This document gathers errata of the behavior annex document. Each erratum is described with an identifier, a rationale, and a ptoposed resolution. The identifier is used in the revised version of the document to identify modifications.

D.3 Behavior Specification

ID	D.3-01
Rationale	Section D.3, paragraph (24): ... "A transition out of the initial state is triggered by the initialize action defined in the core AADL standard."... Let us consider an initial complete state called "S". D.3 (L7) states that "Transitions out of complete states must have dispatch conditions." Then transitions out of S must have a dispatch condition. Eventually, when is a transition out of S triggered? When the initialization action is performed (semantics precision (24)) or when the dispatch occurs (legality rule D.3.L7)?
Proposed Resolution	D.3 (28): In a behavior specification, the initial state can be a complete state (i.e. an initial complete state). Such a state is an implicit superposition of two states - an initial state and a complete state - connected by an implicit transition. This implicit transition – from the implicit initial state towards the implicit complete state - is triggered by the initialize action defined in the core standard. No condition can be associated to this implicit transition. No other action than the initialization action defined in the core standard (i.e. call to the Initialize_Entrypoint as defined by a property in the core language) can be associated to the implicit transition. Note that entering (resp. exiting) an initial complete state stands for entering (resp. exiting) the implicit complete state. This means that no transition can reach the implicit initial state.

ID	D.3-02
Rationale	Section D.3, paragraph (24): ... "A dispatch trigger will result in a transition out of a complete state to zero or one execution state."... What is the meaning here? Does the rule mean it can be a complete state? Or does it mean it cannot be a complete state?
Proposed Resolution	D.3 (28): A dispatch trigger will result in a transition out of a complete state and to one of the states defined in the behavior annex (either an execution state or a complete state). A dispatch trigger can be the arrival of input on ports, a subprogram call initiated by another thread, or a timed event in case of a periodic, hybrid or timed Dispatch_Protocol. Reaching a complete state can be interpreted as calling the Await_Dispatch run-time service. Thus a component is suspended if it performs a transition to a complete state, after having executed the action associated to the transition. The next dispatch will restart the thread from that state when a dispatch condition leaving that state is met.

ID	D.3-03
Rationale	Section D.3, paragraph (24): It would be nice to indicate if the final state is reached when the finalize action defined in the core AADL standard is performed, the same way it was explained for the initialization step.
Proposed Resolution	Added in D.4 (8): In a behavior specification, a final state can be a complete state (i.e. an final complete state). Such a state is an implicit superposition of two states – a complete state and a final state - connected by an implicit transition. This implicit transition – from the implicit complete state towards the implicit final state – can only be triggered by the reception of a stop event. No other action than the finalization action (represented by the finalize_entrypoint property from the core language) can be associated to this implicit transition. No execution condition can be associated to this implicit transition. Note that entering (respectively exiting) a final complete state stands for entering (resp. exiting) the implicit complete state.

ID	D.3-04
Rationale	Section D.3, when defining a local variable, it could be interesting to define or override property associations, the same way we can do this in the core of the language when defining the property associations for a subcomponent.
Proposed Resolution	syntax proposal `behavior_variable ::= local_variable_declarator { , local_variable_declarator }* : data_unique_component_classifier_reference [ { { data_classifier_property_association }+ } ] ;`

ID	D.3-05
Rationale	Timeout-catching transitions are allowed to contain timeout-guarded actions. Consider an example s1 -[timeout]-> s2 { action1 } timeout t2 Consider action1 takes more than t2 time. We will have a timeout state to be true (which means "-[timeout]->" transitions are enabled) and we are staying in the state s1. We can get an infinite cycle in the state s1 with true timeout state.
Proposed Resolution	added D.3 (L11): Timeout on behavior actions are not allowed on behavior transitions with timeout conditions. added D.6 (L10): Timeout on behavior actions cannot be used in a transition exiting from a complete state if the dispatch condition of this transition is based on timeout (dispatch_relative_timeout_catch or completion_relative_timeout_catch).

ID	D.3-06
Rationale	It is legal to define a transition with the timout-guarded actions with no timout-catching transition. Note that conjunctions of timeout and other conditions may lead to uncaught timeout and thus non-deterministic behaviour.
Proposed Resolution	added D.6(L9): When a timeout is associated to a behavior action, at least one transition out of the source state should describe the behavior of the component in case this timeout is raised. This condition of this mandatory transition should only be guarder by the timeout catch (not in conjunction or disjunction of other condition).

