MODELING AND CONTROL OF ELEVATORS BY STATECHARTS

Yi-Sheng Huang, Sheng-Luen Chung, and Mu-Der Jeng

ABSTRACT

Statechart has been utilized as a visual formalism for the modeling of complex and interactive systems for its illuminating features on describing properties of causality, concurrency, and synchronization. This paper presents the application of statechart to the modeling, design and implementation of an elevator system, whose system behavior involves aggregating complexity of state descriptions, and imposition of underlying control policy. Based on the operational flow of an elevator, we derive the associated statechart model by looking into the inherent hierarchically structure of the elevator. The advantage of the proposed approach is the clear presentation of system behavior in terms of conditions and events that cause the transitions in system dynamics. Implementation of the controlled elevator based on the modeled statechart is also presented.

Key Words: Statechart, interactive systems, elevator.

I. INTRODUCTION

Elevator systems form a class of DES’s whose complexity makes them difficult to model, analyze and optimize [10]. In a general elevator system, two types of buttons determine the elevator movement: hall call buttons and car call buttons. The hall call buttons are installed on the panel entering the elevator at the hall of the building, and the car call buttons are installed inside the elevator. An elevator system has a pair of hall call buttons on each floor, one for up hall call and the other for down hall call. Once a hall call button is pressed, the elevator moves to serve the hall call immediately. After entering the elevator, the passenger presses car call button to select his destination floor. The elevator then moves up/down to the destination floor. To increase utilization of the elevator, the elevator may stop at floors that request service of same movement directions. The elevator stops at the nearest requested floors for all queued request. When all calls are served, the elevator stops and waits for the next call to arrive, and the above operation repeats again.

In characterizing a complicated system like the elevator system mentioned above, a rigorous modeling method is required. Finite state machines (FSM’s) have been used for the modeling of discrete event systems because of their inherent simplicity. Based on FSM’s, Ramadge and Wonham [12-14] and the work that followed [15,17] analyze control problems in the framework of state machines. However, in other control problems where a large number of components operating concurrently, the state-machine based modeling approach becomes awkward as the resultant number of states will grow exponentially with the number of parallel components. This problem of state explosion constitutes a severe shortcoming of the state-machine modeling framework [1].

To alleviate the modeling complexity of the state-machine formalism while preserving many of its appealing feature, a statechart modeling framework is introduced by Harel [7], which extends ordinary state-machines by endowing them with natural constructs of: orthogonality, depth, broadcast synchronization and many other sophisticated features that strengthen its modeling power. In summary, Harel’s statechart is best described as follows:

\[ \text{Statecharts} = \text{FSM} + \text{depth} + \text{orthogonality} + \text{broadcast} . \]

In particular, (1) States are organized in a hierarchy of superstates and substates thereby achieving depth. (2) States are composed in parallel thereby achieving concurrency. (3) Transitions are allowed to take place at all
levels of the hierarchical structure, thereby achieving descriptive economy. Overall, statechart provides insights into causality, concurrency and synchronization of the modeled systems.

Based on Harel’s statecharts, related study includes the following. Brave et al. in [1] proposed Hierarchical State Machines (HSM’s), which is a simplified version of statecharts that extend state-machines by adding only hierarchy and orthogonality features. HSM’s are well suited for modeling and specification of complex processes such as manufacturing systems, communication network and air traffic control systems, etc. Related to modeling analysis, Masiero et al. [9] proposed an algorithm to create a reachability tree for statecharts and also showed how to use this tree to analyze dynamic properties of statecharts. Related to software implementation of statechart-based systems, Harel et al. [8] introduced a set of STATEMATE tools, with a heavy graphical orientation, intended for the specification, analysis, design and complex reactive systems. In [6] Drusinsky et al. proposed that statecharts can be beneficially used as a behavioral hardware description language. Coleman et al. [3] showed us how to use statecharts in object-oriented design. Moreover, Borges et al. [2] and Sowmya et al. [16] extended statecharts to deal with reactive systems.

To illustrate its description power on practical control system, this paper presents the modeling, control and implementation of an elevator system. A statechart-based for an elevator controller design and implementation is proposed. The intended advantage of the proposed approach is the detailed accounts of system’s behavior over time, including the dynamics of activities, their control and timing behavior, the states of the elevator system, and the conditions and events that cause states to change and other occurrences to take place.

The rest of the paper is organized as follows: section II presents basic definitions of statecharts that are related to this paper. Section III presents the elevator system modeling. Section IV presents the implementation of the elevator controller. Conclusions are presented in section V.

