ソフトウェア工学 58－19
（１９８８．２．５）

統合化プログラミング環境ＭＵＳＥの構築

佐藤　康臣*，天満　隆夫*，永良　裕*，坪谷　英昭*，田中　稔**，市川　忠男**
*広島大学大学院　　**広島大学工学部

ソフトウェア開発の生産性および信頼性の向上を目的として、統合化プログラミング環境ＭＵＳＥを構築した。ＭＵＳＥは、プログラミング言語の構文や静的意味に関する知識を取り込むことにより、各言語に対し専用化された機能を提供する。

本稿では、ＭＵＳＥの基本機能、ユーザインタフェース、および静的意味チェックの記述のために新たに導入された属性リレーションについて述べる。

Construction of an Integrated Programming Environment, MUSE

Yasuomi SATO, Takao TENMA, Yutaka NAGARA,
Hideaki TSUBOTANI, Minoru TANAKA, and Tadao ICHIKAWA

Faculty of Engineering, Hiroshima University
Shitami, Saijo-cho, Higashi-Hiroshima, 724, Japan

In order to increase productivity and reliability in software development, we have already constructed an integrated programming environment named MUSE. MUSE provides the facility specific to a paticular programming language by utilizing the knowledge on syntax and static semantics of the language.

In this paper, we describe basic facilities of MUSE, the user-interface, and attribute-relation newly introduced for describing static semantic checking.

1．はじめに

ソフトウェア開発の生産性および信頼性の向上を目的として、

（1）構文チェックやコード生成などの特定のプログラミング言語向きの多くの機能を提供する、

（2）対話的かつインクリメンタルなプログラム開発を支援する、

（3）ツールを密に結合することによって、プログラムの編集、デバッグ、実行をモードの切り換えなしに行なえる、

といった特長を持つ多くの統合化プログラミング環境が開発されている[1]-[3]。

このような特定のプログラミング言語に依存した機能を提供する統合化プログラミング環境は、個々のプログラミング言語ごとに環境を構築しなくてはならないといった問題がある。環境の構築の負荷を軽減するために、プログラミング言語に関する記述から統合化環境やツールを生成するシステムの研究がなされている[4]-[6]。

筆者らは、プログラミング言語の構文や静的意味等に関する知識を取り込むことにより、その言語向きの機能を提供する環境ＭＵＳＥ（Multiple-Language Support Environment)[7]を開発した。ＭＵＳＥは、上記の統合化プログラミング環境の特長(1)-(3)に加え、

（4）統合化プログラミング環境を与えられたプログラミング言語に関する知識から生成する、

（5）新たな機能の追加や既にある機能の変更が容易である、

（6）ディレクトリ情報やバージョン情報といったソフトウェア資源の管理のための情報を扱うことができる、

といった特長を持つ。

本稿では、ＭＵＳＥの基本機能、ユーザインタフェース、および静的意味チェックの記述のために導入された属性リレーションについて述べる。

2．ＭＵＳＥの概要

2．1　ＭＵＳＥの目的

ＭＵＳＥはソフトウェア開発における設計、コーディング、デバッグ、テスト、再利用等を支援し、このようなソフトウェア開発活動の中でプログラマの知的な助手となるようなプログラミング環境の提供を目的としている。現時点において開発しているシステムＭＵＳＥはコーディング、デバッグのフェーズを支援している。このフェーズの中で、

（1）特定の言語向きの編集機能

（2）ブレイクポイントの設定やステップ実行といったデバッグ機能

（3）ディレクトリ、構成管理、バージョン管理など作成されたプログラムを管理する機能

を提供している。

2．2　ＭＵＳＥの特長

ＭＵＳＥには以下の特長がある。

（1）プログラムの内部表現の統一化

ＭＵＳＥはプログラムを属性付き抽象構文木（以下、構文木という）として保持する。これにより異なる言語間でのツールの共有が可能となる。また、ディレクトリやバージョン管理などの管理情報をプログラムと同様な構文木で表すことにより、環境内部でこれらをプログラムと同様に扱うことができる。

