Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 3, pp. 292-300

A Novel Model for Adaptive Control Systems A State Machine Approach

F. Valles-Barajas

Fernando Valles-Barajas
Universidad Regiomontana
Information Technology Department
15 de Mayo 567 pte., C.P. 64000 colonia centro, Monterrey, Nuevo León, México
E-mail: fernando.valles@acm.org, fernando.valles@ieee.org

Abstract: In this paper a new model, based on state machines, of adaptive control systems is presented. Due to its high level of expressiveness, UML was chosen as the modeling language. In particular the paper presents a model of an indirect adaptive control system. This model can be used to document and to have a better understanding of adaptive control systems.
Keywords: adaptive control systems, state machines, Personal Software Process (PSP), software design

1 Introduction

Adaptive control is one of the research areas of control engineering that deals with time-varying systems [9]. To control a process using this technique, the first step is to obtain a model of the process based on measurements of the input and output of the process (these measurements are stored in a vector called measurement or observation vector $\phi(k)$); this step is called identification and is done by using a parameter adaptation algorithm (PAA) like the recursive least squares algorithm. Once the parameters of the process are obtained, the controller is designed using these parameters.

The Personal Software Process[1] is a modern methodology that helps the engineer to ensure quality software products [7]. The design phase of this methodology requires that the software engineer builds four models. Every of these models represents a different view of a system; for example there model that represents the states and the transitions between these states of a particular entity using state machines, which represent the internal dynamic view of a system.
State machines are used in software engineering to analyze the behavior of complex systems, for example the rational unified process (RUP)[2] uses state machines to model a use case, an operation or an object [2].

In this paper the concepts of state machines will be applied to model an adaptive control system (ACS). There are several notations to represent state machines (see for example [5], [6], [7]), in this paper the notation of the UML will be used. The reason for this, is the maturity of this modeling language and the success shown by UML to model complex systems.

Motivation and contribution of the paper:

1. the graphical model obtained with the state machine will help the control engineer to have a better understanding of the ACS control law.

2. this model could be used by a software engineer as a base to implement the software of the ACS.

[1]© Software Engineering Institute
[2]© IBM

3. by using this new model, a better understanding by the software engineer of an adaptive control system will be obtained.

4. this model can be used as documentation of the control system.

5. a better communication between the software engineer and the control engineer will be obtained by using the proposed model.

Conventions used in the paper: In all the paper, for the purpose of clarity, all the parts of the state machines (states, transitions, events, guard conditions and actions) are printed in *italic font*. The key concepts of the state machines will be written in **bold font**.

Related works: In [4] an adaptive strategy based on state machines is presented.
The **events** that are considerd in the strategy are: *threshold crossings*, *commands from the operator of the system* and *the occurrence of several patterns in the process signals* (control error $e(k)$, process input $u(k)$ and process output $y(k)$) among others. The occurrence of one of these events may provoke that the adaptive strategy changes from one state to another.
The **states** modeled in this strategy are: *initial* state, *open loop* state, *closed loop* state and *final* state. The *open* and *closed loop* states are **composite states**, which are states composed of other states [12].
In that paper the author shows a successful example of his proposal.

The paper of [13] shows an application of state machines in fault-adaptive control systems. As stated in that paper this kind of system can deal with time-varying systems and also with systems that present faults; the performance of the control system must remain unaffected in spite of these two problems. To deal with these problems that paper proposes to add to the control system a supervisory controller module and a reconfiguration management module. The relevant point in the paper is that the supervisor controller module is represented as a finite state machine (FSM).
That paper contains an example of the application of the roll control of a simulated airplane.

Organization of the paper: Section 2 presents the necessary concepts of the adaptive control theory to understand the proposed model. In section 3 an explanation of the building blocks for state machines is given. Section 4 contains the proposed model for an adaptive control system using state machines. The last section gives some concluding remarks.