II. AN OVERVIEW OF STATECHARTS

The statecharts method was introduced as a visual formalism for specifying the behavior of complex reactive systems [7]. As in the conventional state-transition diagrams, the notion of states constitutes the basic component of statecharts. However, states may be embedded into superstates thus creating hierarchies of states. Superstates may be one of two types: AND-states or XOR-states. The former captures the notion of independence, e.g. image and sound may be at the same time in a website. The latter corresponds to refinement of states, e.g. a light bulb either stays on OFF or ON state. The components of an AND-state are called orthogonal components and have the distinctive feature that a system in an AND-state is also in all its orthogonal components. This means that a state S is described as consisting of two or more orthogonal components, and to be in state S entails being in all of those components simultaneously. The notation is represented by a dashed line that partitions the state into its components. Conversely, if the system is in a XOR-state, it has to be in only one of its substates.

States interact through transitions, which consist of three basic parts: an event expression, a condition, and action statements. The general syntax of an expression labeling a transition in a statechart is

$$\alpha[C]/\beta$$

where $\alpha$ is the event that triggers the transition, $C$ is a condition that guards the transition from being taken unless it is true when $\alpha$ occurs, and $\beta$ is an action that is carried out if the transition is taken. All of these are optional. Events and conditions can be considered as inputs, and actions as outputs. For example, if $\beta$ appears as an action along some transition, but it also appears as a triggering event of a transition in some orthogonal component, then executing the action in the first transition will immediately cause the second transition to be taken simultaneously. In fact, a transition is relevant in a time step if its source states are active and it fires when both the event expression and the condition are evaluated to be true in the same time step. Execution of an action may generate other events, which are broadcasted to the orthogonal components, possibly firing new transitions. Statecharts also allow a transition to leave a superstate, i.e. to leave all the substates in the superstate. Conditions are formed by a combination of logical and relational operators involving variables and special conditions, e.g. being in a state. Event-expressions are formed from a basic set of primitive events, which can be combined using logical disjunctions and conjunctions. Other special conditions and events, as in(s) and en(s), are taking into account in this paper.

To summarize the definitions above, a statechart example is modeled and shown in Fig. 1. This statechart contains a state S, consisting of the two orthogonal components $U$ and $V$; being in $S$ is being in both. Both components are XOR-state: the first consisting of $A$, $B$ and $C$, and the other consisting of $W$, $X$, and $Y$. It then follows that to be in $S$ is to be in one of $A$, $B$ or $C$, and at the same time being in one of $W$, $X$, or $Y$. Such a tuple of states, each from a different orthogonal component, is called a state configuration. We say that $S$ is the parent of the components $U$ and $V$, or that $U$ and $V$ are the substate of $S$. Notice that the components $U$ and $V$ are not different from any other states; they may have their own substates, default entrances, internal transition, and so on.
currency. That is, a single event transition that is effectively a form of synchronized concurrency. This means the event $a$ triggers two simultaneous transitions. If $c$ now occurs, the new configuration is changed to $(B, W)$; the transition $c$ is taking place in the $V$ component, independently of what might be happening in the $U$ component. Now, if event $\gamma$ occurs, the system exits $B$ and the new configuration is changed to $(C, W)$. Notice the $in(X)$ in $U$ represents if the system is in $(B, X)$; the $b$ transition from $B$ to $A$ occurs only if the system is in $(B, X)$ causing $in(X)$ to be true. Moreover, it is worthy to mention that the event $en(C)/\beta$ takes place only if the system is entering in $(C, X)$. This means the event $en(C)/\beta$ occurs, when the system transfers to $(C, X)$, while simultaneously causing action $\beta$ to take place.

Figure 1 shows the event $en(C)$ occurs at the entrance to $(C, Y)$, and the accompanied action $\beta$ causes entrance to the initial configuration $(A, Y)$.

III. MODELING OF ELEVATION SYSTEMS

The dynamics of an elevator system is generally complicated in that the system behavior involves aggregating complexity of state descriptions, and can vary very differently when imposing different control policy. However, we find that statechart being a nice model for an elevator. In general, an elevator group may consist of multiple elevators [4,10-11], each of which has its own car controller while the group controller is to coordinate the behavior of the elevator system. In this paper, however, we focus on a single elevator. This section describes the basic control problems in elevator system and how to obtain its control model by statecharts.

3.1 The elevator state transitions

The elevator control problem considered in this paper consists of a single elevator in a building with eight floors, numbered one to eight. In this system, capacity issue is considered here also: constraint on the maximum elevator load is considered to ensure safe services. As a result of that, a full elevator cannot load any more passengers. For passenger selection on destination, the elevator hall and car are provided with hall buttons and car buttons, respectively. Once a hall/car button is pressed, the elevator serves the hall/car call immediately by moving to the selected destination floor. When all calls are served, the elevator stops and waits for the next call to arrive, and then repeats the above operation again.

