A UML2.0 PROFILE FOR DEVS: PROVIDING CODE GENERATION CAPABILITIES FOR SIMULATION

Mara Nikolaidou, Vassilis Dalakas, George-Dimitrios Kapos, Loreta Mitsi, Dimosthenis Anagnostopoulos
{mara, vdalakas, gdkapos, mitsi, dimosthe@hua.gr}

Harokopio University of Athens
70 El. Venizelou Str, 17671
Athens, GREECE.

Abstract

DEVS (discrete event system specification) is a popular method for specifying discrete event simulation models. Though there are a number of simulators for DEVS, they usually do not provide an easy-to-use graphical interface, while, even if they do, it is simulator-specific. Thus, even if the modeler specifies the same model using DEVS, he/she has to create it from scratch for every discrete simulator. On the other hand, UML is a standard modeling language providing graphical representation of models and code generation capabilities. In this paper, we discuss our effort to combine UML with DEVS to build simulation models, promoting software engineering methods in the simulation world. Such an endeavor should facilitate a standard method to define DEVS models and promote interoperability between DEVS simulators. The first step towards this, is the formal definition of the DEVS UML 2.0 profile proposed in this paper.

Keywords: Software Engineering, Simulation Models, DEVS, UML.

1 INTRODUCTION

Discrete event simulation is a popular method to conduct simulation experiments. DEVS (Discrete Event System Specification) is a standard method for specifying simulation models (Zeigler, Praehofer, and Kim 2000) Though, DEVS models and the code needed to be written by the system modeler for DEVS simulators, such as DEVSim++ (Kim, Ham, and Kim 1993), are two discrete entities. DEVS simulators do not usually provide an easy-to-use graphical interface, while, even if they do, it is simulator-specific. Thus, even if the modeler specifies the same model using DEVS, he/she has to create it from scratch for every discrete simulator.

Unified Modeling Language (UML) (OMG 2004) is considered as the most popular methodology for software modeling, thus a fundamental skill for software engineers. There is an effort to combine UML with DEVS. A mapping between DEVS models and UML state charts has been introduced in (Schulz, Ewing, and Rozenblit 2000), while in (Hong and Kim 2004), the representation of atomic DEVS models using UML sequence diagrams is proposed. Most of these efforts, focus on mathematical proofs, that mapping DEVS to UML is possible (Zinoviev 2005). In (Feng 2004), an attempt has been undertaken to develop DCharts as a graphics language for DEVS models. DCharts is a UML-like language that does not follow any UML standard. The formal method, proposed by OMG to extend or to restrict UML models, is the definition of a UML profile (OMG 2004), emphasizing the use of UML to describe a specific "world", as the DEVS formalism. Since UML 2.0 profiles are based on formal UML extension mechanisms, can be implemented in any standard UML modelling tool providing automated code generation for DEVS simulators.

In the following, we propose a UML 2.0 profile supporting the description of DEVS simulation models. Our aim is a) to offer a graphical, standardised environment for the definition of DEVS models using the proposed profile and b) if the profile is embedded in a UML modelling tool, to be able to generate code for DEVS simulators. The generated code should correspond to DEVS model definitions forwarded to DEVS simulators.

The paper is structured as follows: A brief description of DEVS formalism and simulation tools is provided in section 2. The scope of DEVS UML 2.0 profile and related implementation issues are presented in section 3. Coupled DEVS model and atomic DEVS model description are focused in section 4 and section 5 respectively. Conclusions and future work reside in section 6.

