Annexe D : BNF du formalisme H-COSTAM

Version 1.0,
Kordon Fabrice & El Kaim William,
Laboratoire LIP6, Thème Systèmes Répartis Coopératifs
E-mail : Fabrice.Kordon@lip6.fr - William.El-Kaim@lip6.fr

1. Preliminary definitions

char_alpha	::=	**A**|**B**|**C**|**D**|**E**|**F**|**G**|**H**|**I**|**J**|**K**|**L**|**M**|**N**|**O**|**P**|**Q**|**R**|**S**|**T**|**U**|**V**|**W**|**X**|**Y**|**Z**|_ [1]
char_num	::=	**0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**
char	::=	char_alpha | char_num
identifier[2]	::=	alpha_char**{**char**}**
list_identifier	::=	identifier | identifier**,** list_identifier
integer_value	::=	positive_value | negative_value
positive_value	:=	char_alpha**{**char_alpha**}**
negative_value	::=	**-**positive_value
string	::=	**'{**char**}'**

Comments are possible in H-COSTAM. They start anywhere in a line by **//** and finish at the end of the current line. Comments may occur anywhere in any description.

In this document :

- Predefined constructions presented above are referenced using *italic* characters;
- Characters space, tab et carriage return will be considered as separator except in comments or strings;
- Keywords and delimitors are displayed this way : **KEY_WORD**;
- You should pay attention on the comment position. Comments may not take place anywhere in the attributes.

1.1. List of special words that cannot be identifiers in H-COSTAM

Identifiers may not be H-COSTAM Keywords.

Actually, there are external and internal keywords in H-COSTAM. External keywords are the ones defined in the grammar this document describes. Internal Keywords are the ones that are necessary to identify part of text that comes from a graphical description (the one provided by Macao).

Hereafter is the list of unauthorised identifiers in H-COSTAM.

(1) Lower and upper cases are consireded equal.
(2) An identifier cannot be a reserved word of H-COSTAM declaration.

2. Declarations in H-COSTAM pages

Declarations in H-COSTAM are associated to nodes that cannot be removed from the specification. Their shape is the one defined on the left. Of course, there are two types of declarations : one for macro-level pages and one for micro-level pages.

Each declaration objet has five attributes :

- ***name*** that gives the page id (optional except for the root page). This attribute has to be an identifier;
- ***author*** that is a one line free text zone;
- ***version*** that specifies the version of the current page. This attribute has to respect the following format : `integer[.integer[.integer]]`;
- ***comments*** that is a free text multiline zone to write comments about the current page;
- ***declaration*** that defines declarative items. This attribute value has to respect a syntax provided in section 2.2., page 343;

A declarative item is composed with the following elements :

i. generic types and generic constants (both macro and micro levels);
ii. types and constants (micro and macro levels);
iii. local instanciation (macro-level only);
iv. context definition (micro-level only).

2.1. Visibility rules of declarative items

Visibility rules of any declaration item in an H-COSTAM specification do respect the following rules :

- If a page is not generic, any item having class (i) or (ii) on any upper level is visible. Local overwriting of any definition is forbbiden. New items will be added to the other one;
- If a page is generic, all declarative items having class (i) or (ii) on any upper level are forbidens. Only generic parameters and new declarations are visible;
- The visibility of declaration items having class (iii) or (iv) is reduced to the current page.

As mentionned, the default rule is a convenient top down visibility. genericity is dedicated to component reuse. For that reason, visible types are the one explicitely specified by the system designer.

Figure 110 illustrates the visibility rules in H-COSTAM. This example relies on a three pages model. In page **(2)**, it is possible to use types XXX, XXX2 and ZZZ as well as constants YYY and TTT. In page **(3)**, only types AAA and CCC, or constants BBB and DDD are visible. There is no inheritance of definitions coming from page **(1)** because the page is generic. However, it is specified (page **(1)**) that in page **(3)**, AAA renames XXX and BBB renames YYY. So, page **(3)**, types XXX (ranamed AAA) and CCC, as well as constants YYY (renamed BBB) and DDD are visible.