To model the complicated system behavior involved, we first give some basic state definitions:

1). IDLE state: the elevator stops at a certain floor.
2). OPEN state: when the car of the elevator reach passenger’s destination floor or the requested hall call position, the elevator door is to open.
3). WAIT state: the elevator stays at a certain floor, waiting for the passenger inside to leave or the passenger outside the hall to enter.
4). CLOSE state: the elevator door is to close after the opening time, which is set at 10 seconds, is over.
5). UP state: the elevator is moving up.
6). DOWN state: the elevator is moving down.
7). OVERLOAD state: the weight of all passengers inside the elevator is overloaded.

To understand how a state is transferred to another state, we use a state transition diagram (e.g. Fig. 2) to explain their relations. Assume the initial state of the elevator system is in IDLE state. A hall button is pressed when a passenger wants to enter the elevator. Upon entering the elevator, he or she selects a destination floor by pressing a car button, and the elevator state is transferred to Up or Down state accordingly, before starting to move up or down. Once the elevator arrives at the destination floor, the elevator state changes to OPEN state and the door ready to open. When the door is opened, the elevator is now at the WAIT state: waiting for passenger to come out of the car, or passengers outside to enter the car. In this state, three cases are possible: one is overloading, one that is the waiting time is over, and the other that is a close button pressed inside the car. For the first case, the elevator changes into an OVERLOAD state, which will then trigger an overload signal; the elevator will then go back to WAIT state. For the other two cases, the elevator changes to the CLOSE state and then the door is closing. After the door is closed, the elevator changes to IDLE state, thus completing an operation cycle.

3.2 The elevator control strategy

The control objective for the elevator system is to direct elevator movement: At each time, the elevator either moves up/down or stays at the same floor, based
on control decision, which in turns depends on the requested service and the control policy utilized. The control problem for an elevator system may be viewed as a disturbance rejection control problem [5], of which the basic objective is to empty the passengers in the elevator. In other words, the main control aim has to guarantee that all the buttons pressed be served and the elevator be empty.

The control policy we use complies with the common elevator rule: Service requests are served at a first-come-first-serve basis, with the exception when the elevator is heading for a particular direction, the elevator will pick up all requests heading for the same direction along the way. After that, the elevator will serve requests of the opposite direction. Elevator will move up and down until all the outstanding requests are served.

Based on the above description, our control strategy for the elevator control problem can be stated as follows:

1) The persons staying nearer to the elevator gain higher priority if their location is on the same direction as what the elevator is heading.
2) However, passengers’ request should temporarily be ignored if they are at the opposite direction of the elevator’s movement.
3) The first new disturbance has the highest priority to service when the elevator is inactive.

Our control strategy satisfies the basic objective of emptying the elevator. Note that our control model is separated into a control plane and a controller. In the next subsection, we will show how to obtain an elevator system model based on statecharts.

### 3.3 The elevator system hierarchical tree diagram

In this subsection, main equipments of the elevator system are introduced in the following:

1) Buttons: the buttons include both hall calls and car calls. Press on any of the buttons represents a service request, and is considered as a disturbance input in the elevator system.
2) Elevator Car: the controlled plane that is always in one of the seven states shown in the previous subsection.
3) Display Lights: the lights, which display all of the active buttons and current location of the elevator car.
4) Sensors: the sensors, which report the elevator car position and the elevator state.

To clarify relations among these subsystems, we construct a hierarchical tree diagram as shown in Fig. 3, based on the definitions proposed by Brave et al [1]. The notation $A^+$ represents the subset of XOR-states, $A^⊥$ the subset of AND-states and $A_{\text{basic}}$ the subset of basic states.

This hierarchical tree diagram contains useful information like the following:

1) The elevator system consists of three main mechanisms: car mechanism, input mechanism and display mechanism. In terms of statechart modeling, all of mechanisms are viewed as AND states. As such, the state ELEVATOR is an AND-state with three sub-states: INPUT, CAR, and DISPLAY. We describe the AND relation as below:

\[ \{\text{INPUT, CAR, DISPLAY} \in \text{ELEVATOR}^+\} \]

2) The input mechanism consists of button and sensor components. The input mechanism state changes when any button is pressed or sensor state changes. Since both conditions occur independently, they are related by AND relation. We show the AND-state as below:

\[ \{\text{BUTTON, SENSOR} \in \text{INPUT}^+\} \]

3) The car mechanism consists of a capacity limited elevator car and car call buttons. As discussed above, car
can be at any of the seven substates. The state CAR is an XOR-state: CLOSE, OPEN, UP, DOWN, WAIT, or OVERLOAD. We show the XOR-state as below:

\{CLOSE, DOWN, UP, IDLE, OPEN, WAIT, OVERLOAD \in CAR^\ast\}.