（2）特定の言語向きの機能の提供

ＭＵＳＥは言語の構文・意味に関する知識を用いることにより、その言語向きの機能を提供する。言語や管理情報に対する知識の集まりのことをモジュールと呼ぶ。またプログラムごとにモジュールを切り換えることによって、環境内で複数の言語を扱うことができる。

（3）オブジェクト指向の導入

ＭＵＳＥでは、言語に関する知識の表現、環境内部の処理のためにオブジェクト指向[8]の考えが導入されている。すなわち、特定言語に関する知識はその言語の各構文要素に対するクラスで表され、構文木内のノードは対応するクラスのインスタンスとなる。クラスにメソッドを記述し、ノード間でメッセージ通信を行なうことにより、環境内部の処理を行なうことが可能である。また、クラス間の継承により記述の量を減少させることができる。

（4）内部処理のための様々な機能の提供

ＭＵＳＥでは、メッセージパッシングによって起動されるメソッド、デモンの一種であるプローブ、属性リレーションなどの記述によって実行コードの生成、静的意味チェックなどの内部処理が可能である。

（5）ゲートの導入

ＭＵＳＥでは異なる構文木を関係づけるためにゲート[9]の考えを導入した。ゲートは別の構文木全体を表す１つのノードである。これによりプログラムと管理情報といった異なる種類の構文木を関係づけることが可能となる。

2．3　ユーザインタフェース

ＭＵＳＥは、マルチウインドウベースのユーザインタフェースを提供している。ユーザは主にマウスを用いてプログラムに対する操作を行なう。ユーザの操作する１つのプログラムに対して１つのウインドウが開かれる。このウインドウのことをコンテキストウインドウと呼ぶ。複数のプログラムを操作する場合、複数のコンテキストウインドウがスクリーン上に開かれる。

図１にコンテキストウインドウの例を示す。コンテキストウインドウは次の３つの領域からなる。

（1）プログラム領域

プログラム領域にはプログラムのテキスト表現が表示される。ユーザが操作を行なう対象領域は反転表示によっ

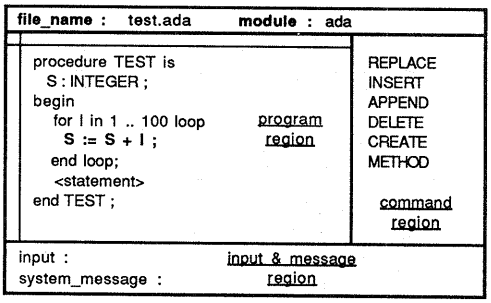


図1　コンテキストウインドウ

て示される。この領域は構文木内の１つの部分木に対応する。ユーザはテキストの一部をマウスでクリックすることによって、対象領域を移動させることができる。

（2）コマンド領域

コマンド領域には、編集やデバッグのためにユーザが実行することができるコマンドが表示される。ユーザは表示されているコマンドのひとつをマウスでクリックすることによって、そのコマンドを実行させる。

（3）入力・メッセージ領域

ユーザがプログラム中に名前やリテラル値を入力するための領域である。また、エラーメッセージなどのシステムからの応答の表示にも用いられる。

２．４　システム構成

図2にＭＵＳＥのシステム構成を示す。ＭＵＳＥは、言語独立な４つのサブシステム、および個々のプログラミング言語やユーザ用に定義される３種類のデータを保持する領域からなる。

（1）オブジェクト管理部（Object Manager）

プログラミング言語に関する知識（モジュール）およびコンテキストを管理する。また、編集機能、およびオペレーティングシステムのファイルシステムとのインタフェースを提供する。