2 DEVS REVIEW

The DEVS formalism is a conceptual framework consisting of mathematical sets to describe the structure and behavior of a model. Simulation models are specified in a modular and hierarchical form. Two types of models are defined: *atomic models* (behavioural representation), from which larger ones are built and describe basic model functionality, and *coupled models* (structural representation) expressing how basic models are connected in a hierarchical form.
An atomic model consists of inputs, outputs, state variables and functions. Each model is described as:

- set of input ports for receiving external events
- set of output ports for sending external events
- set of state variables and parameters
- internal transition function, which specifies the next state to which the system will transit

- external transition function, which specifies the next system state when an input is received (the next state is computed on the basis of the present state, the elapsed time, and the content of the external input event)
- output function, which generates an external output just before an internal transition occurs
- time advance function, which controls the timing of internal transitions

A coupled DEVS model contains the following information:

- set of components
- set of input and output ports
- external coupling, which connects the input/output ports of the coupled model to one or more input/output ports of the components
- internal coupling, which connects output ports of the components to input ports of other components – when an output is generated by a component it may be sent to the input ports of designated components (in addition to being sent to an output port of the coupled model).

DEVS simulators facilitate system modellers to generate simulation code in one-to-one correspondence with DEVS formalism. The code imported in DEVS simulators consists of DEVS entity declarations (structural and behavioural). DEVS simulators support object-oriented simulation, thus the system modeller defines DEVS models as a set of classes and methods in an object-oriented language, as C++ or Java (Kim 1998). In such a case, all DEVS model entities are defined as ancestors of a predefined class hierarchy provided in DEVS libraries (figure 1).

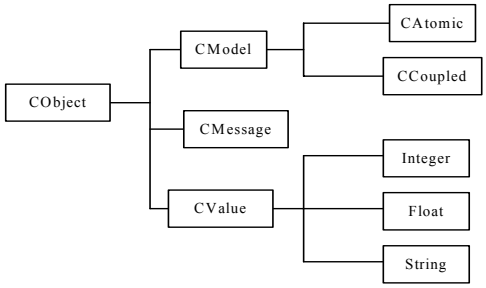

Figure 1: DEVS Simulator Class Hierarchy

CObject is the root of DEVS hierarchy and all the other classes derive from it. CModel defines the construction needed for DEVS modeling. This is specified into: CAtomic and CCoupled. Input and output ports are defined as corresponding attributes. In the atomic model, states are described as combinations of state variable values, thus state variables are defined as attributes of CAtomic class. CMessage manages transmissions of events between models, defining the method for sending output events and receiving input events. CValue class is the basic class for all data type classes. Coupled model class supports methods for defining ports and the coupling between them. Atomic model class, besides methods for handling structural information, for example ports or state variables, also includes methods for the description of system behavior. The implementation of methods for handling structural information is provided in DEVS libraries. The modeler has to specify the implementation of methods corresponding to internal transition, external transition and output functions, namely InTransFn, ExtTransFn and OutputFn, which are model-related and thus can not be predefined. In most cases, system modeler has to write himself object-oriented code in C++ or Java using DEVS libraries.

3 DEVS UML 2.0 PROFILE

Although DEVS simulators support well-defined simulation modeling formalisms, they lack of a standardized, easy-to-use interface facilitating system modelers to define simulation models, independently of their internal characteristics and implementation language. On the other hand, UML provides graphical representation of models regardless of their implementation and automated code generation in most common object-oriented languages, as C++ and Java. Thus, it may support a standardized, easy-to-use, graphical environment for defining DEVS models, that can be consequently executed in existing DEVS Simulators. In such a case, the modeler could describe DEVS model using a popular UML modeling tool, generate the corresponding code in C++ or Java and execute the model in a DEVS simulator. This has no effect in the Simulator, while code generated by the UML modeling tool would serve as a skeleton for the definition of classes and methods corresponding to the model. Class definition can be fully generated, while the same applies to most method implementations corresponding to DEVS function. Only application-specific functionality, unrelated to simulation process, for example statistics computation, would be filled by system modeler. Furthermore, the modeler defines his/her model independently of simulation implementation. The same model can be used to generate code for different simulators built using the same or different programming languages.