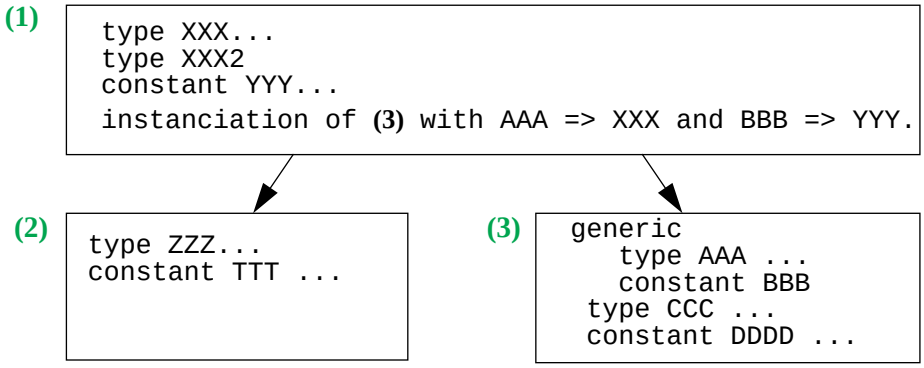


Figure 110 : Visibility rules of declarative items in H-COSTAM.

2.2. BNF of the declaration attribute

Declaration of micro page entities is very similar to the one for macro page entities. The main difference is that instanciation of enclosed «boxes» (processes or subsystems) are only allowed in macro pages and context definition (for a process) is only allowed in micro pages.

Both macro and micro pages may declare local types or constants and may take advantage of genericity.

page_declaration	::=	micro_page_decl | macro_page_decl
macro_page_declaration	::=	**[**generic_declaration**]** local_declaration **[**instanciations**]**
micro_page_declaration	::=	**[**generic_declaration**]** local_declaration context_declaration initial_state_declaration

Generic part

Generic parameters of a page may be either generic types or generic constants. If a page is generic, it must contains at least one type declaration.

generic_declaration	::=	**GENERIC** one_generic_type **{**one_generic_type**}** **{**one_generic_constant**}**

Generic type parameters may be either elementary (it is a finite set of values) or composed (it is a product of at least two others types). Composed types are used to define tuples.

The definition of a generic type parameter enables or disables some operations on it. There are two types of generic parameters :

- ***Elementary generic types*** that enable the use of =, ≠, <, ≤, >, ≥, ++(3), --(4), .all(5) and product operators;
- ***Composed generic types*** that only enable the use of =, ≠, #(6) and product operators. However, # function is only possible if some hypothesis are defined on the composed types. Each hypothesis specifies, for one given field, the type of information it contains. It is then possible to partially define the prototype of a composed token.

(3) Circular successor function.
(4) Circular predecessor function.
(5) Broadcast function.
(6) Extraction of one field in the composed type.

one_generic_type ::= **TYPE** *identifier* **IS** type_definition**;**

type_definition ::= **ELEMENTARY** |
COMPOSED [WITH one_hypothesis **{,** one_hypothesis**}]**

one_hypothesis ::= **FIELD** *positive_value* **HAS TYPE** *identifier*

one_generic_constant ::= **CONSTANT** *identifier* **HAS TYPE** *identifier***;**

Declaration part

Local declaration may be empty. If not, it contains at least one type definition.

local_declaration ::= **DECLARATION** declaration_body

declaration_body ::= **NONE;** |
type_declaration **{**type_declaration**} [{**constant_declaration**}]**

There are three sort of types in H-COSTAM :

- ***integer types*** (elementary types) that corresponds to a finite range of values;
- ***enumerative types*** (elementary types) that group a finite set of enumerative values;
- ***product types*** (composed types) that defines tuples composed of either elementary or composed fields.

type_declaration ::= **TYPE** *identifier* **IS** type_definition**;**

type_definition ::= **RANGE** *integer_value* **..** *integer_value* |
(*list_identifier***)** |
PRODUCT *list_identifier*

constant_declaration ::= **CONSTANT** *identifier* **:** *identifier* **:=** cst_value**;**

Constants may correspond to :

- an immediate value (integer or enumeration litteral);
- a reference to another constant;
- a one level agregate value, it is then associated to a product type. Aggregates are composed constants in the sense of [Ada 83, Ada 95].

cst_value ::= *integer_value* |
identifier |
constant_reference |
(one_agregate_value **{,** one_agregate_value**})**

constant_reference ::= **$***identifier*

one_agregate_value ::= *integer_value* |
identifier |
constant_reference

Instanciation part (macro pages only)

The instanciation part of a macro-level page defines the generic parameter association of any generic box enclosed in the current page. Each generic box in the page must be instanciated at least once. An instanciation may be nammed for readability convenience(7) (optional directive at the end of the instanciation instruction).

instanciations ::= **ENCLOSED** one_instanciation **{**one_instanciation**}**

one_instanciation ::= **IN GENERIC** *identifier* **USE (**par_association **{,** par_association**})**
[TO MAKE *identifier***]**;

par_association ::= *identifier* **=>** one_agregate_value

(7) This enables production of more readable programs in code generation.