ID	D.3-07
Rationale	D.3 (30) states : An otherwise execute condition becomes true when all the other execute conditions associated with transitions from the same state are false.
Proposed Resolution	Added legality rule D.3 (L9): A behavior transition with an otherwise condition should not be assigned a priority: such a transition implicitly has a lower priority than all other transitions from the same source state. Added legality rule D.3 (L10): From each state of a Behavior_Specification annex subclause, at most one outgoing transition may use an otherwise condition.

ID	D.3-09
Rationale	Forbid the existence of complete state, that is not a mode, in a mode transition
Proposed Resolution	Added legality rule D.3 (L14): In case of a Behavior_Specification annex subclause without an in modes statement, if a complete state identifier is a mode identifier, then all complete state identifiers in that annex subclause must also be mode identifiers.

ID	D.3-10
Rationale	BA subclauses in subprogram may have more than one final state
Proposed Resolution	removed legality rule D.3(L1): A behavior annex specification for a subprogram must define an initial state and one final state. added "and one or more final states" in D.3 (9)

D.4 Dispatch Behavior Specification

ID	D.4-01
Rationale	Section D.4 (L1): "The dispatch_relative_timeout_catch statement must only be declared for timed threads, and must be declared in only one outgoing transition of a complete state". Can this rule be extended to make a link between each kind of thread (Timed, sporadic, periodic, and hybrid) and the associated dispatch conditions that are allowed? `Exp: s1 -[on dispatch timeout]-> s2;`
Proposed Resolution	A table was added in D.4(L1) to show which kind of dispatch condition is allowed for each dispatch protocol of a thread.

ID	D.4-02
Rationale	Similarly to the issue D.4-01, it would be usefull to make a link between each kind of thread (Timed, sporadic, periodic, and hybrid) and the possibility to add a completion_relative_timeout_catch dispatch condition. `Exp: s1 -[on dispatch timeout 100 ms]-> s2;`
Proposed Resolution	A table was added in D.4(L1) to show which kind of dispatch condition is allowed for each dispatch protocol of a thread.

ID	D.4-03
Rationale	Behavior annex is supposed to enable the specification of complex mode switch logics (complex meaning different than a systematic "or" combination carried on by the core standard). Behavior transitions with logical combinations of event ports should then be allowed when states match the name of opeRationale modes. for example: m1 -[on e1 and (e2 or e3)]-> m2;

Proposed Resolution	Modified D.3 (16): Transitions can be guarded by dispatch conditions, execute conditions, external conditions, or internal conditions.
	
	Add D.3 (19): In other categories of components than threads and subprograms, external conditions specify transition conditions out of execution states, triggered on reception of events, or event data, on event or event data ports. For instance, this type of condition can be used to represent mode transitions in process components.
	
	`behavior_condition ::=`
	`  dispatch_condition`
	`  | execute_condition`
	`  | external_condition`
	
	`external_condition ::=`
	`  on trigger_logical_expression`
	
	`trigger_logical_expression ::=`
	`  event_trigger { logical_operator event_trigger }*`
	
	`event_trigger ::=`
	`  in_event_port_name`
	`  | in_event_data_port_name`
	`  | ( trigger_logical_expression )`
	
	Consistency rule has been added in D.3 (C6) to explain that mode conditions can be used for any component category except threads and subprograms.

ID	D.4-04
Rationale	Mode transitions allow to reference ports of subcomponents as part of a mode transition condition. This should be reflected in the behavior annex language as well. for example: `m1 -[on e1 and (sub.e2 or sub2.e3)]-> m2;`

Proposed Resolution	syntax proposal
	
	`port_component_reference ::=`
	`  subcomponent_name . port_element`

ID	D.4-05
Rationale	AADL now support internal features. These internal features could also trigger a transition

Proposed Resolution	Modified D.3 (16): Transitions can be guarded by dispatch conditions, execute conditions, external conditions, and internal conditions.
	
	Add:
	
	D.3 (20): Internal conditions specify transition conditions out of execution states in order to abstract the reaction of a component on the occurrence of internal events. Internal conditions model interactions among different annexes of the same component. The abstract behavior based on internal conditions may be refined in another Behavior_Specification annex subclause by describing under which circumstances such internal events (or internal event data) can be sent. The internal condition is specified by a disjunction of internal events. The transition is taken when at least one internal event occurred, which resets all the listed internal events (i.e. event is considered as not present).
	
