A NEW APPROACH TO MODELING DYNAMIC STRUCTURE SYSTEMS

Oumar Maïga[1,3]
[1]Université des Sciences,
des Techniques et des Technologies
Bamako, Mali
maigabababa78@yahoo.fr

Hamzat Olanrewaju Aliyu[2,3]
[2]School of Info. & Comm. Tech.
Federal University of Technology
Minna, Nigeria
hamzat.aliyu@futminna.edu.ng

Mamadou Kaba Traoré[3]
[3]LIMOS CNRS UMR 6158
Université Blaise Pascal
Clermont-Ferrand, France
traore@isima.fr

KEYWORDS

Dynamic Structure Systems, DEVS, DSDEVS, HiLLS

ABSTRACT

This paper presents a novel approach to modeling Dynamic Structure Systems (DSS) using the High Level Language for System Specification (HiLLS), a graphico-textual language that combines system-theoretic and software engineering concepts from DEVS (Discrete Events System Specifications) and Object-Z respectively to form a unified language for specifying Discrete Events Systems for multiple analysis methodologies.We take benefit of the expressive and user-friendly notations of HiLLS to alleviate the complexity of modeling dynamic-structured systems and use Dynamic Structure DEVS (DSDEVS) as one of its semantics domains to take advantage of the latter's simulation protocol and its associated tool(s). We provide as a case study, the modeling of a Cash Deposit Machine of an automated teller machine to illustrate how HiLLS is used to model DSS and its equivalent DSDEVS model for simulation.

INTRODUCTION

We present a simple approach to the Modeling and Simulation (M&S) of structural changes in complex Discrete Events Systems (DES) using a graphical modeling language. A DSS is a system whose structural properties, such as number of components and/or connections/interactions between them may change dynamically at runtime. Examples of such systems exist in communication networks where connections are dynamically established and broken between nodes, automatic switching systems where a component may be linked to different peer components under different conditions, etc.

In M&S, modeling structural changes in a system is usually an herculean task as most modeling tools lack the required constructs to easily express such properties. With DEVS (Zeigler et al. 2000) as starting point, a number of formalisms have been defined to provide sound theoretical background for the simulation of DSSs; notably among them are DSDEVS (Barros 1995), dynDEVS (Uhrmacher 2001) and rhoDEVS (Uhrmacher et al. 2006). They however lack easy-to-use concrete syntaxes for defining models independently from mathematical equations or program codes; we complement these abstract formalisms with a user-friendly front end.

We propose a graphical modeling language, HiLLS, that supports the specification of a broad range of complex systems including DSSs. HiLLS combines system-theoretic and software engineering concepts from DEVS and Object-Z (Smith 2000) respectively to define an integrated vocabulary of systems constructs for analyses within disparate semantic contexts. We will however limit our discussions about HiLLS in this paper to the M&S of DSSs. This is achieved by adopting DSDEVS as a semantics domain for HiLLS. We prefer DSDEVS among its contemporaries for its safe and simple approach to maintaining consistency between systems' structure and behavior (Muzy and Zeigler 2014).

We present overviews of DEVS-based formalisms for modeling DSSs in the next section, followed by the abstract and concrete syntaxes of HiLLS as well as its mapping to DSDEVS (for simulation semantic). Then we illustrate the approach with a case study before presenting our conclusions and perspectives in the last section.

STATE OF THE ART

DSSs are characterized by operations that manipulate the set of components, couplings, interfaces and behavior of a system (Hu et al. 2005). We present overviews of existing formalisms for modeling DSSs.

For a better understanding of this section, we assume the reader has a basic understanding of the DEVS formalism; a brief introduction is however provided in the appendix for interested reader.

DSDEVS is a variant of DEVS with capability for modeling structural changes. It retains the specification of atomic DEVS while introducing a "*network executive*" model into the coupled network specification to manage structural changes in the latter. The state variables of network executive store the structural information of the network so that there is a one-to-one correspondence between the executive's instantaneous states and the network's structure.

DSDEVS coupled network is described as:

$DSDN = (\chi, M_\chi)$ where χ is the name of the network executive and M_χ is the dynamics of network.

$$M_\chi = \langle X_\chi, S_\chi, s_{o,\chi}, Y_\chi, \gamma, \Sigma^*, \delta_\chi, \lambda_\chi, \tau_\chi \rangle$$

