



COTRE Language Reference Manual F241-2

Airbus France
TNI-Valiosys
E.N.S.T. Bretagne
I.R.I.T.
L.A.A.S.
O.N.E.R.A. - D.T.I.M.

Authors	Jean Paul Bodeveix, Mamoun Filali, Pierre Gaufillet, Pierre Michel, François Vernadat
Summary	This document describes the structure & semantics of the COTRE language based on AADL version 0.96.

Attention : la responsabilité des entreprises et des organismes ayant participé à l'élaboration de ce document ne peut en aucun cas être engagée en cas de dommages ou de pertes résultant de l'utilisation ou de l'exploitation des informations qui y sont contenues.

Disclaimer: Contractors participating to this report shall incur no liability whatsoever for any damage or loss which may result from the use or exploitation of Information and/or Rights contained in this report.

Contents

	Page
1 Preface.....	3
1.1 Record of revisions.....	3
1.2 References of applicable documents	4
1.3 Table of Abbreviations.....	5
2 Introduction.....	6
2.1 Purpose of the document.....	6
2.2 Notation.....	6
3 COTRE & AADL.....	6
3.1 Comparison with AADL.....	6
3.2 Language extension principles.....	7
4 COTRE Language Reference.....	8
4.1 Lexical elements.....	8
4.1.1 Key words.....	8
4.1.2 Operators.....	8
4.1.3 Case.....	9
4.1.4 Identifiers and constants.....	9
4.2 File importation	9
4.3 Data typing.....	9
4.3.1 Recognised types.....	9
4.3.2 Simple data.....	10
4.3.3 Structured data.....	10
4.4 Behavior description	10
4.4.1 Overall behavior description	10
4.4.2 Behavior description operation by operation.....	11
4.4.3 Condition & synchronisation.....	11
4.4.4 Actions.....	11
4.4.5 Definition & initialisation of variables.....	12
4.4.6 Behavioral automaton.....	12
4.4.7 Exceptions.....	13
4.4.8 Examples of behavioral sections	13
4.5 Expression of contracts.....	14
4.5.1 Assertions.....	14
4.5.2 Behaviors.....	14
4.5.3 Environmental constraints.....	15
4.5.4 Operating guarantees.....	15
4.6 COTRE-specific property sets	15
Table 1 – Comparison of COTRE & AADL concepts.....	7@~
Table 2 – Coverage of COTRE attributes.....	16@~

1 Preface

1.1 Record of revisions

Version	Date	Description & justification of the modifications	Modified pages
1	05/03/2004	New document	-
2	01/04/2004	Copyright update Update of behavioral equivalences section	All 4.5.2

1.2 References of applicable documents

Reference	Title & Version	Author or originator	Year
AADL	ARCHITECTURE ANALYSIS & DESIGN LANGUAGE (AADL) AS5506 Version 0.96	SAE AS-2C	2004
Cprop1	COTRE properties	COTRE team	2003
Cprop2	Properties and behaviours in AADL	COTRE team	2003
F221	Rapport de description étendue du méta-modèle	A.Bouabdallah, Y.Kermarrec, A.Schach	2003

1.3 Table of Abbreviations

Abbreviation	Description
AADL	Avionic Architecture Description Language
COTRE	COmposants Temps Réel Embarqués
OPCS	OPeration Control Structure (concept HOOD)
SAE	Society of Automotive Engineers
PCP	Priority Ceiling Protocol

2 Introduction

2.1 Purpose of the document

This manual describes the COTRE language. In particular, it contains a description of the syntax and semantics of each construction, and a comparison between COTRE and AADL.

2.2 Notation

The following notation system is used throughout this document:

- Generic elements, `<xxx>`,
- Optional elements, `[xxx]`,
- Groups of elements, `(xxx)`,
- A certain number of elements, `xxx*`,
- At least one element, `xxx+`.

The COTRE symbols which may be confused with the notation presented above shall be shown in bold.

Example: UNITS (...)

3 COTRE & AADL

3.1 Comparison with AADL

To ensure that the real time software architectures can be modeled as finely as necessary, the COTRE project requires an ADL definition. Using a similar approach, the main aim of the AADL project, led by the SAE AS-2C committee, is to define such a language. The similar nature of the needs expressed has led us to study the correspondence between the two approaches. The table below compares the concepts of the two projects.

Exchanges between the COTRE & SAE teams have resulted in: the addition of COTRE-specific requirements in the AADL language & the integration of AADL generalities in the COTRE language.

COTRE object	AADL object	Notes
Action ¹	-	
Application	process	No notion of time partitioning
Attribute ¹	-	
Automaton Modes	modes subclause	
Component	Component type/implementation	AADL provides the concept of component type & component type implementation. This notion is missing from COTRE.
Cyclic	thread	
Data ²	-	No internal data in AADL. This level should feature in a COTRE appendix.
Feature	feature	
Mode	mode	
ModeTransition	mode transition	
Module	package / Thread group / ?	The Module enables prioritisation of both the component declarations and their instance, since COTRE does not contain the notion of instantiation. COTRE's prioritisation requirement is therefore only partially covered by AADL. A structuring of passive instances is missing.
OPCS ¹	-	Leaving aside the AADL execution path definitions, cannot be exploited in practice
Operation	subprogram	AADL introduces the customer/server nuances made necessary by the weakness of the behavioral description. This can be deduced from the COTRE call graph.
Parameter	subprogram parameter	
Periodic	thread	
Process	thread	
Properties (contracts) ³	-	
Protected	data	Check the AADL specific rules for definition of DATAs. Mutual exclusion is not necessarily pertinent. Apart from that, the correspondence should be correct.
Sporadic	thread	
State ¹	-	
State constraints ¹	-	
Synchronization Constraints ¹	-	
System_configuration	system	The AADL system is bigger than the COTRE system, because it can be made up of sub-systems.
Type	data	Partial correspondence since AADL does not enable specific definition of types. It is only a declaration, and the internal details are transferred to the subjacent implementation language.
The OPCS content is not dealt with as there is no AADL equivalent		
Idem for contracts		
	¹	Covered by the COTRE.BEAHVIOR appendix
	²	Covered by the COTRE.TYPE appendix
	³	Covered by the COTRE.ASSUMES or COTRE.GUARANTEES appendices

Table 1 – Comparison of COTRE & AADL concepts

3.2 Language extension principles

Based on the differences identified between the COTRE concepts and the AADL language, the SAE committee has defined mechanisms enabling the language to be extended easily. Two structures are available to users:

- *Properties*: each element of the architecture (component or sub-component) may be associated statically with name/value pairs. The name of these constants, as well as their possible values and the elements they apply to shall be defined in *property sets*.

THREAD IMPLEMENTATION t.t1

PROPERTIES

```

Period                => 13.96ms;
thread_p::Priority    => 1;
cotre::Phase         => 0.0ms;
Dispatch_Protocol    => Periodic;

```

```
Compute_Entrypoint => "t_t1_ep";
```

...

- *Appendices*: this approach enables specific sections to be added to every AADL component. The content of these sections is at the user's discretion. The tools shall simply ignore the content of unknown appendices, which ensures that a minimum degree of compatibility is maintained between them.

```
SYSTEM IMPLEMENTATION ex_deadlock.default
```

```
ANNEX <nom> IS
  <syntaxe libre>
END ANNEX <nom>;
```

```
END ex_deadlock.default;
```

4 COTRE Language Reference

Only the extensions that are specific to the COTRE language are described here. For structures that are common to both COTRE & AADL, refer to [AADL].

4.1 Lexical elements

4.1.1 Key words

All the AADL elements are used by COTRE, and, in addition:

alive	observational
array	periodic_wait
before	potentially
behavior	reachable
branching	receive
called	reset
computation	resume
delay	send
equivalence	skip
exceptions	state
final	states
import	stop
inits	timeout
integer	transition
invariant	transitions
language	unavoidably
leads	when
livelock	

4.1.2 Operators

The following operators, in descending order of priority, line by line, are supported by COTRE:

```
'.'
'-' et '+' unaires NOT
'*' '/' modulo
'+' '-'
'<' '>' '<=' '>='
'==' '!='
AND
OR
```

With the exception of ' - ' and NOT which are right associative, all these operators are left associative.

4.1.3 Case

The COTRE language does not take account of the case (upper or lower).

4.1.4 Identifiers and constants

The COTRE variable and status identifiers shall correspond to the rational expression `[a-z_][a-z0-9_]*`.

Examples: `Lionel_Terray_04`
`__baklava__`
`etat_de_repos`

The constants take the following form:

Whole	' - ' ? [0-9] +
Floating-point	' - ' ? [0-9] + \. [0-9] + (e (' - ' '+') ? [0-9] +) ?
Character chains	\ " . * \ "

4.2 File importation

AADL is not capable of file inclusion. This point is covered in COTRE using the `IMPORT` instruction:

```
IMPORT "<fichier>";
```

`<fichier>` being an absolute or relative path (& expressed in this case based on the location of the file containing the instruction).

4.3 Data typing

Data are typed using the specific appendix *cotre.type*.

4.3.1 Recognised types

The following types are recognised:

Nom	Description
BOOLEAN	Boolean type which can take the values TRUE or FALSE
DATA <id>	DATA component in the AADL sense.
ARRAY <exp1> .. <exp2> OF <type>	Table with issues ranging from exp1 to exp2 & with elements expressed as <type>.
REAL [<exp1> .. <exp2>] [UNITS (<unit1>, <uniti> => <unitj> * <mult>...)]	Real type. May be limited to the interval [<exp1> ; <exp2>], and/or may be associated with units (<unit1>)/sub-units(<uniti>)
INTEGER [<exp1> .. <exp2>] [UNITS (<unit1>, <uniti> => <unitj> * <mult>...)]	Whole type. May be limited to the interval [<exp1> ; <exp2>], and/or may be associated with units (<unit1>)/sub-units (<uniti>)

Examples: **REAL** 0..+INFINITY
INTEGER 10..20
INTEGER UNITS (s, ms => s * 0.001, us => ms * 0.001)
ARRAY 0..10 OF **BOOLEAN**

4.3.2 Simple data

Simple data is typed as follows:

```
DATA IMPLEMENTATION int.s32
    ANNEX cotre.type IS
        <type>;
    END ANNEX cotre.type;
END int.s32;
```

4.3.3 Structured data

Structured data are defined as follows:

```
DATA IMPLEMENTATION int.sub
    ANNEX cotre.type IS
        <champ 1> : <type 1>;
        <champ 2> : <type 2>;
        <champ 3> : <type 3>;
    END ANNEX cotre.type;
END int.sub;
```

4.4 Behavior description

The behavior of objects is described in the appendix *cotre.behavior*, which is broken down as follows:

```
ANNEX cotre.behavior IS
    <section variables>

    <section initialisations>

    <comportement global>|<comportement opération>*
END ANNEX cotre.behavior;
```

4.4.1 Overall behavior description

In this case, the description takes the form:

```
...
STATES
    <déclaration des états>

TRANSITIONS
    <définition des transitions>

EXCEPTIONS
    <définition des exceptions>
...
```

All the components that have behaviours can be described in this way. It is the only possible description for threads.

