TRANSFORMING STATECHARTS TO DEVS

Spencer Borland and Hans Vangheluwe
Modelling, Simulation and Design Lab
School of Computer Science
McGill University
http://msdl.cs.mcgill.ca

Keywords: Statecharts, DEVS.

Abstract

This document discusses the transformation of a model in the Statechart formalism to a model in the DEVS formalism. Only the core elements of Statecharts are analyzed. In each section, an essential element of the Statechart formalism is described as well as how to convert it to its DEVS equivalent. Finally, a general procedure for converting from the Statechart formalism to the DEVS formalism is given.

1 INTRODUCTION

The DEVS formalism [1], created by Zeigler, is a modular, discrete-event formalism. It rigourously defines model structure as well as operational semantics. The Statechart formalism [2] was developed by Harel for modelling reactive systems such as embedded electronics. It uses Higraphs to extend state machines with hierarchy, orthogonality and broadcast communication. Today, there exist several different versions of Statechart semantics [3]. By transforming Statecharts into DEVS, insight is gained into this semantics. Previous efforts to define an equivalence between Statecharts and DEVS [4] illustrate a conversion by means of examples. Here, a more explicit transformation procedure is given.

2 NOTATION

The following is a summary of some of the notations used in this document. First, some of the symbols used are described followed by a description of the notations used in the DEVS figures.

- ε: Denotes a DEVS state where $ta(\varepsilon) = 0$.
- ω: Denotes a DEVS state where $ta(\omega) = \infty$.
- $\langle\varphi\rangle$: Indicates a location which can hold any state in the state space.
- $\leftarrow$: Boolean operator used to indicate the replacement of the left-hand set with the right-hand set.

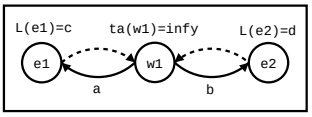

Figure 1: A DEVS diagram.

Figure 1 depicts an atomic DEVS model. The dashed lines denote internal transitions while the solid lines denote external transitions. State names starting with 'e' usually represent ε states and those starting with 'w' represent an ω state. The label $ta(w1) = infy$ means the time-advance of the ω_1 state is ∞. $L(e2) = h$ means $\lambda(\varepsilon_2) = h$. The external transition of ω_1 is as follows.

$$\delta_e((\omega_1, e), \langle\varphi\rangle) \quad = \quad \begin{cases} \varepsilon_1 & \text{if } \langle\varphi\rangle = a \\ \varepsilon_2 & \text{if } \langle\varphi\rangle = b \end{cases}$$

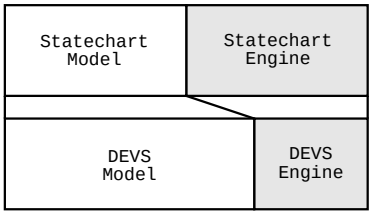

Figure 2: Transforming Statecharts to DEVS.

3 MOTIVATION

The Statechart formalism makes use of very high level constructs not seen in many formalisms developed to this day. DEVS, on the other hand, has few high level syntactic elements. Thus, it is natural to transform the high-level to the low-level and try to learn about the semantics of the high-level formalism which is hidden in

its execution/simulation engine. In effect, part of the semantics of Statecharts is removed from the engine and is modeled using DEVS (see figure 2. Furthermore, not only Statecharts, but also many other formalisms can be mapped onto DEVS [5]. Thanks to the closure under coupling of the DEVS formalism, this allows for multi-formalism modelling.

4 0-TIME OUTPUTS

Outputs may only be generated after an internal transition, in DEVS. Thus, a Statechart transition must correspond to a DEVS model with an extra intermediate state, ε, with $ta(\varepsilon) = 0$. This can be referred to as an ε-*insertion*. Moreover, $after(x)$ on a transition originating from A, corresponds to a DEVS transition originating from A such that, $ta(A) = x$.