	D.3(21): The quard of a transition may rely on an internal condition in order to abstract the reaction of a component on the occurrence of internal events (or internal event data). The objective is to enable interactions among different annexes of the same component. The abstract behavior based on internal conditions may be refined in another Behavior_Specification annex subclause by describing under which circumstances such internal events (or internal event data) can be sent. The internal condition is specified by a list of internal events (or internal event data): the transition is fired if at least one of the internal events (or internal event data) was used. In this case, all the listed internal events (or internal event data) are reset to their initial configuration (i.e. event is considered as not present).
	
	D.3 (L13): Only transitions out of execute states may have internal conditions.
	
	Syntax modifications:
	
	`behavior_condition ::=`
	`  ...`
	`  | internal_condition`
	`internal_condition ::=`
	`  on internal internal_port_name (or internal_port_name)*`

ID	D.4-06
Rationale	BLESS: §6.6 timeout [(event_port_name*) behavior_time] Reason: The very first BLESS behavior attempted (VVI single-chamber cardiac pacing) could not be expressed with BA's timeout. Instead BLESS timeout dispatch triggers have port lists and explicit timeout durations. Events arriving at, or leaving from, any port in the list reset the timeout.
Proposed Resolution	syntax: `completion_relative_timeout_catch ::= timeout [ ( timeout_reset_port { or timeout_reset_port }* ) ] behavior_time` Description: Added to D.4 (6): A port list may be attached to a completion_relative_timeout_condition_and_catch declaration. In this case, the behaviour is as follows: if no event was received in the time interval specified in the declaration, the timeout expires and the timeout transition from the source state is triggered. Otherwise the timeout value is reset to zero. Note that event or event data ports in the list are only used to reset the timeout: they do not interfere with the dispatch condition of the transition. Thus we can use both input and output ports in this list. Receiving/sending an event/event data port just resets the timeout.

D.5 Component Interaction Behavior Specification

ID	D.5-01
Rationale	Section D.5 (12): !< and !> operators can be used in different transitions, for instance: s1 -[on dispatch]-> s2 {!<R}; s2 -[]-> s1 {R!>}; This would raise problems if s2 is a complete state.
Proposed Resolution	Added a legality rule (D.6, (L7)) stating that series of states reached between the usage of !< and !> operators should not be complete states.

ID	D.5-02
Rationale	D.5 (7) states "Ports causing a dispatch event are implicitly frozen at dispatch time. It is also possible to explicitly freeze additional ports" But Input_Time property may state that the port is frozen some time (i.e. 5 us) after dispatch.
Proposed Resolution	Modified D.5(7) to: Ports causing a dispatch event are implicitly frozen at the time specified by the Input_Time property if the property specifies a deterministic value. It is also possible to explicitly freeze additional ports if it is consistent with their Input_Time property. As long as it remains consistent with the Input_Time property of a port, an explicit call to the Receive_Input service can be performed thanks to the frozen statement of the dispatch condition. With the same consistency constraints with respect to the Input_time property value attached to a port, an explicit call to the Receive_Input service can be performed thanks to the input freeze operator (>>) as a transition action.

ID	D.5-03
Rationale	D.5.(7) states "If the Input_Time property of a port has the value Dispatch_Time, then an explicit call to the Receive_Input service can be performed thanks to the with statement of the dispatch condition." Besides, the grammar gives a "frozen" keyword but no "with" keyword.
Proposed Resolution	Rephrased: "As long as it remains consistent with the Input_Time property of a port, an explicit call to the Receive_Input service can be performed thanks to the frozen statement of the dispatch condition" Note that it requires to add a « Dynamic » input_time property value in the property sets section of the AADL core standard.

ID	D.5-04
Rationale	D.5 (9) describes two different behaviours for nominal execution of the dequeue operation (nominal means not empty). Why? Is it consistent with the next_value operation?

Proposed Resolution	the resolution is four folds:
	
	PART 1) redefine input ports ports operators as follows:
	
	`p>> is equivalent to a call to the Receive_Input runtime service`
	`p'updated is equivalent to a call to the Updated runtime service`
	`p'count is equivalent to a call to the  Get_Count runtime service`
	`p'fresh is equivalent to the following expression: Get_Count > 0`
	`p in a value expression is equivalent to a call to the Get_Value runtime service`
	`p? is equivalent to a call to the Next_Value runtime service`
	`p?(var) is equivalent to a call sequence to Get_Value (with var being assigned the result of this call) and then to Next_Value runtime service`
	PART 2) define error types for runtime services on input ports
	
	`Receive_Input: DefaultException`
	`Updated: DefaultException`
	`Get_Count: NotFrozen,DefaultException`
	`Get_Value: NoValue,NotFrozen,DefaultException`
	`Next_Value: NoValue,NotFrozen,DefaultException`
	note that NoValue is already part of the core standard. Should NotFrozen be added?
	