4) The display mechanism consists of all lights of the elevator display system. The display mechanism not only shows all of the passengers demand for service but also shows the elevator’s current position. It is clear that the system permits all button events happening wherever the elevator locates. This means those substates are concurrent. We show the AND-state as below:

\{\text{LIGHT, CARPOSITION} \in \text{DISPLAY}^\perp\}.

5) The states at the bottom of the hierarchical tree, such as ON, OFF, DISABLE and ENABLE, are not shown in this paper. These states are called basic states. The ENABLE and DISABLE states are used to indicate if any button is pressed. The parts of the basic states and their hierarchy relation are shown as follows:

\{\text{UON}^\text{basic}, \text{UOFF}^\text{basic} \in \text{UP}^\ast\}, \{\text{DON}^\text{basic}, \text{DOFF}^\text{basic} \in \text{DOWN}^\ast\},

\{\text{OPON}^\text{basic}, \text{OPOFF}^\text{basic} \in \text{OPEN}^\ast\},

\{\text{CLON}^\text{basic}, \text{CLOFF}^\text{basic} \in \text{CLOSE}^\ast\},

\{\text{ON}^\text{basic}, \text{OFF}^\text{basic} \in \text{WAIT}^\ast\},

\{\text{BPON}^\text{basic}, \text{BPOFF}^\text{basic} \in \text{OVERLOAD}^\ast\},

\{\text{DISABLE}^\text{basic}, \text{ENABLE}^\text{basic} \in \text{BUTTON}^\perp\},

\{\text{ON}^\text{basic}, \text{OFF}^\text{basic} \in \text{LIGHT}^\perp\}.

Based on above discussion, we can construct the associated statechart model accordingly by hierarchical tree and their mathematical models. In the next subsection, we will show how to construct an elevator system statechart model through top-down procedure.

3.4 The elevator system model by statecharts

Based on the hierarchical tree in Fig. 3, we construct the associated statechart in Fig. 4, which serves as overview of the elevator system statechart model. The top-level state ELEVATOR is decomposed into three INPUT, CAR and DISPLAY substates. Based on the AND-state \{\text{CAR, INPUT, DISPLAY} \in \text{ELEVATOR}^\ast\}, we know these substates are all orthogonal components. The elevator statecharts are depicted as rectilinear box with rounded corners. The two dash lines are used to divide it into three orthogonal behavioral components.

The INPUT substate is decomposed two parallel behavioral components, BUTTON and SENSOR; each of these is further decomposed into exclusive substates that are to be depicted later. The CAR substate is decomposed into seven exclusive states. The DISPLAY substate is decomposed two orthogonal components, BUTTON and CARPOSITION. We specify the initial state by using a small arrow emanating from a small solid circle, meaning that the system must be in the initial states of the five substates simultaneously. In more details, we can zoom in to the INPUT, CAR and DISPLAY states, and show the next-level state decomposition of the elevator system. The name of the state appears inside their rectilinear box. The transitions are drawn as splined arrows with the triggers severing as labels. The triggers of the transitions in this paper are all events, which are regarded as instantaneous occurrences. There are two kinds:

1). External events coming from external sources, such as the demands from the passengers via hall calls or car calls.

2). Internal events coming from internal sources, such as down, up, open, close, overload, and so on.

In our elevator system, there are 24 buttons: 10 car call buttons and 14 hall call buttons. The former includes eight floor’s number buttons, one open and one close buttons. The latter includes seven up hall call buttons and seven down hall call buttons. It is worthy to note that these 24 buttons are external events, which are considered as disturbance input in the elevator system. In this paper, except these 24 disturbance inputs, other events are all internal events. Meanwhile, there are 24 button lights to display the above buttons and eight lights to display the current position of the elevator. In particular, there are eight relays at each floor and two relays in the door mechanism and one relay in the weighbridge mechanism to sense the current state of the elevator.
Recall the AND-state \{BUTTON, SENSOR \in INPUT\}, we know the main states of the INPUT is decomposed into two orthogonal components: BUTTON and SENSOR. For more details, first, the BUTTON state consists of several basic states those are depicted as follows:

\begin{align*}
\{&BOPEN_1^{\text{basic}}, BCLOSE_2^{\text{basic}}, BC_3^{\text{basic}}, \ldots, BC_8^{\text{basic}}, \\
&BUHU_1^{\text{basic}}, \ldots, BUHU_7^{\text{basic}}, BHD_2^{\text{basic}}, \ldots, \\
&BHD_8^{\text{basic}} \in \text{BUTTON}^\perp\}.
\end{align*}