4.4.2 Behavior description operation by operation

```

...
SUBPROGRAM <nom opération>
  VARS
    <déclaration des variables locales>

  INITS
    <initialisation des variables locales>

  STATES
    <déclaration des états>

  TRANSITIONS
    <définition des transitions>

  EXCEPTIONS
    <définition des exceptions>
...

```

Local variables cannot of course be listed outside of the operation's behavior description, and can only be local variables.

The call constraints between the operations are described at object level using the boolean property *cotre* : *:protected* defined in the standard file *std.cotre*. When this property is true, the operations are mutually exclusive; when it is absent or false, the operations can be performed concurrently.

Operation reentrance is defined at the level of the operation in question by the property *cotre* : *:reentrant*, which may take the values *true* or *false*.

4.4.3 Condition & synchronisation

In COTRE, conditions are expressed as follows:

```

WHEN <condition booléenne> => <événement de synchronisation>

```

The recognised synchronisation events are:

Event	Comments
<nom opération> ! [(<paramètres>)]	Call up of the named operation with possible parameters. The name of the operation is specified using the <i>objet.opération</i> notion (with point). The parameters are separated by commas.
CALLED ?	The operation corresponding to the behavior description is called up.
RESUME [(<paramètres>)]	The caller regains control (but the operation continues).
SEND <identifiant port> [(<paramètres>)]	Message sent on the port indicated.
RECEIVE <identifiant port> [(<paramètres>)]	Message received on the port indicated.

4.4.4 Actions

The actions performed when a transition or exception is fired are noted as shown below:

```

{ <action> ( ; <action> )+ }

```

The possible actions are:

Action	Comment
COMPUTATION(<durée max ou intervalle>)	Consumes a CPU time lower than <durée max> or between the limits of the <intervalle>.
DELAY(<durée max ou intervalle>)	Places the object in a pending state for less than the “durée max” (max. duration) or for a length of time between the limits of the interval.
PERIODIC_WAIT	In the case of a periodic thread, ensures it is blocked until the start of the following period. Equivalent of SKIP in the opposite case.
STOP	The thread is stopped.
SKIP	No action.
<l_exp> := <exp>	Modifies the value of a variable.

4.4.5 Definition & initialisation of variables

Internal variables can be of any type supported by the typing extension COTRE (section 4.3.1). The definition takes the form:

```
ANNEX cotre.behavior IS
  VARS
    <variable> : <type>;
    ...

  INITS
    <variable> := <valeur d'initialisation>;
    ...
```

4.4.6 Behavioral automaton

The automaton enables us to describe the different processing stages for objects and operations. The model used is that of Mealy machines, and the firing conditions and actions are carried by the transitions only. The states, including the initial state, must be declared prior to their use.

```
STATES
  <nom état>(, <nom état>)* : STATE;
  <nom état> : INITIAL STATE;

  TRANSITIONS
    <transition 1>;
    <transition 2>;
    ...
    <transition n>;
```

The transitions take the form:

```
(<label>:)* <état départ> -[ <franchissement> ]-> <état arrivée> { <actions> };
```

<label>	Transition identifier
<franchissement>	Condition to be satisfied in order to activate the transition
<état départ>	Initial state of the transition
<état d'arrivée>	State reached once the transition has been fired.
<actions>	Actions to be performed when the transition is fired.

4.4.7 Exceptions

Exceptions enable us to describe specific behaviors such as time-outs, error detection reactions or, more generally, every process involving general monitoring of the behavior of an executable entity and an interruptive reaction. Exceptions are defined according to the format below:

EXCEPTIONS

```
<sensibilisation> -[ <franchissement> BEFORE <désensibilisation> ]->
    <état d'arrivée> { <actions> };
```

...

<sensibilisation>	Condition to be satisfied in order for an exception to be "sensitized"
<franchissement>	Condition to be satisfied in order to activate the exception when it is "sensitized"
<désensibilisation>	Condition to be satisfied in order to "desensitize" the exception
<état d'arrivée>	State reached once the exception has been fired. The '=' symbol gives the option not to modify the current state.
<actions>	Actions to be performed when the exception is fired.

4.4.8 Examples of behavioral sections

Behavioral description operation by operation:

ANNEX cotre.behavior IS

VARs

```
tokens : INTEGER 0..+infinity;
```

INITS

```
tokens := 1;
```

SUBPROGRAM wait

STATES

```
s0, s1 : STATE;
s0 : INITIAL STATE;
```

TRANSITIONS

```
s0 -[ WHEN tokens > 0 => CALLED ? ]-> s1 { tokens := tokens - 1 };
s1 -[ RESUME ]-> s0;
```

EXCEPTIONS

```
CALLED ? -[ TIMEOUT(300ms) BEFORE WHEN STATE(s1) ]-> s1 { computation(1ms, 2ms) };
```

SUBPROGRAM release

STATES

```
s0, s1 : STATE;
s0 : INITIAL STATE;
```

TRANSITIONS

```
s0 -[ WHEN tokens < capacity => CALLED ? ]-> s1 { tokens := tokens + 1 };
s1 -[ RESUME ]-> s0;
```

END ANNEX cotre.behavior;

General behavioral description (thread case):

ANNEX cotre.behavior IS STATES

```
s0, s1, s2, s3, s4, s5, s6, s7, s8 : STATE;
s0 : INITIAL STATE;
```

TRANSITIONS

```
s0 -[ ]-> s1 { PERIODIC_WAIT };
s1 -[ ]-> s2 { COMPUTATION(1.9ms, 1.9ms) };
s2 -[ s1.wait ! (-1.0ms) ]-> s3;
s3 -[ ]-> s4 { COMPUTATION(0.1ms, 0.1ms) };
t1: s4 -[ s2.wait ! (-1.0ms) ]-> s5;
s5 -[ ]-> s6 { COMPUTATION(2.5ms, 2.5ms) };
t5: s6 -[ s2.release ! ]-> s7;
s7 -[ ]-> s8 { COMPUTATION(1.5ms, 1.5ms) };
s8 -[ s1.release ! ]-> s9;
```

END ANNEX cotre.behavior;

4.5 Expression of contracts

COTRE contracts can express obligations and guarantees, in the form of assertions or behavioral equivalences.

4.5.1 Assertions

The following assertions are supported:

Assertion	Formal Description	Comment
potentially reset	AG EF <i>init</i>	The component can return to its initial state from any other state.
unavoidably reset	AG AF <i>init</i>	The component must return to its initial state from any other state.
is alive	AG EF EX _c true	Component actions must always be possible in the future. Applied to a root component, this assertion implies absence of inter-blocking.
no livelock	AG AF EX _c true	The component must not remain inactive indefinitely.
invariant <exp>	AG <exp>	<exp> must always be true.
<exp1> leads to <exp2> [within <exp3>]	AG(e1=>AF _{<=d>} e2)	The occurrence of <exp1> always leads to the occurrence of <exp2> within a period of time less than <exp3>.
reachable <exp1> [from <exp2>] [within <exp3>]	AG(e1=>EF _{<=d>} e2)	The occurrence of <exp1> may lead to the occurrence of <exp2> within a period of time less than <exp3>.
<exp1> after <exp2>	AG(¬EU(¬e2, e1))	The occurrence of <exp1> is always preceded by the occurrence of <exp2>.

4.5.2 Behaviors

A component's abstract behavior can also be described by an automaton specifying the constraints – sequence, non-determinism, parallelism, conflict, etc. – between the different actions (not necessarily *all* of them) to be observed.

The check then consists of ensuring that there is equivalence between the abstract behavior and the real behaviour. Several behavioral equivalences, offering increasing discrimination power, are considered:

Convention	Comment
language	Ensures that the executions that can be observed from the abstract & real behaviors are identical. Does not take into account either the possibility of blocking (state in which no action can be observed) or the possibility of divergence (infinite execution involving only non-observable actions).
observational	In addition to <i>language</i> equivalence, takes into account blocking states and the conservation of non-determinism. Divergent executions are ignored.
branching	In addition to <i>observational</i> equivalence, takes into account divergent executions.

The associated syntax is as follows:

```

...
BEHAVIOR (<convention> equivalence)
STATES
  <nom état>(, <nom état>)* : STATE;
  <nom état> : INITIAL STATE;

TRANSITIONS
  <transition 1>;
  <transition 2>;
  ...
  <transition n>;
END;
...

```

4.5.3 Environmental constraints

Use the appendix *cotre.assumes* if you want a component to ask its environment to conform to certain constraints.

4.5.4 Operating guarantees

A component announce the properties it guarantees using the appendix *cotre.guarantees*.

4.6 COTRE-specific property sets

As shown in the table below, analysis of the attributes identified in the COTRE meta-model and those of the AADL language has revealed a certain number of points which are not covered.