5 GLOBAL SCOPE 0-TIME EVENT RELAY

Events generated in a Statechart model have global scope. DEVS, however, is modular and all event communication occurs through interfaces. One simple and naive approach to global events in DEVS is to simply output all external messages that are intercepted by each coupled model. This is done by maintaining a state which remembers which external event was generated and then outputs this event on the next internal transition. This can be represented by the following atomic DEVS. Note that the input and output sets, X and Y respectively, are the set of all events.

$$\begin{aligned} S &= \{\omega, (\varepsilon_{\langle\varphi\rangle})\} \\ \delta_e((\omega, e), \langle\varphi\rangle) &= \varepsilon_{\langle\varphi\rangle} \\ \delta_i(\varepsilon_{\langle\varphi\rangle}) &= \langle\varphi\rangle \\ \lambda(\varepsilon_{\langle\varphi\rangle}) &= \langle\varphi\rangle \\ ta(\omega) &= \infty \\ ta(\varepsilon_{\langle\varphi\rangle}) &= 0 \end{aligned}$$

The parent coupled DEVS must ensure that all events that are output from each of its atomic DEVS are routed to the event relay and returned, in 0-time, to all subcomponents. This is done by properly specifying the influencee sets and the output-to-input transfer functions. If $r \in D$ is the event relay component in the coupled model, $\{X_{self}, Y_{self}, D, M_i, I_i, Z_{i,j}, select\}$, then the Z and I functions would be as follows.

$$\begin{aligned} I_i &= r\ , \forall i \in (D \cup \{self\}) \setminus \{r\} \\ I_r &= (D \cup \{self\}) \setminus \{r\} \\ Z_{self,r}(\langle\varphi\rangle) &= \langle\varphi\rangle \\ Z_{r,self}(\langle\varphi\rangle) &= \langle\varphi\rangle \\ Z_{i,r}(\langle\varphi\rangle) &= \langle\varphi\rangle\ , \forall i \in (D \setminus \{r\}) \\ Z_{r,i}(\langle\varphi\rangle) &= \langle\varphi\rangle\ , \forall i \in (D \setminus \{r\}) \end{aligned}$$

Note how an optimized event relay would only send messages if a particular component can actually respond to external events.

6 BOOLEAN EXPRESSIONS & CURRENT STATE REFERENCES

Statechart transitions can have guards which are boolean expressions. Boolean expressions are easily encoded in a DEVS framework. *AND* expressions correspond to transitions linked consecutively while *OR* expressions are converted to several transitions emanating from one state.

A state reference, is a special boolean expression. There are three ways of modelling state reference in DEVS. The first and most naive method is to completely flatten the entire model, including orthogonal components, so that each state becomes a long tuple. This method is not desirable as the number of states grows exponentially with the number of orthogonal components. It is important to come up with a solution which exploits the DEVS hierarchy.

6.1 The Pull Method

The pull method utilizes a query event which is narrowcast to the parent of the state in question. This component's current state must then have an external transition which reacts to the query event. The query event will trigger the component to switch the current state to a 0-time-advance state, ε. The internal transition of ε will return to the previous state and will also output the value of the current state encoded in the output event. This mechanism is called a *Current State Response Loop*.

Below are the abbreviated definitions for the component doing the *referencing* in which there is a transition from $A \in S$ to $B \in S$ which has trigger $\mathtt{a}[X_1.X_2...X_n]$.

$$\begin{aligned} S &\leftarrow S \cup \{\varepsilon, \omega\} \\ \delta_i(\varepsilon) &= \omega \\ \delta_e((A, e), a) &= \varepsilon \\ \lambda(\varepsilon) &= CS(X_1, ..., X_{n-1}) \\ \delta_e((\omega, e), \langle\varphi\rangle) &= \begin{cases} B & \text{if } \langle\varphi\rangle = CSR(X_1, ..., \langle X_n\rangle) \\ A & \text{otherwise} \end{cases} \\ X &\leftarrow X \cup \{CSR(X_1, ..., \langle X_n\rangle)\} \\ Y &\leftarrow Y \cup \{CS(X_1, ..., X_{n-1})\} \end{aligned}$$