Next, the SENSOR state is decomposed into three orthogonal components, FLOOR, DOOR and ALARM, in AND-state \{FLOOR, DOOR, ALARM \in SENSOR\}. As we zoom in these states, we obtain each exclusive substate of these states as depicted in following:

\begin{align*}
\{&F_{1}^{\text{basic}}, F_{2}^{\text{basic}}, \ldots, F_{8}^{\text{basic}} \in \text{FLOOR}^\perp; \\
&\text{SELFHOLD}^{\text{basic}}, \text{OPENING}^{\text{basic}}, \text{OPENED}^{\text{basic}}, \\
&\text{CLOSING}^{\text{basic}}, \text{CLOSED}^{\text{basic}} \in \text{DOOR}^\perp; \\
&\text{DISABLE}^{\text{basic}}, \text{ENABLE}^{\text{basic}} \in \text{ALARM}^\perp
\end{align*}

In summary, statecharts for BUTTON and SENSOR are depicted in Fig. 5 and Fig. 6, respectively. Interpretations on states and events are listed in the Appendix. Fig. 5 shows that these states are all orthogonal components. All buttons are initially at the basic states DISABLE, meaning that the elevator has no disturbance inputs at this moment: the elevator system is idling and waiting for passengers’ demands.

Notice how these events are sensed immediately by the orthogonal component. Events generated by actions in one component are sensed by all other orthogonal components. For example, when a hall call is pressed at third floor (i.e., a phu3/hu3on event is triggered), this single event triggers two simultaneous transitions: one is the BHU3 state transferred from the basic state DISABLE into ENABLE, the other is the LHU3 of the LIGHT state transferred from the basic state OFF into ON. It is interesting to note that some of events in the BUTTON statechart are with label conditions that depend on states in orthogonal components. Both the events po and pc are with in (WAIT), meaning that both events are triggerable, only if the elevator car stays in the WAIT state. In other words, the open and close buttons can be triggered if the elevator car stops at a certain floor.

Figure 6 consists of three orthogonal components: FLOOR, DOOR and ALARM states. Based on the default arrows, it shows the elevator stays at the ground floor, the door is closed, and no alarm event happens.

As to the analysis of SENSOR statechart, we again suppose a hall call is pressed on the third floor as in the previous example. The phu3/hu3on event does not only turn on a button light but also urge the elevator to move up. The relay rf2 will be sensed when the elevator car passes through the second floor: the event rf2 makes the basic state F1 change to F2. Meanwhile, the triggered action f2 is active and is to turn on the second floor light (shown in LIGHT statecharts later). As to the complex events when the elevator car arrivals at the third floor, the operation details will be depicted later.

The statechart for the CAR state with its transitions and triggers are shown in Fig. 7. Notice how these events are sensed immediately by the exclusive components. With the initial state being in the IDLE state, we also assume that the elevator car stays on some floor. Once a hall call button is pressed (external event), there are three possible events: up, down and open/\alpha. We are to explain one by one.
First, if the hall call’s position is higher than the elevator car’s current position, the internal up event will be triggered and makes the state changed to UP state, with the elevator car moving up at the same time. Notice that the default arrow indicates UON state is the initial state of the UP state. That is, the elevator car is moving up. Once the elevator car arrives at the destination (i.e., the hall call’s position), the state changes to UOFF state by the internal arrive event, and then the state changes to OPEN state by the en(UOFF)/α event immediately. The en(UOFF)/α event makes the Up state change to the OPEN state with the α action being carried out instantaneously. Occurrence of the α action also affect the orthogonal component of SENSOR states. The α action takes the subset SELFHOLD of the DOOR state to the OPENING state. Meanwhile, the other orthogonal component, CAR state, enters the OPON of OPEN state according to the default arrow. Thus the door will be opened immediately. When the door is opened, the relay rd1 becomes active (i.e., rd1/openover is triggered), the internal openover event is triggered and the state be brought to OPOFF state, and then the state changes to WAIT state by the en(OPOFF)/γ event immediately. Notice that the action γ brings the OPENED state back to SELFHOLD state. This means the controller keeps the elevator car’s door in OPENING state. When the system stays in WAIT state, four cases are possible: (1) an internal overload event makes the WAIT state change to OVERLOAD state; (2) an external pressopen event (i.e., an open button be pressed) makes the WAIT state changes to OPEN state; (3) an external pressclose event (i.e., an close button be pressed) makes the WAIT state changes to CLOSE state; and (4) an internal timeout event (about 10 seconds) makes the WAIT state change to the CLOSE state. Notice that the two external events are active if the events take place before the internal timeout event active. When the system stays at the CLOSE state, two cases are possible: one is an internal closeover event making the CLOSE state changes to IDLE state; the other is an external pressopen event making the CLOSE state change to the OPEN state. It is worthy to note that the external pressopen event has to take place before the internal closeover event becomes active.