（2）プリティプリンタ（Pretty-Printer）

プログラムの内部表現である構文木(図中、ＡＳＴ)をテキスト表現に変換し、その表示データをウインドウ管理部へ送る。

（3）ウインドウ管理部（Window Manager）

実際の画面出力、およびマウスのクリック等のユーザの操作（イベント）の処理を行なう。

（4）コマンド管理部（Command Interpreter）

ウインドウ管理部からイベントを受け取り、実行されるコマンド、および次のコマンド候補などをコマンド記述に従って決定する。

（5）モジュールスペース（Module Space）

モジュールスペースは複数のモジュールを保持する。モジュールは特定の言語の構文や意味などに関する知識である。

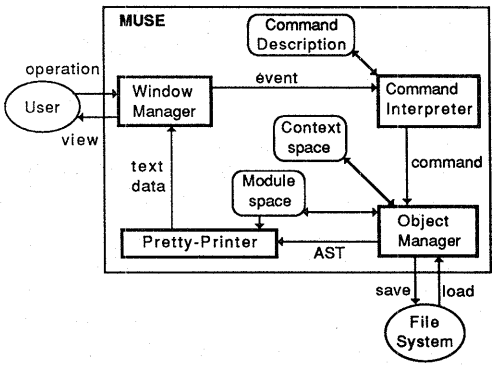


図2　ＭＵＳＥのシステム構成

（6）コンテキストスペース（Context Space）

コンテキストスペースは複数のコンテキストを保持する。コンテキストは操作対象である構文木ごとに作られるオブジェクトである。１つのコンテキストごとに１つのコンテキストウインドウが割り当てられる。コンテキストは構文木のほかウインドウ識別子、実行コード、ファイル名、モジュール名などの情報を保存する。

（7）コマンド記述（Command Description）

ＭＵＳＥではイベントに対するシステムのふるまいを有限状態遷移モデルで表している。つまり、イベントの発生によって、コマンドを実行するとともに状態を遷移する。コマンド記述には、システムの状態、イベント、実行されるコマンド、次の状態の関係等が記述される。

図3にコンテキスト、コンテキストウインドウ、モジュールの関係を示す。１つのコンテキストは、ウインドウ識別子およびモジュール名を保持している。これらによって、コンテキストはスクリーン上のコンテキストウインドウ、およびモジュールスペース中のモジュールと対応づけられる。

次にシステムの動作の概略について述べる。ユーザの操作によって発生したイベントは、ウインドウ管理部によってコマンド管理部に送られる。コマンド管理部はコマンド記述を参照して、起動すべきコマンドを決定する。オブジェクト管理部は、ユーザが操作を行なったコンテキストウインドウに対応するコンテキストの構文木に対してそのコマンドを実行する。そのとき必要に応じて、構文木に対応するモジュール中の各クラスが参照される。コマンドの終了後、プリティプリンタはクラスに記述された出力フォーマットを参照し、構文木をテキスト表現に変換しウインドウ管理部へ送る。ウインドウ管理部はそのテキスト表現をコンテキストウインドウに表示する。

３．プログラムの内部表現と言語に関する知識

プログラムの内部表現である属性付き抽象構文木と言語に関する知識について述べる。

３．１　プログラムの内部表現

ＭＵＳＥでは、プログラムや種々の管理情報は属性付き抽象構文木で表わされる。構文木内のノードは、次節で述べるクラスからのインスタンスであり、どのクラスからのインスタンスであるかのラベルを持つ。図４に抽象構文木の一部とそれに対応するテキスト表現を示す。

構文木内の各ノードは、構造リンクおよび意味リンクによって関係づけられている。図４でwhileノードのconditionリンク、bodyリンクといった構造リンクはノード間の構造的な関係を表す。変数Ａのdeclリンクのような意味リンクは、変数とその宣言部といった意味的な関係を表わす。変数の型やリテラル値などはノードの属性によって表わされる。各ノードがどのような構造リンク、意味リンク、属性を持つかはそのノードに対応するクラスに記述されている。

３．２　言語に関する知識

言語に関する知識は、言語の各構文要素に対するクラスによって表される。クラスは、そのクラスに属するノードの性質やメッセージを受けたときのノードのふるまいに関する情報を保持する。クラスは、クラス名、スーパークラス、構造リンク型、意味リンク型、属性、メソッド、プローブ、属性リレーション、出力フォーマットからなる。図５にwhileクラスの記述例を示す。