COTRE	Type COTRE	AADL	Type AADL	Notes
Application:name	String	Nom composant	string	
Component:actif	Boolean	thread		
Component:description	String	-	-	Documentary role
Component:generic	Boolean	-	-	Concept not supported by AADL
Component:heapsize	Integer	Source_Stack_Size	size_range	
Component:instance_of_generic	String	-		Concept not supported by AADL
Component:isClass	Boolean	tous les composants sont des classes	-	
Component:name	String	Nom composant	string	
Component:ods	String	-	-	Covered by the behavioral extensions
Component:size	Integer	Source_Code_Size	size_range	
Data:readOnly	Boolean	Data:Provided_Access	(read_only, write_only, read_write, by_method)	
Data:value	String	-	-	Covered by the behavioral extensions
Feature:description	String	-	-	Documentary role
Feature:name	String	Nom sous-composant	string	
Feature:provided	Boolean	-	-	The notion of protection only appears at the level of package components. The features are always public. This should not pose a problem as far as COTRE is concerned. It is simply necessary to divide the component up into several sub-components, some of which implement exported operations while others supply internal operations.
Mode:isInitial	Boolean	Déclaration des modes	-	
Mode:mds	String			Semantics?
Mode:name	String	Déclaration des modes	-	
ModeTransition:condition	String	-	-	Documentary role
Operation:abstract	Boolean	-		
Operation:budget	Integer			Semantics?
Operation:faultEventRate	Float			Semantics?
Operation:heapsize	Integer	Source_Stack_Size	size_range	
Operation:inherited	Boolean	construction inherits		
Operation:size	Integer	Source_Code_Size	size_range	
Operation:wcet	Float	Subprogram_Execution_Time	time_range	
Parameter:mode_	EnumMode	Déclaration des sous-programmes : utilisation des mots clefs IN, OUT et IN OUT.	-	
Periodic:delta	Float			Semantics?
Periodic:period	Float	thread:Period	time	
Process:actif(h)	Boolean	implicite (thread)	-	
Process:deadline	Float	thread:deadline	time	
Process:priority	Integer	-	-	
Protected:ceiling_priority	Integer	-	-	
Sporadic:minTime	Float	-	-	
Type:predefine	Boolean	Source_Name	string	Refers back to a source code declaration. May therefore provide a link between a COTRE type & an implemented type.
Type:values	EnumValue	-	-	Covered by the behavioral extensions

Table 2 – Coverage of COTRE attributes

The following *properties* are defined explicitly by the COTRE language:

Name	Type	Applies to	Default value	Notes
cotre::ceiling_priority	integer	DATA	-	Applies to DATA components that are protected by the PCP access protocol. Enables definition of the priority taken by the threads accessing data during their critical section.
cotre::Description	string	All components & sub-components	-	Informal comments, requirement traceability info. etc.
cotre::Min_Time	time	THREAD	-	Minimum time between 2 executions of a sporadic thread.
cotre::Phase	time	THREAD	0.0s	Period start time shifted for periodic threads.

cotre::Priority	integer	THREAD	-	Basic thread priority (semantics depend on scheduling strategy).
cotre::Protected	boolean	SUBPROGRAM	false	True if the sub-program executions are exclusive with the component's other sub-programs; false in the opposite case.
cotre::Reentrant	boolean	SUBPROGRAM	false	True if the sub-program is reentrant, false in the opposite case.

Annexe A Grammaire du langage

```

%token UNKNOWN_CHAR /* anything not recognized, to raise a parser error */
%token ACCESS
%token ANNEX_COTRE_ASSUMES
%token ANNEX_COTRE_BEHAVIOR
%token ANNEX_COTRE_GUARANTEES
%token ANNEX_COTRE_TYPE
%token ANNEX_UNKNOWN
%token APPLIES
%token BEHAVIOR
%token BINDING
%token BOOLEAN
%token BUS
%token CASE
%token CLASSIFIER
%token CLIENT
%token CLOSE_GOTO_NEXT /* ]-> */
%token COMPONENT
%token CONNECTIONS
%token CONSTANT
%token DATA
%token DELAYED_CONNECTION /* ->> */
%token DELTA
%token DEVICE
%token DOT_DOT /* .. */
%token DOUBLE_COLON /* :: */
%token END
%token END_ANNEX_COTRE_ASSUMES
%token END_ANNEX_COTRE_BEHAVIOR
%token END_ANNEX_COTRE_GUARANTEES
%token END_ANNEX_COTRE_TYPE
%token END_ANNEX_UNKNOWN
%token ENUMERATION
%token EVENT
%token EXTENDS
%token FALSE
%token FLOAT_LITERAL
%token GOTO /* -> */
%token GROUP
%token IDENTIFIER
%token IMPLEMENTATION
%token IN
%token INFINITY
%token INHERIT
%token INITIAL_MODE
%token INTEGER
%token INTEGER_LITERAL
%token IS
%token LIST
%token LIST_SET_TO /* +=> */
%token MEMORY
%token MODE
%token MODES
%token NONE
%token OF
%token OPEN_GOTO_NEXT /* -[ */
%token ORLESS ORMORE
%token OTHERS
%token OUT
%token PACKAGE
%token PORT
%token PRIVATE
%token PROCESS
%token PROCESSOR
%token PROPERTIES
%token PROPERTY
%token PROVIDES
%token PUBLIC
%token RANGE
%token REAL

```

```
%token REFINED
%token REFINES
%token REQUIRES
%token SERVER
%token SET
%token SET_TO          /* => */
%token STRING
%token STRING_LITERAL
%token SUBCOMPONENTS
%token SUBPROGRAM
%token SYSTEM
%token THREAD
%token TO
%token TRUE
%token TYPE
%token UNITS

%left '.'
%right NEG POS NOT
%left '*' '/' MODULO
%left '+' '-'
%left '<' '>' LTE GTE
%left EQ NEQ
%left AND
%left OR

/* COTRE extensions */

/* Contracts */
/* Assertional expressions */
%token AFTER
%token ALIVE
%token CALLED
%token FALSE
%token FROM
%token GTE
%token INVARIANT
%token LEADS
%token LIVELOCK
%token LTE
%token NEQ
%token NO
%token POTENTIALLY
%token REACHABLE
%token RESET
%token STATE
%token TRANSITION
%token TRUE
%token UNAVOIDABLY
%token WITHIN

/* Behavioral expression */

%token COMPUTATION
%token DELAY
%token FINAL
%token PERIODIC_WAIT
%token RECEIVE
%token RESUME
%token SEND
%token SKIP
%token STOP
%token TIMEOUT
%token WHEN

%token ARRAY
%token ASSIGN
%token BEFORE
%token STATES
%token TRANSITIONS
%token EXCEPTIONS
%token VARS
```

```

%token IMPORT
%token INITS

%token IF
%token ELSE
%token ELSIF
%token THEN

%token BRANCHING
%token OBSERVATIONAL
%token LANGUAGE
%token EQUIVALENCE

%%

/*-----*/
/*-----*/
/*-- AADL SPECIFICATIONS */
/*-----*/
/*-----*/

/*-----*/
/*-- AADL Specifications */
/*-----*/

/* AADL_specification ::=
   { AADL_declaration }+ */

/* add-on COTRE for file importation here */

aadl_specification:
    aadl_declaration
    | cotre_directive
    | aadl_specification aadl_declaration
    | aadl_specification cotre_directive;

cotre_directive:
    IMPORT string_literal ';';

/* AADL_declaration ::=
   component_classifier | package_spec | system_instance | property_set */

aadl_declaration:
    component_classifier
    | package_spec
    | system_instance
    | property_set;

/* component_classifier ::=
   component_type | component_type_extension |
   component_implementation | component_implementation_extension */

component_classifier:
    component_type
    | component_type_extension
    | component_implementation
    | component_implementation_extension;

/*-----*/
/*-- Packages */
/*-----*/

/* package_spec ::=
   package defining_package_name
   ( public package_items [ private package_items ]
     | private package_items )
   end defining_package_name ;
*/

package_spec:

```

```

PACKAGE defining_package_name PUBLIC package_items package_property_associations END
defining_package_name ';;'
| PACKAGE defining_package_name PUBLIC package_items package_property_associations PRIVATE
package_items package_property_associations END defining_package_name ';;'
| PACKAGE defining_package_name PRIVATE package_items package_property_associations END defin-
ing_package_name ';;';

/* package_items ::=
   ( component_type |
     component_type_extension |
     component_implementation |
     component_implementation_extension |
     annex_specification )
  { ( component_type |
    component_type_extension |
    component_implementation |
    component_implementation_extension |
    annex_specification) } *
  [ properties
    ( property_association { property_association }
    | none_statement ) ]

*/

package_items:
  package_items package_item
  | package_item;

package_item:
  component_type
  | component_type_extension;
  | component_implementation
  | component_implementation_extension
  | annex_specification;

package_property_associations:
  /* void */
  | PROPERTIES property_associations
  | PROPERTIES NONE;

defining_package_name:
  /*package_identifier*/ IDENTIFIER '.' defining_package_identifier
  | defining_package_identifier;

/* package_name ::=
   { package_identifier . } * package_identifier
*/

/* DEVIATION : '.' replaced by '::' */

package_name:
  package_name /*'.'*/ DOUBLE_COLON /*package_identifier*/ IDENTIFIER
  | /*package_identifier*/ IDENTIFIER;

/* none_statement ::= none ; */

none_statement:
  NONE ';;';

/*-----*/
/*-- Component Types */
/*-----*/

/* component_type ::=
  component_category defining_component_type_identifier
  [ provides ( { feature }+ | none_statement ) ]
  [ requires ( { required_subcomponent_access }+ | none_statement ) ]
  [ properties ( { component_type_property_association }+ | none_statement ) ]
  { annex_subclause } *
  end defining_component_type_identifier ;

*/

```

```

component_type:
    component_category defining_component_type_identifier
    component_type_internals
    END defining_component_type_identifier ';';

/* component_type_extension ::=
    component_category defining_component_type_identifier
    extends unique_component_type_name
    [ provides ( { feature | feature_refinement }+ | none_statement ) ]
    [ requires
        ( { required_subcomponent_access
            | required_subcomponent_access_refinement }+ | none_statement ) ]
    [ properties ( { component_type_property_association }+ | none_statement ) ]
    { annex_subclause }*
    end defining_component_type_identifier ;

*/

component_type_extension:
    component_category defining_component_type_identifier
    EXTENDS unique_component_type_name
    component_type_extension_internals
    END defining_component_type_identifier ';';

component_type_internals:
    component_type_features
    component_type_requires
    component_type_properties
    annex_subclauses;

component_type_extension_internals:
    component_type_ext_features
    component_type_ext_requires
    component_type_ext_properties
    annex_subclauses;

component_type_ext_properties:
    component_type_properties;

component_type_features:
    /* void */
    | PROVIDES features
    | PROVIDES none_statement;

component_type_ext_features:
    /* void */
    | PROVIDES rfeatures
    | PROVIDES none_statement;

rfeatures:
    feature_refinement
    | feature
    | rfeatures feature
    | rfeatures feature_refinement;

features:
    feature
    | features feature;

component_type_requires:
    /* void */
    | REQUIRES required_subcomponent_accesses
    | REQUIRES none_statement;

component_type_ext_requires:
    /* void */
    | REQUIRES r_required_subcomponent_accesses
    | REQUIRES none_statement;

r_required_subcomponent_accesses:
    required_subcomponent_access
    | required_subcomponent_access_refinement
    | required_subcomponent_accesses required_subcomponent_access_refinement