Second, if the hall call’s position is lower than the elevator car’s position, the internal down event will be triggered and makes the state change to DOWN state, causing the elevator car move down at the same time. Once the elevator car arrives at the destination floor (i.e., assume the hall call’s position is at third floor), the state changes to the DNOFF state by the internal arrive event, and then the state changes to OPEN state by the en(DNOFF)/α event immediately. The rest of the state transitions are similar to those stated above and are omitted.

Third and finally, if the hall call’s position is equal to the elevator car’s current position, the internal open event will be triggered and makes the state change to OPEN state, thus making the elevator open the door at the same time. Similarity, the rest of the state transitions are similar to those stated above in case one, and are omitted.

We show the DISPLAY statecharts in Fig. 8, which consists of two orthogonal components LIGHT and CARPOSITION.

IV. IMPLEMENTATION OF ELEVATOR CONTROLLER

The block diagram of the elevator control system is shown in Fig. 9. In this control system, a number of sensors (i.e., relays) are used to determine the elevator’s position and the door’s state. The primary function of an elevator controller, modeled by the statecharts, is to coordinate two kinds of inputs: (1) external input, which is the random demand from the passenger(s) via hall call or car call. (2) internal input, which is the feedback signal of the system status reported by relays.

We defined the plant of the control system as the part of the system that is to be controlled. Elevator position is changed by events triggered and/or active actions. As the control objective, we want the system output, i.e., elevator position, to be equal to the external inputs. In doing so, each of the operation sequences can be regarded as a concatenation of operations performed at the elevator system. In terms of statecharts modeling, each operation can be constructed by:

1) A triggered event signifying the “event” of the intended operation is initiated and followed by
2) An active action representing the status of concurrent transitions being taken.
The elevator states are changed by the transitions. It is interesting to note that a transition can be labeled not only with the trigger that causes it to happen, but also optionally with an action. If an action is present when the transition takes place, the specified action is carried out instantaneously.

To illustrate how these three components of the control block work together, let us consider the following example: Assuming the elevator system stays at IDLE state, and the elevator is also at ground floor as shown in Fig.10, an passenger on ground floor wants to go up to the eighth floor. Once a hall call up button is pressed (i.e., a transition phu1/hulon generated), the event phu1 and action hulon would take place concurrently. Fig. 11 helps us understand the operation of the events and actions. The phu1 event makes the system change from IDLE state to OPEN state and also generate an internal transition open/α. In addition, the action hulon turns on the hall call button light.

Moreover, the α action takes the substate SELF-HOLD of the DOOR state to the OPENING state. At the same time, the door will be opened immediately. Fig. 12 shows the result of α action taking place. When the door is opened, the relay rd1 becomes active (i.e., rd1/openover is triggered), the internal openover event is triggered, the state is brought to OPOFF state, and then the state changes immediately to WAIT state by the en(OPOFF)/γ event. This is the control sequence of the example. Fig. 13 depicts the elevator staying on UP state and the elevator is moving up. Fig. 14 shows the passenger reaches his/her destination.

In order to design and implement the elevator control system, the object-orient programming technique is used. The demonstration program is written with Visual Basic 6.0 taking about 420K bytes, whose operation flow is modeled by statecharts.

V. CONCLUSIONS

This paper presents the modeling, control and implementation of an elevator system using statecharts. The
complexity nature due to state aggregation and the imposition of underlying control policy is modeled by statechart with emphasis on event synchronization among states of different depth. Based on the operational flow of an elevator, we derive the associated statechart model by looking into the inherent hierarchical structure of the elevator. The advantage of the proposed approach is the clear presentation of system behavior and readiness for implementation.

APPENDIX A

Table A1. Interpretation of states in the elevator statecharts.