各クラスには以下が記述される。

（１）クラス名

クラスの名前。

（２）スーパークラス

言語の構文はスーパークラスと構造リンク型で記述される。スーパークラスはＯＲオペレータのみからなる構文規則から導かれる。例えば、

```
statement ::= if | assignment | loop
```

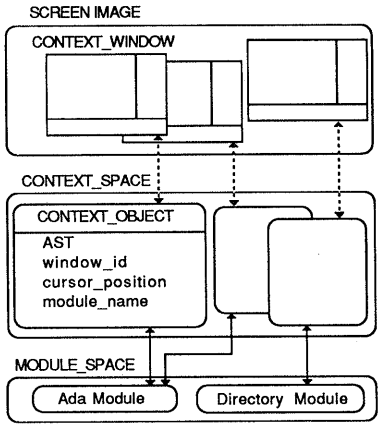
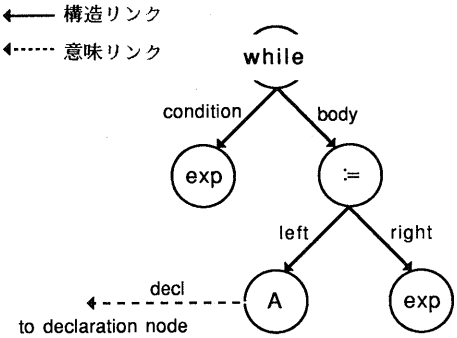


図３　コンテキスト、コンテキストウインドウ、モジュールの関係



（ａ）構文木

```
while <exp> loop
  A:=<exp>;
end loop;
```

（ｂ）テキスト表現

図４　プログラムの内部表現とそのテキスト表現

```
( defclass while
    : super                (loop)
    : struct-link          ((condition : class expression : range (1 1))
                            (body : class statement : range (1 infi)))
    : semantic-link        nil
    : attribute            ((code : default nil))
    : method               ((code-gen (lambda () . . .)))
    : probe                ( ((condition type) (lambda () . . .)) )
    : attribute-relation   nil
    : format               (("while " : level 0) (seq condition : level -1)
                            "loop" (para body : level 1)
                            ("end loop;" : level 0)) )
```

図５　whileクラスの記述

のような構文規則から、if、assignment、loopのスーパークラスとしてstatementが定義されることが導かれる。このときif、assignment、loopはstatementの一種であり、statementの性質を継承すると考えられる。クラスは、そのスーパークラスから、構造リンク型、意味リンク型、属性、メソッド、プローブ、属性リレーションを継承する。ＭＵＳＥでは、複数のスーパークラスからの多重継承を許している。継承により記述の量を減少させることができる。

（3）構造リンク型

構造リンク型は構文規則の中の生成規則から導かれる。例えば、

```
while ::= while  expression  loop
            statement { statement }
          end loop;
```

のような構文規則は、whileノードがexpressionノードを１つだけ持ち、statementノードを１つ以上持つことを意味する。この規則から図５におけるcondition構造リンク型、body構造リンク型が導かれる。

１つの構造リンク型は、リンク名、リンククラス、リンク数制約からなる。リンククラスは、そのリンクに結合されるノードのクラスが、リンククラスまたはそのサブクラスでなければならないという制約を示し、リンク数制約はそのリンク名のリンクと結合できるノード数の範囲を示す。図５の場合、conditionリンクに結合できるノードは１つであり、このノードはexpressionクラス、またはそのサブクラスのノードでなければならない。

