7. Remote accessing

An attribute of an object is identified completely by the following items of information:

1) the object,

2) a class which is outer to or equal to that of the object, and

3) an attribute identifier defined in that class or in any class belonging to its prefix sequence.

Item 2 is textually defined for any attribute identification. The prefix level of the class is called the "access level" of the attribute identification.

Consider an attribute identification whose item 2 is the class C. Its attribute identifier, item 3, is subjected to the same identifier substitutions as those which would be applied to an uncommitted occurrence of that identifier within the main part of C, at the time of concatenation. In that way, name conflicts between attributes declared at different prefix levels of an object are resolved by selecting the one defined at the innermost prefix level not inner to the access level of the attribute identification.

An uncommitted occurrence within a given object of the identifier of an attribute of the object is itself a complete attribute identification. In this case items 1 and 2 are implicitly defined, as respectively the given object and the class associated with the prefix level of the identifier occurrence.

If such an identifier occurrence is located in the body of a procedure declaration (which is part of the object), then, for any dynamic instance of the procedure, the

occurrence serves to identify an attribute of the given object, regardless of the context in which the procedure was invoked.

Remote accessing of attributes, i.e. access from outside the object, is either through the mechanism of "remote identifiers" ("dot notation") or through "connection". The former is an adaptation of a technique proposed in [3], the latter corresponds to the connection mechanism of SIMULA I [2].

A text reference is (itself) a compound structure in the sense that it has attributes accessible through the dot notation.

7.1 Remote identifiers

7.1.1 Syntax

```
<attribute identifier> ::= <identifier>
<remote identifier> ::=
      <simple object expression>.<attribute identifier>|
      <simple text expression>.<attribute identifier>
<identifier 1> ::= <identifier>|
                   <remote identifier>
<variable identifier 1> ::= <identifier 1>
<simple variable 1> ::= <variable identifier 1>
<array identifier 1> ::= <identifier 1>
<variable> ::= <simple variable 1>|
               <array identifier 1>[<subscript list>]
<procedure identifier 1> ::= <identifier 1>
<function designator>::=
      <procedure identifier 1><actual parameter part>
<procedure statement> ::=
      <procedure identifier 1><actual parameter part>
<actual parameter> ::= <expression>|
                       <array identifier 1>|
                       <switch identifier>|
                       <procedure identifier 1>
```

7.1.2 Semantics

Let X be a simple object expression qualified by the class C, and let A be an appropriate attribute identifier. Then the remote identifier "X.A", if valid, is an attribute identification whose item 1 is the value X and whose item 2 is C.

The remote identifier X.A is valid if the following conditions are satisfied:

1) The value X is different from none.

2) The object referenced by X has no class attribute declared at any prefix level equal or outer to that of C.

Condition 1 corresponds to a run time check which causes a run-time error if the value of X is none.

Condition 2 is an ad hoc rule intended to simplify the language and its implementations.

A remote identifier of the form

<simple text expression>.<attribute identifier>

identifies an attribute of the text reference obtained by evaluating the simple text expression, provided that the attribute identifier is one of the procedure identifiers listed in section 10.1.

Example 1:

Let G5 and G10 be variables declared and initialized as in example 1 of section 6.1.2.2. Then an expression of the form

G5.integral(.....) or G10.integral(.....)

is an approximation to a definite integral obtained by applying respectively a 5 point or a 10 point Gauss formula.

Example 2:

Let P1 and P2 be variables declared and initialized as in example 2 of section 6.1.2.2. Then the value of the expression

```
P1.plus (P2)
```

is a new "point" object which represents the vector sum of P1 and P2. The value of the expression

```
P1 qua polar.plus (P2)
```

is a new "polar" object representing the same vector sum.

7.2 Connection

7.2.1 Syntax

```
<connection block 1> ::= <statement>
<connection block 2> ::= <statement>
<when clause> ::=
            when <class identifier>do<connection block 1>
<otherwise clause> ::= <empty>|
                       otherwise<statement>
<connection part> ::= <when clause>|
          <connection part><when clause>
<connection statement> ::=
          inspect <object expression>
          <connection part><otherwise clause>|
          inspect <object expression> do
          <connection block 2><otherwise clause>|
          <label>:<connection statement>
```

A connection block may itself be a connection statement, which, in that case, is the largest possible connection statement.

7.2.2 Semantics

The purpose of the connection mechanism is to provide implicit definitions of the above items 1 and 2 for certain attribute identifications within connection blocks.

The execution of a connection statement may be described as follows:

1) The object expression of the connection statement is evaluated. Let its value be X.

2) If when-clauses are present they are considered one after another. If X is an object belonging to a class equal or inner to the one identified by a when-clause, the connection block 1 of this when-clause is executed, and subsequent when-clauses are skipped. Otherwise the when-clause is skipped.

3) If a connection block 2 is present it is executed, except if X is none in which case the connection block is skipped.

4) The statement of an otherwise clause is executed if X is none, or if X is an object not belonging to a class included in the one identified by any when-clause. Otherwise it is skipped.

A statement which is a connection block 1 or a connection block 2 acts as a block, whether it takes the form of a block or not. It further acts as if enclosed in a second fictitious block, called a

"connection block". During the execution of a connection block the object X is said to be "connected". A connection block has an associated "block qualification", which is the preceding class identifier for a connection block 1 and the qualification of the preceding object expression for a connection block 2.

Let the block qualification of a given connection block be C and let A be an attribute identifier defined at any prefix level of C. Then any uncommitted occurrence of A within the connection block is given the local significance of being an attribute identification. Its item 1 is the connected object, its item 2 is the block qualification C.

It follows that a connection block acts as if its local quantities are those attributes of the connected object which are defined at prefix levels outer to and including that of C. (Name conflicts between attributes defined at different prefix levels of C are resolved by selecting the one defined at the innermost prefix level.)

Example:

Let "Gauss" be the class declared in the example of section 2.2. Then within the connection block 2 of the connection statement

```
inspect new Gauss(5) do begin ..... end
```

a procedure "integral" is available for numeric integration by means of a 5 point Gauss formula.