```

```

| required_subcomponent_accesses required_subcomponent_access;

/* WARNING : required_subcomponent_access not present in the AADL document */
/* Using the subcomponent_access rule */
required_subcomponent_access:
  subcomponent_access;

required_subcomponent_access_refinement:
  subcomponent_access_refinement;

required_subcomponent_accesses:
  required_subcomponent_access
  | required_subcomponent_accesses required_subcomponent_access;

annex_subclauses:
  /* void */
  | annex_subclauses annex_subclause;

component_type_properties:
  /* void */
  | PROPERTIES component_type_property_associations
  | PROPERTIES none_statement;

component_type_property_associations:
  component_type_property_association
  | component_type_property_associations component_type_property_association;

/*-----*/
/*-- Component category */
/*-----*/

/* component_category ::=
      software_category
      | platform_category
      | composite_category
*/

component_category:
  software_category
  | platform_category
  | composite_category;

/* software_category ::= data | thread | thread group | process
*/

software_category:
  DATA
  | THREAD
  | THREAD GROUP
  | PROCESS;

/* platform_category ::= memory | processor | bus | device
*/
platform_category:
  MEMORY
  | PROCESSOR
  | BUS
  | DEVICE;

/* composite_category ::= system
*/
composite_category:
  SYSTEM;

/* unique_component_type_name ::=
  [ package_name :: ] component_type_identifier
*/
unique_component_type_name:
  package_name DOUBLE_COLON /*component_type_identifier*/ IDENTIFIER
  | /*component_type_identifier*/ IDENTIFIER;

/*-----*/

```

```

/*-- Component Implementations                                     */
/*-----*/

/* component_implementation ::=
component_category implementation
component_type_identifier . defining_component_implementation_name
[ refines type
  ( { feature_refinement
    | required_subcomponent_access_refinement }+
    | none_statement ) ]
[ subcomponents ( { subcomponent }+ | none_statement ) ]
[ connections ( { connection }+ | none_statement ) ]
[ modes ( mode_subclause | none_statement ) ]
[ properties
  ( { property_association }+ | none_statement ) ]
{ annex_subclause }*
end component_type_identifier.defining_component_implementation_name ;

*/
component_implementation:
  component_category IMPLEMENTATION /*component_type_identifier*/ IDENTIFIER '.' defining_compon-
ent_implementation_name
  component_implementation_internals
  END /*component_type_identifier*/ IDENTIFIER '.' defining_component_implementation_name ''';

component_implementation_internals:
  component_implementation_refines
  subcomps
  connections
  modes
  implementation_properties
  annex_subclauses;

/* component_implementation_extension ::=
component_category implementation
component_type_identifier . defining_component_implementation_name
extends unique_component_implementation_name
[ refines type
  ( { feature_refinement
    | required_subcomponent_access_refinement }+
    | none_statement ) ]
[ subcomponents
  ( { subcomponent | subcomponent_refinement }+ | none_statement ) ]
[ connections
  ( { connection | connection_refinement }+ | none_statement ) ]
[ modes ( mode_subclause | none_statement ) ]
[ properties
  ( { property_association }+
    | none_statement ) ]
{ annex_subclause }*
end component_type_identifier.defining_component_implementation_name ;

*/
component_implementation_extension:
  component_category IMPLEMENTATION /*component_type_identifier*/ IDENTIFIER '.' defining_com-
ponent_implementation_name
  EXTENDS unique_component_implementation_name
  component_implementation_extension_internals
  END /*component_type_identifier*/ IDENTIFIER '.' defining_component_implementation_name ''';

component_implementation_extension_internals:
  component_implementation_refines
  subcomps_ext
  connections_ext
  modes
  implementation_properties
  annex_subclauses;

subcomps_ext:
  /* void */
  | SUBCOMPONENTS none_statement
  | SUBCOMPONENTS subcomponents_ext;

subcomponents_ext:

```

```

subcomponents_ext subcomponent_ext
| subcomponent_ext;

subcomponent_ext:
subcomponent
| subcomponent_refinement;

connections_ext:
/* void */
| CONNECTIONS none_statement
| CONNECTIONS connections_ext_g;

connections_ext_g:
connections_ext_g connection_ext
| connection_ext;

connection_ext:
connection
| connection_refinement;

component_implementation_refines:
/* void */
| REFINES TYPE none_statement
| REFINES TYPE refinements;

refinements:
refinements refinement
| refinement;

refinement:
feature_refinement
| required_subcomponent_access_refinement;

subcomps:
/* void */
| SUBCOMPONENTS none_statement
| SUBCOMPONENTS subcomponents;

subcomponents:
subcomponents subcomponent
| subcomponent;

connections:
/* void */
| CONNECTIONS none_statement
| CONNECTIONS connections_g;

connections_g:
connections_g connection
| connection;

modes:
/* void */
| MODES none_statement
| MODES mode_subclause;

implementation_properties:
/* void */
| PROPERTIES none_statement
| PROPERTIES property_associations;

/* unique_component_implementation_name ::=
   [package_name :: ]
   component_type_identifier . component_implementation_name
*/
unique_component_implementation_name:
/*component_type_identifier*/ IDENTIFIER '.' component_implementation_name
| package_name DOUBLE_COLON /*component_type_identifier*/ IDENTIFIER '.' component_implementation_name;

/* defining_component_implementation_name ::=

```

```

        defining_component_implementation_identifier | others
*/
defining_component_implementation_name:
    defining_component_implementation_identifier
    | OTHERS;

/* component_implementation_name ::=
    component_implementation_identifier | others | binding
*/
component_implementation_name:
    component_implementation_identifier
    | OTHERS
    | BINDING;

/*-----*/
/*-- Subcomponents */
/*-----*/

/* subcomponent ::=
    defining_subcomponent_identifier_list :
        component_category classifier_reference
        [ required_subcomponent_access_resolution ]
        [ { { subcomponent_property_association
            | contained_property_association }+ } ]
        [ applies_to_modes ] ;
*/
subcomponent:
    defining_subcomponent_identifier_list ':' component_category classifier_reference
    required_subcomponent_access_resolution_section subc_pasb subc_atm ';';

required_subcomponent_access_resolution_section:
    /* void */
    | required_subcomponent_access_resolution;

subc_pasb:
    /* void */
    | '{' subc_pas '}';

subc_pas:
    subc_pas subc_pa
    | subc_pa;

/* BEWARE : contained_property_association can not be differentiated
    from subcomponent_property_association -> only one rule kept here */

subc_pa:
    /*subcomponent_property_association
    | */contained_property_association;

subc_atm:
    /* void */
    | applies_to_modes;

/* subcomponent_refinement ::=
    defining_subcomponent_identifier_list : refined to
        component_category classifier_reference
        [ required_subcomponent_access_resolution ]
        [ { { subcomponent_property_association
            | contained_property_association }+ } ]
        [ applies_to_modes ] ;
*/
subcomponent_refinement:
    defining_subcomponent_identifier_list ':' REFINED TO component_category classifier_reference
    required_subcomponent_access_resolution_section subc_pasb subc_atm ';';

/* identifier_list ::= identifier { , identifier }*
*/
identifier_list:
    identifier_list ',' IDENTIFIER

```

```

| IDENTIFIER;

/* classifier_reference ::=
   [ unique_component_type_name [ . component_implementation_name ] ]
*/
classifier_reference:
/* void */
| unique_component_type_name
| unique_component_type_name '.' component_implementation_name;

/* required_subcomponent_access_resolution ::=
   ( required_access_binding { , required_access_binding }* )
*/
required_subcomponent_access_resolution:
(' required_subcomponent_access_resolution_b ');

required_subcomponent_access_resolution_b:
required_subcomponent_access_resolution_b ',' required_access_binding
| required_access_binding;

/* required_access_binding ::=
   required_identifier =>
   ( required_dataorbus_subcomponent_identifier |
     dataorbus_subcomponent_identifier |
     subcomponent_identifier . provided_dataorbus_subcomponent_identifier )
*/
/* BEWARE : required_dataorbus_subcomponent_identifier can not be differentiated
   from dataorbus_subcomponent_identifier -> only one rule kept here */
required_access_binding:
/*required_identifier SET_TO required_dataorbus_subcomponent_identifier
|*/ required_identifier SET_TO dataorbus_subcomponent_identifier
| required_identifier SET_TO /*subcomponent_identifier*/ IDENTIFIER '.' provided_dataorbus_sub-
component_identifier;

/*-----*/
/*-- Annexes */
/*-----*/

/* annex_subclause ::=
   annex annex_identifier is
     annex_specific_language_constructs
   end annex annex_identifier;

   annex annex_identifier {**
     annex_specific_language_constructs
   **} ;
*/
/* unknown annexes (everyone by default) are discarded by the scanner */
annex_subclause:
ANNEX_UNKNOWN annex_specific_language_constructs END_ANNEX_UNKNOWN
| ANNEX_COTRE_BEHAVIOR cotre_behavior_constructs END_ANNEX_COTRE_BEHAVIOR
| ANNEX_COTRE_BEHAVIOR none_statement END_ANNEX_COTRE_BEHAVIOR
| ANNEX_COTRE_ASSUMES cotre_assumes_constructs END_ANNEX_COTRE_ASSUMES
| ANNEX_COTRE_ASSUMES none_statement END_ANNEX_COTRE_ASSUMES
| ANNEX_COTRE_GUARANTEES cotre_guarantees_constructs END_ANNEX_COTRE_GUARANTEES
| ANNEX_COTRE_GUARANTEES none_statement END_ANNEX_COTRE_GUARANTEES
| ANNEX_COTRE_TYPE cotre_type_constructs END_ANNEX_COTRE_TYPE
| ANNEX_COTRE_TYPE none_statement END_ANNEX_COTRE_TYPE
/* the following rule is covered by the scanner */
/* | ANNEX annex_identifier ANNEX_OPEN annex_specific_language_constructs ANNEX_CLOSE ';' */;

/* annex_specification ::=
   annex annex_identifier is
     annex_specific_reusable_constructs
   end annex annex_identifier;

   annex annex_identifier {**
     annex_specific_reusable_constructs
   **} ;
*/