（4）意味リンク型

意味リンク型は、変数とその宣言部といった意味的な関係を表す意味リンクに対する制約を与える。意味リンク型は構造リンク型と同じフォーマットをとる。

（5）属性

クラスのインスタンスであるノードが持つ型やリテラル値といった属性が定義される。属性は属性名とデフォルト値からなる。

（6）メソッド

メソッドはＭＵＳＥ内部の様々な処理に用いられる手続きであり、ノードがメッセージを受け取ったとき起動される。メソッドはメソッド名、手続き本体からなる。

（7）プローブ

プローブは、主にプログラムの静的意味チェックを行なうために用いられ、リンク名、属性名またはリンク名、手続き本体からなる。プローブは、リンク名で指定されたリンクで結合されているノードの、属性名またはリンク名で指定された値が変更されたとき自動的に起動される。図５の場合、conditionリンクで結合されているノードのtype属性が変更されたときプローブが起動され、手続き本体が実行される。また、ノードそれ自身の属性を指示するにはリンク名selfを用いる。

（8）属性リレーション

属性リレーションは型チェックなどの静的意味チェックのために用いられる。このクラスのノードの属性と、このノードとリンクによって結合されているノードの属性の値の関係が記述される。詳細は４．２で述べる。

（9）出力フォーマット

構文木がプリティプリンタによってテキスト表現に変換されるときの変換規則が記述される。出力フォーマットの詳細については省略する。

４．ＭＵＳＥにおけるプログラムの作成

ＭＵＳＥが提供する機能とそのメカニズムについて述べる。

４．１　編集機能

ＭＵＳＥは、テンプレートを用いた構造的な編集機能を提供する。テンプレートはクラス名によって呼び出される。ユーザはテンプレートをプログラム中に挿入したり、あるテンプレートを別のテンプレートで置換したり、テンプレートを削除するといった操作によってプログラムを作成する。内部的には、これらの操作は、構文木内のノードの挿入、置換、削除に対応する。またＭＵＳＥは、ユーザの各編集操作ごとに構文チェック、および型チェックなどの静的意味チェックを行なうことによって、プログラムの構文的、静的意味的な正しさを保証する。

次にＭＵＳＥの編集機能の特徴について述べる。

（1）構文チェック

構文チェックは、ＭＵＳＥがクラスに記述されているスーパークラスと構造リンク型を参照することによって行なわれる。ノードが置換されるとき、置換するノードのクラスが、置換されるノードとその親ノード間のリンクのリンククラスのサブクラスであるかどうかチェックされる。ノードが挿入、または削除されるとき、リンク数制約を満足するかどうかチェックされる。

（2）静的意味チェック

プログラミング言語には、構文的な制約の他に静的意味的な制約がある。例えば、代入文の左辺と右辺の型は同じでなければならない、while 文の条件部の型は論理型でなくてはならないといった型制約や未宣言変数は使用できないといった制約である。

一般に、静的意味制約のチェックは、属性文法などを用いたノードの属性値の計算とその値のチェックによって行なわれる。ＭＵＳＥでは静的意味チェックには属性リレーションや、属性値の変更によって起動されるプローブを用いている。４．２で属性リレーションを用いた型チェックについて述べる。

（3）内部クラス

従来、プログラム中で宣言された変数や手続きは変数や手続きを表すノードの名前属性に値を入力することよ

って表されていた。ＭＵＳＥでは、プログラム中で宣言がなされると、それに対応して新しいクラスが作られる。このようにプログラム中の宣言部によって作られるクラスを内部クラスと呼ぶ。これにより、手続き呼び出しや変数の参照などは、内部クラスのインスタンスとして扱われる。

内部クラスの導入により、

（１）未宣言変数や未宣言手続きは使用できない、

（２）構文チェックや静的意味チェックが容易になる、

（３）宣言された名前でテンプレートが呼び出せる、

（４）呼び出されたテンプレートははじめから構文的に正しい形をしている、

といった利点が得られる。例えば、

```
procedure P(X:in INTEGER; Y:in out FLOAT);
```

のような手続きＰの宣言部によってprocedure-callクラスのサブクラスとしての内部クラスＰが作られる。図６に内部クラスＰのテンプレートと対応する構文木を示す。クラスＰのノードには、ＸとＹという２つのリンクが定義される。Ｘのリンククラスは、expressionクラスとなる。なぜならパラメータＸのモードがin、すなわち値による呼び出しを意味し、引数として式をとることができるからである。同様にパラメータＹのモードはin-out、すなわち参照による呼び出しと同じ意味を持ち、引数として変数しか許さないので、Ｙのリンククラスは、variableクラスとなる。このように定義された内部クラスＰのノードをプログラム中のstatement クラスやprocedure-callクラスのノードと置き換えることにより、Ｐの手続き呼び出し文をプログラム中に展開することができる。