2 Adaptive control systems

Fig. 1 shows the configuration of an indirect adaptive control system. In this kind of adaptive control, the model of the process $G_p(z^{-1})$ is obtained based on a set of input-output measurements ($u(k)$, $y(k)$) and then the controller is designed with this model. The parameter adaptation algorithm (PAA) block is responsible for obtaining this model. The controller design block designs the controller $G_c(z^{-1})$ based on the model obtained by the PAA and on the desired performance specified by the operator of the system. The adaptive control system must control the process in spite of the disturbances $d_1(k)$, $d_2(k)$, the noise $n(k)$ and the parametric variations of the process.

3 State machines

State machines are graphical models that allow the specification of the states of one entity, the legal transitions between these states, the events that can occur in the life of the entity and how the entity manages these events [10]. The transitions from one state to another can be fired by an event. Some

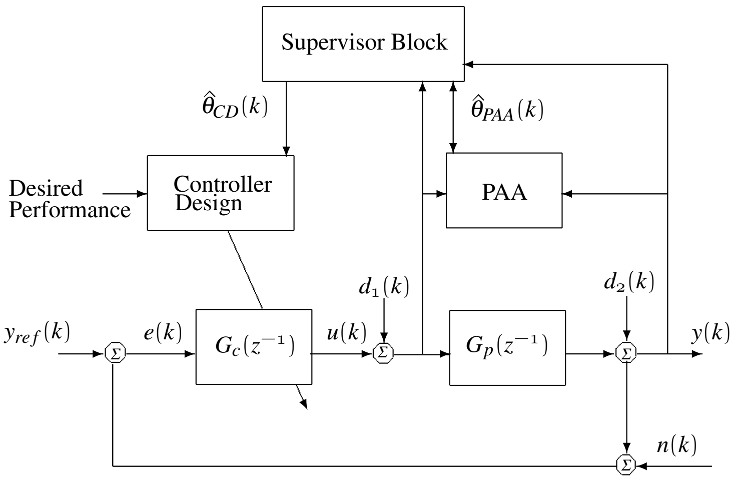

Figure 1: Indirect adaptive control system equipped with a supervisor block

condition (called guard condition) can be specified as a requirement to the occurrence of the transition.

To model an entity using a state machine the entity is isolated from the world and the communication with the rest of the world is specified by detecting events and responding to them. In UML state machines are represented using state diagrams [12].

A state machine can also be used to model use cases, which are useful to specify the functional requirements of a system. A state machine that describes the external events and its sequence in a use case is a kind of state diagram of use cases [10]. The external events happen between the actors, which are any external entity that interacts with the system, and the system [8]. Events that pass between objects residing in the system are referred to as internal events. Some authors like [10] recommend to use state machines to model the external and time events and the responses to them instead of using state machines to model the internal events of objects (instances of a class).

A state machine is drawn inside a rectangle with the name label in the top left-hand side (see fig. 2), though the frame may be omitted if the context is clear [1]. If the context is an instance of a class, all the classes related with it are candidate targets for actions and for inclusion in guard conditions [3].
A state is drawn as a rounded rectangle with two compartments: one for the state name and the other for the internal transitions and the internal behavior (see fig. 2). The second compartment may define entry and exit actions, activities, internal transitions and deferred events.
There must be only one initial state[3], but there may be any number of final states, including no final state at all. Initial states are pseudostates, they do not have all the properties of a real state. Final states are real states; they posses all the properties of a real state; this means that while the entity is in a final state it can perform some activity and can response to some events. The difference between final states and the rest of the real states is that they do not have transitions out. An initial state is drawn using a solid black circle and a final state is drawn using a shallow black dot (see fig. 2).
A state machine can have an exit point (drawn as a circle with a cross, as in fig. 2) that indicates the occurrence of an exception [1].

[3]It is important to mention that [1] considers that a state machine may have more than one initial state only if each of these states are labeled with the event that created the entity.