Below are the definitions for the component being *referenced*. A state, $\langle X_n \rangle \in S$, which lies in orthogonal region $X_1.X_2...X_{n-1}$, which can be queried, should have the following definitions included in the atomic DEVS in which it is defined to build its *Current State Response Loop*. Note that, $\langle X_n \rangle$ represents the current state in component X_{n-1} and each of $X_1, \ldots, X_{n-1}$ are the names of components.

$$
\begin{aligned}
S &\leftarrow S \cup \{\varepsilon\} \\
\delta_e((\langle X_n \rangle, e), CS(X_1, ..., X_{n-1})) &= \varepsilon \\
\delta_i(\varepsilon) &= \langle X_n \rangle \\
ta(\varepsilon) &= 0 \\
\lambda(\varepsilon) &= CSR(X_1, ..., \langle X_n \rangle) \\
X &\leftarrow X \cup \{CS(X_1, ..., X_{n-1})\} \\
Y &\leftarrow Y \cup \{CSR(X_1, ..., \langle X_n \rangle)\}
\end{aligned}
$$

6.2 The Push Method

A Statechart model that includes a transition with a state reference, may wish to store one boolean switch in an atomic DEVS for each different state reference. In this case, the components with the states being referenced must send notifications every time these states move in and out of the current state tree.

First, a boolean switch is installed into the coupled DEVS to track if state X_n is current or not. This switch resides in its own atomic DEVS. Wherever state X_n resides, it must have the same machinery installed to indicate to other components whether or not it is a current state. This can be done by building extra states and internal transitions for each incoming and outgoing transition. Consider a transition, t, whose destination is state X. Force t to point at a new, extra state, ε. Then create an internal transition from ε to X. Finally, $\lambda(\varepsilon)$ should be a value which indicates to other components that X is now a current state. Transitions leaving X have a similar, converse structure.

6.3 Summary of State Reference

Note the distribution of messages relayed for the two different methods of the state reference implementation. The Pull Method pumps a message through its event relay and then receives a message. The Push Method can receive several messages, but only needs to communicate locally to determine the boolean value of a state reference. If the cost of passing messages is not the same everywhere, it may be more advantageous to use the push over the pull method, or vice versa.

7 FLATTENING

Flattening a Statechart is the process of transforming a hierarchical Statechart into a new Statechart which has a depth of 1. A simple and effective way of transforming Statechart features such as an interlevel transition is to flatten the surrounding contours. Orthogonal regions should never be flattened when transforming to DEVS as the number of states in the resultant model grows exponentially.

Two issues need to be resolved when flattening: nondeterminism and state references made to non-basic states.

7.1 Flattening with Non-Determinism

Different types of non-deterministic transitions are allowed thanks to priority schemes used in different semantics [3]. Irrespective of the scheme, one transition is completely redundant because the priority scheme will never allow that transition to fire. Thus, this redundant transition can simply be eliminated in the flattened DEVS. Moreover, by giving each transition a numbered priority, it becomes easy to represent different semantics by simply changing these values.

7.2 References to Non-Basic States

Consider figure 3. If D is flattened, the D contour, referred to by $[in(U.D)]$ will cease to exist. There are two different ways of correcting this problem. One solution is a preprocessing step while the other is a solution modeled in DEVS.

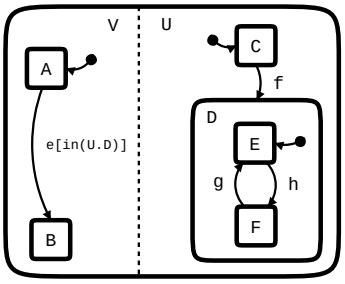

Figure 3: $U.D$ is referenced.

