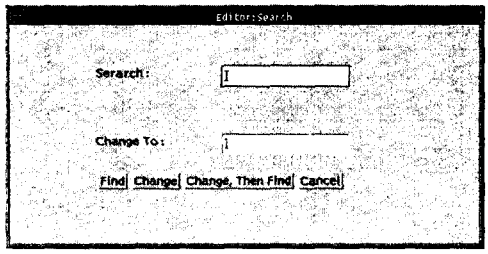
본 논문에서 구현한 소프트웨어 개발을 위한 통합 점진 시스템은 편리한 사용자 인터페이스와 점진 해석을 수행하는 점진 해석기와 프로그램 편집을 위한 에디터와 디버깅시 필요한 정보를 제공하는 디버거로 구성된다. SUN 1000에서 X 윈도우로 사용자 인터페이스를 구현하고 C, Lex, 및 Yacc으로 통합 점진 시스템을 구축하였다.

4.1. 사용자 인터페이스

소프트웨어 개발을 편리하게 수행하기 위해 사용자 인터페이스에서 사용 가능한 메뉴에는 File, Edit, Interpreter, Incrementor, Debug, help가 있다. File 메뉴는 파일을 열거나 저장하는 등의 파일 처리에 관련된 메뉴이고 Interpreter 메뉴는 전체 프로그램 파일을 해석하여 실행하는 메뉴이고 Incrementor 메뉴는 프로그램이 수정되었을 때 점진적으로 실행하기 위한 점진 해석기(점진기)에 관련된 메뉴이고 Debug 메뉴는 디버깅을 수행하기 위한 메뉴이다. 본 점진 해석기의 경우 사용자가 Interpreter 메뉴와 Incrementor 메뉴 중 하나를 선택하여 프로그램을 실행할 수 있다.

4.2 에디터 및 디버거

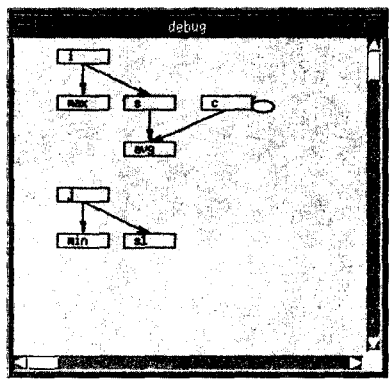
edit 메뉴에는 cut, clear, copy, paste, search가 있다. 특히 search 메뉴를 누르면 실행이 되는 search popup dialog가 (그림 3)에 나와 있고 이 search popup dialog에서 찾고자하는 문자열 입력부분과 바꾸고자 하는 문자열을 입력부분에 입력을 하면 문자열이 다이나믹하게 바뀌게 된다.



(그림 3) search 팝업 셀

프로그램을 개발할 때 의미 오류 정보를 적절하게 주기 위해 디버거가 필요하다. 본 논문에서 사용한 종속 차트를 통해 변수의 값에 변화가 생기면 그 변수 값이 변함에 따라 영향받아 값이 변하는 변수들과 그 값을 표시해 주어 사용자가 의미 오류를 발견하는 작업을 도울 수 있다.

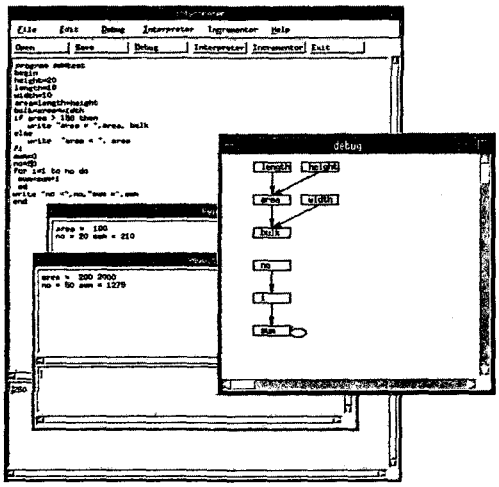
프로그램을 수정하면서 변수 값이 변함에 따라 값이 변하는 변수들을 (그림 4)에서 처럼 시각적으로 보여주면 디버깅 과정이 훨씬 간단하게 수행될 수 있다.



(그림 4) 종속성 팝업 셀

디버거는 "Debug" 메뉴를 선택했을 때 작동하는 것으로 종속 차트 중 종속성을 나타내는 포인터를 시각적으로 보여주게 된다.

디버깅하면서 점진 해석기를 수행한 화면이 (그림 5)에 표시되어 있다.



(그림 5) 디버거 실행 인터페이스

4.3 점진 해석기의 실행 효율성 분석

점진 해석의 실행 효율성을 다양하게 분석하기 위해 세 가지 유형의 프로그램에 대해 점진 해석이 수행되는 결과를 살펴보았다. 점진 실행의 효율성을 분석하기 위해 세 가지 형태의 수정인 배정문들, IF 문, 루프이 변경되는 경우에 세 가지 유형의 프로그램 예제에 대해 점진 해석을 실행해 보았고 점진 해석의 성능을 <표 1>에서 표시하였다.