Context definition part (micro pages only)

The context definition of a micro-level page defines a set of variables. It is possible to associate a default value to these variables.

context_declaration	::=	**CONTEXT** context_body
context_body	::=	**NONE** | context_element **{**context_element**}**
context_element	::=	*identifier* **:** *identifier* **[:=** one_agregate_value**];**

initial_state definition part (micro pages only)

The initial state declarative part does define the set of initial instances for the current process in the micro level page. If it is set to «none», it means that there are no initial instances for the current process. Otherwise, the system designer has to specify the number of instances having a given profile.

Each profile should contains the following definitions :

- the initial process-state of the process;
- a set of default values for some context piece. If some context_piece is not enumerated, no value will be preaffected to it(8).

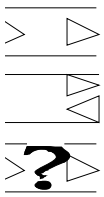
initial_state_declaration	::=	**INITIAL_STATE** init_state_content
init_state_content	::=	**NONE;** | one_proc_instance_profile **{**one_proc_instance_profile**}**
one_proc_instance_profile	::=	*positive_value* **INSTANCE HAS STATE =>** *identifier* **AND CONTEXT => (**one_process_profile**);**
one_process_profile	::=	**NONE** | one_ctx_elem_default_value **{,** one_ctx_elem_default_value**}**
one_ctx_elem_default_value	::=	*identifier* => one_agregate_value

3. Media and Factory definition (micro and macro level)

Media in H-COSTAM define a communication mechanism. There are two classes of media : ***passive*** media that describe asynchroneous communication, and ***active*** media that operate more complex communication mechenism that could need to be managed by a specific server.

Passive media are FIFO links, LIFO links and Random links. There is only one Active media : the multi-rendez-vous(9).

3.1. Passive media



Passive media all contain the same attributes that respect the same rules. These attributes are :

- ***name*** that identify the media;
- ***data_type*** that identify the class of data that goes through the media;
- ***capacity*** that defines the maximum number of messages the media may contains;
- ***initial_state*** that defines messages that are initially contained in the media.

All attributes are reachable when a media take place in a macro level page. Attributes name and data_type are the only one visible in a micro level page.

(8) Their initial value is then a random one.

(9) a RPC communication mechanism should be added soon.

BNF of the name attribute

This attribute does not have to be set in a macro-level page[10]. If it is not, an automatic name is generated. If a name is provided, it has to be an identifier.

passive_media_name ::= *identifier*

BNF of the data_type attribute

This attribute has a default valuee : **NONE** that means only non typed information (events) is transfered. Otherwise, this attribute has to refer a type previously defined in the declaration part of the page.

passive_media_data_type ::= *identifier* |
NONE

BNF of the capacity attribute

This attribute should be an integer. It has a special value : **INFINITY** that means the media is never full. This special value is the one assumed if the attribute is not set

passive_media_capacity ::= *positive_value* |
INFINITY

BNF of the initial_state attribute

This attribute is not yet implemented. The oinly available value is **EMPTY** that means there is no initial messages in the media when the system starts.

passive_media_initial_state ::= **INFINITY**

3.2. Active media

Multi-rendez-vous is the only possible active media. It is defined using the following attribute :

- ***name*** that identify the media;
- ***condition*** that defines the activation rules that are related to the multi-rendez-vous;
- ***interface*** that defines the type of message sent and received by means of this attribute.

The first attribute (name is required in both macro and micro level pages). Condition is required in a macro-level page when the media does not represent an interface. Interface only appear in a micro level page. The name attribute do respect the rule that is defined for media (see definition page 346).

BNF of the condition attribute

The condition attribute defines rules that are used to evaluate a multi-rendez-vous activation and the information exchange performed when this activation occurs.

mrdv_condition_attribute ::= **[**input_output_declaration**]**
[bool_expression**]**[11]
[data_exchange**]**

It is composed with a declarative part that can remain undefined when ***the_condition*** is reduced to TRUE or FALSE. The ***data_exchange*** part is optional.

input_output_declaration ::= one_declaration
{one_declaration**}**

one_declaration ::= direction unit **:** variable **HAS TYPE** type_reference**;**

(10) It has to be set in a micro level page because it may be refered in some expression (condition of modifiers). To get more informations, see XXX

(11) Here, keyword **[** and **]** are expected.

direction ::= **TO** |
FROM

unit ::= *identifier*

variable ::= *identifier*

type_reference ::= *identifier*

Each variable is declared either as an input (**FROM**) of an output (**TO**) variable. An input variable may appear in either ***bool_expression*** or the right hand of an affectation in the ***data_exchange*** part. An output variable can only appear in the left hand of an affectation in the ***data_exchange*** part.