X_χ and Y_χ are the input and output interfaces of the system, S_χ is the set of states, $s_{o,\chi}$ is the initial state, Σ^* is the set of possible structures of the network, $\gamma : S_\chi \longrightarrow \Sigma^*$ is the function which associates a unique network structure to each state of the executive. $\forall \Sigma \in \Sigma^* \wedge s_x \in S_\chi : \gamma(s_\chi) = \langle D, \{M_i\}_{i\in D}, \{I_i\}_{i\in D}, \{Z_i\}_{i\in D} \rangle$
$\delta_\chi : Q_\chi \times (X \cup \phi) \longrightarrow S_\chi$ is the transition function
$\lambda_\chi : S_\chi \longrightarrow Y$ is the output function and
$\tau_\chi : S_\chi \longrightarrow R^+ \cup \{+\infty\}$ is the time advance function.
In DSDEVS, only single central network executive is responsible for the management of structural changes in the network. Any other component is not allowed to modify the network structure or its own structure and/or behavior. This prevents ambiguity when different components require structural change. The single executive also ensures structural and behavioral consistencies.
dynDEVS (Uhrmacher 2001) and rho-DEVS (Uhrmacher et al. 2006, Muzy and Zeigler 2014) are other DSS modeling approaches based on the DEVS formalism. Unlike DSDEVS, these formalisms allow naturally dynamic behavior at atomic level and dynamic structure at network level. In addition, rho-DEVS supports modeling changes in components interfaces. (Muzy and Zeigler 2014) proposed a more general and decentralized approach to modeling DSS in which each component can change its own behavior and network structure.
While these formalisms provide the formal background for M&S of DSS, model specification can be very laborious because modeler must list all the possible structures of the system as a part of the state space of the network executive which can grow exponentially depending on the number of components and dynamic couplings between them. Like DEVS itself, they don't have concrete syntaxes and logical semantics.
HiLLS proposes a simpler approach to modeling structural changes without need for a separate "executive" model or special transition functions to capture structural information: these information are inherent in the system's configuration and appropriate structural changes occur naturally with state transitions. The ability to specify this aspect in a graphical language also makes our approach easier to use and accessible to a larger audience.

HiLLS

HiLLS evolves from the DEVS-Driven Modeling Language (DDML) (Traoré 2009, Maïga et al. 2012, Ighoroje et al. 2012), a graphical modeling language built on DEVS to facilitate the use of the latter by domain experts via user-friendly graphical concrete syntax to describe system models. The goal of HiLLS is to be able to create multi-semantic models that can be used for simulation, formal analysis and enactment.
HiLLS' syntax combines system-theoretic and Software Engineering concepts adopted from the DEVS and Object-Z respectively. Our choices of system constructs from Object-Z and DEVS are motivated by their universalities in their respective domains; while the former claims suitability for modeling most kinds of state-based systems for formal analysis, the later has been proven to be a common denominator to most DES simulation formalisms (Vangheluwe 2000). Another advantage of the combination is that Object-Z provides constructs such as predicates and expressions that are reused for the refinement of abstract constructs such as states and transitions functions adopted from DEVS. This feature also aids the synthesis of executable program codes for enactment.
In addition to the DEVS-based system-theoretic concepts in HiLLS, the syntax also adds concepts to describe structural changes in DSSs. HiLLS' approach to modeling DSSs is unique in that it provides a simple and graphical means of doing it; we demonstrate this wit a case study in a later section.

Abstract Syntax

Figure 1 is an excerpt of the HiLLS' abstract syntax. In the bottom-right segment of the diagram (within the dashed-box), a DES is described as an *HSystem* which may be an atomic unit or composed of interacting components (*hcomponents*). It may have input and/or output ports (class *Port*) for interacting with its environment by means of exchanging messages called *Event*s. By its inheriting the class *HClassifier*, an *HSystem* may have a *StateSchema* in which state variables are declared with possible constraints, an *AxiomaticSchema* that defines global parameters, and *Operations* that manipulate the system's variables and parameters.
The behavior of an HSystem is described by a finite set of configurations and transitions between them. A *Configuration* is a cluster of all states satisfying a unique system *property* defined on the state variables. The *sojournTime* of a configuration is the duration for which it may be assumed before a scheduled transition occurs. A configuration is regarded as *Passive*, *Transient* or *Finite* if its *sojournTime* is positive infinity ($+\infty$), zero (0) or positive real number greater than zero($\mathbb{R}^+$) respectively. A *ConfigurationTransition* belongs to one of three categories: an *internal* transition occurs at the expiration of the *sojournTime* of the current configuration, an external transition occurs whenever an *input* is received before the end of the *sojournTime* while a confluent transition occurs when the reception of an input coincides with the expiration of the *sojournTime*. A transition is accompanied by a sequence of *computations* that manipulate the state variables to satisfy the *property* of the target configuration; it may also involve

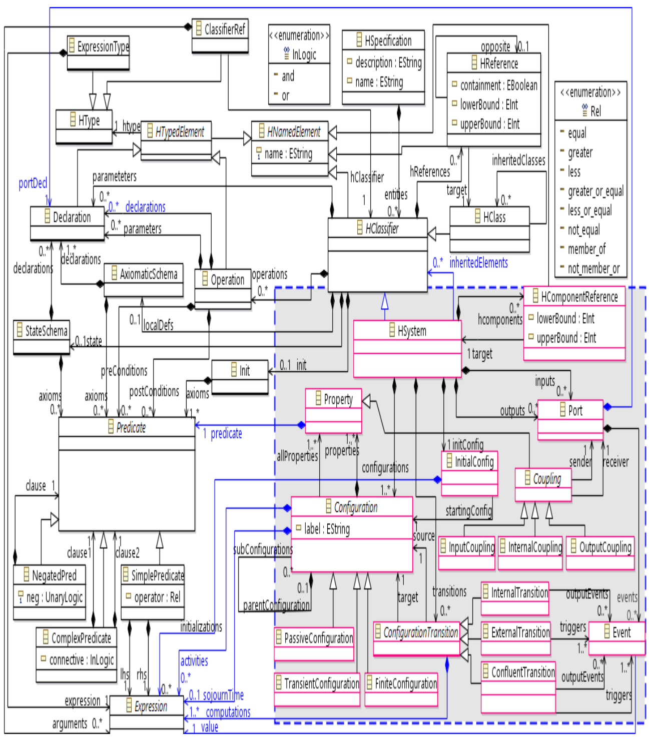
Figure 1: Abstract syntax of HiLLS

sending *events* to some output ports.

