Modeling State Transitions with Colored Petri Nets*

Rowland Pitts

George Mason University, Fairfax Virginia, USA
rpitts@gmu.edu

Abstract. Modeling state-dependent systems can quickly become unwieldy as the number of states and transitions increase. This paper introduces a reusable approach to modeling state transitions that results in simpler models and allows designers to focus on a design's functional behavior rather than the details of state transition.

Keywords: State Transition · Executable Modeling · Colored Petri Net.

Introduction A significant challenge associated with modeling state transitions, in both UML and CPN, is that the complexity of the models tends to grow exponentially [1]. Efforts to model state transition behavior typically result in one-to-one conversions of UML elements into arcs, transitions and places [4, 1], resulting in complex and unweildy models. This paper introduces an approach to modeling state transitions that results in simpler models and allows designers to focus on functional behavior rather than the details of state transition. A reusable component is modeled in CPN Tools [2].

Related Research André, Benmoussa and Choppy proposed a formalization of UML state machines using Colored Petri Nets [1]. Meghzili, Chaoui, Strecker and Kerkouche presented an approach to transforming State Machine Diagrams into CPN models, and proved certain structural properties in the transformation [3]. Pettit and Gomaa described an approach to systematically map state dependent objects and their corresponding state charts from UML into CPN [4].

This work differs from others in that state transition behavior is abstracted into a reusable CPN component that defines state transitions and their associated actions in metadata, requiring no complex network definition.

Modeling State Transitions UML is the *de facto* standard for modeling systems [1], but it is not executable. Petri nets *are* executable, however models of state transitions become increasingly complex because they are typically modeled using an explicit one-to-one correspondence with the State Transition Diagram.

CPN's hierarchical capabilities facilitate the building of models benefiting from abstraction and separation of concerns. The functional behavior associated

* Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

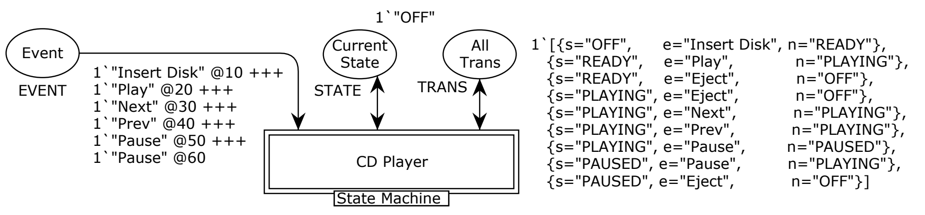
1`"OFF"
Event
EVENT
Current State
STATE
All Trans
TRANS
1`"Insert Disk" @10 +++
1`"Play" @20 +++
1`"Next" @30 +++
1`"Prev" @40 +++
1`"Pause" @50 +++
1`"Pause" @60
CD Player
State Machine
1`[{s="OFF", e="Insert Disk", n="READY"},
{s="READY", e="Play", n="PLAYING"},
{s="READY", e="Eject", n="OFF"},
{s="PLAYING", e="Eject", n="OFF"},
{s="PLAYING", e="Next", n="PLAYING"},
{s="PLAYING", e="Prev", n="PLAYING"},
{s="PLAYING", e="Pause", n="PAUSED"},
{s="PAUSED", e="Pause", n="PLAYING"},
{s="PAUSED", e="Eject", n="OFF"}]

Fig. 1. Programmable State Machine for a CD player. Initial markings of the All Trans place define the various state transitions (s equals the current state, e is the triggering event, and n is the next state). The markings on the Event place define a test sequence.

with processing events, issuing actions, and changing state can therefore be encapsulated within a reusable CPN, allowing the supported event-driven state changes to be defined in dynamically programmable meta data. In other words, rather than define a new net for each state driven scenario, each new scenario can simply be programmed into the reusable state machine. Figure 1 depicts the external view of the reusable state machine, with events and transitions defined to model a CD player.

Conclusions and Future Work The approach simplifies modeling in a number of ways. For example, the reusable State Machine eliminates the the need to explicitly define the state transition relationships with a massive number of arcs and transitions. The designer need only define the transition data, and then focus on the functional aspects of their design.

This paper addresses state transitions in only the most general sense. That is where the modeling complexity is first and most apparent. The incorporation of the Entry, Exit and Do actions associated with each state results in similar complexity, and early efforts to model them using the same simplified approach described above show similar improvements. Therefore, the main future work includes incorporating transitional actions.

References

1. André, É., Benmoussa, M.M., Choppy, C.: Formalising concurrent uml state machines using coloured petri nets. Formal Aspects of Computing **28**(5), 805–845 (Sep 2016). https://doi.org/10.1007/s00165-016-0388-9
2. CPN Tools website (May 2020), http://cpntools.org
3. Meghzili, S., Chaoui, A., Strecker, M., Kerkouche, E.: On the verification of uml state machine diagrams to colored petri nets transformation using isabelle/hol. In: 2017 IEEE International Conference on Information Reuse and Integration (IRI). pp. 419–426 (Aug 2017). https://doi.org/10.1109/IRI.2017.63
4. Pettit, R.G., Gomaa, H.: Modeling state-dependent objects using colored petri nets. In: Proceedings of Workshop on Modelling of Objects, Components, and Agents. pp. 105–120. University of Aarhus, Denmark (2001)