bool_expression ::= elementary_bool_expr |
not bool_expression |
bool_expression **AND** bool_expression |
bool_expression **OR** bool_expression |
(bool_expression**)**

elementary_bool_expr ::= **TRUE** |
FALSE |
reference cmp_operator reference

cmp_operator ::= **=** | **/=** | **>** | **<** | **>=** | **<=**

reference ::= variable_reference |
constant_reference(12) |
identifier |
integer_value

variable_reference ::= **&***identifier* |
&*identifier***#***positive_value*

It is possible to refer a complete variable (expression «&var») or a component if this variable belongs to a composed type. The position of this component is then referenced. So, the expression «&var#3& means «the third component of variable var».

Complex boolean expressions have to fully parenthesed.

data_exchange ::= one_data_affectation
{one_data_affectation**}**

one_data_affectation ::= variable_reference **:=** expression**;**

expression ::= one_agregate_expression |
(one_agregate_expression **{,** one_agregate_expression**})**

one_agregate_expression ::= reference |
variable_reference **++** *positive_value* |
variable_reference **--** *positive_value*

++ and -- operators are recpectively successor and predecessor. So, the expression «&var ++ 4» means : «the successor of rank 4 for the variable var». It is possible to compose agregates but they are limited to one level depth only.

Important Remark : when the attribute is not set, the following default value is assumed : [true].

BNF of the interface attribute

Not yet implemented.

(12) See the definition of this rule page 344.

4. Automata definition (micro level)

A micro level page in H-COSTAM describes a sequential state machine. To achieve that, we use a state transition model composed with two classes of nodes :

- State-processes represent one state in the automata;
- Action-processes represent one action that is performed by a process.

4.1. State-process

A state process has only one attribute : its name that can remain unset. In that case, a default name is assumed. However, it is necessary to provide a name to the state-processes that are initial states of a static instance (see the definition page 345).

The name attribute do respect the rule that is defined for media (see definition page 346).

4.2. Action-process

An action process has three attributes :

- ***Name*** identifies it. This attribute respects the rules defined for media (page 346);
- ***Condition*** defines the activation rules that are related to the condition;
- ***Modifier*** describes what hapens when an action has fired.

BNF of the condition attribute

This attribute describes the conditions that should be respected when the actions fires.

act_proc_condition_attribute::= **[**ap_bool_expression**]**(13)

Unlikely to multi-rendez-vous conditions, inputs are not declared in the condition itself. The condition is applied on both elements the process context (see page 345) and input media.

ap_bool_expression	::=	ap_elementary_bool_expr | **not** ap_bool_expression | ap_bool_expression **AND** ap_bool_expression | ap_bool_expression **OR** ap_bool_expression | **(**ap_bool_expression**)**
ap_elementary_bool_expr	::=	**TRUE** | **FALSE** | ap_reference cmp_operator ap_reference
ap_cmp_operator	::=	**=** | **/=** | **>** | **<** | **>=** | **<=**
ap_reference	::=	ap_context_reference | ap_media_reference | constant_reference(14) | *identifier* | *integer_value*
ap_context_reference	::=	**&***identifier* | **&***identifier***#***positive_value*
ap_media_reference	::=	**%***identifier* | **%***identifier***#***positive_value*

It is possible to refer a complete entity (expression «&ctx_elem» or «%media_ref») or a component if this entity values is of a composed type. The position of this component is

(13) Here, keyword **[** and **]** are expected.
(14) See the definition of this rule page 344.

then referenced. So, the expression «&ctx_elem#3& means «the third component of context element ctx_elem».

Complex boolean expressions have to fully parenthesed.

Important Remark : when the attribute is not set, the following default value is assumed : [true].

BNF of the modifier attribute

This attribute defines the actions to be performed when an action has fired. It may affect any element of the process context. Messages that are produced to output media do have to be specified here.

act_proc_modifier_attribute	::=	**[**one_modification **{**one_modification**}]**
one_modification	::=	modif_reference **:=** expression**;**
expression	::=	one_m_agregate_expression | **(**one_m_agregate_expression **{,** one_m_agregate_expression**})**
one_m_agregate_expression	::=	ap_reference | ap_reference **++** *positive_value* | ap_reference **--** *positive_value*

Unlikely to the data exchange performed when a multi-rendez-vous has been fired (see page 347), one modification involves all the output media connected to the process-action. Ther may also content some modification of elements in the current process context.

Each modification expression is composed with a left hand and a right hand part. The left hand part should only refer either an element of context or an output media. The right hand part should only refer to read-only elements that are either element of the process context or input media of the action.