Coupling describes the relations between the ports of the components of a system. Systems influence on one another exchanging events (i.e., messages, impulse, etc.) through their input and output ports. Therefore, a coupling is a property that establishes a relation between a source port (*sender*) and a target port (*receiver*) for the exchange of events.

In DSSs (e.g. automatic switching systems), certain couplings are characteristics of some states of the system, hence coupling and decoupling of components occur during state transitions. Specifications of structural changes follow a similar fashion in HiLLS; a couplings is a kind of property that defines a configuration. Therefore, the couplings associated with the configurations of an HSystem naturally define the instantaneous relationships between its components; this is in fact similar to the real-time behavior of such systems. *InputCoupling*, *InternalCoupling* and *OutputCoupling*, have the same definitions as DEVS' *EIC*, *IC* and *EOC* respectively.

In addition to the amenability of Object-Z to formal analysis, the level of functional refinement provided by the segment of the meta-model outside the dashed-box helps to precisely and completely model systems' behavior in a generic form that can be refined to executable program code for the enactment of systems. These concepts are reused for the refinement of the system-theoretic concepts through their associations with them; examples are the associations between the following pairs of components: (*Property*, *Predicate*), (*Port*, *Declaration*) and (*Event*, *Expression*).

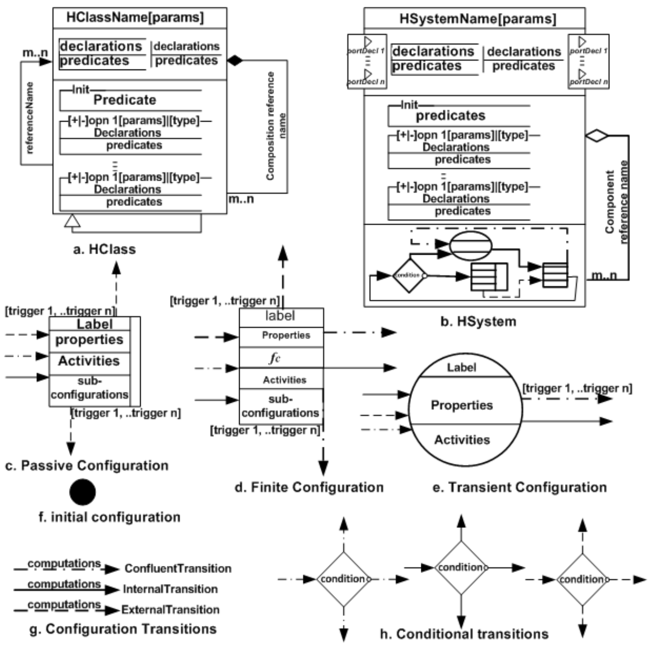

Figure 2: Concrete syntax of HiLLS

Concrete Syntax

The concrete notations to express HiLLS' concepts are described in Figure 2(a-h). *HClass* (a) is denoted by a box with three compartments similar to the UML class symbol. The first compartment contains the HClass' name and parameters if any. The second compartment houses the state and axiomatic schema if any. We adopt the notations of the state schema and axiomatic schema as used in Object-Z. The third and last compartment houses the definitions of the class' operations if any. An operation is similar to the state schema but with additional information indicated on its top side. The top bears the name attribute of the operation, the list of parameter declarations (if any) and the type of the operation. Similarly, an empty type bracket denotes an operation that does not produce any output. Associations such as inherited class, composition, and class reference use the corresponding notations in as used in the UML class diagram notations.

The *HSystem* (b) notation extends that of HClass; it has four compartments with the first three serving similar functions as in HClass while the fourth contains the transition diagram that describes the system's behavior. The input and output interfaces are denoted by windows attached to the left and right sides respectively of the second compartment. In each rectangular window, a port is denoted by a small arrowhead labelled with the port declaration.

The notation for a finite configuration (d)is a box with five compartments for label, properties, sojournTime, activities, and sub-configurations respectively from top to bottom. Passive configuration (c) is similar to a finite configuration except that the compartment for sojournTime is not represented; a vertical stripe is attached to its right side as an indication of its infinite sojournTime. Transient configuration (e) is denoted by a circle with three compartments for its label, properties and activities if any. Its shape depicts its zero sojournTime.

Configuration transitions are represented by arrow-ended lines(g) emanating from source to target configurations with associated computations as textual labels.A conditional statement in the path of a transition may initiate a choice between one of two targets. In such cases, the condition is enclosed within a diamond (h). To disambiguate the flow of the computations, the transition arrow flows into the left corner of the diamond and flows out from the circle attached to the right corner if the condition is true; otherwise, it flows out from either the top or bottom corner.

Semantic Mapping of HiLLS to DSDEVS

HiLLS has a family of semantics domains, each element providing a context for system analysis; however the focus of this paper is limited to DSDEVS as a semantics domain for the simulation of DSS. By generating an equivalent DSDEVS model from a HiLLS specification, we take benefit of the simulation protocol of the former and its associate tools. Due to space limitations, a detailed formal specification of the semantic mapping cannot be given in this paper but the description provided in this section will help the reader follow subsequent sections.

