

ソフトウェア開発支援環境用 データベース管理システムのカスタマイズ機構

Customization Facilities of a Database Management System for Software Development Environments

前田 和昭*, 野呂 昌満**

Kazuaki MAEDA, Masami NORO

要 旨

ソフトウェア開発環境を統合化するために、データベースを中心としてツールを統合化するアプローチが一般的に取り入れられている。ソフトウェア開発の形態はプロジェクト毎に異なるものであり、すべてのソフトウェア開発に対して万能普遍的なデータベース管理システムを考えることは不可能である。むしろ、データベース管理システムをカスタマイズするための機構を提供することにより多様な開発形態に対応したほうがよい。本論文では、ソフトウェア開発の多様な形態に適応可能なオブジェクト指向データベース管理システムLeOのカスタマイズ機構について述べる。

1. はじめに

ソフトウェア開発を支援するため、これまで多くの統合化ソフトウェア開発支援環境(以下、SDEと略記する)が提案されてきた。[Hen84, Hen86, Hen88, Tay90]。最近のSDEは、データベース(以下、DBと略記する)を中心に置き、それを各ツールが共通に利用し、ユーザは統一されたユーザインタフェースからツールを利用するといった機構になっている。この機構の各要素(DB、各ツール、ユーザインタフェース)は、有機的に結合されていて、ユーザからはSDE自体が一つのツールとして見える。

SDE用DBに保存されるデータは、ソフトウェア開発の途上で作成されるもの全て(プログラ

ム、マニュアル、仕様書、データフロー図等)で、木構造やグラフ構造を用いて表現される。これらのデータは、商用DBの代表である関係DB等では扱いにくい以下の性質を持っている。

1) 蓄えられるデータの構造が複雑である

プログラムを抽象構文木として表現し、DBに保存することを考えてみよう。この場合、その木は、比較的深く、またノード間の参照関係が複雑である。

2) 一つのトランザクションが比較的長い

エディタでDBに保存されているプログラムを編集することを考えてみよう。この場合、その編集が数秒で終わることは稀で、数日または数週間かかることもある。従来のDBでのスケジュールのようにトランザクションを直列化するために、編集中のプログラムへのアクセスを禁止したとすると、他の開発者の生産性が著しく低下してしまう。

3) 矛盾のあるデータが存在してよい

作業中のプログラムを複数人で検討しながら開発を進める場合を考えてみよう。DBは共有する情報を保存する場所であるため、作業中のプログラムをDBに保存することもありうる。このような作業中のプログラムは、その完成度

* 経営情報学科 — ソフトウェア工学

Department of Business Administration and Information Science — Software Engineering

** 南山大学経営学部情報管理学科

— ソフトウェア工学

Department of Information Systems and Quantitative Sciences, School of Business Administration
Nanzan University — Software Engineering

によっては、コンパイルエラー等の矛盾があっても構わない場合（例えばコードレビューの途中）がある。

SDE用のデータベース管理システム（以下、DBMSと略記する）を設計する際には、これらの特徴を効率よく扱う機構が必要となる。1980年代初め頃までは、そのような機構を持ったSDE用DBMSは開発されていなかった。

近年、SDE用DBとして、オブジェクト指向データベース（以下、OODBと略記する）が研究用及び商用に開発されている[KWG*87, BMO*89, AHS91, SZR91]。このOODBとは、補助記憶中に格納されたオブジェクトを操作、管理する機能を持ったプログラミング言語システムのことである。OODBを利用すれば、プログラム、マニュアル、仕様書、データフロー図等のデータをオブジェクトとして統一的に扱うことが可能になる。また、主記憶中に存在するオブジェクトと補助記憶中に存在するオブジェクトの扱い方が同じになるため、インピーダンスミスマッチも解決できる（インピーダンスミスマッチとは、アプリケーション記述言語とデータベース操作言語の意味的融りによりプログラマにかかる負担が増大してしまう問題のことである）。したがって、SDEのツール開発者に対して、より開発しやすい環境を与えることになる。さらに、協調作業に適した並行制御機構やオブジェクト管理法もOODBの研究としていくつか提案されている。