In order to be able to use any standard UML modelling tool for defining DEVS models, a formal method to extend UML semantics for DEVS formalism must be used. This is accomplished by the definition of DEVS UML 2.0 profile (OMG 2004), which is properly loaded in any standard UML 2.0 modelling tool. Within the profile, all discrete DEVS entities should be described in an object-oriented fashion, while common DEVS simulator class hierarchy, as presented in figure 1, should also be taken into account. DEVS UML 2.0 profile must provide for the description of all the entities included in figure 1. Alternative UML diagrams are used depicting different aspects of atomic and coupled DEVS models. DEVS model entities are defined as stereotypes of UML 2.0 entities with additional attributes, while constraints are used to restrict UML semantics to DEVS formalism. An except of the stereotypes defined, is presented in figure 2.

The proposed DEVS UML 2.0 profile is implemented using Magic Draw modelling tool (Magic draw, 2007), which fully supports UML 2.0 and provides a Java API. Figures depicting UML diagrams in the rest of the paper are snapshots of the environment. As indicated in figure 3, the tool facilitates the definition of DEVS Profile stereotypes in a standard fashion (left part of the figure), the customization of menus to include DEVS specific diagrams (see corresponding tool bar) and the definition of constraints using OCL (OMG 2003) and Java. The definition of all DEVS constraints using OCL is not trivial and, thus, it was avoided. Some of DEVS constraints are implemented using OCL and other (the more complicated ones) using the provided Java API. The API was also used for the implementation of advanced capabilities, such as the automated generation of states in the Internal Transition and Output Function Diagram (sections 5.1.3 and 5.1.4). Code skeleton generation for C++ DEVS simulators is currently under development, while Java code will be produced as well.

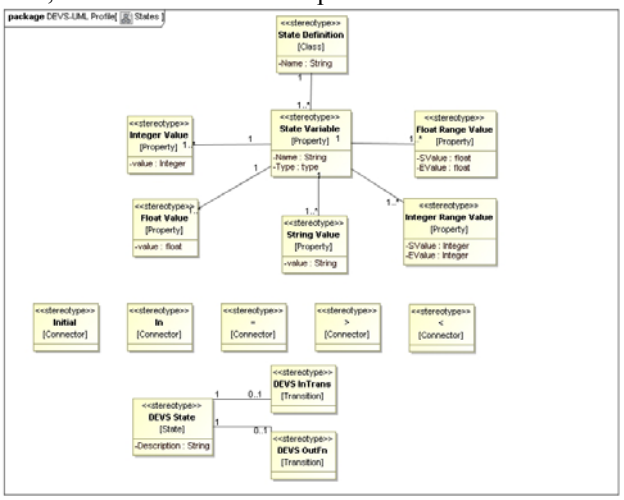

Fig. 2: Except of DEVS stereotypes

4 COUPLED DEVS MODEL

In the context of coupled DEVS, emphasis is given on the component models (atomic or coupled), their interconnections through connection points, called ports, and compositional capability. Thus, UML diagrams that depict structural composition and dependencies of distinct elements, such as class, object, communication or component diagrams, could be used for DEVS CM representation.

UML 2.0 component diagrams provide the means to naturally describe system composition. In UML 2.0 component diagrams, components may be connected or decomposed into other components. Also, they have ports used as the end-points of inter-component connections. Ports facilitating sending or receiving messages and are associated to interfaces indicating whether each port produces (output) or requests (input) data. Figure 3 depicts an example of defining a coupled DEVS model using a UML 2.0 Component Diagram. Both atomic and coupled DEVS models, composing the models, are represented as stereotypes of the UML component element, namely *DEVS AM* and *DEVS CM*. Each stereotype has additional DEVS specific attributes corresponding to the ones describing DEVS entities depicted in Classical DEVS Model (figure 2). Constraints are defined to depict the relationships between DEVS entities (as depicted in figure 2) and restrict UML component diagram functionality to effectively correspond to DEVS formalism. All stereotypes defined for coupled DEVS model reside in table 1 (Appendix A).