４．２　属性リレーションを用いた型チェック

型チェックに属性リレーションを用いることによって、

（１）型の決まっていない変数の型をプログラムの文脈によって決めることができる、

（２）型制約を満たす変数の候補を列挙することができる、

といった機能が実現できる。

属性リレーションは、ノード間の属性の値の関係を表す。図７にPLUSクラスの属性リレーションを表形式で示す。表の属性名（以下ではノードの属性名と混乱を防ぐため項目と呼ぶ）はリンク名とノードの属性名である。この表の１行目の項目はリンク名selfと属性名typeである。この表は、PLUSクラスのノードの型属性、leftリンクと結合されているノードの型属性、right リンクと結合されているノードの型属性の取りうる値の組み合わせを表している。

代入文の両辺の型が同じでなくてはならないという制約は、代入文の左(右)辺の型が決まったとき右(左)辺の型もその型と同じでなくてはならないということを示している。このように、あるノードの型が決まったとき、それによって他のノードの型の値の取りうる範囲が決まることがある。この範囲のことを型制約と呼び、ノードの属性として保持する。

P (X => <exp> , Y => <var>) ;

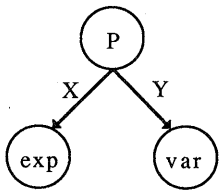


図６　内部クラスＰのテンプレート

self . type	left . type	right . type
integer	integer	integer
float	float	float
float	float	integer
float	integer	float

図７　PLUSクラスの属性リレーション

あるノードの型が決まったとき、そのノードの属性リレーション上での関係演算によって他のノードの型制約が求まる。さらに、そのノードの型制約が求まったことによって、別のノードの型制約が求まる。このように、型制約はノードを次々と伝播していく。型制約の違反は、あるノードの型属性が決まったとき、その型属性の値が型制約属性の値に含まれるかどうかを調べることによって検出される。

型制約の伝播のための手順を以下に示す。

１．あるノードの型制約属性が決まったとき、それに関係する属性リレーションを決定する。

２．その属性リレーション上で、項目で指定されたノードの型制約属性の値で関係演算セレクションを行なう。次に項目それぞれに対して関係演算プロジェクションを行ない、それぞれの項目に対する型制約の値を求める。

３．計算によって求めた型制約を、項目で指定されたノードの型制約属性にセットする。このとき既にノードにセットされている型制約の値が伝播する型制約と同じ場合、型制約の伝播は停止する。

４．変更された型制約それぞれに対し、１－４を繰り返す。

次に型制約伝播の例を説明する。図８(ａ)においてテンプレートは

〈var〉 := 〈exp〉 + 〈exp〉

と展開されている。プログラム中でfloat型の変数Ａとcharacter型の変数Ｂが宣言されているとする。assignmentクラスとplusクラスにはそれぞれ図８のような属性リレーションが記述されている。（図ではintegerをｉ、floatをｆ、characterをｃと略している。）２つの属性リレーションによって図の構文木の各ノードには(ｉｆ)という型制約がかかっている。

代入文の左辺を変数Ａに展開すると、図８(ｂ)のようになる。Ａに展開されたとき、Ａの型が型制約(ｉｆ)に含まれるかどうかチェックされる。この場合Ａの型はｆなので型制約は満たされる。これにより、assignmentノードの左辺のノードの型制約が(f) となる。assignmentの属性リレーション上で