ソフトウェアの開発法や管理法は多種多様で、唯一の機構からなる開発支援環境だけでまかなうことは難しい。ソフトウェアの開発手法が変わると、SDEの中心となるDBに保存するデータも変わってくるため、DBMS自体に変更が必要になる場合もある。我々は、このようなソフトウェア開発の多様性に柔軟に適応可能にするためには、DBMSにカスタマイズ機能を持たせるべきだと信じている。DBMSのカスタマイズ機構の整備が十分であれば、比較的少ない作業で、どのようなソフトウェア開発にも適応可能になるはずである。

我々は、このような要求を満足し、多様なソフトウェア開発に柔軟に対応できるように、カスタマイズ可能なオブジェクト指向データベース管理システムLeO及びその記述言語Liyaの設計を行った。本論文では、このLeOの基本構想とLiyaによるカスタマイズ機能について述べる。

2. 基本構想

LeOは、ツールインタフェースとLiyaインタプリタで構成されている。その構成を図1に示す。

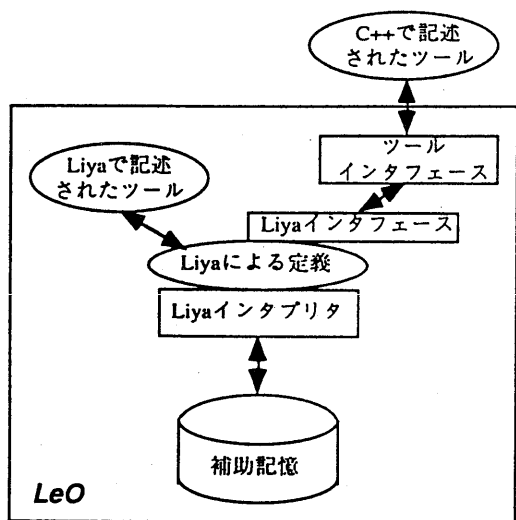


図1 LeOの構成図

OODBの多くは、プログラミング言語システムとして実現されているため、アプリケーションを記述するための言語を特定している場合が多い。ユーザが従来利用してきたソフトウェア開発ツールや過去のソフトウェア資産の利用を考えると、OODBを使うために新しいプログラミング言語を採用するというのは現実的ではない。そこでLeOは、複数のプログラミング言語からのアクセスを可能にするための特別なツールインタフェースを準備している（このツールインタフェースについては、本論文の主旨を越

えるため省略する)。LeOのカスタマイズは、Liyaの自己反映機能〔Mae 87, WY 88〕を用いて行う。Liyaで記述されたプログラムは、LeOが提供するLiyaインタプリタ上で実行され、その結果、ツールから見ると、LeOは、カスタマイズされたDBMSとなる。Liyaは単にDBMSをカスタマイズするためだけの言語ではなく、一般のプログラミング言語としての機能を持っているので、ツールを記述することも可能である。

オブジェクト指向言語におけるクラスは、オブジェクトを定義し、生成する役割を果たしている。この生成されたオブジェクトにメッセージを送ると、そのオブジェクトが所属しているクラスで定義されているメソッドを使って計算が行われる。オブジェクトは、値を保持するだけで、その処理方法を知らない。オブジェクトとクラスの間にはクラス-インスタンス関係が存在して初めて処理方法が決定する。LeOでは、オブジェクト間の関係がオブジェクトの動作を引き起こすという考えを基本にしている。オブジェクトがメッセージを受け取った際に、そのオブジェクトとクラスの間にはクラス-インスタンス関係がその事を察知し、クラスからそのメソッドを捜し出し、処理を行う。このクラス-インスタンス関係はカスタマイズ可能になっており、自由に変更することができる。Liyaには、クラスやインスタンスの構造を記述する構造定義部とクラス-インスタンス関係やスーパクラ