<표 1> 세 가지 예제 프로그램의 실행 시간 단위: ms

프로그램 유형	배정문		IF 문		Loop 문		평균		개선 비율
	전체	점진	전체	점진	전체	점진	전체	점진	
M1	6	0.5	7	0.5	8	9	7	3.3	52.9%
M2	12	1	13	1	15	17	13.3	6.3	52.6%
P1	8	0.5	9	0.5	11	12	9.3	4.3	53.8%
P2	13	1	15	1	17	19	15	7	53.3%
S1	11	1	12	1	14	16	12.3	6	51%
S2	20	1	22	1	24	27	22	9.7	55.9%

M1,M2: 최대값, 최소값, 합과 평균을 구하는 프로그램
P1, P2: 급료을 계산하는 프로그램
S1, S2: 합계를 구하는 프로시저를 호출하는 프로그램

<표 1>을 통해 알 수 있듯이 배정문들이나 IF문을 수정했을 경우는 수정된 부분만을 번역하여 결과를 얻는 시간은 프로그램 전체를 실행한 시간에 비하면 0.5~1 ms로 아주 적은 시간이다. 그러나 루프를 수정했을 경우는 점진 평가에 대한 오버헤드까지 포함되기 때문에 전체 프로그램을 실행한 시간보다 더 많은 시간이 소요된다. 프로그램을 수정한 후 프로그램 전체를 실행하는 것과 점진 실행을 수행하는 것의 실행 효율성을 비교하여 보면 51%~55.9% 만큼 빠르게 실행되는 것을 알 수 있다.

5. 결론

본 논문에서는 소프트웨어의 생산성을 향상시키기 위해 에디팅, 인터프리팅, 디버깅 및 실행과정이 하나의 인터페이스에서 수행될 수 있는 통합 점진 시스템을 구축하였다. 본 시스템에서는 사용자가 편리하게 소프트웨어 시스템을 개발할 수 있도록 유용한 사용자 인터페이스를 제공해 준다. 에디터와 파일 처리에 관련된 메뉴와 에디팅을 위한 메뉴를 제공하고 디버깅시 필요한 정보를 주는 디버거는 수정된 변수 값에 영향받아 변하는 변수들을 시각적으로 보여주게 된다. 특히, 점진 해석을 수행하는 점진기와 전체 프로그램을 수행할 수 있는 인터프리터 메뉴를 제공하여 주어, 사용자는 프로그램을 수정한 후 전체 프로그램을 다시 실행하거나 점진기를 사용해서 점진 해석을 수행할 지 선택할 수 있다.

본 논문에서 핵심 도구가 되는 점진 해석기는 새로운 IMPLO 언어를 정의하고 이 언어에 대한 점진 해석기를 구현하였다. 수정된 부분만을 토큰으로 나누어 파싱하여 의미 분석 과정을 거치게 된다. 이 과정에서 수정된 부분에서 값이 변한 변수가 있을 경우 종속 차트를 통해 그 변수에 영향받는 변수가 속해 있는 명령문들을 다시 실행하게 된다. 이러한 점진 해석기의 실행 결과는 전체 프로그램을 번역했을 때와 같은 결과를 얻을 수 있었다. 세 가지 유형의 프로그램을 사용해서 본 점진 해석기의 실행 효율성을 검사해 보았으며 평균적으로 고려해 볼 때 전체 프로그램을 실행했을 경우 보다 약 50% 정도의 속도 개선 효과를 얻을 수 있었다.

참고문헌

[1] 한정란 "작용 식 기반 점진 해석" Ph. D Thesis 이화여대 1999.

[2] 이기호 한정란 "순환 속성 문법의 효율적 점진 평가 기법" 정보과학회 논문지 제 21권 6호 pp. 1116~1126 1994.

[3] 한정란 이기호 "작용 식 기반 점진 해석기" 정보과학회 논문지 제 26권 8호 pp. 1018~1027 1999.

[4] 한정란 이기호 "종속 차트를 사용한 점진 해석 시스템 구축" 정보과학회 춘계 학술발표논문집 제 26권 1호 pp. 87~89 1999.

[5] Han junglan "The Design and Implementation of an Advanced Incremental Interpreter" In Proceeding of ICITA 2001 pp.101~108 KIPS 2001.

[6] T. Teitelbaum and T. Reps "The Cornell *Program Synthesizes*: A Syntax-directed Environment" Communication ACM Vol 24(9) pp. 563~573 1981.

[7] Raul Medina Rora and David S. Notkim "ALOE users' and implementers' guide" Carnegie-mellon Computer Science Depart. Research Report CS-81-145 1981.

[8] A. N. Habermann "The Gandalf Research project" Computer Science research Review Carnegie-Mellon University 1979.

[9] Charles N. F., Greg J. and Jon M. "An Introduction to Editor Allen Poe" Univ. Wisconsin-Madison TR 451 1981.

[10] John F. Beetem and Anne F. Beetem "Incremental Scanning and Parsing With Galaxy" IEEE Transactions on Software Engineering Vol. 17 No. 7 pp. 641~651 1991.

[11] Roger Hoover "Alphonse : Incremental Computation as a Programmer Abstraction" ACM SIGPLAN Notices pp. 261~272 1992.