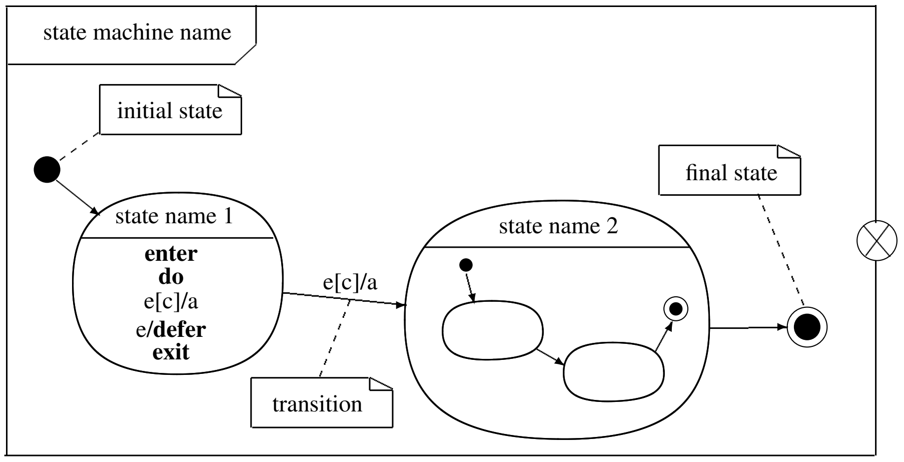

Figure 2: A state machine example

State machine can model parallel operations; this is important in control systems if we consider the situation when it is necessary that the system responses to commands from the operator and at the same time operates the plant.

This section has presented some basic concepts of state machines. The next section contains the model of an adaptive control system based on state machines. Also in the next section advanced concepts of state machines will be illustrated using the proposed model.

4 The model

This section explains how an adaptive control system can be modeled by a state machine. The explanation is divided in three section, which correspond to every of the building blocks of a state machine: the states, the events and the transitions. The actions that occur while a control system is in some state will be explained in the section of the states and the actions executed when a transition occurs will be explained in the section of the transitions.

4.1 States

A state describes the condition of an object [11]. This condition for a control system can be expressed in any of the following ways:

- in the terms of the control system attributes:
  - the values of the attributes that define the control system's structure. For example the values of the poles of the close loop transfer function (see the polynomial $A_m(z^{-1})$ in eq. 1) define if the system is in *stable* or *unstable* state.

    $$H_{cl}(z^{-1}) = \frac{z^{-d+1}B_m(z^{-1})}{A_m(z^{-1})} \tag{1}$$

    The control system will be in the *stable* state if all the roots of the transfer function denominator of eq. 1 are inside the unit circle, as it is stated in the eq.

    $$1 + a_{m1}z^{-1} + \cdots + a_{mn}z^{-n} = 0 \Rightarrow |z| < 1 \tag{2}$$

Otherwise the state of the system will be in the *unstable* state.

- the value of the attributes that define the object's relationships. In this point it is important to mention that the entire control system is represented with a class called ACS. When a fault occurs in the system a fault detection and isolation (FDI) object of the class FDI will be created to manage this fault. A FDI reference variable will be defined inside the ACS class, so when the fault occurs an instance of the class FDI will be created inside a method of the class ACS. If there is no fault in the system, the reference variable of FDI will have the value of null and the system will be in the *no fault* state, but when a fault appears the reference variable will point to the FDI object that will handle the fault and the system will be in the *fault* state.

- in terms of a behavior that the object is engaged in:
  - a period of time during which a control system performs some outgoing activity. For example, one of the activities that a controller must do to achieve robustness against the external agents to the system (disturbances, noise, faults) is to shape the sensitivity functions. This state will be called *shaping the sensitivity functions*. The output sensitivity function $\mathcal{S}$ is the transfer function between the output disturbance $d_2(k)$ and the plant output $y(k)$; this transfer function is usually named as sensitivity function and is given by:

$$\begin{aligned} S_{yd_2}(z^{-1}) &= \frac{A(z^{-1})S(z^{-1})}{A(z^{-1})S(z^{-1}) + z^{-d}B(z^{-1})R(z^{-1})} \\ &= \frac{A(z^{-1})S(z^{-1})}{P(z^{-1})} \end{aligned} \tag{3}$$