ス-サブクラス関係を記述する関係定義部が準備されている。以下、構造定義部と関係定義部について説明する。

2.1. 構造定義部

構造定義部では、クラスやインスタンスのデータ構造を定義する。この定義部では、構造体とその間の参照関係を定義し、グラフ構造を作り上げることができる。この構造体は、複数のスロットを持ち、各スロットに対してスロット名とその型を定義していく。組み込み型 (Integer, Boolean, Attribute など) がいくつか準備されており、これを使って構造体を定義し、その構造体をユーザ定義型として利用してデータ構造を記述していく。その定義の一部を図2に示す。

2.2. 関係定義部

構造定義部で定義したインスタンスやクラスを用いて、クラス-インスタンス関係やスーパクラス-サブクラス関係を定義するのがこの関係定義部である。

この定義部では、

- ・変化が起きた時に自動的に起動されるトリガ
- ・常に真となる条件である表明
- ・関係に特別な機能を持たせるためのメソッド
- ・関係独自の値を保持するための変数を定義することができる。その定義の一部を図3に示す。

・クラスの定義

```
structure Class
    attrs          : seqof Attribute;      -- 属性定義用スロット
    methods        : seqof Method;        -- メソッド定義用スロット
    methodSearch    : Message;            -- メソッド検索用スロット
end;
```

ここで、root とは、このデータ構造定義の始まりを示す予約語である。
また、seqof * は、* のリスト、-- は、注釈文を表す。

・インスタンスの定義

```
structure Instance
    attrs          : seqof AttributedValue; -- 属性値保存用スロット
    message        : Message;              -- 受信したメッセージ保存用スロット
end;
```

図2 構造定義例

・クラス-インスタンス関係の定義

```
relation INS (c:Class; i:Instance)
```

```
trigger
```

```
on i.message
```

```
  mess:Message:=i.message;
```

```
  meth:Method;
```

```
begin
```

```
  foreach m in c.methods do
```

```
    if m.name=mess.selector then
```

```
      apply(i,m);
```

```
      return;
```

```
    end;
```

```
  end;
```

```
  meth:=c.methodSearch(mess);
```

```
  if meth/=Null
```

```
    apply(i,meth);
```

```
    return;
```

```
  else
```

```
    return MethodSearchError;
```

```
  end;
```

```
end;
```

```
end relation;
```

-- i がメッセージを受信した時、以下を実行する。

-- c が保持する全てのメソッドに対して検査する。

-- メソッド名とメッセージのセレクト名が一致した時

-- そのメソッドを使って、実行する。

-- c に求めるメソッドがないため、スーパークラスへ処理を依頼する。

-- メソッドが見つかった時、

-- そのメソッドを使って、実行する。

-- メソッドが見つからない時、

-- エラーコードを返す

・スーパークラス-サブクラス関係の定義

```
relation ISA (super:Class; sub:Class)
```

```
trigger
```

```
on sub.methodSearch
```

```
  mess:Message:=sub.methodSearch;
```

```
begin
```

```
  foreach m in super.methods do
```

```
    if m.name=mess.selector then
```

```
      return m;
```

```
    end;
```

```
  end;
```

```
  m:=super.methodSearch(mess);
```

```
  return m;
```

```
end;
```

```
end;
```

-- super : スーパークラス、sub : サブクラス

-- sub がメソッド検索を依頼された時、以下を実行する。

-- super が保持する全てのメソッドに対して検査する。

-- メソッド名とメッセージのセレクト名が一致した時

-- そのメソッドを返す。

-- 求めるメソッドがない時、スーパークラスへ処理を依頼する。

図3 関係定義例

3. カスタマイズ機能

2節で述べた構造定義部と関係定義部を修正することで、多様なカスタマイズが可能となる。ここでは、クラス-クラス関係と並行制御とスキーマ進化のカスタマイズについて述べる。