DEVS, being a mathematical formalism solely for system specification has no specific constructs for representing objects. Since the formalism also does not prescribe any concrete syntax, the user may take advantage of the freedom to represent an object as a mathematical structure with its essential attributes and operations constituting the elements of the structure. HiLLS' operations are also specified as mathematical functions that may be called from the DEVS-specific functions such as the transition and output functions.

An *HSystem* maps to *Atomic* or *Coupled DSDEVS* models if its hComponents is empty or not respectively. In the case that it maps to a Coupled DSDEVS, the set *HSystem.hComponnents.target* maps to the set of its components. HiLLS' ports map to the set of corresponding input and output ports of the equivalent DSDEVS model. If an HSystem maps to an Atomic DSDEVS, each configuration, c, translates to a subset of the state set S of the latter while the sojourn time, f_c, of c translates to the time advance function, $ta(s)$, of the states in this set .

Internal, External and Confluent transitions in HiLLS are extracted to build DSDEVS δ_{int}, δ_{ext} and δ_{conf} functions respectively. The output computations accompanying internal and confluent transitions in HiLLS are used to build the λ function in the DSDEVS model. In the case that an HSystem maps to a coupled DSDEVS, the configurations and transitions of the former map to the states and state transitions of the network executive of the latter.

CASE STUDY: THE CASH DEPOSIT MACHINE

The Cash Deposit Machine (CDM) allows a customer to deposit a bundle of currency notes into an account. It comprises six components that collaborate to process the currency bills. It first checks the genuineness of the bills by testing for some security properties, unrecognised bills are returned to the customer user while accepted bills are temporarily held in the machine to request a confirmation of the transaction from the user. If confirmed, the bills are permanently stored in the machine while the transaction runs to completion. Otherwise, the bills are returned while the transaction is being cancelled. The following are the components and their respective roles in processing the bills.

1. *Bundle Accepter (BA)* receives a bundle of bills (max. 50 per transaction),releases them into the BC one a time for validation and notifies the controller after sending the last bill. It also receives returned bills from the REJ and presents them in a bundle to the customer in the event of unrecognised bills and/or cancellation of transaction. It is guarded by a shutter that opens only when bills are taken from or returned to the user.

2. *Bill checker(BC)* receives a bill at a time from the BA to test its genuineness. Accepted and rejected bills are passed on to the ES and REJ respectively.

3. *Escrow(ES)* is a temporary storage for valid bills (max. 50) until the transaction is confirmed or cancelled. If confirmed, the bills are sent to the CAS; Otherwise, they are released into the REJ. ES has only one output slot that may be linked to either the REJ or the CAS depending on the situation.

4. *Reject box(REJ)* stacks rejected or cancelled bills and returns them in a bundle to the BA for onward delivery to the customer upon receiving the appropriate control instruction.

5. *Cassette (CAS)* is a permanent storage for deposited bills.

6. *Control Board (CON)* is a micro-electronic board that coordinates the activities by sending instruction signals to other components.

Note: This case study has a static set of components but the couplings between certain components vary dynamically. The ES has one output port that may be

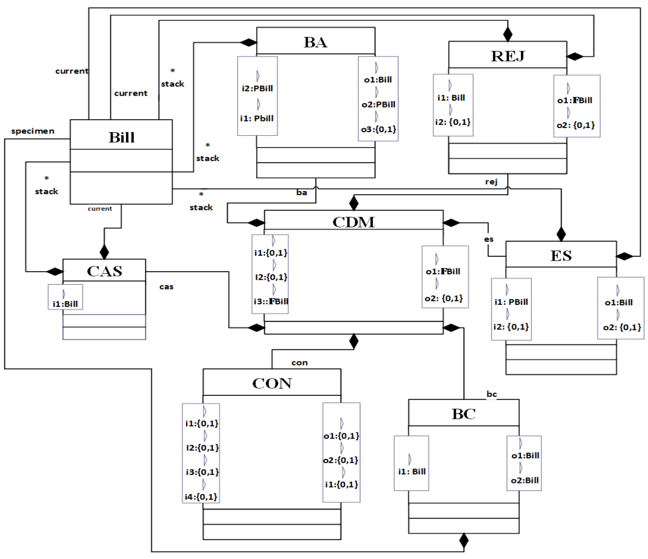

Figure 3: Hierarchical composition of the CDM model

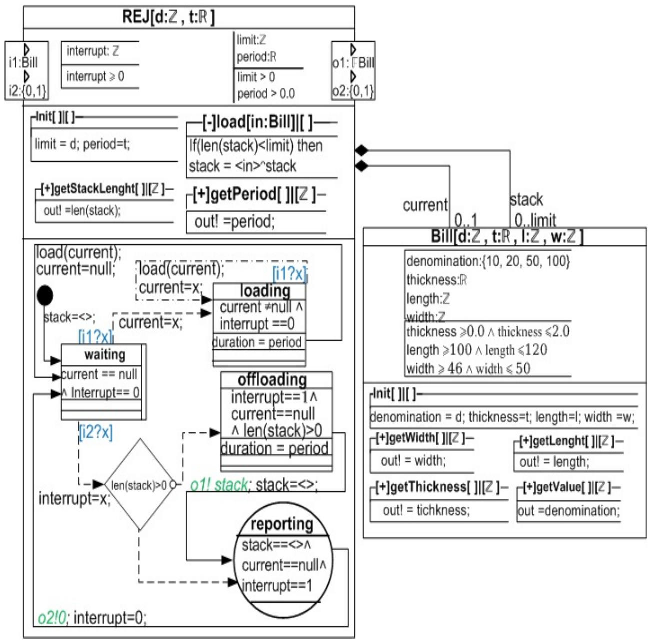

Figure 4: Reject box and Bill models

coupled to CAS or REJ depending on whether the user confirms or cancels the transaction.