In component diagrams, ports can be defined for each component, related to two different types of interfaces determining whether the port requires input or produces output. The same applies to DEVS CM and DEVS AM as well (*DEVS port* stereotypes). As indicated in figure 3:

- Input ports of the external coupled DEVS model (Teller Queue) connect to input ports of the contained DEVS component stereotypes (*External DEVS input port* stereotype of port UML entity). Similarly, output ports of the external coupled DEVS model connect to output ports of the contained components (*External DEVS output port* stereotype of port UML entity). External input (output) ports of the external DEVS CM both realize and use (use and realize) the same interface in order to propagate messages into (out of) the external model.
- All other connections (between internal DEVS models, either DEVS CM or DEVS AM) are made using the interfaces specified between two connected ports, defining the messages that may be sent. *DEVS input ports* are defined as stereotypes of ports realizing the respective interfaces (circle symbol), while *DEVS output ports* are defined as stereotypes of ports using them (arc symbol). Standard UML notation of interface entity is adopted in this case.

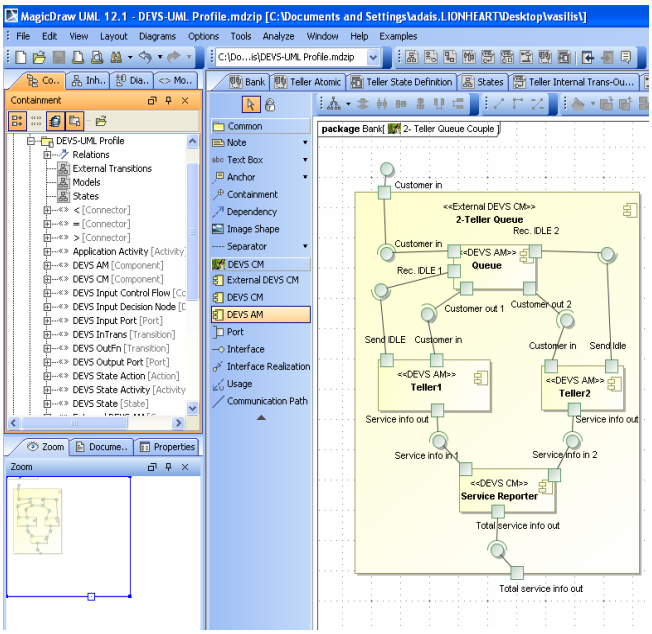

Fig. 3: Coupled DEVS Model

Figure 3 depicts a simple example of a coupled DEVS model. *Two Teller Queue* model is defined as the coupling of atomic (such as *teller*) and coupled (such as *Service Reporter*) models. Coupling is described by the definition of the correspondence among input and output ports of the components and the coupled model. *Queue* receives a customer from *Two Teller Queue* input port to its *customer in* port. *Teller* atomic model serves the received customers from *Queue* atomic model. It waits for a customer in *idle* state. Upon receiving a customer in the input port "*customer in*, *Teller* changes its status to *busy* state and increases *customer* variable. Service time is an exponentially distributed random variable. When the service is finished, it sends customer to the output port *service info out* and *send idle* message to *Queue* informing its state. Component diagrams can not accommodate the description of such functionality, thus a more complex representation of Atomic Model was sought.

5 ATOMIC DEVS MODEL

Defining an atomic DEVS model is divided in two parts:

- Static characteristics definition, such as states, input and output ports and messages.
- Behaviour definition in response to input messages or time advancement.

The diversity of atomic DEVS models leads to the integration of more than one UML diagrams for their definition Component diagrams are used for defining static characteristics (in/out ports) and integrating all other UML diagrams (External View), while a variety of diagrams are related to the corresponding component diagram, to represent atomic model behavior. The diagrams proposed for atomic model description are discussed in the following.

5.1.1 External View

The external view of an atomic DEVS models is defined using a component diagram, in a way similar to coupled DEVS models. For each component stereotype (*External DEVS AM*), used to depict an atomic model, two subdiagrams must be defined: A composite structure diagram facilitating state definition and a state diagram facilitating the definition of internal and output function. Both are discussed in the following. For each input port (*external DEVS input port*), an external function must be defined. This is accomplished through an activity diagram related to each input port.