<table>
<thead>
<tr>
<th>State</th>
<th>Interpretation</th>
<th>State</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALARM</td>
<td>showing alarm states</td>
<td>IDLE</td>
<td>showing idle states</td>
</tr>
<tr>
<td>BC1~BC8</td>
<td>car call button states</td>
<td>INPUT</td>
<td>showing input signal states</td>
</tr>
<tr>
<td>BHD1~BHD8</td>
<td>hall call down button states</td>
<td>LC1~LC8</td>
<td>car call button light states</td>
</tr>
<tr>
<td>BHU1~BHU7</td>
<td>hall call up button states</td>
<td>LCLOSE</td>
<td>showing close light states</td>
</tr>
<tr>
<td>BCLOSE</td>
<td>showing close button states</td>
<td>LF1~LF8</td>
<td>car position light states</td>
</tr>
<tr>
<td>BOPEN</td>
<td>showing opening button states</td>
<td>LHD2~LHD8</td>
<td>hall call down button’s light</td>
</tr>
<tr>
<td>BPOFF</td>
<td>beep off state</td>
<td>LHU1~LHU7</td>
<td>hall call up button’s light</td>
</tr>
<tr>
<td>BPON</td>
<td>beep on state</td>
<td>LIGHT</td>
<td>showing button light states</td>
</tr>
<tr>
<td>BUTTON</td>
<td>showing button states</td>
<td>LOPEN</td>
<td>showing open light states</td>
</tr>
<tr>
<td>CAR</td>
<td>showing elevator car states</td>
<td>OPEN</td>
<td>showing open states</td>
</tr>
<tr>
<td>CARPOSITION</td>
<td>showing car position states</td>
<td>OPENED</td>
<td>the door be opened state</td>
</tr>
<tr>
<td>CLOFF</td>
<td>closing off state</td>
<td>OPENING</td>
<td>the door be opening state</td>
</tr>
<tr>
<td>CLON</td>
<td>closing on state</td>
<td>OPOFF</td>
<td>opening off state</td>
</tr>
<tr>
<td>CLOSE</td>
<td>showing close states</td>
<td>OPON</td>
<td>opening on state</td>
</tr>
<tr>
<td>CLOSED</td>
<td>the door be closed state</td>
<td>ON</td>
<td>Timer on (10sec)</td>
</tr>
<tr>
<td>CLOSING</td>
<td>the door be closing state</td>
<td>OVERLOAD</td>
<td>showing overload states</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>showing light states</td>
<td>SELFHOLD</td>
<td>showing selfhold states</td>
</tr>
<tr>
<td>DNOFF</td>
<td>elevator moving down off</td>
<td>SENSOR</td>
<td>showing sensor states</td>
</tr>
<tr>
<td>DN NON</td>
<td>elevator moving down on</td>
<td>UOFF</td>
<td>elevator moving up off</td>
</tr>
<tr>
<td>DOOR</td>
<td>showing door states</td>
<td>UON</td>
<td>elevator moving up on</td>
</tr>
<tr>
<td>DOWN</td>
<td>showing down states</td>
<td>UP</td>
<td>showing up states</td>
</tr>
<tr>
<td>F1~F8</td>
<td>floor’s sensor states</td>
<td>WAIT</td>
<td>showing wait states</td>
</tr>
<tr>
<td>FLOOR</td>
<td>showing all floors be changed states</td>
<td>WTEND</td>
<td>timer off</td>
</tr>
</tbody>
</table>

Fig. 12. A snapshot of opening the door of the elevator.

Fig. 13. The elevator on Up state and moving up.

Fig. 14. The elevator arrival at the top floor.
Table A2. Interpretation of events/actions in the elevator statecharts.