3.1. クラス-クラス関係のカスタマイズ

大抵のオブジェクト指向言語が提供しているクラス-クラス関係としてis-a関係がある。このis-a関係は、クラス間の論理的な包含関係を表現するために利用されている。このis-a関係

を特殊化した関係として、TAMEデータモデル[NBG90]では、ファセットに基づくis-a関係を定義している。ファセットに基づくis-a関係とは、is-a関係に以下のような制約を与えたものである。

スーパークラスのインスタンス変数をファセットの指標として宣言した時、サブクラスはそのインスタンス変数の型を特殊化できるだけで、他の属性を再定義できない。

LeOでは、この制約をis-a関係に追加するだけで、ファセットに基づくis-a関係が利用可能

```

relation FacetISA (super:Class; sub:Class) is ISA -- FacetISA は、ISA を特殊化したもの
assert
  forall a in sub.attrs do      -- サブクラス sub の属性全てに対して以下の条件が成り立つ
    exists b in super.attrs do  -- スーパークラス super の属性の中で以下の条件を満たすものが存在する
      a.name=b.name and        -- a と b の名前が同じで、かつ、
      a.type<b.type;           -- a の型は、b の型を特殊化したもの
    end;
  end;
end;

```

図4 ファセットに基づく is-a 関係の定義例

となる。その定義の一部を図4に示す。

3.2. 並行制御

LeOの初期設定では、インスタンスに渡されたメッセージをINS関係に転送し、そのメッセージの解釈を任せている。メッセージを受け取ったINS関係は、そのメッセージに対応するメソッドを、インスタンスが所属しているクラス内から探し出す。探し出したメソッドとインスタンスを使って、実際に計算が行われる。従って、INS関係はインスタンスの管理をしているとみなすことができる。

このINS関係を利用すると、並行制御を定義することができる。例えば、インスタンスにアクセスしたい場合には、必ずロックをかけてからアクセスを行うとしよう。ロックがかかっているかどうかは、インスタンス自体の情報ではなく、インスタンスを管理する側が保持すべき情報である。従って、この情報をINS関係に持たせる。オブジェクトにロックが要求された時、その要求をINS関係で解釈し、INS関係内でその所有者を設定する。INS関係がロック要求を解釈する際に、既に使用中ならば、その要求を拒絶する。これらの手続きも、Liyaで記述されているためカスタマイズ可能である。その定義の一部を図5に示す。

3.3. スキーマ進化

スキーマ進化とは、あるクラスの定義を変更し、既存のインスタンスとの間に矛盾が生じた時に、この矛盾を解決するために与えられた機

構のことである[BKK'86, PS87, SZ87]。LeOでは、このスキーマ進化にも柔軟に対処できる。まず、あるクラスの定義を変更したとしよう。その際、そのクラスに対応するINS関係がその変更を察知し、矛盾のあるオブジェクト全ての変更を行い、クラスとインスタンスの間の矛盾をなくす。この機構は、ISA関係とINS関係内で定義されており、カスタマイズ可能である。LeOの初期設定では、クラスに変更があった時、スキーマ進化の処理をすぐに行うようになっている。その定義の一部を図6に示す。

4. 終わりに

本論文では、カスタマイズ可能なオブジェクト指向データベース管理システムLeOの基本構想とそのカスタマイズ機能について述べた。既発表のカスタマイズ可能なOODBとして、KOMS[YSL91]が提案されている。KOMSでは、モデル自体をモデル化する機能が提供されている。このモデル化は、メタクラスを使って行われ、OODBの構成要素(クラス、アソシエーション等)の構造をカスタマイズできる。KOMSでは、クラスなどの構造のカスタマイズを強調しており、OODB上での処理(メソッド探索など)のカスタマイズが考慮されていない。LeOにおけるカスタマイズは、記述言語Liyaを用いる。Liyaでは、クラスやインスタンスの構造の定義だけではなく、メソッド探索などの処理方法を記述できるため、KOMSよりも広くカスタマイ