(left=f and (self=i or self=f)
and (right=i or right=f))

という制約でセレクションが行なわれる。次にself、rightそれぞれでプロジェクションが行なわれ、assignmentノードには(f) 、plusノードには(f) が型制約として伝播される。plusノードの型制約が(f) となるので、plusの属性リレーション上で同様の関係演算が行なわれ、plusノードのleftリンク、rightリンクに結合されているexpressionノードにそれぞれ型制約(ｉｆ)が伝播される。

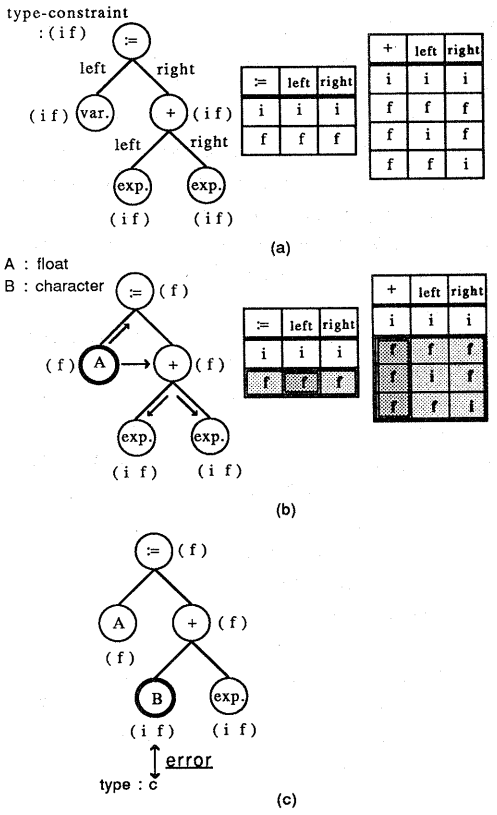


図８　型チェックの例

次にplusノードのleftリンクに変数Ｂが展開されるとＢの型が型制約(ｉ　ｆ)に含まれるかどうかチェックされる。この場合Ｂの型はｃなので型制約エラーとなり、エラーメッセージが表示され、図８(ｂ)の状態に戻る。

型チェックを属性リレーションで行なうことの利点は、

（１）型の依存関係を明示的に表すことができるので型制約の記述が容易になる、

（２）あらかじめ型制約に違反しない置き換え可能な変数を示すことが可能になる、

といったことである。

一方、問題点として次のようなことがある。

（１）予め与えられた型しか扱えない。つまり、プログラム中で型が宣言された場合、それを扱うことができない。これを解決するために、属性リレーション中で変数を扱えるように検討している。

（２）型制約の再計算に時間がかかる。型を持ったノードが型を持たないノードに置換されたとき、あるいは宣言部において型が変更されたとき、型制約を再計算しなくてはならない。現在は、変更されたノードが影響を与える可能性のある部分木の全てのノードの型制約属性をnil にして型制約の再計算を行なっている。このため部分木が大きくなると型制約の再計算に時間がかかる。この問題に対し、再計算が行なわれる部分木をより小さくする方法を検討している。

４．３　ゲート

ＭＵＳＥでは特定の言語向きの機能やインクリメンタルで対話的なプログラム開発を支援するといった機能のほか、構成管理、バージョン管理、資源管理といった機能の実現を目的としている。

ＭＵＳＥでは、プログラムやこれらの管理情報間を関連づけるためにゲートを導入した。ゲートそれ自身はノードであり、別の構文木全体を表している。図９にゲートの例を示す。この図においてディレクトリ木とプログラムがゲートにより関連づけられている。

ユーザがゲートに対してオープンというメッセージを送ることによって、ゲートが表す構文木のための新しいコンテキストが作られ、コンテキストウインドウが開かれ、この構文木に対して操作を加えることが可能となる。