HiLLS Model of the Case Study

The HiLLs model in Figure 3 shows the hierarchical compositions between the CDM and its autonomous components that interact with one another by exchanging bills and low voltage signals. Due to space limitation, we will present the detailed specification of one of the components, REJ and that of the CDM to show how dynamic couplings between some components are specified.

Bill Model
Since it has no autonomic behavior, the bank note (Bill) is modelled as an HClass as shown in Figure 4. It has four parameters d, t, l and w representing the denomination, thickness, length and width respectively of the bank note. The state schema declares the state variables and constraints that define the features of the bill.

REJ Model
The REJ model is shown in Figure 4, The input interface has two ports *i1* and *i2* for receiving bills and digital signals respectively. Similarly, the output interface has two ports *o1* and *o2* for sending bundles of bills and digital signals respectively.
State Variables and Constants: REJ declares variables *interrupt* of type positive integer,*current* of type Bill and *stack* which is a list of Bills. By default, every instance of HSystem declares an implicit variable, *duration* of type positive real number (with possibility of positive infinity) to hold the instantaneous values of the sojourn times of active configurations. The axiomatic schema in the second compartment defines two constants: *limits* and *period* representing the component's capacity and time taken to exchange bills respectively as specified in the system's synopsis. Note that limit is used as the upper limit of the cardinality of the state variable stack (see the composition relation 'stack' from REJ to Bill indicating the maximum number of bills that may be stacked in the REJ at any instant and for any transaction. The Init operation initializes the global constants with the parameters provided at instantiation.
Operations: The third compartment defines three operations; *getStackLenght*, *getPeriod* and *load*. In HiLLS, any operation with type different from void has an intrinsic variable, *out*, of the same type as the operation's type; this variable must be assigned the computed value of the output of the operation. It serves the same purpose as *return* in Java.
Configurations The fourth and last compartment contains the specification of REJ's behavior described by configurations and transitions. The state space is partitioned into four configurations *waiting*, *loading*, *offloading* and *reporting* with their respective properties indicated in their second compartments. As stated previously, the computed sojourn time for each configuration is assigned to the implicit variable duration. *loading* and *offloading*, being finite configurations, have their sojourn times explicitly defined in the third compartments while waiting and reporting have implicit sojourn times of positive infinity $+\infty$ and 0 respectively.
Configuration Transitions: With *waiting* as the initial configuration, there are seven (7) transitions as shown in the diagram. The *triggers* of external and confluent transitions are represented by the port and event names at the source end; e.g., an external transition from *wait-*