Method 1: Guard Transformation

The first corrective measure is to change the actual boolean expression in the guard itself. Simply, *OR* together other *in* statements with the guard's boolean expression for each basic state within the contour. For example, the guard on the transition from A to B would become, $[in(U.D.E) \vee in(U.D.F)]$.

Method 2: Current Response Loops
For every contour around a basic state, simply add a current state response loop to that basic state in the transformed DEVS model. For example, the basic states, E and F, within D in figure 3 should have extra current state response loops for the state D.

Method 3: State Reference Switches
This solution also makes use of the Push Method used to model state references. In this scheme, state reference switches are maintained for each state reference, just as discussed in section 6.1. However, a message should be sent to the state reference switch indicating the referenced state is current when *any* state within the referenced state is current. Conversely, a message should be sent to the state reference switch indicating the referenced state is not current when no state within the referenced contour is current.

8 FLATTENING ALGORITHM

Note that flattening is a preprocessing step. This means that flattening can be viewed as a function, F, defined on the set of all Statecharts, S, as $F : S \rightarrow S$. The flattening algorithm is not included here for brevity.

9 ORTHOGONALITY

It is desirable to maintain orthogonal regions in separate atomic DEVS components. The first reason is that it is a natural and intuitive transformation. Atomic DEVS which lie at the same hierarchical depth represent a cross product state set just as orthogonal regions in a Statechart contour.

Another reason for keeping orthogonal regions in separate atomic DEVS components is that the dynamics within each orthogonal region can be simulated or executed in parallel. The simulation/execution engine for DEVS is well-defined and there already exists several DEVS software packages[1].

A problem arises from this arrangement, however. This problem can best be illustrated with an example. Notice that in Figure 4 there exist two contours with orthogonal regions, C and D. Thus, it is natural to put the orthogonal regions into two separate atomic DEVS. These are denoted by E, F and K, L.

Now the question remains, where do the remaining states A and B go? They must reside within an atomic DEVS, since only atomic models have a state set. No matter where one puts the atomic DEVS containing the dynamics for $R \setminus (C \cup D)$, it will appear as though they

[1] http://www.sce.carleton.ca/faculty/wainer/standard/

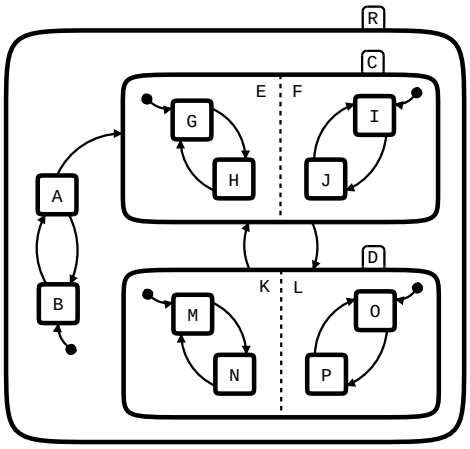

Figure 4: A Statechart exhibiting orthogonality.

are orthogonal to C and D. However, A, B, C and D are not orthogonal to each other as can be seen in the Statechart model.

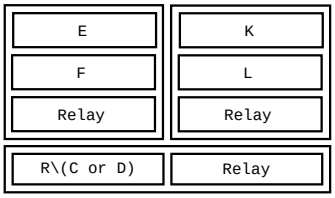

Figure 5: A topological view of the altered DEVS model.

After flattening, the resultant topology of the DEVS model can be seen in figure 5. Since only one OR-state can be current within a Statechart contour, something must be done to indicate that one of A, B, C or D is current.