ズが可能となる。

LeO上で動くアプリケーションプログラムは、Liyaを用いて記述できる。しかし、既存のソフトウェア開発支援ツールやソフトウェア資

産を活用するために、多言語サポートを実現することも重要な検討事項である。そのために、複数のプログラミング言語とのインタフェースを生成する機構も現在設計中である。

```

relation INS (c:Class; i:Instance)
attribute
    owner:UID;
trigger
    on i.message
        mess:Message:=i.message;
        meth:Method;
    begin
        if owner/=Null then
            return LockError;
        end;
        foreach m in c.methods do
            if m.name=mess.selector then
                apply (i,m);
                return;
            end;
        end;
        meth:=c.methodSearch (mess);
        if meth/=Null then
            apply (i,meth);
            return;
        else
            return MethodSearchError;
        end;
    end;
primitive method lock:Boolean is
    begin
        if owner=Null then
            return LockError;
        else
            owner:=getOwner (i.message);
        end;
    end;
primitive method unlock:Boolean is
    begin
        if owner=Null then
            return UnLockError;
        else
            owner:=Null;
        end;
    end;
end;
end;

```

-- i がメッセージを受信した時、以下を実行する。

-- c が保持する全てのメソッドに対して検査する。
 -- メソッド名とメッセージのセレクト名が一致した時
 -- そのメソッドを使って実行する。

-- c に求めるメソッドがないため、スーパークラスへ処理を依頼する
 -- メソッドが見つかった時、
 -- そのメソッドを使って、実行する

-- メソッドが見つからない時、
 -- エラーコードを返す

-- ロック設定のためのメソッド。ただし、実行中は他へ制御が移らない

-- 既に使用中である時、
 -- ロックエラーのコードを返す
 -- それ以外の時
 -- 所有者コードを設定する

-- ロック解除のためのメソッド。ただし、実行中は他へ制御が移らない

-- 使用中でない時、
 -- アンロックエラーのコードを返す
 -- それ以外の時、
 -- ロックを解除する

図5・並行制御の定義例

```

relation INS (c:Class; i:Instance)
trigger
on c.attrs
  attrAttribute:=c.attrs;          -- cのスロット attrs に変更があった時、以下を実行する
  newAttr:AttributedValue;
begin
  case triggerType (attr)          -- トリガが起動された原因を調べる
  whenKadd =>                      -- 属性が追加された時、
    newAttr:=AttributedValue.new (attr.Vadd); -- 新しい属性を作成する
    appendrearSEQAttributedValue (i.attrs,newAttr); -- 新しい属性を i に設定する
  whenKremove =>                  -- 属性が削除された時、
    foreach a in i.attrs do        -- i が持つ全ての属性に対して検査する
      if a.name=attr.Vdelete.name then -- 削除された属性に対応するものがあれば、
        removeSEQAttributedValue (i.attrs,a); -- その属性を i から削除する
      return;
    end;
  end;
end;
end;
end;

```

図6 スキーマ進化の定義例

参考文献

[AHS91] Tim Andrews, Craig Harris, and Kiril Sinkel. "ONTOS: A Persistent Database for C++." In Rajiv Gupta and Ellis Horowitz, editors, Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD, pages 387-406. Prentice-Hall, 1991.

[BKK*86] Jay Banerjee, Hyoung-Joo Kim, Won Kim, et al. "Schema Evolution in Object-Oriented Persistent Databases." In Proceedings of the 6th Advanced Database Symposium, pages 23-31. August 1986.

[BMO*89] Robert Bretl, David Maier, Allen Otis, et al. "The Gemstone Date Management System." In Won Kim and Frederick H. Lochovsky, editors, Object-Oriented Concepts, Databases, and Applications, pages 283-308. ACM Press, 1989.