5.1.2 State Definition

Composite structure diagrams are used for defining atomic DEVS set of states. This is done indirectly through the definition of state variables and their state distinguishing values. State variables are defined as stereotypes of UML 2.0 part entity (*state variable*), while state distinguishing values are defined as stereotypes of UML 2.0 property entity. Since state variables are of certain type, 4 different value stereotypes (*integer value, float value, string value, range value*) of property entity are defined. As shown in figure 8, state variables are represented as parts using solid line rectangles. Each *state variable* is associated with one or more *values* represented as properties using dashed line rectangles. As *values*, one should define the values or value range of each state variable that may lead to different model states (state distinguishing values). Corresponding stereotypes reside in table 2. State variables and values are associated using one of the following associations: *Initial,* $=$, $>$, $<$, $\geq$, $\leq$, $\in$. There must be exactly one *Initial* association for every state variable. On the other hand, the "state determining" associations are as many as the cases where the value (or value range) of the respective state variable determines a distinct occasion in internal state transition. When the value of a state variable does not determine such an occasion, then no such association and property exist. As parts and properties must be contained in a UML 2.0 class entity, *state variables* and *values* must be contained in a *State Definition* entity (stereotype of UML 2.0 class entity)

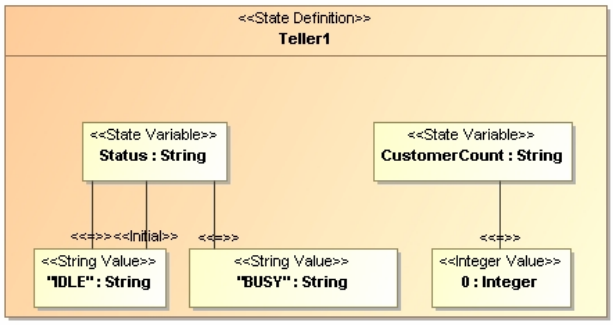

Fig. 4: State Definition Model

Using this diagram, discrete model states can be defined by combing discrete values of all the state variables defined in it. Thus, the computed discrete model states can be automatically inserted in the state diagram, corresponding to Internal Transition and Output functions, discussed in the following paragraph. Two state variables are defined for the teller model. As show in the figure, only *status* state variable actively participates is discrete state definition, since for customerCount only an initial value is defined. Thus, two discrete states are expected to participate in the *Int-Trans/Out* state diagram.

5.1.3 Internal Transition and Output Function Definition

State diagrams are used for the definition of the internal transition function. DEVS states are computed based on State Definition diagram and automatically inserted in the diagram. The modeller specifies internal transitions by inserting simple transitions between states. The initial state is determined by the initial values of each state variable. It

was decided to include Output Function within the diagram rather than define a discrete one for each output port, since output generation is strictly related to internal state transition. Corresponding stereotypes reside in table 3. The *InTrans/Out* diagram for Teller atomic model is depicted in figure 5. There are two discrete states computed. The initial state is defined based on the initial value of status variable. One internal transition is defined accommodated with the definition of two outputs. For each one of them the output port and corresponding output value is defined.

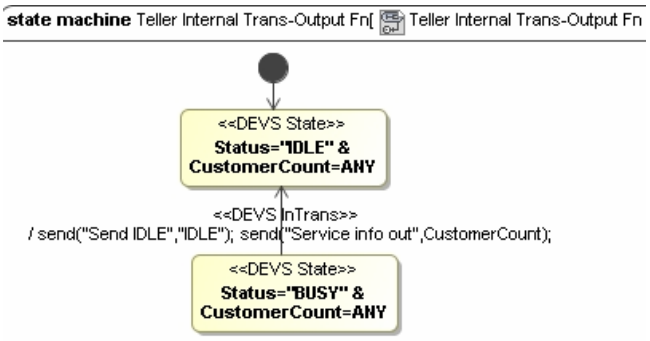

Fig. 5: Teller Internal Transition and Output Function

5.1.4 External transition function

The external transition function of *Customer In* input port of *Teller* model is depicted in figure 6.