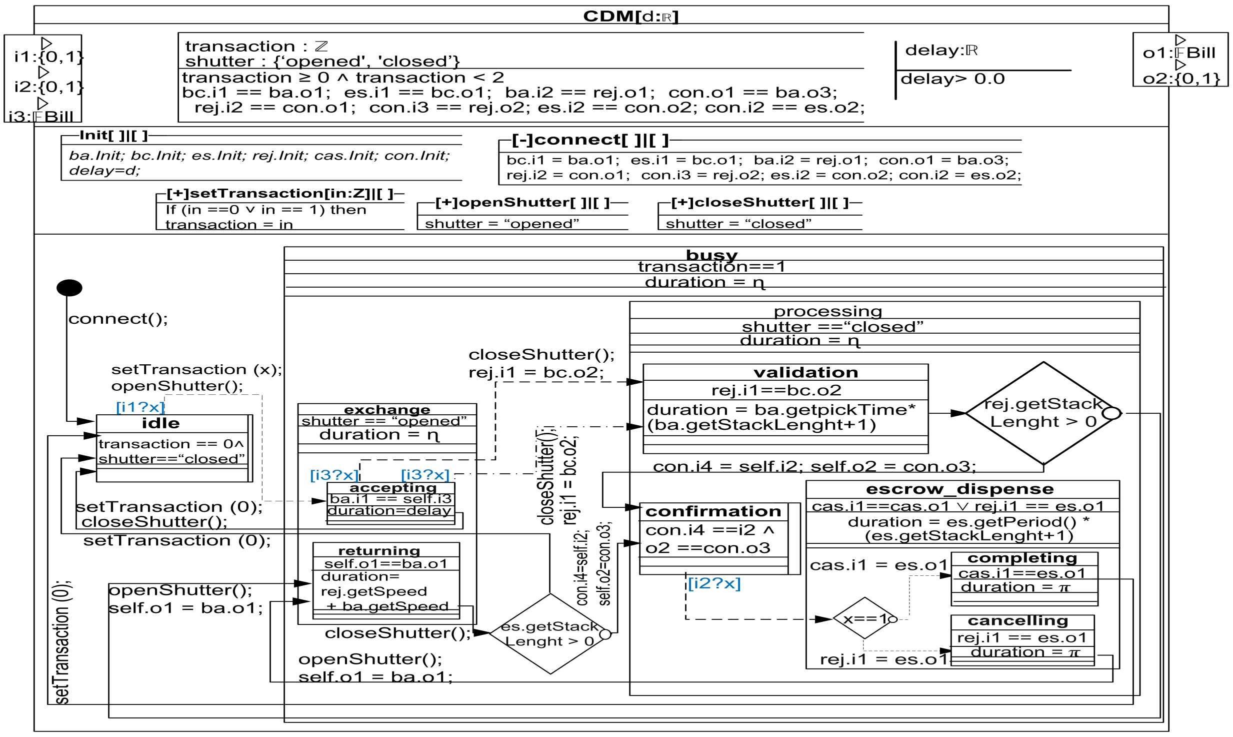
Figure 5: CDM model

ing $\longrightarrow$ *loading* is triggered by [*i1?x*](i.e., when input x is received on port i1). Output events that may accompany internal and confluent transitions are represented as assignment of values to appropriate port variables in the transition computations. for example, during the transitions *offloading* $\longrightarrow$ *reporting* and *reporting* $\longrightarrow$ *waiting*, $o1!stack$ and $o2!0$ are outputs on ports *o1* and *o2* respectively.

CDM Model

We have seen in Figure 3 that the *CDM* has six components: **ba**, **bc**, **es**, **rej**, **cas** and **con** which are instances of *BA*, *BC*, *ES*, *REJ*, *CAS* and *CON* respectively. Figure 5 shows the internal details of CDM's specification. In addition to the six components, *CDM* declares two state variables, *transaction* and *shutter* that keep tracks of the presence of an active transaction and the status of the shutter respectively. The shutter is the opening through which bills are presented to or taken from the machine. *delay* is a parameter of the CDM that describes the maximum period for which the shutter remains opened when accepting bills from the user. The second line of predicates in the state schema specifies the invariants for static couplings between certain components of the system. These invariants are realized by the initialization of corresponding port variables in the *connect* operation. For example; a coupling specification $A.port_r = B.port_s$ implies that port **s** of system **B** influences port **r** of system **A**. In addition to initializing the system's parameter,*delay*, the *Init* operation of *CDM* also invokes the *Init* of each of its components.

In addition to the explicit and implicit specifications of sojourn times for configurations demonstrated in the *REJ* specification, the *CDM* presents two special functions, η and π, to specify the sojourn times in some specific cases. η is used in cases where a configuration has some sub-configurations where its own sojourn time cannot be precisely specified. Hence, the function indicates that the sojourn time at any instant is equal to that of its active sub-configuration at that time. For example, the sojourn time of processing at any instant is equal to that of whichever is active among validation, confirmation and escrow-dispense. π on the other hand is used when all sub-configurations under the same parent have identical sojourn time; the sojourn time may be specified once on the parent configuration while all its children inherit this property from it. For instance, configurations completing and cancelling have identical sojourn times which is specified once on escrow-dispense from which all its sub-configurations inherit the property. Configuration transitions of the *CDM* can be read just the same way those of *REJ* were presented previously.

DSDEVS Equivalent Model of the CDM

We present in this sub-section, the DSDEVS model of the *CDM* derived from the HiLLS specification.

$DSDN_{CDM} = (\chi, M_\chi)$ where $\chi = CMD$ and $M_\chi = \langle X_\chi, S_\chi, s_{o,\chi}, Y_\chi, \gamma, \Sigma^*, \delta_\chi, \lambda_\chi, \tau_\chi \rangle$

$X_\chi = \{(i_1, \{0,1\}), (i_2, \{0,1\}), (i_3, \mathbb{P}Bill)\}$
$Y_\chi = \{(o_1, \mathbb{P}Bill), (o_2, \{0,1\})\}$
$S_\chi = \{(transaction, \mathbb{Z}^+), (shutter, String), (conf, CONF)\}$
where: $CONF = \{idle, accepting, returning, validating, confirmation, completing, cancelling\}$
$s_{(o,\chi)} = (transaction = 0, shutter = "closed", conf = idle)$
$\Sigma^* = \{M_{conf}\}_{conf \in CONF}$ with
$M_{conf} = \langle D_{conf}, \{M_i\}_{i \in D_{conf}}, \{I_i\}_{i \in D_{conf}}, \{Z_i\}_{i \in D_{conf}} \rangle$
Note that out of the three state variables, the system's structure depends of on the value of *conf.* i.e.,
$\Sigma^* = \{M_{idle}, M_{accepting}, M_{returning}, M_{validating}, M_{cancelling}, M_{completing}, M_{confirmation}\}$
Let $D_o = \{M.name\}_{M \in Comps}$ where $Comps = \{ba, bc, es, rej, cas, con\}$
$\forall M_{conf} \in \Sigma^*, D_{conf} = D_o \wedge \{M_i\}_{i \in D_{conf}} = Comps$
$\forall s_x \in S_\chi, \gamma(s_x) = M_{conf}$
Let $sI_{CDM} = \{\}, sI_{ba} = \{rej\}, sI_{bc} = \{ba\}, sI_{es} = \{bc, con\}, sI_{rej} = \{con\}, sI_{con} = \{ba, rej, es\}, sI_{cas} = \{\}$.