[Fer89] Jacques Ferber. "Computational Reflection in

Class based Object Oriented Languages." In OOPSLA'89 Conference Proceedings, pages 317-326, October 1989.

[Hen84] Peter Henderson, editor. "Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments," November 1984.

[Hen86] Peter Henderson, editor. "Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments," December 1986.

[Hen88] Peter Henderson, editor. "Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments," November 1988.

[KWG*87] Won Kim, Darell Woelk, Jorge Garza, et al. "Enhancing The Object-Oriented Concepts For Database Support." In 3rd Data Engineering, pages 291-292, 1987.

- [Mae87] Pattie Maes. "Concepts and Experiments in Computational Reflection." OOPSLA '87 Conference Proceedings, pages 147-155, October 1987.
- [NBG90] Masami Noro, Victor R. Basili, and Parke Godfrey, "TAME 経験ベースのデータモデル品質改良パラダイムに基づくソフトウェアプロジェクト管理・支援システム実現の第一歩." In 日本ソフトウェア科学会第7回大会論文集, pages 201-204, September 1990.
- [PS87] D. Jason Penney and Jacob Stein. "Class Modification in the GemStone Object-Oriented DBMS." In OOPSLA'87 Conference Proceedings, pages 111-117, November 1987.
- [SZ87] Andrea H. Skarra and Stanley B. Zdonik. "Type Evolution in an Object-Oriented Database." In Bruce Shriver and Peter Wegner, editors, Research directions in Object-Oriented Programming, pages 393-415. MIT Press, 1987.
- [SZR91] Andrea H. Skarra Stanley B. Zdonik, and Steven P. Reiss. "ObServer: An Object Server for an Object-Oriented Database System." In K.R. Dittrich, U. Dayal, and A.P. Buchmann, editors, On Object-Oriented Database Systems, pages 275-290. Springer-Verlag, 1991.
- [Tay90] Richard N. Taylor, editor. "Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments," December 1990.
- [WY88] Takuo Watanabe and Akinori Yonezawa. "Reflection in Object-Oriented Concurrent Language." In OOPSLA'88 Conference Proceedings, pages 306-315, September 1988.
- [YSL91] Rahim Yaseen, Stanley Y.W. Su, and Herman Lam. "An Extensible Kernel Object Management System." In OOPSLA'91 Conference Proceedings, pages 247-263, October 1991.

付録 Liyaの構文規則

以下の構文規則は、yaccの記法を使用している。ただし、

IDENTIFIERは、識別子で英字で始まる英数字、
 CHARCONSTは、文字定数、
 INTCONSTは、整数定数、
 REALCONSTは、実数定数、
 BOOLCONSTは、論理定数、
 STRCONSTは、文字列定数、
 を表すものとする。

```
unitList      : /* empty */
               | unit
               | unitList unit

unit          : structure
               | relation

structure     : 'structure' IDENTIFIER parentName
               'attribute' structFeatureList 'end' ';'

parentName    : /* empty */
               | 'is' IDENTIFIER

idList        : IDENTIFIER
               | idList ',' IDENTIFIER

structFeatureList : structFeature
                   | structFeatureList structFeature

structFeature  : varDecl ';'
                | sharedAttrDecl ';'
                | constAttrDecl ';'

relation       : 'relation' IDENTIFIER '(' varDeclList ')'
                parentName relFeatureList 'end' ';'

relFeatureList : relFeature
                | relFeatureList relFeature

relFeature     : triggerDecl
                | attrDecl
                | methodDecl ';'
                | assert

attrDecl       : 'attribute' attrDeclList

attrDeclList   : eachAttrDecl
                | attrDeclList eachAttrDecl

eachAttrDecl   : varDecl ';'
                | sharedAttrDecl ';'
                | constAttrDecl ';'
                | 'private' varDecl ';'
                | 'private' methodDecl ';'