In general, consider a Statechart contour, R, with states $X_1, \ldots, X_n, O_1, \ldots, O_m$ where the X states do not consist of orthogonal regions and the O states do consist of orthogonal regions. Moreover, the ith O state consists of $O_i^1, \ldots, O_i^{k_i}$ orthogonal regions. After flattening, the following regions are obtained which are in a one-to-one correspondence with the atomic DEVS which will appear in the resulting DEVS model. Let $\Psi = \{O_1^1 \cup \ldots \cup O_1^{k_1} \cup \ldots \cup O_m^1 \cup \ldots \cup O_m^{k_m}\}$. There will be one atomic DEVS for each item in set Ψ, and one atomic DEVS for $R \setminus \Psi$. Each of these atomic DEVS is given an extra state, π, to denote whether or not it is out-of-scope. The practical use of this approach is

explained in more detail in the following two sections about hyperedges and history.

10 HYPEREDGES

Each hyperedge segment can have a different trigger and guard. This means these events must occur simultaneously (see section 13). All of these triggers and conditions must be monitored so that it is known when the transition fires.

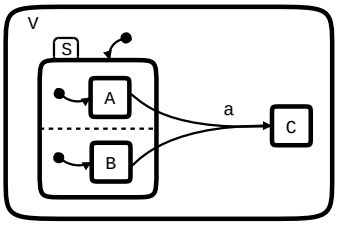

Figure 6: A Statechart exhibiting a hyperedge.

Consider the Statechart in figure 6. The hyperedge transition moves from state (A,B) to C. The mechanism which monitors the occurrence of the hyperedge's trigger could appear at any scope level. A natural location is at the same scope level as the destination of the hyperedge transition. This mechanism will track the occurrence of the event, a. Once a has occurred, events ($OUTSCOPE(S)$ and ($INSCOPE(V)$) will be generated signaling S to move out of scope and C to move into scope. $INSCOPE(V)$ is a short hand notation to denote that $V \setminus S$ is in scope. This mechanism is quite simple. It waits in an ω state then moves to an ε state where it outputs scoping events for the current coupled model.

11 HISTORY

The general topology of a Statechart model is quite different from the topology of its equivalent DEVS model. In the DEVS framework that has been built so far, two situations arise with respect to history. A history state may be present within an orthogonal region or within a composite state which is to be flattened.

12 NON-ORTHOGONAL HISTORY

Each History entity in a Statechart model must correspond to an atomic DEVS memory unit. The memory is stored in an atomic DEVS for each History node since flattening non-orthogonal states destroys the boundaries which History entities use to determine the last known current state configuration.

If a separate atomic DEVS monitors the history of a certain contour, all that needs to be done is to communicate changes in the history as well as retrieving the history value itself. Mechanisms which perform similar tasks have already been discussed in section 6.

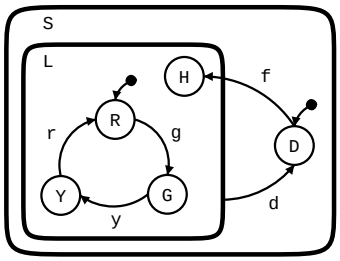

Figure 7: A Statechart exhibiting history.

A Traffic Light example is present in figure 7. The only time the memory mechanism must be notified of a history change is when the L contour is departed. In a flattened model, this corresponds to leaving R, Y or G to a state outside L, namely D. These transitions must have ε-insertions in order to facilitate an output notification. Moreover, the transition from D to the history node should have an ε,ω-insertion. This ensures a history value query is issued to the memory mechanism and the response indicates the next current state.

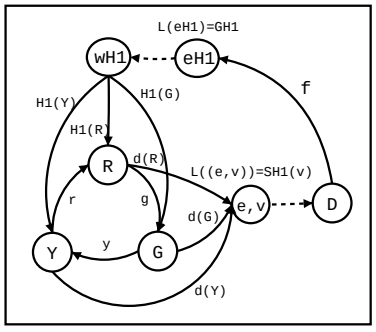

Figure 8: A DEVS model capable of history.