Recall that M_{conf} is the structure of the system for the actual value of the *conf* variable; due to space constraint, we present the details of $M_{completing}$ and others can be obtained in similar manner.
$M_{completing} = \langle D_{completing}, Comps_{completing}, I_{completing}, Z_{completing} \rangle$ where:
$I_{completing} = \{sI_{CDM} \cup \{ba\}, sI_{bc}, sI_{cas} \cup \{es\}, sI_{es}, sI_{ab} \cup \{CDM\}, sI_{rej} \cup \{bc\}, sI_{con} \cup \{CDM\}\}$
$Z_{completing,CDM} : Y_{ba} \rightarrow Y_{CDM}$; $Z_{completing,ba} : X_{CDM} \times Y_{rej} \rightarrow X_{ba}$; $Z_{completing,bc} : Y_{ba} \rightarrow X_{bc}$; $Z_{completing,es} : Y_{bc} \rightarrow X_{es}$; $Z_{completing,rej} : Y_{bc} \times Y_{con} \rightarrow X_{rej}$; $Z_{completing,cas} : Y_{es} \rightarrow X_{cas}$; $Z_{completing,con} : X_{CDM} \times Y_{rej} \times Y_{ba} \times Y_{es} \rightarrow X_{con}$

Transition function: $\delta_\chi : Q_\chi \times (X_\chi \cup \phi) \longrightarrow S_\chi$ with $\delta_\chi((0, closed, idle), e, i) = (1, opened, accepting), \forall e > 0$.

This transition corresponds to the external transition between the *idle* configuration and the *accepting* configurations when an event is received on port i_1. This transition creates also a new input coupling between the input port i_3 of CDM and the input port i_1 of its component *ba*. The coupling creation is part of the changes specified in the set of different possible structures. $\delta_\chi((1, opened, accepting), e, i) = (1, closed, validation), \forall i \in \{0,1\}, e \geq 0$. This transition takes into account the two transitions (the external one and the confluent one) from the accepting configuration to validation configuration. This transition specifies a new internal coupling between the output port o_2 of the *bc* and the input port i_1 of the reject box *rej*.
The external transition from *confirmation* when an input is received on port i_2 is described as a piecewise function: $\delta_\chi((1, closed, confirmation), e, i) =$
$$\begin{cases} (1, closed, completing) & if\ i = 1; \\ (1, closed, cancelling) & if\ i = 0; \end{cases}$$
The target of the transition is *completing* or *cancelling* if the input received on port i_2 is 1 or 0 respectively. The transition is accompanied by the establishment of couplings between the port o_2 of *es* (i.e.,$es.o_2$) and $cas.i_1$ or $rej.i_1$ if the value of the received input is 1 or 0 respectively.
The internal transition from the completing configuration to the idle configuration is represented as: $\delta_\chi((1, closed, completing), 0, \phi) = (0, closed, idle)$. Other transitions can be mapped in similar manner.

Time advance function: $\tau_\chi : S_\chi \longrightarrow \mathbb{R}^+ \cup \{+\infty\}$
$\forall s_x \in S_\chi, \tau_\chi(s_x) = duration(conf)$ i.e.,
$$\tau_\chi(s_x) = \begin{cases} +\infty & if\ conf \in \{idle, confirmation\} \\ t_{ba} & if\ conf = validating \\ t_{es} & if\ conf \in \{cancelling, completing\} \\ t_{rej} & if\ conf = returning \\ delay & if\ conf = accepting \end{cases}$$
$t_{ba} = ba.getPeriod() * (ba.getStackLength() + 1)$
$t_{es} = es.getPeriod() * (es.getStackLength() + 1)$
$t_{rej} = rej.getSpeed() + ba.getSpeed()$
delay is a parameter of the CDM.

The output function: $\lambda_\chi : S_\chi \rightarrow Y_\chi$
$\forall tr \in \{0,1\}, shut \in \{opened, closed\}$:
$\lambda_\chi(tr, shut, returning) = 0$
$\lambda_\chi(tr, shut, completing) = 1$
$\lambda_\chi(tr, shut, cancelling) = rej.getStack()$

CONCLUSIONS AND PERSPECTIVES

We have presented a new approach to modeling DSSs with HiLLS. The advantage of the proposed approach is the possibility of simple and communicable graphical notations to describe structural changes in complex systems which are usually expressed either as mathematical equations or program codes by existing solutions.

HiLLS models are amenable to analyses by simulation, formal methods and enactment but our discussions in this paper were limited to the provision of simulation semantics for DSSs using the DSDEVS as the semantics domain. This way, we take benefits of the ease and simplicity of HiLLS for specification and the simulation protocols for DSSs defined by DSDEVS and possibly, its associated tool called DELTA Environment (Barros and Mendes 1997).

We presented a case study to illustrate how HiLLS' concepts and notations are used to achieve the graphical modeling of DSSs. It is however important to state here that our approach is limited to what is possible with DSDEVS as far as structural changes is concerned; for example, structural changes in the input and output interfaces are not supported. We are currently working on a HiLLS editor that will allow for the specification of complex DESs and subsequent generation of simulation codes based on existing tools.

APPENDIX

Discrete Events System Specification (DEVS)

DEVS (Zeigler et al. 2000) is a system-theoretic mathematical formalism for specifying DESs as abstract mathematical objects for simulation. It supports the specification of a full range of DESs as other formalisms for systems in this category have been proven to have equivalent DEVS representations (Vangheluwe 2000).