```

```

annex_specification:
  ANNEX_UNKNOWN annex_specific_reusable_constructs END_ANNEX_UNKNOWN
  | ANNEX_COTRE_BEHAVIOR cotre_behavior_constructs END_ANNEX_COTRE_BEHAVIOR
  | ANNEX_COTRE_BEHAVIOR none_statement END_ANNEX_COTRE_BEHAVIOR
  | ANNEX_COTRE_ASSUMES cotre_assumes_constructs END_ANNEX_COTRE_ASSUMES
  | ANNEX_COTRE_ASSUMES none_statement END_ANNEX_COTRE_ASSUMES
  | ANNEX_COTRE_GUARANTEES cotre_guarantees_constructs END_ANNEX_COTRE_GUARANTEES
  | ANNEX_COTRE_GUARANTEES none_statement END_ANNEX_COTRE_GUARANTEES
  | ANNEX_COTRE_TYPE cotre_type_constructs END_ANNEX_COTRE_TYPE
  | ANNEX_COTRE_TYPE none_statement END_ANNEX_COTRE_TYPE
/* the following rule is covered by the scanner */
/* | ANNEX annex_identifier ANNEX_OPEN annex_specific_reusable_constructs ANNEX_CLOSE ';' */;

/*-----*/
/*-- COTRE type annex */
/*-----*/
cotre_type_constructs:
  ct_fields
  | ct_type ';;';

ct_fields:
  IDENTIFIER ':' ct_type ';;'
  | ct_fields IDENTIFIER ':' ct_type ';;';

ct_type:
  BOOLEAN
  | ct_number_type
  | ARRAY ct_range OF cb_var_type
  | DATA data_classifier_reference;

ct_range:
  cotre_expression DOT_DOT cotre_expression;

ct_number_type:
  REAL
  | REAL ct_range
  | REAL UNITS unit_designator
  | REAL ct_range UNITS unit_designator
  | INTEGER
  | INTEGER ct_range
  | INTEGER UNITS unit_designator
  | INTEGER ct_range UNITS unit_designator;

/*-----*/
/*-- COTRE assumes annex */
/*-----*/
cotre_guarantees_constructs:
  cotre_assumes_constructs;

/*-----*/
/*-- COTRE assumes annex */
/*-----*/
cotre_assumes_constructs:
  cotre_assumes_constructs cotre_assumes_construct
  | cotre_assumes_construct;

cotre_assumes_construct:
  IDENTIFIER ':' cotre_contract
  | cotre_contract
  | IDENTIFIER ':' IDENTIFIER ':' string_literal /* external format contract */;

cotre_contract:
  POTENTIALLY RESET ';;'
  | UNAVOIDABLY RESET ';;'
  | IS ALIVE ';;'
  | NO LIVELOCK ';;'
  | INVARIANT cotre_assumes_expression ';;'
  | cotre_assumes_leadsto_prefix ';;'
  | cotre_assumes_leadsto_prefix WITHIN cotre_assumes_expression ';;'
  | cotre_assumes_reachable_prefix ';;'
  | cotre_assumes_reachable_prefix WITHIN cotre_assumes_expression ';;'
  | cotre_assumes_reachable_from_prefix ';;'
  | cotre_assumes_reachable_from_prefix WITHIN cotre_assumes_expression ';;'

```

```

| cotre_assumes_expression AFTER cotre_assumes_expression ';'
| cotre_behavioral_specification;

cotre_behavioral_specification:
BEHAVIOR '(' cotre_equivalences ')'
cb_states_dcl_block
TRANSITIONS cb_transitions END ';;';

cotre_equivalences:
cotre_equivalences ',' cotre_equivalence
| cotre_equivalence;

cotre_equivalence:
LANGUAGE EQUIVALENCE
| OBSERVATIONAL EQUIVALENCE
| BRANCHING EQUIVALENCE;

cotre_assumes_reachable_from_prefix:
cotre_assumes_reachable_prefix FROM cotre_assumes_expression;

cotre_assumes_reachable_prefix:
REACHABLE cotre_assumes_expression;

cotre_assumes_leadsto_prefix:
cotre_assumes_expression LEADS TO cotre_assumes_expression;

cotre_assumes_expression:
cotre_expression;

cotre_expression:
 '(' cotre_expression ')'
 | cotre_expression AND cotre_expression
 | NOT cotre_expression
 | cotre_expression OR cotre_expression
 | cotre_expression LTE cotre_expression
 | cotre_expression GTE cotre_expression
 | cotre_expression '<' cotre_expression
 | cotre_expression '>' cotre_expression
 | cotre_expression EQ cotre_expression
 | cotre_expression NEQ cotre_expression
 | cotre_expression '+' cotre_expression
 | cotre_expression '-' cotre_expression
 | cotre_expression '*' cotre_expression
 | cotre_expression '/' cotre_expression
 | cotre_expression MODULO cotre_expression
 | IF cotre_expression THEN cotre_expression elsifs ELSE cotre_expression END
 | '-' cotre_expression %prec NEG
 | '+' cotre_expression %prec POS
 | TRUE
 | FALSE
 | STATE '(' IDENTIFIER ')' /* return TRUE if the current state
is simple_identifier */
 | TRANSITION '(' IDENTIFIER ')'
 | CALLED
 | CALLED '(' IDENTIFIER /* local subprogram identifier */ ')'
 | cotre_field
 | cotre_number_term
 | cotre_field '[' cotre_expression ']'
 | INFINITY;

cotre_number_term:
unsigned_numeric_literal
| unsigned_numeric_literal units_identifier;

cotre_field:
cotre_field '.' IDENTIFIER
| IDENTIFIER;

elsifs:
/* void */
| ELSIF cotre_expression THEN cotre_expression elsifs;

/*-----*/

```

```

/*-- COTRE behavior annex */
/*-----*/

/* From Properties and behaviours in AADL, November 2003 */
/* and the yacc grammar of VCOTRE, 02/2004 */

cotre_behavior_constructs:
  cb_vars_block cb_inits_block cb_op_sections
  | /* thread have not any operation, only an entry point */
  cb_vars_block cb_inits_block cb_states_dcl_block TRANSITIONS cb_transitions
  cb_exceptions_block;

cb_exceptions_block:
  /* void */
  | EXCEPTIONS cb_exceptions;

cb_exceptions:
  cb_exceptions cb_exception
  | cb_exception;

cb_exception:
  cb_exception_cond OPEN_GOTO_NEXT cb_exception_cond BEFORE cb_exception_cond
  CLOSE_GOTO_NEXT cb_exception_dest_state cb_actions ';;';

cb_exception_dest_state:
  '='
  | /* state_identifier */ IDENTIFIER;

cb_exception_cond:
  WHEN cb_guard SET_TO cb_interaction
  | cb_interaction
  | WHEN cb_guard
  | TIMEOUT '(' cb_time_expression ')';

cb_op_sections:
  cb_op_sections cb_op_section
  | cb_op_section;

cb_op_section:
  SUBPROGRAM IDENTIFIER cb_vars_block cb_inits_block cb_states_dcl_block
  TRANSITIONS cb_transitions cb_exceptions_block;

cb_states_dcl_block:
  STATES cb_states_declaration cb_initial_state cb_terminal_state;

cb_vars_block:
  VARS cb_vars_declaration
  | /* void */;

cb_vars_declaration:
  /* void */
  | cb_vars_declaration IDENTIFIER ':' cb_var_type ';;';

cb_inits_block:
  /* void */
  | INITS cb_inits;

cb_inits:
  cotre_expression ASSIGN cotre_expression ';'
  | cb_inits cotre_expression ASSIGN cotre_expression ';;';

cb_var_type:
  ct_type;

cb_states_declaration:
  /* state_identifier_list */ identifier_list ':' STATE ';;';

cb_initial_state:
  IDENTIFIER ':' INITIAL_MODE STATE ';;';

cb_terminal_state:
  /* void */
  | IDENTIFIER ':' FINAL STATE ';;';

```

```

cb_transitions:
  cb_l_transition ';'
  | cb_transitions cb_l_transition ';;';

cb_l_transition:
  /* label_identifier */ IDENTIFIER ':' cb_transition
  | cb_transition;

cb_transition:
  /* state_identifier */ IDENTIFIER OPEN_GOTO_NEXT cb_transition_cond
  CLOSE_GOTO_NEXT /* state_identifier */ IDENTIFIER cb_actions;

cb_transition_cond:
  WHEN cb_guard SET_TO cb_interaction
  | cb_interaction
  | WHEN cb_guard
  | /* void */;

cb_interaction:
  cb_operation_call
  | cb_operation_accept
  | cb_operation_resume
  | cb_send_statement
  | cb_receive_statement;

cb_operation_call:
  cb_unique_subprogram_name '!'
  | cb_unique_subprogram_name '!' '(' cb_terms ')';

/* subcomponent.subprogram */
cb_unique_subprogram_name:
  IDENTIFIER '.' IDENTIFIER;

cb_operation_accept:
  /* cb_operation_identifer */ CALLED '?';

cb_operation_resume:
  RESUME
  | RESUME '(' cb_terms ')';

cb_send_statement:
  SEND /* port_identifer */ IDENTIFIER
  | SEND /* port_identifer */ IDENTIFIER '(' cb_terms ')';

cb_receive_statement:
  RECEIVE /* port_identifer */ IDENTIFIER
  | RECEIVE /* port_identifer */ IDENTIFIER '(' cb_terms ')';

cb_terms:
  cb_term
  | cb_terms ',' cb_term;

cb_actions:
  '{' cb_action_b '}'
  | /* void */;

cb_action_b:
  cb_action
  | cb_action_b ';' cb_action;

cb_action:
  COMPUTATION '(' cb_time_range ')'
  | COMPUTATION '(' cb_time_expression ')'
  | DELAY '(' cb_time_range ')'
  | DELAY '(' cb_time_expression ')'
  | PERIODIC_WAIT
  | STOP
  | SKIP
  | cotre_expression ASSIGN cotre_expression
  /*| cb_transition_model*/;