  The noise sensitivity function $\mathcal{T}$ is the transfer function between the measurement noise $n(k)$ and the plant output $y(k)$ and is given by:

$$\begin{aligned} S_{yn}(z^{-1}) &= \frac{-z^{-d}B(z^{-1})R(z^{-1})}{A(z^{-1})S(z^{-1}) + z^{-d}B(z^{-1})R(z^{-1})} \\ &= \frac{-z^{-d}B(z^{-1})R(z^{-1})}{P(z^{-1})} \end{aligned} \tag{4}$$

  An equation that relates these two transfer functions is

$$S_{yd_2}(z^{-1}) - S_{yn}(z^{-1}) = \mathcal{S} + \mathcal{T} = 1 \tag{5}$$

  - a period of time during which an object waits for some event or events to occur. Suppose that the ACS is equipped with a fault detection and isolation module to manage the faults that can appear in the system. The FDI system can be modeled as an object. One of the states of the FDI is the *waiting for a fault* state.

The response of a control system to an event depends on its state. For example if the system is in *closed loop* state and an the event *change reference* $y_{ref}(z^{-1})$ occurs the system will take $y(k)$ close to $y_{ref}(k)$ but if the system is in *closed loop* state and the same event occurs the system will do nothing.

4.2 Events

An event is any occurrence that provokes the reaction of an object, this reaction can be the transition from one state to another, the execution of one action or do nothing. It is important to mention that although events starting with UML 2.0 are called triggers; in this paper we will use the term events.
There are four types of events: signals events, time events, call events and change events. They will be explained in the following sections.

Signal events

A signal is an asynchronous event; it is not necessary that a sender waits for the answer of a receiver when a sender sends a signal to a receiver. Signals can also be triggered by an operation; this kind of signals are called exceptions. Signals are a one-way asynchronous communication between active objects.

An exception is the most common kind of signals that specifies the abnormal behavior of an operation [8]. Exceptions are modeled in UML as stereotyped classifiers. The parameters of the exceptions contain information of the abnormal behavior, and these are modeled as attributes in the exceptions. Because exceptions are modeled as classifiers, a hierarchy of exceptions can be drawn. Fig. 3 shows some of the exceptions that can be generated in an adaptive control system. These exceptions form a hierarchy.