Basically, DEVS defines two abstraction levels for DESs - *atomic* and *coupled* DEVS. An atomic DEVS has a time base; state, input and output sets; and functions that define successive states and outputs events. A coupled DEVS on the other hand is an hierarchical composition of two or more atomic and/or coupled DEVS as components with couplings between their input/output ports to enable their interactions.

Atomic DEVS, AM, is defined as:
$AM = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta \rangle$ where:
$X = \{(p, v) | p \in IPorts \wedge v \in dom(p)\}$,
$Y = \{(q, v) | q \in OPorts \wedge v \in dom(q)\}$,
S: *Abstract state set* , $\delta_{int} \colon S \longrightarrow S$,
$\delta_{ext} \colon Q \times X^b \longrightarrow S$ with $Q = \{(s, e) | s \in S \wedge 0 \leq e < ta(s)\}$,
$\delta_{conf} \colon S \times X^b \longrightarrow S$, $\lambda : S \longrightarrow Y^b$ and $ta \colon S \longrightarrow \mathbb{R}^+_{0,\infty}$
X and Y are the sets of input and output *events* respectively with $IPorts$ as set of input ports and $OPorts$ as set of output ports. An event, v, in this context is a value generated in form of a message that triggers an action by its recipient. S is the set of states; at any given moment, the system is in a state $s \in S$. The time advance function, ta, maps each state to a specified duration after which a scheduled internal state transition, δ_{int}, is automatically fired. The external transition function, δ_{ext}, specifies the system's response to the input event(s) before the expiration of the ta of current state; Q is called the set of total states and e is the elapsed time since the last state transition. If the input event coincides with the expiration of the ta), then confluent transition function, δ_{conf} is invoked instead. The subscript b of X_b denotes a bag of input events. The function λ defines the outputs that may accompany internal and/or confluent state transitions. Similarly, the subscript b of Y_b denotes a bag of output events.

The Coupled DEVS, CM is defined as:
$CM = \langle X, Y, D, \{M_d\}_{d \in D}, EIC, EOC, IC \rangle$ where:
$EIC = \{((CM, ip_{CM}), (d, ip_d)) | ip_{CM} \in IPorts_{CM} \wedge ip_d \in IPorts_d\}_{d \in D}$,
$EOC = \{((d, op_d), (CM, op_{CM})) | op_{CM} \in OPorts_{CM} \wedge op_d \in OPorts_d\}_{d \in D}$,
$IC = \{((a, op_a), (b, ip_b)) | op_a \in OPorts_a \wedge ip_b \in IPorts_b\}$ with $a, b \in D$
X and Y are as defined for atomic DEVS and D is the set of names of components of CM such that M_d is the DEVS specification referred to by d for all $d \in D$. An EIC, External Input Coupling, is a connection between an input port of CM and an input port of one of its components, an EOC, External Output Coupling, is a connection between an output port of CM and an output port of one of its components and an IC, Internal Coupling, is a connection between the output port of a component of CM and an input port of another peer component. More details on DEVS and its simulation protocols can be found in Zeigler et al. (2000).

REFERENCES

Barros F., 1995. *Dynamic Structure Discret Event System specifications: A new formalism for dynamic structure modeling and simulation.* In *27th WSC*. IEEE, 781–785.

Barros F.J. and Mendes M.T., 1997. *Forest fire modelling and simulation in the DELTA environment. Simulation Practice and Theory*, 5, no. 3, 185–197.

Hu X.; Zeigler B.P.; and Mittal S., 2005. *Variable structure in DEVS component-based modeling and simulation. Simulation*, 81, no. 2, 91–102.

Ighoroje U.B.; Maïga O.; and Traoré M.K., 2012. *The DEVS-driven modeling language: syntax and semantics definition by meta-modeling and graph transformation.* In *Symposium on TMS*. SCS International, 49.

Maïga O.; Ighoroje U.B.; and Traoré M.K., 2012. *DDML: A support for communication in Modeling and Simulation.* In *3rd IEEE Track on Collaborative Modeling and Simulation- CoMetS12*. IEEE, 238–243.

Muzy A. and Zeigler B.P., 2014. *Specification of dynamic structure discrete event systems using single point encapsulated control functions. IJMSSC*, 5, no. 03, 1450012.

Smith G., 2000. *The Object-Z specification language. Advances in Formal Methods.*

Traoré M.K., 2009. *A graphical notation for DEVS.* In *SpringSim Multiconference*. SCS International, 162.

Uhrmacher A.M., 2001. *Dynamic structures in modeling and simulation: a reflective approach. TOMACS*, 11, no. 2, 206–232.

Uhrmacher A.M.; Himmelspach J.; Rohl M.; and Ewald R., 2006. *Introducing variable ports and multi-couplings for cell biological modeling in DEVS.* In *WSC 06*. IEEE, 832–840.

Vangheluwe H., 2000. *DEVS as a Common Denominator for Multi-Formalism Hybrid Systems Modeling.* In *International Symposium on Computer-Aided Control System Design*. IEEE, 129–134.

Zeigler B.P.; Praehofer H.; and Kim T.G., 2000. *Theory of Modeling and Simulation, 2nd Edition: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press. ISBN 0-12-778455-1.