```

```

cb_guard: cotre_expression;

cb_time_expression: cotre_expression;

cb_term: cotre_expression;

cb_time_range:
  cb_time_expression ',' cb_time_expression;

/*-----*/
/*-- Property */
/*-----*/
/* property_set ::=
  property set defining_property_set_identifier is
    { property_type_declaration | property_name_declaration }+
  end defining_property_set_identifier ;
*/
property_set:
  PROPERTY SET defining_property_set_identifier IS
  property_set_internals
  END defining_property_set_identifier ';;';

property_set_internals:
  property_set_internals property_xxxx_declaration
  | property_xxxx_declaration;

property_xxxx_declaration:
  property_name_declaration
  | property_type_declaration;

/* property_type_declaration ::=
  defining_property_type_identifier_list : type property_type ;
*/
property_type_declaration:
  /*defining_property_type_*/ identifier_list ':' TYPE property_type ';;';

/* property_type ::=
  boolean | string
  | enumeration_type | units_type
  | number_type | range_type
  | classifier_type
  | component_property_type
*/
property_type:
  BOOLEAN
  | STRING
  | enumeration_type
  | units_type
  | number_type
  | range_type
  | classifier_type
  | component_property_type;

/* enumeration_type ::=
  enumeration ( defining_enumeration_literal_identifier
    { , defining_enumeration_literal_identifier }* )
*/
enumeration_type:
  ENUMERATION '(' defining_enumeration_literal_identifiers ')';

defining_enumeration_literal_identifiers:
  defining_enumeration_literal_identifiers ',' defining_enumeration_literal_identifier
  | defining_enumeration_literal_identifier;

/* units_type ::=
  units units_list
*/
units_type:
  UNITS units_list;

/* units_list ::=
  ( defining_unit_identifier
    { , defining_unit_identifier => unit_identifier * numeric_literal }* )

```

```

*/
units_list:
  '(' defining_unit_identifier ')'
  | '(' defining_unit_identifier ',' defining_subunit_identifiers ')';

defining_subunit_identifiers:
  defining_subunit_identifiers ',' defining_subunit_identifier
  | defining_subunit_identifier;

defining_subunit_identifier:
  defining_unit_identifier SET_TO unit_identifier '*' numeric_literal;

/* number_type ::=
   real [ simple_range ] [ units unit_designator ]
   | integer [ simple_range ] [ units unit_designator ]
*/
number_type:
  REAL
  | REAL simple_range
  | REAL UNITS unit_designator
  | REAL simple_range UNITS unit_designator
  | INTEGER
  | INTEGER simple_range
  | INTEGER UNITS unit_designator
  | INTEGER simple_range UNITS unit_designator;

/* unit_designator ::=
   units_unique_property_type_identifier
   | units_list
*/
unit_designator:
  units_unique_property_type_identifier
  | units_list;

/* simple_range ::= lower_bound .. upper_bound
*/
simple_range:
  lower_bound DOT_DOT upper_bound;

/* lower_bound ::= numeric_literal | -infinity
*/
lower_bound:
  numeric_literal
  | '-' INFINITY %prec NEG;

/* upper_bound ::= numeric_literal | +infinity | infinity
*/
upper_bound:
  numeric_literal
  | '+' INFINITY %prec POS
  | INFINITY;

/* range_type ::=
   range of number_type
   | range of number_unique_property_type_identifier
*/
range_type:
  RANGE OF number_type
  | RANGE OF number_unique_property_type_identifier;

/* classifier_type ::=
   classifier [ ( component_category { , component_category }* ) ]
*/
classifier_type:
  CLASSIFIER
  | CLASSIFIER '(' component_categories ')';

component_categories:
  component_categories ',' component_category
  | component_category;

/* component_property_type ::=
   component [ ( component_category { , component_category }* ) ]

```

```

*/
component_property_type:
  COMPONENT
  | COMPONENT '(' component_categories ')';

/* unique_property_type_identifier ::=
   [ property_set_identifier :: ] property_type_identifier
*/
unique_property_type_identifier:
  /*property_set_identifier*/ IDENTIFIER DOUBLE_COLON property_type_identifier
  | property_type_identifier;

/* property_name_declaration ::=
   defining_property_name_identifier_list : [ access ] [ inherit ]
   [ list of ] property_type_designator
   [ => default_property_expression ]
   applies to ( property_category { , property_category }* ) ;
*/
property_name_declaration:
  /*defining_property_name_*/identifier_list ':' pnd_modifier property_type_designator
  pnd_default_property_expression APPLIES TO '(' property_categories ')';

pnd_modifier:
  /* void */
  | ACCESS
  | ACCESS INHERIT
  | ACCESS LIST OF
  | ACCESS INHERIT LIST OF
  | INHERIT
  | INHERIT LIST OF
  | LIST OF;

pnd_default_property_expression:
  /* void */
  | SET_TO default_property_expression;

/* property_type_designator ::=
   property_type | unique_property_type_identifier
*/
property_type_designator:
  property_type
  | unique_property_type_identifier;

property_categories:
  property_category
  | property_categories ',' property_category;

/* property_category ::=
   component_category
   | [ event ] [ data ] port
   | [ server | client ] subprogram
   | [ connection_type ] connections
*/
property_category:
  component_category
  | PORT
  | EVENT PORT
  | DATA PORT
  | EVENT DATA PORT
  | SUBPROGRAM
  | SERVER SUBPROGRAM
  | CLIENT SUBPROGRAM
  | CONNECTIONS
  | connection_type CONNECTIONS;

/* connection_type ::=
   subprogram
   | [ event ] [ data ] port
*/
connection_type:
  SUBPROGRAM
  | PORT

```

```

| EVENT PORT
| DATA PORT
| EVENT DATA PORT;

property_associations:
  property_associations property_association
  | property_association;

/* property_association ::=
  [ property_set_identifier :: ] property_name_identifier ( => | +=> )
  [ constant ] property_expression { , property_expression }*
  [ applies_to_modes ] ;
*/

property_association:
  /* property_name_identifier */ IDENTIFIER pa_set_to_or_list_set_to pa_constant
  property_expressions pa_applies_to_modes ';'
  | IDENTIFIER /* property_set_identifier */ DOUBLE_COLON /* property_name_identifier */ IDENTIFIER pa_set_to_or_list_set_to pa_constant
  property_expressions pa_applies_to_modes ''';

/* access_property_association ::=
  [ property_set_identifier :: ] property_name_identifier ( => | +=> )
  [ constant ] access property_expression { , property_expression }*
  [ applies_to_modes ] ;
*/

access_property_association:
  IDENTIFIER DOUBLE_COLON /* property_name_identifier */ IDENTIFIER pa_set_to_or_list_set_to
  pa_constant
  ACCESS property_expressions pa_applies_to_modes ''';
  | /* property_name_identifier */ IDENTIFIER pa_set_to_or_list_set_to pa_constant
  ACCESS property_expressions pa_applies_to_modes ''';

pa_set_to_or_list_set_to:
  SET_TO
  | LIST_SET_TO;

pa_constant:
  /* void */
  | CONSTANT;

property_expressions:
  property_expressions ',' property_expression
  | property_expression;

pa_applies_to_modes:
  /* void */
  | applies_to_modes;

/*contained_property_association ::=
  [ property_set_identifier :: ]
  { contained_unit_identifier . }+
  property_name_identifier ( => | +=> )
  [ constant ] property_expression { , property_expression }*
  [ applies_to_modes ] ;
*/

contained_property_association:
  property_set_identifier DOUBLE_COLON cpa_cuis /*property_name_identifier*/
  cpa_op cpa_cte property_expressions subc_atm ''';

cpa_cuis:
  cpa_cuis '.' contained_unit_identifier
  | contained_unit_identifier
  | /* void */;

cpa_op:
  SET_TO
  | LIST_SET_TO;

cpa_cte:
  /* void */
  | CONSTANT;

```

```

/* property_expression ::=
   boolean_term
   | number_term
   | string_term
   | enumeration_term
   | range_term
   | property_term
   | subcomponent_classifier_term
   | component_term
   | case_term
*/
/* BEWARE : enumeration, property and component can not be
   recognized. They have been melt into only one rule : qualified_identifier_term */
property_expression:
   boolean_term
   | number_term
   | string_term
   /*| enumeration_term*/
   | range_term
   /*| property_term*/
   | subcomponent_classifier_term
   /*| component_term*/
   | qualified_identifier_term /* new rule replacing enumeration and so on */
   | case_term;

/* boolean_term ::=
   true
   | false
   | not boolean_term
   | boolean_term and boolean_term
   | boolean_term or boolean_term
   | numeric_literal orless ( boolean_term { , boolean_term }* )
   | numeric_literal ormore ( boolean_term { , boolean_term }* )
   | ( boolean_term )
*/
boolean_term:
   TRUE
   | FALSE
   | NOT boolean_term
   | boolean_term AND boolean_term
   | boolean_term OR boolean_term
   | numeric_literal ORLESS '(' boolean_terms ')'
   | numeric_literal ORMORE '(' boolean_terms ')'
   | '(' boolean_term ')';

boolean_terms:
   boolean_terms ',' boolean_term
   | boolean_term;

/* number_term ::= numeric_literal [ units_identifier ]
*/
number_term:
   numeric_literal
   | numeric_literal units_identifier;

/* string_term ::= string_literal
*/
string_term:
   string_literal;

/* enumeration_term ::= enumeration_identifier
*/
/*enumeration_term:
   enumeration_identifier;*/

/* range_term ::=
   number_term .. number_term [ delta number_term ]
*/
range_term:
   number_term DOT_DOT number_term
   | number_term DOT_DOT number_term DELTA number_term;

/* property_term ::= [ property_set_identifier :: ] property_name_identifier