```

```

        | 'private' sharedAttrDecl ';'

triggerDecl      : 'trigger' triggerList

triggerList      : trigger
                  | triggerList trigger

trigger          : 'on' interestField relFeatureList
                  'begin' statementList 'end' ';'

interestField    : IDENTIFIER '.' IDENTIFIER

varDecl          : singleVarDecl
                  | multiIdList ':' IDENTIFIER initValue
                  | multiIdList ':' 'seqof' IDENTIFIER initValue

initValue        : /* empty */
                  | ':' expr

singleVarDecl    : IDENTIFIER ':' IDENTIFIER initValue
                  | IDENTIFIER ':' 'seqof' IDENTIFIER initValue

multiIdList      : idList ',' IDENTIFIER

sharedAttrDecl   : 'shared' varDecl
                  | 'shared' varDecl ':' expr

varDeclList      : varDecl
                  | varDeclList ';' varDecl

methodDecl       : methodType IDENTIFIER 'is' localDecl
                  'begin' statementList 'end'
                  | methodType IDENTIFIER '(' varDeclList ')' 'is' localDecl
                  'begin' statementList 'end'
                  | methodType IDENTIFIER ':' IDENTIFIER 'is' localDecl
                  'begin' statementList 'end'
                  | methodType IDENTIFIER '(' varDeclList ')' ':' IDENTIFIER
                  'is' localDecl 'begin' statementList 'end'

methodType       : 'method'
                  | 'primitive' 'method'

constAttrDecl    : 'const' varDecl ':' expr

assert           : 'assert' assertCond

assertCond       : quantifier singleVarDecl 'in' expr
                  'do' assertOrExpr 'end' ';'

quantifier        : 'forall'
                  | 'exists'

assertOrExpr      : expr
                  | assertCond

```

```

statementList      : /* empty */
                    | statement
                    | statementList statement

statement          : assignment ';'
                    | conditional ';'
                    | loop ';'
                    | case ';'
                    | break ';'
                    | return ';'
                    | call ';'
                    | assert ';'

localDecl          : /* empty */
                    | varDecl initValue

assignment         : expr ':= ' expr

conditional        : 'if' expr 'then' statementList elsifPart elsePart 'end'

elsifPart          : /* empty */
                    | elsifPart 'elseif' expr 'then' statementList

elsePart           : /* empty */
                    | 'else' statementList

loop               : 'while' expr 'loop' statementList 'end'
                    | 'loop' statementList 'end'
                    | 'foreach' IDENTIFIER 'in' expr 'do' statementList 'end'

case               : 'case' expr whenPartList elsePart 'end'

whenPartList       : /* empty */
                    | whenPart
                    | whenPartList whenPart

whenPart           : 'when' exprList '=>' statementList

assert             : 'assert' '(' IDENTIFIER ')' expr 'end'

break              : 'break'

return             : 'return'
                    | 'return' expr

call               : IDENTIFIER '(' exprList ')'
                    | IDENTIFIER '.' IDENTIFIER '(' exprList ')'
                    | cexpr '.' IDENTIFIER argVals

argVals            : /* empty */
                    | '(' exprList ')'

exprList           : expr
                    | exprList ',' expr

expr               : cexpr

```

```
      | nexpr

cexpr      : CHARCONST
      | INTCONST
      | REALCONST
      | BOOLCONST
      | STRCONST
      | array
      | '(' expr ')'

nexpr      : 'not' expr
      | expr '<' expr
      | expr '>' expr
      | expr '<=' expr
      | expr '>=' expr
      | expr '=' expr
      | expr '/=' expr
      | expr 'and' expr
      | expr 'or' expr
      | '-' expr
      | '+' expr
      | expr '+' expr
      | expr '-' expr
      | expr '*' expr
      | expr '/' expr
      | IDENTIFIER
      | selectedName
      | call

selectedName : IDENTIFIER '.' IDENTIFIER

array      : cexpr '[' exprList ']'
      | '[' exprList ']'
```