The corresponding DEVS model can be viewed in figure 8. Notice that each path from R, Y and G to D is interrupted by a 0-time state, $(\varepsilon,\langle\varphi\rangle)$, which outputs a $SH_i(\langle\varphi\rangle)$ (Set History) event. Furthermore, the path from D back to the history node corresponds to an ε-ω sequence. The 0-time ε state sends a GH_i (Get History) request and the ω state waits for the reply. Below is the formal definition of the δ_e for the ω_{H_1} state.

$$\delta_e((\varepsilon_{H_1},e),\langle\varphi\rangle) = \begin{cases} G & \text{if } \langle\varphi\rangle = H_1(G) \\ Y & \text{if } \langle\varphi\rangle = H_1(Y) \\ R & \text{otherwise} \end{cases}$$

Notice that if $\langle\varphi\rangle$ is equal to neither $H_1(G)$ nor $H_1(Y)$, the current state will become R. This means that R becomes the current state even if the L contour does not have any history information. This is precisely the Statechart semantics described in [6].

12.1 Orthogonal History

If the history is present in a top level orthogonal component, the out-of-scope state, π, will track the history for that area. Upon re-entering an orthogonal domain, an *INSCOPE* event can be parameterized to indicated that the system should move to a history state.

13 SIMULTANEOUS EVENTS

In a Statechart model several events may be generated before the maximal set of transitions have fired. These events are considered to have been generated in 0 simulation time or in parallel. Statechart transitions can react to multiple simultaneous events. This syntactic element must be constructed in corresponding DEVS models.

Events have no duration in a DEVS model and must therefore be remembered. However, to catch simultaneous events it is important to reset the memory mechanism after any elapsed time. This is natural in an external transition since one of its parameters is the elapsed time since the last external transition.

To monitor the simultaneous event chain $a_1 \wedge a_2 \wedge \ldots \wedge a_n$ as a trigger on a transition from A to B, the following must be encoded into the atomic DEVS $\{S, ta, \delta_i, X, \delta_e, Y, \lambda\}$.

$$S \leftarrow S \cup \{\Omega\}$$

$$\delta_e((A,e),a_i) = \Omega(\{i\}) \text{ if } 1 <= i <= n$$

$$\delta_e((\Omega(X),e),a_i) = \begin{cases} \Omega(\{X \cup \{i\}\}) & |X \cup \{i\}| < n \wedge \\ & 1 <= i <= n \\ B & |X \cup \{i\}| = n \wedge \\ & 1 <= i <= n \\ A & e > 0 \\ \Omega(X) & \text{otherwise} \end{cases}$$

This model, in effect, simulates an event queue. When all the required events have been accounted for, it is safe to proceed to state B.

14 TRANSFORMATION ALGORITHM

The algorithm presented here is a very loose description and needs to defined in more detail. Basically, each section described above represents a step in the algorithm, but the order is very important.

1. Flatten
2. Transform boolean expressions
3. Transform simultaneous events
4. Add hyperedge trackers
5. Transform history states
6. Add event relays (optimization is optional)

15 CONCLUSION

A general approach to mapping Statechart models onto DEVS was presented. More work is needed to address the details of the Statechart formalism as well as UML Statecharts.

References

[1] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, second edition, 2000.

[2] David Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the formal semantics of statecharts. *Symposium on Logic in Computer Science*, pages 54 – 64, June 1987.

[3] M. Von Der Beek. A comparison of statechart variants. *Springer-Verlag: Lecture Notes in Computer Science*, 863:128 – 148, September 1994.

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

[5] Jean-Sébastien Bolduc and Hans Vangheluwe. Expressing ODE models as DEVS: Quantization approaches. In *Proceedings of the AIS'2002 Conference*, pages 163 – 169. SCS, April 2002. Lisboa, Portugal.

[6] David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293 – 333, October 1996.

[7] H. Praehofer and D. Pree. Visual modelling of devs-based multiformalism systems based on higraphs. In *Proceedings of the 1993 Winter Simulation Conference*, pages 595–603. SCS, 1993. Los Angeles, CA.