```

```

*/
/*property_term:*/
/* property_name_identifier */ /*IDENTIFIER*/
/*| IDENTIFIER DOUBLE_COLON*/ /* property_name_identifier */ /*IDENTIFIER;*/

/* subcomponent_classifier_term ::=
   component_category
   [ unique_component_type_identifier
   [ . ( component_implementation_identifier | others ) ] ]
*/
subcomponent_classifier_term:
  component_category
  | component_category unique_component_type_identifier
  | component_category unique_component_type_identifier '.' component_implementation_identifier
  | component_category unique_component_type_identifier '.' OTHERS;

qualified_identifier_term:
  component_term
  | IDENTIFIER DOUBLE_COLON /* property_name_identifier */ IDENTIFIER;

/* component_term ::=
   subcomponent_identifier { . subcomponent_identifier }*
*/
component_term:
  component_term '.' /*subcomponent_identifier*/ IDENTIFIER
  | /*subcomponent_identifier*/ IDENTIFIER;

/* case_term ::=
   ( { case binding case_label :
     property_expression { , property_expression }* ; }+ )
*/
case_term:
  '(' case_bindings ')';

case_bindings:
  case_bindings case_binding
  | case_binding;

case_binding:
  CASE BINDING case_label ':' property_expressions ';';

/* case_label ::=
   [ extends ] subcomponent_classifier_term
   | others
*/
case_label:
  EXTENDS subcomponent_classifier_term
  | subcomponent_classifier_term
  | OTHERS;

/*-----*/
/*-- Modes */
/*-----*/

/* mode_subclause ::= mode_spec mode_behavior
*/
/* DEVIATION : MODE BEHAVIOR has been added between the 2 sections
/* to avoid conflicts */
mode_subclause:
  mode_spec MODE BEHAVIOR mode_behavior;

/* mode_spec ::= initial_mode { additional_mode }*
*/
mode_spec:
  initial_mode additional_modes;

additional_modes:
  /* void */
  | additional_modes additional_mode;

/* initial_mode ::=
   defining_mode_identifier : initial mode [ mode_property_associations ] ;
*/

```

```

initial_mode:
  defining_mode_identifer ':' INITIAL_MODE MODE mode_property_associations ';'
  | defining_mode_identifer ':' INITIAL_MODE MODE ';' ;

/* additional_mode ::=
  defining_mode_identifer { , defining_mode_identifer }* : mode
  [ mode_property_associations ] ;
*/
additional_mode:
  defining_mode_identifiers ':' MODE mode_property_associations ';'
  | defining_mode_identifiers ':' MODE ';' ;

/* mode_property_associations ::=
  { { mode_properties_association }+ }
*/
mode_property_associations:
  '{' mode_property_associations_b '}' ;

mode_property_associations_b:
  mode_property_association
  | mode_property_associations mode_property_association ;

defining_mode_identifiers:
  defining_mode_identifiers ',' defining_mode_identifer
  | defining_mode_identifer ;

/* mode_behavior ::= { mode_transition }*
*/
mode_behavior:
  mode_behavior mode_transition
  | /* void */ ;

/* mode_transition ::=
  source_mode_identifer { , source_mode_identifer }*
  -[ unique_port_identifer { , unique_port_identifer }* ]->
  destination_mode_identifer ;
*/
mode_transition:
  source_mode_identifiers OPEN_GOTO_NEXT unique_port_identifiers CLOSE_GOTO_NEXT
  destination_mode_identifer ';' ;

source_mode_identifiers:
  source_mode_identifiers ',' source_mode_identifer
  | source_mode_identifer ;

/* unique_port_identifer ::=
  component_type_port_identifer
  | subcomponent_identifer . port_identifer
*/
unique_port_identifer:
  component_type_port_identifer
  | /*subcomponent_identifer*/ IDENTIFIER '.' port_identifer ;

unique_port_identifiers:
  unique_port_identifiers ',' unique_port_identifer
  | unique_port_identifer ;

/* applies_to_modes ::=
  applies to { ( mode_identifer { , mode_identifer }* ) | none_statement }
*/
applies_to_modes:
  APPLIES TO '{' mode_identifiers '}'
  | APPLIES TO '{' none_statement '}' ;

mode_identifiers:
  mode_identifiers ',' mode_identifer
  | mode_identifer ;

/* applies_to_modes_and_transitions ::=
  applies to { ( mode_or_transition { , mode_or_transition }* ) | none_statement }
*/
applies_to_modes_and_transitions:
  APPLIES TO '{' mode_or_transitions '}'

```

```

| APPLIES TO '{' none_statement '}';

/* mode_or_transition ::=
   mode_identifer | ( old_mode_identifer -> new_mode_identifer )
*/
mode_or_transition:
  mode_identifer
  | '(' old_mode_identifer GOTO new_mode_identifer ')';

mode_or_transitions:
  mode_or_transitions ',' mode_or_transition
  | mode_or_transition;

/*-----*/
/*-- Features */
/*-----*/

/* feature ::=
   port_spec |
   clientserver_subprogram |
   subprogram_spec |
   provided_data_or_bus_subcomponent_access
*/
feature:
  port_spec
  | clientserver_subprogram
  | subprogram_spec
  | provided_data_or_bus_subcomponent_access;

/* feature_refinement ::=
   port_refinement |
   clientserver_subprogram_refinement |
   subprogram_refinement |
   provided_data_or_bus_subcomponent_access_refinement
*/
/* BEWARE : provided_data_or_bus_subcomponent_access_refinement, as
   subcomponent_access_refinement leads to a reduce/reduce conflict
   should be fixed. Commented out for this first version */
feature_refinement:
  port_refinement
  | clientserver_subprogram_refinement
  | subprogram_refinement
  /* | provided_data_or_bus_subcomponent_access_refinement*/;

/* port_spec ::=
   defining_port_identifer_list : ( in | out | in out ) port_type
   [ { { port_property_association }+ } ] ;
*/
port_spec:
  /*defining_port_*/identifer_list ':' ps_prefix port_type ';'
  | /*defining_port_*/identifer_list ':' ps_prefix port_type '{' port_property_associations '}'
  ';' ;

port_property_associations:
  port_property_associations port_property_association
  | port_property_association;

ps_prefix:
  IN
  | OUT
  | IN OUT;

/* port_refinement ::=
   defining_port_identifer_list : refined to
   ( in | out | in out ) port_type
   [ { { port_property_association }+ } ] ;
*/
port_refinement:
  /*defining_port_*/identifer_list ':' REFINED TO ps_prefix port_type ';'
  | /*defining_port_*/identifer_list ':' REFINED TO ps_prefix port_type '{' port_property_asso-
  ciations '}' ';' ;

```

```

/* port_type ::=
   ( data port [data_classifier_reference ] )
   | ( event data port [data_classifier_reference ] )
   | ( event port )
*/
port_type:
  DATA PORT
  | DATA PORT data_classifier_reference
  | EVENT DATA PORT
  | EVENT DATA PORT data_classifier_reference
  | EVENT PORT;

data_classifier_reference:
  data_unique_component_type_identifier
  | data_unique_component_type_identifier '.' data_implementation_identifier;

/* subprogram_spec ::=
   defining_subprogram_identifier_list : subprogram
   [ subprogram_signature ]
   [ { { subprogram_property_association }+ } ] ;
*/
subprogram_spec:
  /*defining_subprogram*/identifier_list ':' SUBPROGRAM ss_subprogram_signature ';'
  | /*defining_subprogram*/identifier_list ':' SUBPROGRAM ss_subprogram_signature
  '{' subprogram_property_associations '}' ';' ;

subprogram_property_associations:
  subprogram_property_associations subprogram_property_association
  | subprogram_property_association;

/* subprogram_refinement ::=
   defining_subprogram_identifier_list : refined to subprogram
   [ subprogram_signature ]
   [ { { subprogram_property_association }+ } ] ;
*/
subprogram_refinement:
  /*defining_subprogram*/identifier_list ':' REFINED TO SUBPROGRAM ss_subprogram_signature ';'
  | /*defining_subprogram*/identifier_list ':' REFINED TO SUBPROGRAM ss_subprogram_signature
  '{' subprogram_property_associations '}' ';' ;

/* clientserver_subprogram ::=
   defining_subprogram_identifier_list : subprogram_direction subprogram
   [ subprogram_signature | unique_subprogram_name ]
   [ { { subprogram_property_association }+ } ] ;
*/
clientserver_subprogram:
  /*defining_subprogram*/identifier_list ':' subprogram_direction SUBPROGRAM
  ss_ss_usn ';'
  | /*defining_subprogram*/identifier_list ':' subprogram_direction SUBPROGRAM
  ss_ss_usn '{' subprogram_property_associations '}' ';' ;

/* clientserver_subprogram_refinement ::=
   defining_subprogram_identifier_list : refined to
   subprogram_direction subprogram
   [ subprogram_signature | unique_subprogram_name ]
   [ { { subprogram_property_association }+ } ] ;
*/
clientserver_subprogram_refinement:
  /*defining_subprogram*/identifier_list ':' REFINED TO subprogram_direction SUBPROGRAM
  ss_ss_usn ';'
  | /*defining_subprogram*/identifier_list ':' REFINED TO subprogram_direction SUBPROGRAM
  ss_ss_usn '{' subprogram_property_associations '}' ';' ;

ss_ss_usn:
  /* void */
  | subprogram_signature
  | unique_subprogram_name;

/* subprogram_direction ::= client | server
*/
subprogram_direction:
  CLIENT
  | SERVER;