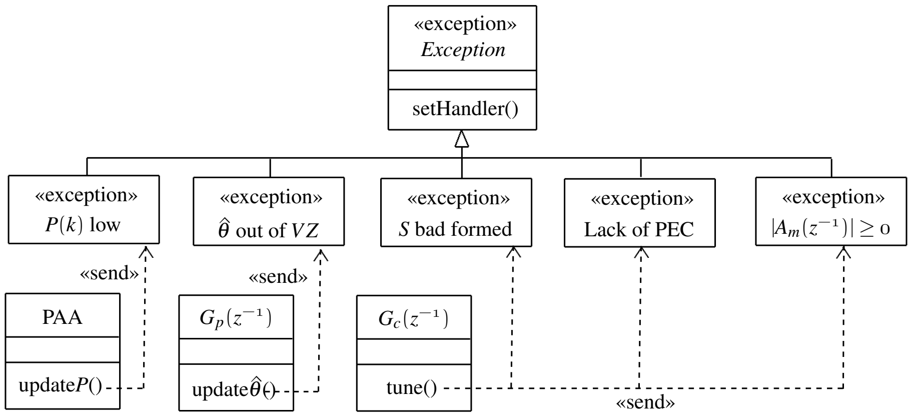

Figure 3: Modeling the exceptions of an adaptive control system

The root of this hierarchy is the abstract class "Exception" which was specified by typing the name of the class in italic font. The "Exception" class has a method called setHanlder(); this method is useful to specify the entity that is going to manage the exception, once it occurs. In this proposal, the supervisor is the entity that handles all the exceptions of an ACS.
Fig. 3 shows some of the exceptions that can be thrown in the methods of the classes PAA, $G_p(z^{-1})$ and $G_c(z^{-1})$. When $G_c(z^{-1})$ is tuned, the resulting controller may provoke that the closed loop system changes to the *unstable* state (the poles of $A_m(z^{-1})$ are outside the unit circle; see eq. 1). To specify that an operation can trigger an exception the exception can be drawn as a stereotyped class and then a send dependency can be drawn between the operation that can trigger the signal and the signal. See for example that in fig. 3 a send dependency between the operation update$\hat{\theta}$() and the signal "$\hat{\theta}$ is out of VZ" has been drawn to indicate that the operation update$\hat{\theta}$() can trigger the exception "$\hat{\theta}$ is out of VZ" (VZ is the valid zone which is a zone where the parameters of $G_p(z^{-1})$ are allowed. This can be obtained from previous knowledge or by association with the physical parameters of the process). Also the resulting controller may be poorly robust; this can be inspected by looking at the sensitive function $\mathcal{S}$ (see eq. 3) which is a measure of how the disturbance $d_2(k)$ affects the output of the process $y(k)$.

One of the advantages of the configuration of the fig. 3 is that polymorphism can be used to handle the exceptions; the handler of the exceptions can be specified in such a way that it handles all the exceptions of the ACS. Also when we specify a transition, we can specify that any of the exceptions of the ACS can trigger a transition; in this case the transition is polymorphic and can be triggered by the exception "Exception" or by any of its specializations, for example the "Lack of PEC" exception [3].

Signals are used to establish a communication between two objects in an asynchronous way. The reception of a signal is an event for the receiver. In this paper $d_1(k)$, $d_2(k)$, $n(k)$ and the bursting phenomenon will be modeled as signals as fig. 4 indicates. Is is important to mention that because $d_1(k)$, $d_2(k)$ and $n(k)$ are unknown, they must be estimated.
The entity that generates these signals is the environment and the receiver is the adaptive control system. Fig. 4 shows that signals are modeled as stereotyped classifiers and because of this a hierarchy of signals

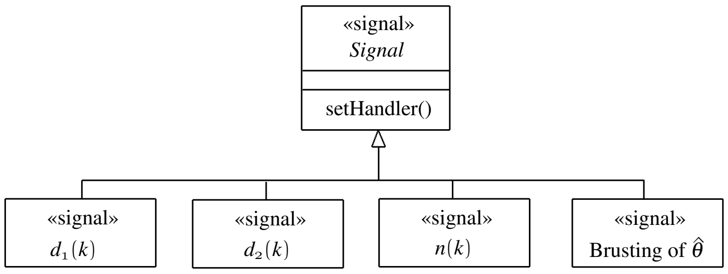

Figure 4: Modeling of the signals for an adaptive control system

can be established. Signals can also have parameters in the form of attributes, these are useful to store information related with the signals.

When we define signals we are making the opposite procedure for exceptions. When signals are defined the entities that handle the signals are defined and when the exceptions are defined the abnormal behavior that can generate an operation is defined [3].

In the UML, you model the signals that an object may receive by naming them in an extra compartment of the class, as shown in fig. 5 where the signals that the supervisor block can handle are defined. Signals do not return values to the caller [8].

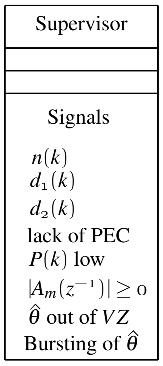

Figure 5: Signals that the supervisor can handle

Time events

A time event evaluates the passage of time as a trigger. It is assumed that the object has some mechanism to monitor the passage of time. Time can be specified either in absolute mode (time of day) or relative mode (time elapsed since a given event). Time can be considered as an event from the environment. The general form of a time event is:

after(exp)[guard condition]/ action.