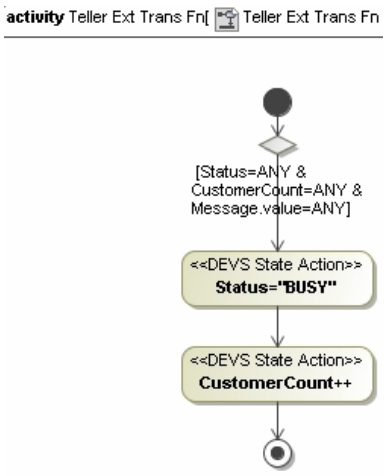

Fig. 6: Teller external transition function

For each input port of the atomic DEVS model, an external transition function must be defined. This is accomplished using an activity sub-diagrams associated with each input port of the atomic DEVS model external view. Two kinds of activities are included in the diagram: a) *DEVS state activity* indicating state variable modification which results in state transitions and b) *application specific activity* corresponding to application specific code (for example statistics computation). Corresponding stereotypes of the UML 2.0 activity entity are defined. A *DEVS state activity* consists of *DEVS actions* (stereotype of UML 2.0 action entity). Each action defines the modification of a specific state variable. Each *DEVS ExtTrans*, defined as a stereotype of UML activity diagram, must start with a decision node indicating the conditions leading to a specific state transition. Conditions represented as *DEVS Input Control Flow* (stereotype of UML 2.0 control flow entity) consists of combinations of state variable and message values, where message entity facilitates the communication between model ports. Stereotypes defined for external transition function definition reside in table 4.

6 CONCLUSIONS – FUTURE WORK

We proposed the use of UML as a standardized, easy-to-use, graphical method to define DEVS simulation models, that can be consequently executed in existing DEVS Simulators. The first step towards this endeavor, is the formal definition of the proposed DEVS UML 2.0 profile and its implementation in Magic Draw tool. Standard profile definition options and the Java API provided were used for the profile implementation. Code skeleton automated generation for DEVSim++ and DEVSJava is currently under implementation.

ACKNOWLEDGMENTS

This research was supported in part by Pythagoras program (MIS 89198) co-funded by the Greek Government (25%) and the European Union (75%).

REFERENCES

Feng, H. 2004, February. Dcharts, a formalism for modeling and simulation based design of reactive software systems. *Master Thesis, McGill University*.

Hong, S.-Y., and T. G. Kim. 2004, July. Embedding UML subset into object-oriented DEVS modeling process. In *Proceedings of SCSC 2004*, 161–166. San Jose, CA.

Kim, T. G. 1998. *DEVSim++ © User's manual. C++ based simulation with hierarchical modular DEVS models*.

Kim, Y. C., K. S. Ham, and T. G. Kim. 1993. Object-oriented memory management in devsim++. In *WSC '93: Proceedings of the 25th conference on Winter simulation*, 670–673. New York, NY, USA: ACM Press.

Magic Draw 2007, *http://www.magicdraw.com/*

OMG 2003, October. UML 2.0 OCL Specification. *Available online via http://www.omg.org/docs/ptc/03-10-14.pdf*

OMG 2004, August. OMG Unified Modeling Language: Superstructure, version 2. *Available online via http://www.omg.org/docs/formal/05-07-04.pdf*

Schulz, S., T. C. Ewing, and J. W. Rozenblit. 2000, April. Discrete event system specification (DEVS) and statemate statecharts equivalence for embedded systems modeling. In *Proceedings of 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 308–316.

Zeigler, B. P., H. Praehofer, and T. Kim. 2000. *Theory of modeling and simulation*. 2nd ed. Academic Press.

Zinoviev, D. 2005, April. Mapping DEVS models onto UML models. In *Proceedings of the 2005 DEVS Integrative M&S Symposium, SpringSim05*, 101–106. San Diego, CA.