```

```

/* subprogram_signature ::=
   ( parameter { , parameter }+ )
*/
subprogram_signature:
  '(' parameters ')';

parameters:
  parameters ',' parameter
  | parameter; /* void ? */

/* parameter ::=
   identifier_list : ( in | out | in out ) [ data_classifier_reference ]
*/
parameter:
  identifier_list ':' ps_prefix
  | identifier_list ':' ps_prefix data_classifier_reference;

/* unique_subprogram_name ::=
   [ package_identifier :: ] [ data_type_identifier . ] subprogram_identifier
*/
unique_subprogram_name:
  subprogram_identifier
  | /*package_identifier*/ IDENTIFIER DOUBLE_COLON subprogram_identifier
  | /*package_identifier*/ IDENTIFIER DOUBLE_COLON data_type_identifier '.' subprogram_identifier
  | data_type_identifier '.' subprogram_identifier;

/* subcomponent_access ::=
   defining_subcomponent_access_identifier_list :
     subcomponent_access_classifier
     [ { { access_property_association }+ } ] ;
*/
subcomponent_access:
  /*defining_subcomponent_access_*/identifier_list ':' subcomponent_access_classifier ';'
  | /*defining_subcomponent_access_*/identifier_list ':' subcomponent_access_classifier
  {' access_property_associations '}' ' ';

access_property_associations:
  access_property_associations access_property_association
  | access_property_association;

/* subcomponent_access_refinement ::=
   defining_subcomponent_access_identifier_list : refined to
     subcomponent_access_classifier
     [ { { access_property_association }+ } ] ;
*/
subcomponent_access_refinement:
  /*defining_subcomponent_access_*/identifier_list ':' REFINED TO subcomponent_access_classifier
  ';'
  | /*defining_subcomponent_access_*/identifier_list ':' REFINED TO subcomponent_access_classifier
  er
  {' access_property_associations '}' ' ';

/* subcomponent_access_classifier ::=
   ( data | bus ) access [ unique_component_type_identifier
     [ . component_implementation_name ] ]
*/
subcomponent_access_classifier:
  data_or_bus ACCESS
  | data_or_bus ACCESS unique_component_type_identifier
  | data_or_bus ACCESS unique_component_type_identifier '.' component_implementation_name;

data_or_bus:
  DATA
  | BUS;

/* subcomponent_access_property_refinement ::=
   defining_subcomponent_access_identifier_list : refined to
     ( data | bus ) access unique_component_implementation_name
     { { access_property_association }+ } ;
*/

/* Where is it used ? */

```

```

/*-----*/
/*-- Connection */
/*-----*/

/* connection ::=
   data_connection
   | event_connection
   | event_data_connection
   | client_server_subprogram_connection
*/
connection:
  data_connection
  | event_connection
  | event_data_connection
  | client_server_subprogram_connection;

/* data_connection ::=
   [ defining_data_connection_identifier :]
   data port source_unique_port_identifier
     ( -> | ->> ) destination_unique_port_identifier
     [ { { property_association }+ } ]
     [ applies_to_modes_and_transitions ] ;
*/
data_connection:
  dc_prefix DATA PORT source_unique_port_identifier dc_connect
  destination_unique_port_identifier dc_pa dc_applies ';;';

dc_pa:
  /* void */
  | '{' property_associations '}'';

dc_applies:
  /* void */
  | applies_to_modes_and_transitions;

dc_connect:
  GOTO /* IMMEDIATE_CONNECTION */
  | DELAYED_CONNECTION;

dc_prefix:
  /* void */
  | /*defining_data_connection_identifier*/ IDENTIFIER ':'';

ec_prefix:
  /* void */
  | /*defining_event_connection_identifier*/ IDENTIFIER ':'';

/*edc_prefix:*/
/* void */
/*| defining_event_data_connection_identifier ':'';*/

cssc_prefix:
  /* void */
  | /*defining_subprogram_connection_identifier*/ IDENTIFIER ':'';

/* event_connection ::=
   [ defining_event_connection_identifier :]
   event port source_unique_port_identifier
     -> destination_unique_port_identifier
     [ { { property_association }+ } ]
     [ applies_to_modes_and_transitions ] ;
*/
event_connection:
  ec_prefix EVENT PORT source_unique_port_identifier GOTO /* IMMEDIATE_CONNECTION */
  destination_unique_port_identifier dc_pa dc_applies ';;';

/* event_data_connection ::=
   [ defining_event_data_connection_identifier :]
   event data port source_unique_port_identifier
     -> destination_unique_port_identifier
     [ { { property_association }+ } ]
     [ applies_to_modes_and_transitions ] ;
*/

```

```

*/
event_data_connection:
  /*edc_prefix*/ ec_prefix EVENT DATA PORT source_unique_port_identifieur GOTO /*
IMMEDIATE_CONNECTION */
  destination_unique_port_identifieur dc_pa dc_applies ';;';

/* client_server_subprogram_connection ::=
  [ defining_subprogram_connection_identifieur ]
  client_server_subprogram source_unique_clientserver_subprogram_identifieur
  -> destination_unique_clientserver_subprogram_identifieur
  [ { { property_association }+ } ]
  [ applies_to_modes_and_transitions ] ;
*/
client_server_subprogram_connection:
  cssc_prefix CLIENT SERVER SUBPROGRAM source_unique_clientserver_subprogram_identifieur GOTO /*
IMMEDIATE_CONNECTION */
  destination_unique_clientserver_subprogram_identifieur dc_pa dc_applies ';;';

/* unique_clientserver_subprogram_identifieur ::=
  component_type_clientserver_subprogram_identifieur
  | subcomponent_identifieur . clientserver_subprogram_identifieur
*/
unique_clientserver_subprogram_identifieur:
  component_type_clientserver_subprogram_identifieur
  | /*subcomponent_identifieur */ IDENTIFIER '.' clientserver_subprogram_identifieur;

/* connection_refinement ::=
  connection_identifieur : refined to { { property_association }+ }
  [ applies_to_modes_and_transitions ] ;
*/
connection_refinement:
  /*connection_identifieur*/ IDENTIFIER ':' REFINED TO '{' property_associations '}' dc_applies
  ';;';

/*-----*/
/*-- System instances */
/*-----*/

/* system_instance ::=
  defining_system_instance_identifieur : system
  system_unique_component_implementation_identifieur
  [ { { component_instance_property_association }+ } ] ;
*/
system_instance:
  defining_system_instance_identifieur ':' SYSTEM unique_component_implementation_identifieur
  system_instance_internals ';;';

system_instance_internals:
  '{' component_instance_property_associations '}'
  | '{' ' ' ';

component_instance_property_associations:
  component_instance_property_associations
  component_instance_property_association
  | component_instance_property_association;

component_instance_property_association:
  IDENTIFIER DOUBLE_COLON contained_component_identifieur_path /* property_name_identifieur */
  IDENTIFIER
  pa_set_to_or_list_set_to pa_constant property_expressions pa_applies_to_modes ';;'
  | contained_component_identifieur_path /* property_name_identifieur */ IDENTIFIER
  pa_set_to_or_list_set_to pa_constant property_expressions pa_applies_to_modes ';;';

contained_component_identifieur_path:
  contained_component_identifieur_path contained_component_identifieur '.'
  | contained_component_identifieur '.';

/*-----*/
/*-- System instances */
/*-----*/

```

```

clientserver_subprogram_identifi er: IDENTIFIER;
component_implementation_identifi er: IDENTIFIER;
component_type_clientserver_subprogram_identifi er: IDENTIFIER;
/*component_type_identifi er: IDENTIFIER;*/
component_type_port_identifi er: IDENTIFIER;
/*connection_identifi er: IDENTIFIER;*/
contained_component_identifi er: IDENTIFIER;
contained_unit_identifi er: IDENTIFIER;
data_implementation_identifi er: IDENTIFIER;
data_type_identifi er: IDENTIFIER;
data_unique_component_type_identifi er: IDENTIFIER;
dataorbus_subcomponent_identifi er: IDENTIFIER;
defining_component_implementation_identifi er: IDENTIFIER;
defining_component_type_identifi er: IDENTIFIER;
/*defining_data_connection_identifi er: IDENTIFIER;*/
defining_enumeration_literal_identifi er: IDENTIFIER;
/*defining_event_connection_identifi er: IDENTIFIER;*/
/*defining_event_data_connection_identifi er: IDENTIFIER;*/
defining_mode_identifi er: IDENTIFIER;
defining_package_identifi er: IDENTIFIER;
defining_property_set_identifi er: IDENTIFIER;
defining_system_instance_identifi er: IDENTIFIER;
/*defining_subprogram_connection_identifi er: IDENTIFIER;*/
defining_unit_identifi er: IDENTIFIER;
destination_mode_identifi er: IDENTIFIER;
destination_unique_port_identifi er: unique_port_identifi er;
destination_unique_clientserver_subprogram_identifi er: unique_clientserver_subprogram_identifi er;
/*enumeration_identifi er: IDENTIFIER;*/
mode_identifi er: IDENTIFIER;
new_mode_identifi er: IDENTIFIER;
number_unique_property_type_identifi er: IDENTIFIER;
old_mode_identifi er: IDENTIFIER;
/*package_identifi er: IDENTIFIER;*/
port_identifi er: IDENTIFIER;
/*property_name_identifi er: IDENTIFIER;*/
property_set_identifi er: IDENTIFIER;
property_type_identifi er: IDENTIFIER;
provided_dataorbus_subcomponent_identifi er: IDENTIFIER;
/*required_dataorbus_subcomponent_identifi er: IDENTIFIER;*/
required_identifi er: IDENTIFIER;
source_mode_identifi er: IDENTIFIER;
source_unique_clientserver_subprogram_identifi er: unique_clientserver_subprogram_identifi er;
source_unique_port_identifi er: unique_port_identifi er;
/*subcomponent_identifi er: IDENTIFIER;*/
subprogram_identifi er: IDENTIFIER;
unique_component_implementation_identifi er: unique_component_implementation_name;
unique_component_type_identifi er: IDENTIFIER;
unit_identifi er: IDENTIFIER;
units_identifi er: IDENTIFIER;
units_unique_property_type_identifi er: IDENTIFIER;

/*defining_subcomponent_access_identifi er_list: identifi er_list;*/
/*defining_subprogram_identifi er_list: identifi er_list;*/
/*defining_port_identifi er_list: identifi er_list;*/
/*defining_property_name_identifi er_list: identifi er_list;*/
/*defining_property_type_identifi er_list: identifi er_list;*/
defining_subcomponent_identifi er_list: identifi er_list;

/*subcomponent_property_association: property_association;*/
component_type_property_association: property_association;
subprogram_property_association: property_association;
port_property_association: property_association;
mode_property_association: property_association;

default_property_expression: property_expression;

ss_subprogram_signature:
/* void */
| subprogram_signature;

annex_specific_language_constructs: /* void */ ;
annex_specific_reusable_constructs: /* void */ ;

```

```
/*provided_data_or_bus_subcomponent_access_refinement:
  subcomponent_access_refinement;*/

provided_data_or_bus_subcomponent_access: subcomponent_access;

numeric_literal:
  '-' unsigned_numeric_literal %prec NEG
  | unsigned_numeric_literal
  | '+' unsigned_numeric_literal %prec POS;

unsigned_numeric_literal:
  INTEGER_LITERAL
  | FLOAT_LITERAL;

string_literal:
  STRING_LITERAL;
/*****/
```