Toward Understanding Dynamic Behavior in Abstract Machines

Sabah Al-Fedaghi, Computer Engineering Department, Kuwait University, Safat, Kuwait

ABSTRACT

This article proposes an abstract machine, known by the acronym FM, for modeling behavior currently described by finite state machines (SMs) in conjunction with related notions such as state and event. Although the SM concept was born more than 50 years ago and is widely applied in hardware design, the literature suggests it is not well understood despite its use in other application fields, especially in software system design. Several SM-based examples are remodeled in terms of the proposed machine, and they reveal a richer conception of the dynamic aspects of systems. FM enhances understanding of concepts in control modeling such as states, events, behavior, and control. The article exposes the limitations of unrefined SM features used in modeling software systems. It is suggested that FM be used as a modeling tool in the software system design process.

KEYWORDS

Conceptual Modeling, Control, Dynamic Behavior, Events, Finite State Machine, Software Modeling, System Specification

INTRODUCTION

The history of the concept of finite state machine illustrates its wide range of applications. The first researchers to consider the concept included biologists, psychologists, mathematicians, engineers, and computer scientists. “They all were interested in modeling the human thought process, whether in the brain or in a computer” (Subhadra, 2016). For example, Turing’s underlying thought was that “the organization of the brain must be realizable as a finite discrete-state machine” (Hodges, 2013; see Maguire et al., 2015). “All of the ‘computers’ that we encounter in everyday life are merely physical finite state machines” (Maguire et al., 2015).

Discrete state machines theoretically underlie almost everything related to computers (Mertz, 2000), but a survey revealed that only 28% of university students considered State Machine Diagrams very useful. Teachers have indicated that students tend to underestimate the relevance of state machines, and novices often reject the concept outright (Wyczca & Marcinkowski, 2007).

Finite state machines have a finite number of unambiguous states and use deterministic rules to transition from one state to another. Other models have been inspired by the ideas of states and diagrams; nevertheless, the state machine remains the dominant model for describing the behavior of control systems. State machines are so widely known as a subject of research and study that a review of their basic concepts seems hardly necessary.

SMs were originally introduced for hardware design and differentiated as Mealy machines (outputs as functions of state only) and Moore machines (outputs as functions of state and inputs). The concept of a state machine has also been used in software design, especially with the introduction of the extended state machine known as Statecharts with its view of sub-states through a state. “Statecharts
is sometimes considered a replacement for the state machine concept. There is no direct translation between a system of state machines and a Statecharts representation of a control system. Statecharts is a new, interesting model of control systems which is used in UML for behavior specification and is not too helpful for creating the software” (Wagner & Wolstenholme, 2004). This paper focuses on the classical (finite) SM as it includes the fundamental concept of incorporating states and events.

**Problem**

Specifying complex state machines can be quite tedious (MIT OpenCourseWare, 2011).

The concept, although born 50 years ago, is still not well understood or interpreted in the software domain, despite its wide application in hardware design. Misunderstandings about state machines have produced several stories and half-truths. The concept of the state machine has been several times (unintentionally?) reinvented for software. (Wagner & Wolstenholme, 2004)

Champandard (2007) describes state machines as low-level technique with limited logic, requiring custom extensions, and hard to standardize, not deliberative, and labor intensive. It seems that software developers do not use state machines very frequently. According to Skorkin (2011), “State machines are great and developers should use them more. How many times in my developer career have I actually used a state machine? The answer is zero times.”

Modeling of systems has many aspects, including the system’s components, the interactions between these components, and the components’ properties and states that form the system’s behavior. Modeling a system is a key step in the requirements phase, in which specifications are gathered and modeled. This paper focuses on the specification and modeling phase (the requirements phase) where the system has been modeled using the finite state machine (FSM) approach. The concept of state has been a central notion in this development. While the many diverse and unique properties of FSM are not overlooked, the notion is not rich enough to capture systems with both software and hardware components. The authors claim that the concept of state raises misconceptions, especially in education. An FSM produces scenarios of system behavior, but it is not a semantically rich representation, and it represents a type of sketch of truth tables, which are suitable for circuit-like structures.

As an alternative, a conceptual-level representation, this paper proposes the use of a flow-based model (with the acronym FM) to depict behavior in systems. The methodology is based on identifying things that flow (to be defined later) and specifying their streams of flow in terms of a maximum of six stages. The resulting conceptual picture functions as a map of different flows that trigger each other. Such semantics can be used for understanding the behavior of a given system in design and educational contexts. The paper explores the nature of state machines in the context of the behavior of a control model. Such an investigation reintroduces related notions such as state, behavior, control, and event and should contribute to the alleviation of some of the difficulties summarized here.

**Summary of Contents**

As background to our proposed approach to control models and for the sake of a self-contained paper, the review in the next section titled FLOWTHING MACHINE (FM) summarizes the general features of this model, which has been adapted for diverse applications (Al-Fedaghi, 2016a-e, 2017a,b).

The section titled MACHINES AND EVENTS provides a simple example of the flow of an order created by a customer and processed by a service provider. The paper includes a static FM for this example as a special description, then superimposes events as “happenings” in time. This is a new conceptualization of the event in a diagram that comes to life (is actualized). To illustrate the concept, the paper presents an example from informing science of a service called Cash Flow in Agribusinesses in order to design system behavior capable of reacting to significant change in its environment.
Bigraphical Reactive Systems Based Approaches for Modeling Context-Aware Systems
www.igi-global.com/article/bigraphical-reactive-systems-based-approaches-for-modeling-context-aware-systems/120651?camid=4v1a

COADA: Leveraging Dynamic Coalition Peer-to-Peer Network for Adaptive Content Download of Cellular Users
www.igi-global.com/chapter/coada-leveraging-dynamic-coalition-peer/68954?camid=4v1a