	PART 3) enable to define error states, error catch transitions, error handle transitions
	
	Error states define states that are reached in case an error occur when using runtime services on a given input port.
	
	Error catch transitions link a given behavior state s with an error state to describe which error state is reached in case an error occurs when executing one of the behaviour transitions out of s.
	
	Error handle transitions link an error state with a behaviour state in order to describe the behaviour actions that are executed depending on the type of error that occurred.
	
	PART 4) In case the error is not handled, the core standard error management is used: the recovery entry point in invoked and the thread is reinitialised.
	
	Resolution of PART 1) has been added to the document, in D.5 (9).
	
	PART 2) to 4) require more discussions. More generally, exception management has been registered in erratum D.6-12

ID	D.5-05
Rationale	D.5 (9) states that p'count is decremented whenever p? is called. This means p'count can get arbitrarily negative.
Proposed Resolution	in D.5 (9), it is now written that p'count is decremented if it is strictly positive.

ID	D.5-06
Rationale	D.5 (15) does not state whether calls to put_value is done when executing p!. If it does, then which state is reached in case put_value is executed on a full queue?

Proposed Resolution	the resolution is four folds:
	PART 1) redefine input ports ports operators as follows:
	`p:=v is equivalent to a call to the put_value service call, v being the value put in the output port variable`
	`p! is equivalent to a call to the send_output runtime service call`
	`p!(var) is equivalent to a call sequence to the put_value service call first (with var being the value put in the output port variable), and then a call to the send_output runtime service`
	PART 2) define error types for runtime services on input ports
	`put_value: DefaultException`
	`send_output: FullQueue,DefaultException`
	PART 3) and PART 4) are the same as PART 3) and PART 4) in the resolution of errata D.5-04
	PART 1) of the resolution has been added to the document, in D.5 (15)
	PART 2) to 4) require more discussions. Exception management has been registered in erratum D.6-12

D.6 Behavior Action Language

ID	D.6-01
Rationale	Section D.6 (L6): "A port name only accepts a one dimension array". This is redundent with the core standard.
Proposed Resolution	Added in D.6 (L6): In conformance with the definition of the core AADL Standard, a port name only accepts a one dimension array.

ID	D.6-03
Rationale	The current version of the BA enables to specify calls to subprograms provided by a data accessible through a data access. The same mechanism should be enabled for data subcomponents, and behavior variables.

Proposed Resolution	syntax proposal:
	`communication_action ::=`
	`  subprogram_prototype_name ! [ ( subprogram_parameter_list ) ]`
	`  | required_subprogram_access_name ! [ ( subprogram_parameter_list ) ]`
	`  | subprogram_subcomponent_name ! [ ( subprogram_parameter_list ) ]`
	`  | subprogram_unique_component_classifier_reference ! [ ( subprogram_parameter_list ) ]`
	`  | output_port_name ! [ ( value_expression ) ]`
	`  | input_port_name >>`
	`  | input_port_name ? [ ( target ) ]`
	`  | required_data_access_name !<`
	`  | required_data_access_name !>`
	`  | required_data_access_name . provided_subprogram_access_name ! [ ( subprogram_parameter_list ) ]`
	▪ changement here
	`  | data_subcomponent_name . provided_subprogram_access_name ! [ ( subprogram_parameter_list ) ]`
	`  | local_variable_name . provided_subprogram_access_name ! [ ( subprogram_parameter_list ) ]`
	`  | *!<`
	`  | *!>`

ID	D.6-04
Rationale	Impossible to reference fields of complex data structures in assignment of subprogram parameter or a local variable (see grammar definition for a "value_variable": incoming_subprogram_parameter_name and local_variable_name should be extended to enable the usage of structure fields using qualified names). Also, impossible to access to the data access features and data access feature prototypes as reference fields.