Appendix A: DEVS UML 2.0 Profile Definition

Table 1: DEVS Model Stereotypes

DEVS Stereotype	UML Entity	Attributes	Constraints
External DEVS CM	External Component	InPorts OutPorts	- Each component diagram contains exactly one external component. Inside the external component, only elements of the stereotypes defined in this table are allowed. - The values of InPorts and OutPorts are automatically computed based on the diagram.
DEVS CM	Component		- Every DEVS CM is associated with an external DEVS CM diagram.
DEVS AM	Component		- Every DEVS AM is associated with an external DEVS AM diagram.
DEVS input port	Port		- The port must be related to a DEVS CM or a DEVS AM and must implement the InOut interface. - The InOut interface implementation of the DEVS input port must be related to an InOut interface usage of a DEVS CM or DEVS AM or an external DEVS input port.
DEVS output port	Port		- The port must be related to a DEVS CM or a DEVS AM and must use the InOut interface. - The InOut interface usage of the DEVS output port must be related to an InOut interface of a DEVS CM or DEVS AM or an external DEVS output port.
External DEVS CM input port	Port		- The port must be related to an external DEVS CM and must also use and implement the InOut interface. - The used interface of the external DEVS input port must be related to the InOut interface implemented by a DEVS CM or a DEVS AM.
External DEVS CM output port	Port		- The port must be related to an external DEVS CM and must also use and implement the InOut interface. - The implemented interface of the external DEVS output port must be related the InOut interface used by a DEVS CM or a DEVS AM.

Table 2: State Definition Stereotypes

DEVS Stereotype	UML Entity	Attributes	Constraints
State Definition	Class		- The class may only contain stereotypes of parts, properties and associations defined in this table.
State Variable	Part	Type	- The part must be related to exactly one Initial association. - The part may be related to $\in$, =, >, <, $\geq$, $\leq$ associations.
Integer, Float, String Value	Property	Value	- Must be associated with a State Variable Part with Integer type through an Initial association and/or a =, >, <, $\geq$, $\leq$ association.
Integer, Float Range Value	Property	SValue EValue	- Must be associated with a State Variable Part with Integer type through a $\in$ association.
Initial	Association		- Associates a state variable with any property stereotype.
$\in$	Association		- Associates a state variable with a range value stereotype.
=, >, <, $\geq$, $\leq$	Association		- Associates a state variable with any property except rang value.

Table 3: State Transition Stereotypes

DEVS Stereotype	UMLEntity	Attributes	Constraints
DEVS InTrans/Out	State diagram		The diagram contains only stereotypes defined in this table. The diagram can not be defined if there is no corresponsing state definition diagram
DEVS State	State	Description	- The state description is a combination of the state variable values (properties) specified in the state definition diagram. Variables that do not have a specific value are equated to the keyword ANY.
DEVS InTrans	State transition		-Only one transition may start from any single state node
DEVS OutFn	DEVS InTrans		- The effect of the transition may contain one or more invocations of the method send(<port>,<value>) followed by semicolon. <port> is the name of an output port of the DEVS AM already define in External view.

Table 4: External Transition Description Stereotypes

DEVS Stereotype	UML Entity	Attributes	Constraints
DEVS ExtTrans	Activity diagram		- The diagram contains only of stereotypes defined in this table. - The diagram is associated to an External DEVS Input node stereotype of an external DEVS CA component diagram
DEVS Input Decision Node	Decision Node		- Receives control flow from the initial node. - Only DEVS Input Control Flow originate from this decision node.
DEVS Input Control Flow	Control Flow		- Starts from DEVS Input Decision Node and ends at a DEVS State Action or DEVS Application Action. - Has a guard condition that is a logical expression built from conditions on state variable values and message received value (the ANY keyword may be used).
DEVS State Activity	Activity		- Contains only sequential DEVS State Actions.
DEVS State Action	Action		- Describes the modification of a state variable value. - Propagates control flow to another DEVS State Action, to a DEVS Application Action, or to the final node.
Application Activity	Activity		- Receives control flow from a DEVS State Activity