

D: A Language Framework for Distributed Programming

Cristina Videira Lopes

PhD Thesis, College of Computer Science, Northeastern University. November 1997.

© Copyright 1997 Xerox Corporation. All rights reserved.

Appendix A. Syntax

I JCore

The only syntactic difference between JCore and Java is the keyword `synchronized`, which does not exist in JCore.

II COOL

Note: some of the productions are defined in §IV.

COOLTranslationUnit:

CoordinatorDeclaration

CoordinatorDeclaration:

*Granularity*_{opt} **coordinator** *ClassName_CommaList* *CoordinatorBody*

Granularity: **per_class**

CoordinatorBody:

```
{  
  CondVarDeclaration_Listopt  
  VariableDeclaration_Listopt  
  SelfExclusiveMethodsopt  
  MutuallyExclusiveMethodSet_Listopt  
  MethodManager_Listopt  
}
```

CondVarDecl:

condition *VariableDeclarator_CommaList* ;

VariableDeclarator:

Identifier = *CondVarInitializer* |
Identifier [] = *CondArrayInitializer*

CondVarInitializer:

true / **false**

CondArrayInitializer:

{ *CondVarInitializer_CommaList* }

VarDeclaration:

PrimitiveType VariableDeclarator_CommaList ;

VariableDeclarator:

Identifier = VarInitializer |
Identifier [] = ArrayInitializer

VarInitializer:

Expression

ArrayInitializer:

{ VarInitializer_CommaList }

SelfExclusiveMethods:

selfex *QualifiedName_CommaList ;*

MutuallyExclusiveMethodSet:

mutex { *QualifiedName_CommaList* } ;

MethodManager:

QualifiedName_CommaList :
Requires_{opt} OnEntry_{opt} OnExit_{opt}

Requires:

requires *CondVarExpression ;*

CondVarExpression:

VarRef |
Not CondVarExpression |
(CondVarExpression) |
CondVarExpression ConditionalOp CondVarExpression

OnEntry:

on_entry { *Statement_List* }

OnExit:

on_exit { *Statement_List* }

Statement:

IfStatement |
AssignStatement

IfStatement:

if *Expression* { *Statement_List* } /
if *Expression* { *Statement_List* } **else** { *Statement_List* }

AssignStatement:

Identifier = Expression ;

III RIDL

Note: some of the productions are defined in §IV.

RIDLTranslationUnit:

PortalDeclaration

PortalDeclaration:

portal *ClassName PortalBody*

PortalBody:

```
{
  RemoteOperation_List
  DefaultTransfersopt
}
```

DefaultTransfers:

default: *TransferableType_List*

RemoteOperation:

ReturnType *MethodName* (*Parameter_CommaList_{opt}*)
RemoteOperationBody_{opt} ;

ReturnType:

Type /
void

Parameter:

Type ParamName

ParamName:

Identifier /
Identifier []

RemoteOperationBody:

{ *ObjectTransferSpec_List* }

ObjectTransferSpec:

ObjectName : *Mode* ;

ObjectName:

Identifier /
return

Mode:

gref /
copy *CopyDirective*_{opt}

TypeTransferSpec:
ReferenceType : *Mode* ;

CopyDirective:
 {
 *SelectionDirective_List*_{opt}
 }

SelectionDirective:
ClassSelector *SelectionPrimitive* *VariableSelector_CommaList*;

SelectionPrimitive:
only |
bypass

ClassSelector:
ClassName

VariableSelector:
VariableName /
all . *TypeName*

IV General

Type:
PrimitiveType /
ReferenceType /
ArrayType

PrimitiveType:
boolean / **byte** / **char** / **short** / **int** / **long** / **float** / **double** /
String

ReferenceType:
Name

ArrayType:
PrimitiveType [] /
Name [] /
ArrayType []

Name:
Identifier /
FullQualifiedName

ClassName:
Identifier

VariableName:
Identifier

QualifiedName:
ClassName . VisibleElementName

VisibleElementName:
*Identifier | **

FullQualifiedName:
Name . Identifier

Note: the following production accepts more than it should. It accepts, for example,

“3+(x == false)”

which is obviously unwanted. The production is given in such general form because this form is much simpler and shorter than the productions that would be necessary to disambiguate those situations. In any case, the ambiguities that this simplification accepts will be caught by the Java compiler after translation.

Expression:

<i>Literal</i>	/
<i>VarRef</i>	/
<i>UnaryExpression</i>	
<i>(Expression)</i>	/
<i>Expression ArithmeticOp Expression</i>	
<i>Expression ConditionalOp Expression</i>	
<i>Expression RelationalOp Expression</i>	
<i>Expression EqualityOp Expression</i>	

Literal:
 (as in Java’s grammar, [23] pages 19-27)

VarRef:

<i>Identifier</i>	/
<i>ArrayRef</i>	/

ArrayRef:
Identifier[ArrayIndex]

ArrayIndex:

Identifier /
IntegerLiteral /

UnaryExpression:

AccessExpression ++ |
AccessExpression -- |
 ++ *AccessExpression* |
 -- *AccessExpression*

ArithmeticOp:

+ / - | * / / | %

ConditionalOp:

|| / &&

RelationalOp:

< / > / <= / <=

EqualityOp:

== / !=

Not:

!