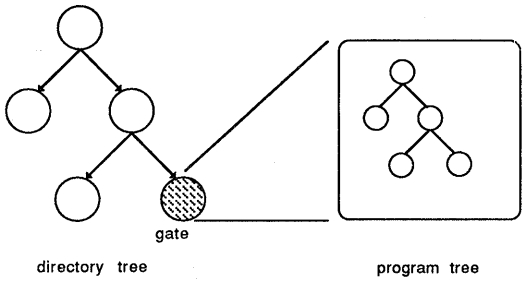


図9　ゲート

5．実験システム

以上のシステムをSun－3のUNIX上にCとLisp（KCL）で実現した。モジュールはライブラリ用モジュール、ディレクトリ用モジュール、Ada（サブセット）用モジュール、Pascal（サブセット）用モジュールが実現されている。システムならびに各モジュールのサイズは次の通りである。

・MUSE本体
　・C　　1200行
　・Lisp　4600行
・モジュール

・ライブラリ	クラス数3	90行
・ディレクトリ	クラス数9	110行
・Ada	クラス数123	1850行
・Pascal	クラス数94	1600行

6．おわりに

本稿では、プログラミング言語の構文および静的意味に関する知識を用いることによって、その言語向きの機能を提供するMUSEについて述べた。MUSEではプログラムの内部表現を統一することにより、異なる言語間でのツールの共有が可能となり、メッセージ、プローブ、属性リレーション、ゲートといった機能により、構文木内での様々な処理が可能となった。またオブジェクト指向の導入により、言語に関する知識の記述が容易になった。

今後の課題として以下が挙げられる。

（1）今回述べた属性リレーションは型チェックにのみ用いられていたが、属性リレーションによる方法を種々の静的意味チェックに利用できるように一般化すること。

（2）メソッド、プローブが手続き的であるため内部処理を記述しにくいので、より宣言的な方法で内部処理を記述できるようにすること。

（3）種々のツールを実現する必要がある。現在デバッガを実現中である。

今後、以上の点を解決するとともに、ソフトウェア開発のデザインフェーズの支援についての研究を行なう予定である。

謝辞　本研究を進めるにあたり、有益な御意見を頂いた広島大学工学部第二類平川正人博士、情報システム研究室の吉野真澄氏、ならびに日頃御討論頂く同研究室の諸氏に感謝する。

参考文献

[1] A. N. Habermann and D. Notkin: "Gandalf: Software Development Environments", IEEE Trans., Software Eng., Vol.SE-12, No.12 pp. 1117-1125 (1986).

[2] T. Teitelbaum and T. Reps: "The Cornel Program Synthesizer: A Syntax-Directed Programming Environment", Commun. ACM, Vol.24, No.9, pp.563-573 (1981).

[3] W. Teitelman and L. Masinter: "The Interlisp Programming Environment", Computer, Vol.14, No.4, pp.25-34 (1981).

[4] T. Reps and T. Teitelbaum: "The Synthesizer Generator", Proc., ACM SIGSOFT/SIGPLAN Software Eng. Symp. on Practical Software Devlop. Env., pp.42-48 (Apr. 1984).

[5] "ALOE Users' and Implementors' Guide", Dep. of Computer Science, Carnegie-Mellon Univ. (Oct. 1984).

[6] R. Bahlke and G. Snelting: "The PSG System: From Formal Language Definitions to Interactive Programming Environments", ACM Trans., Program. Lang. Syst., Vol.8, No.4, pp.547-576 (1986).

[7] 天満，吉野，坪谷，田中，市川："拡張型統合化プログラミング環境の構築"，情報処理学会ソフトウェア工学研究会資料，52-2,(昭62-02).

[8] A. Goldberg and D. Robinson: Smalltalk-80: The Language and Its Implementation, Addison-Wesley, Reading, Mass. (1983).

[9] V. Donzeau-Gouge, G. Kahn, and B. Lang, and B. Meles: "Document Structure and Modularity in Mentor", Proc., ACM SIGSOFT/SIGPLAN Software Eng. Symp. on Practical Software Develop. Env., pp.141-148 (Apr. 1984).