Proposed Resolution	syntax proposal:
	`data_component_reference ::=`
	`  data_subcomponent_name { . data_field }*`
	`  | data_access_feature_name { . data_field }*`
	`  | local_variable_name { . data_field }*`
	`  | data_access_feature_prototype_name { . data_field }*`
	`data_field ::=`
	`  data_subcomponent_name`
	`  | data_access_feature_name`
	`  | data_access_feature_prototype_name`
	`subprogram_parameter ::=`
	`  subprogram_parameter_name {. data_field }*`

ID	D.6-05
Rationale	The current BA enables to spefify time consuming computations with the computation keyword. However, actual time might depend on the hardware on which the component is bound. This variability is not possible with the current version of the annex.

Proposed Resolution	syntax proposal:
	`timed_action ::=`
	`  computation ( behavior_time [ .. behavior_time ] ) [ in binding ( processor_unique_component_classifier_reference { , processor_unique_component_classifier_reference }* ) ]`

ID	D.6-06
Rationale	frozen keyword is the only one keyword introducing an enumeration of entities (ports for instance) without parenthesis. So this issue proposes to add parenthesis around the port enumeration.

Proposed Resolution	syntax proposal:
	`dispatch_condition ::=`
	`  on dispatch [ dispatch_logical_expression ]`
	`  [ frozen ( frozen_ports ) ]`

ID	D.6-07
Rationale	Possibility to use internal feature in the action language to send events, or event and data.

Proposed Resolution	Added D.5 (17) : Internal events (or event data) may be used to represent interactions among annexes. In the scope of a Behavior_Specification annex subclause, an internal feature may be used to describe under which circumstances an internal event (or event data) is sent.
	Added D.6 (L11): Behavior transitions based on internal conditions cannot be present in the same behavior annex subclause as communication actions using internal features. Indeed, these two constructions of the behavior annex language (i.e. internal conditions and sending internal events) are meant to be used in two separate abstractions of the behavior (one being more precise than the other).
	Syntax modifications:
	`communication_action ::=`
	`  ...`
	`  internal_port_name ! [ ( value_expression) ]`
	`  ...`
	`target ::=`
	`  ...`
	`| internal_port_name`

ID	D.6-08
Rationale	We can't know (without freezing a port) whether it has received some values since the last freeze.
Proposed Resolution	An updated keyword has been added, as well as the following paragraph in D.5(9): p'updated returns true if some new values were received in port p since the last freeze of the port. Note that this operator is used to represent a call to the Updated service defined in the core AADL standard. This operation is performed without freezing the port p.

ID	D.6-09
Rationale	There is no semantics defined for "any" keyword in the right part of assignment_action.
Proposed Resolution	Added: D.6 (21) The keyword any should be used to represent non-deterministic behaviors. The objective is to represent easily that the assigned value could take any of the possible value determined by the data type. This could be used for formal verification or for simulation purpose using a randomly generated value. This keyword is incompatible with the use of code generation techniques.

D.7 Behavior Expression Language

ID	D.7-01
Rationale	Impossible to reference port prototypes as value variable or target.

Proposed Resolution	syntax proposal:
	`value_variable ::=`
	`  incoming_port_name`
	`| incoming_port_name ?`
	`| incoming_subprogram_parameter`
	`| incoming_port_prototype_name`
	`| data_component_reference`
	`| port_name ' count`
	`| port_name ' fresh`
	
	`target ::=`
	`  outgoing_port_name`
	`| outgoing_subprogram_parameter`
	`| data_component_reference`
	`| outgoing_port_prototype_name`

ID	D.7-02
Rationale	The BA should be able to reference property values, for instance for enumerated data types but also for property constants for instance the period of the thread, etc.

Proposed Resolution	syntax proposal:
	l property_constant
	`| property_reference`
	`value_constant ::=`
	`  boolean_literal`
	`| numeric_literal`
	`| string_literal`
	