<table>
<thead>
<tr>
<th>Event/action</th>
<th>Interpretation</th>
<th>Event/action</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrive</td>
<td>car arrived one floor</td>
<td>phd2–phd8</td>
<td>hall call down buttons be pressed</td>
</tr>
<tr>
<td>c1on–c8on</td>
<td>car call buttons pressed</td>
<td>phu1–phu7</td>
<td>hall call up buttons be pressed</td>
</tr>
<tr>
<td>closeover</td>
<td>door closed completely</td>
<td>po[in(WAIT)]</td>
<td>the open button be pressed when the state staying in WAIT</td>
</tr>
<tr>
<td>down</td>
<td>car moving down</td>
<td>pressclose</td>
<td>the close button be pressed</td>
</tr>
<tr>
<td>en(BPOFF)</td>
<td>entering BPOFF state</td>
<td>pressopen</td>
<td>the open button be pressed</td>
</tr>
<tr>
<td>en(CLOFF)</td>
<td>entering CLOFF state</td>
<td>r1on</td>
<td>alarm relay on</td>
</tr>
<tr>
<td>en(DNOFF)</td>
<td>entering DNOFF state</td>
<td>r1off</td>
<td>alarm relay off</td>
</tr>
<tr>
<td>en(OPOFF)</td>
<td>entering OPOFF state</td>
<td>rd1</td>
<td>opening relay on</td>
</tr>
<tr>
<td>en(UOFF)</td>
<td>entering UOFF state</td>
<td>rd2</td>
<td>closing relay on</td>
</tr>
<tr>
<td>f1–f8</td>
<td>floor light action</td>
<td>r2–r7</td>
<td>floor’s sensors</td>
</tr>
<tr>
<td>hd2on–hd8on</td>
<td>hall call down button’s light on</td>
<td>rst</td>
<td>beep be reset</td>
</tr>
<tr>
<td>hu1on–hu7on</td>
<td>hall call up button’s light on</td>
<td>timeout</td>
<td>waiting time over</td>
</tr>
<tr>
<td>open</td>
<td>door opening</td>
<td>up</td>
<td>car moving up</td>
</tr>
<tr>
<td>openover</td>
<td>door opened completely</td>
<td>α</td>
<td>opening action</td>
</tr>
<tr>
<td>overload</td>
<td>car overloading</td>
<td>β</td>
<td>closing action</td>
</tr>
<tr>
<td>pcl1–pcl8</td>
<td>Car call buttons be pressed</td>
<td>γ</td>
<td>self holding action</td>
</tr>
<tr>
<td>pc[in(WAIT)]</td>
<td>the close button be pressed when the state staying in WAIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ACKNOWLEDGMENT

This research was supported in part by the grant NSC90-2213-E-011-020 and NSC90-2212-E-014-023 by the National Science Council, Taiwan, R.O.C.

REFERENCES


Yi-Sheng Huang received the B.S. degree in Automatic Control Engineering from Feng Chia University, Taiwan, in 1989, the M.S. degree in Electronic Engineering from Chung Yuan Christian University, Taiwan, in 1991, and the Ph.D. degree in Electrical Engineering from National Taiwan University of Science and Technology, Taiwan, in 2001. He is presently an Associate Professor in the Department of Aeronautical Engineering at National Defense University (Chung Cheng Institute of Technology) in Taiwan. His research interests include discrete event systems, Petri nets, embedded systems and automation.

Sheng-Luen Chung received his B.S. degree in electronic engineering department from the National Chiao-Tung University, Taiwan in 1985, and the M.S.E. and Ph.D. degrees from the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, in 1990 and 1992, respectively. Since 1992, he has been with the Electrical Engineering Department of the National Taiwan University of Science and Technology, Taiwan, where he is now an Associate Professor. As a pioneer in Taiwan semiconductor manufacturing automation industry, he has been an active consultant working with MITRI, Consilium, Applied Material, IBM, and HP in several advanced semiconductor fabs and TFT/LCD fabs in Taiwan. His current research interests include system identification, embedded system design and applications of supervisory control. He is granted Research Award by National Science Council from 1994 to 2001. He was a recipient of IEEE 1994 George E. Axelby Outstanding Paper Award from the Control System Society of the IEEE in 1994 for a paper co-authored with Stephane Lafortune and Feng Lin.

MuDer Jeng received the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1992. Since August 1992, Dr. Jeng has been with National Taiwan Ocean University, Keelung, Taiwan, where he is currently a full Professor at the Department of Electrical Engineering. His current research interests include Petri nets, discrete event systems, computer integrated manufacturing, semiconductor factory automation, embedded systems. Dr. Jeng is the author/co-author of over 120 book chapters, journal papers, and conference papers.

Dr. Jeng received the Franklin V. Taylor Outstanding Paper Award from the IEEE Systems, Man and Cybernetics Society in 1993. He was granted the Research Award by the National Science Council of Taiwan annually from 1994 to 2000. He is an Associate Editor for IEEE Transactions on Systems, Man, and Cybernetics-Part A, IEEE Transactions on Robotics and Automation, IEEE Transactions on Robotics, and serves on the Editorial Board of International Journal of Computer Integrated Manufacturing. He has been a Guest Editor for eight leading journals. Dr. Jeng is the Chair of the Technical Committee on Discrete Event Systems of the IEEE SMC Society, and the Founding Chair of the Technical Committee on Semiconductor Factory Automation of the IEEE Robotics and Automation Society. He served as the Exhibitions Chair of 2003 IEEE International Conference on Robotics and Automation, and the Special Sessions Chair of 2004 IEEE International Conference on Networking, Sensing, and Control. He serves as a Program Co-Chair of 2005 IEEE International Conference on Networking, Sensing, and Control, and the Organization Committee Chair (tentative) of 2006 IEEE International Conference on Systems, Man, and Cybernetics. Dr. Jeng is a Senior Member of IEEE.