An example of a time event is:

after(5 minutes)[$|P(k)|$ *low]/reset* $P(k)$

where 5 minutes is the expression that is evaluated, [$|P(k)|$ low] is called guard condition and reset $P(k)$ is the action that is executed when the expression and the guard condition are true.

Call events

In the UML, you model the call events that an object may receive as operations on the class of the object. For example, in fig. 3 the operation of the class $G_c(z^{-1})$ was modeled as a call event.

Signal events and call events involve at least two objects: the object that sends the signal or invokes the operation and the object to which the event is directed.

Change events

A change event occurs when a boolean expression becomes true. When an event is declared as a change event, the system always monitors that boolean expression. A change event represents a continuous and potential nonlocal computation (action at a distance, because the value or values tested may be distant). Another alternative for a change event is to model this as a guard condition in the transition. The advantage of a guard condition is that it is evaluated only when a transition occurs.
The general form for a change event is:

when(expression) / operation()

where operation() is executed when the expression is true. The persistent excitation condition can be verified by using the following change event:

$$\text{when}\left(lim_{k_1 \to \infty} \frac{1}{k_1} \sum_{k=1}^{k_1} \phi(k)\phi^T(k) \leq 0 \right) / \text{ freezeAdaptation}()$$

5 Conclusions

In this paper a new model for adaptive control systems has been presented. This model is based on state machines and is useful to document and to analyse control systems. The states of an adaptive control system, its transitions and the events that may provoke the transition from one state to another were defined in the proposed model. With this model the software implementation of an adaptive control system, made by a software engineer, will be easier.

Bibliography

[1] S. Bennett, J. Skelton, and K. Lunn. *Schaum's Outline of UML.* McGraw-Hill, USA, 2005.

[2] S. Bergstrom and L. Raberg. *Adopting the Rational Unified Process: Success with the RUP.* Addison Wesley, USA, 2004.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley Professional, 2^{nd} edition, May 19, 2005.

[4] D. Drechsel. An adaptive control system modeled as finite state machine. *Proceedings of XVI Annual Convention and Exhibition of the IEEE In India*, pages 124 – 128, 1990.

[5] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 3^{rd} edition, 2003.

[6] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, pages 231–274, 1987.

[7] W. S. Humphrey. *PSP^{sm} : A Self-Improvement Process for Software Engineers (SEI Series in Software Engineering).* Addison-Wesley Professional, USA, 2005.

[8] IBM. *Object Oriented Analysis and Design using UML.* USA, 2004.

[9] P. Ioannou and B. Fidan. *Adaptive Control Tutorial.* Society for Industrial and Applied Mathematics, 2006.

[10] C. Larman. *Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Prentice Hall, USA, 3^{rd} edition, 2004.

[11] T. Pender. *UML Bible.* Wiley, 2003.

[12] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley Professional, 2^{nd} edition, 2004.

[13] G. Simon, T. Kovácsházy, G. Péceli, T. Szemethy, G. Karsai, and A. Lédeczi. Implementation of reconfiguration management in fault-adaptive control systems. *IEEE Instrumentation and Measurement Technology Conference*, 2002. Anchorage, AK, USA.

Fernando Valles-Barajas obtained a graduate degree in Computer Science at Center for Research and Graduate Programs of La Laguna Institute of Technology (1991). He received an MS in Control Engineering (1997) and a PhD in Artificial Intelligence (2001) from Monterrey Institute of Technology (ITESM) campus Monterrey. He was a research assistant at Mechatronics department of ITESM campus Monterrey (1997-2001). He received certification as a PSP Developer from the Software Engineering Institute of Carnegie Mellon University (2008). He is member of the IEEE and ACM. His research interests include topics in Software Engineering and Control Engineering. Currently, he is full-time professor in the Information Technology Department at Universidad Regiomontana, Monterrey, Nuevo León, México. He also teaches modules at both BS and MS levels in Computer Science and Software Engineering.