	`property_constant ::=`
	`  # [ property_set_identifier :: ] property_constant_identifier`
	`property_reference ::=`
	`  # [ property_set_identifier :: ] property_name.field_record_property_name`
	`| component_element_reference # property_name. field_record_property_name`
	`| { package_identifier :: }+ unique_component_classifier_reference # property_name.field_record_property_name`
	`component_element_reference ::=`
	`  subcomponent_name`
	`| local_variable_name`
	`| binded_prototype_name`
	`| feature_name`
	`| self`
	`property_name ::=`
	`  property_identifier { property_field }*`
	`property_field ::=`
	`  { [ integer_value ] }*`
	`| . item_identifier`
	`| . upper_bound`
	`| . lower_bound`
	We also added some paragraphs to precise the semantics:
	D.6(20) Behavior actions enable to reference property constants, and property values of different elements of an AADL model. Manipulation of such properties is limited to properties with timing units, or without units only.
	D.7(8) Behavior expressions may refer property values, and property constants defined in the core AADL standard. Such references start with a # character, and can reference property constants in a property set, or property values associated to component classifiers, subcomponents, and features.
	D.7(9) When the property is a range, the upper bound or lower bound of the property value can be referenced using upper_bound and lower_bound keywords.
	D.7(10) When a property is a record, the field of a property value can be referenced using a dot separator between the property identifier and the field identifier.
	D.7(11) When a property is an array, elements of the property value can be referenced using an integer value between brackets
	D.7(L7) Property references should only be limited to properties without units, or with a time unit.

ID	D.7-03
Rationale	There is no description of timeout in behaviour actions
Proposed Resolution	Added: D.6 (19) Timeout can be associated with behavior actions of a transition; this timeout represents a timing budget given to the transition for completing its actions.

ID	D.7-04
Rationale	no cand or cor in BA: Irreflexive operator (cor and cand) allowing evaluation of first term, and then (possibly) ignoring the second to test for null value or non-zero division.

Proposed Resolution	The committee agreed to use AndThen as a shortcutting and; and OrElse as a shortcutting Or
	Syntax modifications
	`logical_operator ::=`
	`  and | or | xor | and then | or else`
	Explanation
	D.7 (12) In BA, the and operator and the or operator are considered not to implement short circuits in the evaluation of the boolean clause. For instance, consider two boolean variable A, B in the boolean expression A or B. If A is true, B will be evaluated anyway in order to evaluate A or B. To use boolean operators that implements short circuits in the evaluation of the boolean clause, and then/or else should be used. For instance, consider two boolean variable A, B in the boolean expression A or else B. If A is true, B will NOT be evaluated and the result of A or else B will be returned.

ID	D.7-05
Rationale	unary plus is useless
Proposed Resolution	remove unary plus from BA.

ID	D.7-08
Rationale	In D.7, both rem and mod operators exist. The difference between them is not clear
Proposed Resolution	Add: D.7 (13) The definition of the rem and mod operators follows the Ada definition: "Integer division and remainder are defined by the relation A = (A/B)*B + (A rem B) where (A rem B) has the sign of A and an absolute value less than the absolute value of B. The result of the modulus operation is such that (A mod B) has the sign of B and an absolute value less than the absolute value of B; in addition, for some integer value N, this result must satisfy the relation A = B*N + (A mod B)".

ID	D.7-09
Rationale	Add the possibility to test if a data is not initialised
Proposed Resolution	Add (L8) Behaviour variables or data component references used in the right hand side of an assignement must have been initialized.

ID	D.7-12
Rationale	make sure initialisation is not made statically and in a transition out of the initial state
Proposed Resolution	Added D.7 (15) if a behaviour variable or a data component reference is initialized both statically (i.e. using the initial_value property from the data modelling annex, or using an initalization expression with the variable declaration) and in the behavior action block of one of the transitions going from the initial state to a complete state, then the initial value of the data is the one computed dynamically from the transitions out of the initial state.

ID	D.7-13
Rationale	enable assignment of behaviour variables at declaration time
Proposed Resolution	Added D.7 (14): A behavior variable may be initialized statically either using the Initial_Value property from the Data Modeling annex, or using a static initialization at declaration time of the variable. The value used for static initialization is the initial value of the variable (i.e. the value of the variable before entering the initial state of the Behavior_Specification annex sub clause). If a behavior variable is initialized both with the Initial_Value property from the Data Modeling annex, and at declaration time of the variable, then the initial value of that variable is the one used at declaration time of the variable. change syntax: behavior_variable ::= behavior_variable_identifier { , behavior_variable_identifier }* : type [:= value_constant];

General issues

ID	G-01
Rationale	it is not possible to use feature groups in condition triggers or behavior actions in the current definition of the AADL-BA syntax.

Proposed Resolution	feature_group was added in the syntactic definition of "name":
	`name ::=`
	`  { identifier . }*`
	`  identifier { array_index }*`

	G-02
Rationale	usage of integer constant values in table indexes is not possible.

Proposed Resolution	In the BNF,
	`array_index :: [ integer_value_variable ]`
	has been replaced by
	`array_index ::[ integer_value ]`