Modeling Embedded Applications: An Orderly Simplification of Finite State Automata Description

Eduardo Daniel Cohen, Esteban Volentini and Pablo Gruer
Laboratorio de Microprocesadores, FACET
Universidad Nacional de Tucumán
S. M. de Tucumán, Argentina
dcohen@herrera.unt.edu.ar, evolentini@equiser.com.ar, jpgruer@gmail.com

***Abstract*— This work presents an orderly path for the description and simplification of finite state automata (FSA) representation of embedded systems. This approach takes into account some typical features of this kind of systems, i.e. in a given state of a FSA, usually only a few among a large set of system inputs are taken into account. In addition to this, the system presents identical reaction to many inputs in several states. These and other considerations allow the proposal of a modeling strategy that belongs to the "divide and conquer" paradigm.**

Keywords—embedded systems; finite state automata; modeling methods.

I. INTRODUCTION

In this work, we present some methodological proposals concerning the adoption of an orderly description of finite state machines (FSM) as a formal modeling language for embedded applications. FSM are well suited for modeling and designing embedded applications and are valuable in both hardware and software implementations [1] [2] [3]. Nevertheless, as the complexity of applications grows, classical FSM approach suffers from a lack of hierarchical structure, which produces a combinatorial explosion of the number of states and transitions.

The better a system is described, the more likely it is that a good implementation will emerge [4] [5]. A good FSM description should express with precision the required level of detail to understand the intended system behavior. It is common practice to begin with a global description of the system behavior and then to add details during a refinement process consisting in the addition of states and transitions. Hence, the initially ordered automaton becomes more and more complex, as well as less and less readable and maintainable. Furthermore, this procedure is error-prone, as all the consequences of adding new inputs and/or states are frequently not fully taken into account. Therefore a methodical approach to overcome this limitation becomes mandatory during the modeling and design stages.

An approach to overcome the above-described drawback consists in the application of Statecharts [6], which requires specific professional working environments or toolboxes such as https://www.itemis.com/en/yakindu/statechart-tools/. On the other hand, as proposed in this work, it is possible apply practical rules such as representing many states and transitions as a single activity, combined with a mechanism to filter and/or aggregate input signals.

The diversity of modeled systems makes it difficult to define methods based on a systematic application of a reasonable number of rules. Rather than a rigid prescription of steps and rules, in this work we propose to adopt a flexible approach based on the identification of some features that could characterize many systems to be modeled: (1) often only a few among a set of numerous inputs are taken into account at a given system state and (2) multiple different combinations of inputs usually trigger the same reaction. We also consider the distinction between *bounded stay* states (bounded states for short), in which the automaton remains only for a limited lapse of time, usually controlled by a timer, and *unbounded stay* states (unbounded states for short), where the system stays most of the time waiting for inputs to occur. Usually these unbounded states implement the main functional operations of the system and they can be preempted, generally by user-triggered activities such as reconfiguration.

This work proposes a modeling approach that belongs to the divide and conquer paradigm and is based in the following set of assumptions:

- A given system may assume a high number of states. Many of them are bounded states and therefore the system spends a very low proportion of its execution time on them. On the other hand, unbounded states usually represent the system's behavior in normal conditions that include most of the user's functional specifications, e.g. an alarm system that is in a disarmed state. It is important to incorporate description tools adapted to each one of the situations mentioned above.
- The number of system inputs may be high but most of the states are sensitive only to a reduced subset of input signals. As an example, consider an FSM that controls an elevator: if a state represents a floor, next state depends only in neighboring floor sensors, and only these inputs are relevant for transitions to a new state.
- Usually, there exist many different inputs to which the system produces the same output from many of its internal states. As an example let us consider a system where the input of any critical sensor has to trigger an alarm, no matter in which state the system is.

It would be desirable to be able to take advantage of the characteristics of the assumptions described above.

To illustrate how these features could be exploited to build a simpler FSA model of reactive systems, the case study of a home alarm system will be briefly described.

II. CASE STUDY: A HOME ALARM SYSTEM

A. General Description

In a high level of abstraction, the behavior of a home alarm system may be described by a FSM with only three states: *INACTIVE, DISARMED and ARMED.*

B. Refining the Model

The *ARMED* state can be refined by adding new states that describe different armed behaviors.

- *ARMED PRESENT*: allows the presence of people inside the house, some sensors are inactive but not all.
- *ARMED ABSENT*: all sensors are active.
- *ARMED PRESENT FORCED*: similar to "armed present" but faulty sensors are deactivated.
- *ARMED ABSENT FORCED*: similar to "armed absent" but faulty sensors are deactivated.
- *ARMED TEMPORARY*: intermediate state that allows the transition between PRESENT and ABSENT states. Some sensors are activated after a predefined time interval so that the user may leave the house.

Likewise, the *DISARMED* state needs to be refined by adding a new state *DISARMED TEMPORARY*: some sensors are temporarily disarmed to allow entering the premises.

It should be considered that to perform a transition from an *ARMED* state to a *DISARMED* state, a password consisting of five numerical keys followed by a special "Enter" key has to be typed. Thus, a refined state diagram would have to add six intermediate states. It becomes clear that successive refinement operations would lead to a rapid growth of the number of states and transitions, resulting in an unreadable state diagram.

In many cases, similar to the one described above, the system behavior has to include particular sequences of intermediate states. Each of these sequences will be called an "activity". This kind of activity may induce purely sequential thinking, which is error-prone. A common error of sequential thinking results from not taking into account other possible parallel events that could disrupt the main sequence (e.g. entering a password). In this case the first digit of the password would lock the system in waiting only for the following numerical keys. Taking into account all possible pre-empting events results in even a more complex state diagram. The following sections introduce the definitions and tools necessary to simplify the FSA that represents a given system.

III. SIMPLIFYING TOOLS

A. Initial Definitions

1) Reaction:

Let a system be in state S_i, a reaction to a system combination of inputs may consist on at least one of the following actions: (1) there is a transition from S_i to S_j and (2) there are new outputs from the system.

2) Stimuli:

A stimulus is a combination of input signal values that causes a system reaction at a given state of the system, i.e. there is at least one system state S_i such that if S_i is active and the stimulus takes place, there will be a reaction. It is important to note that, depending on the state assumed by the system, the same combination of inputs might not get any reaction. Likewise, the same stimulus may cause different reactions in different states of the system.

3) Activities:

Let the system be in an initial state S_i, an activity is defined as the complete sequence of stimuli, reactions and bounded states that allows the system to transition to a new state S_j.

We note that usually S_i and S_j are unbounded states, although this is not a necessary condition. On the other hand, states that belong to an activity have to be bounded states.

As an example, consider the activity that takes place in the system described in the case-study to transition from an *ARMED* state to a *DISARMED* one: the user has to input a five number password and an "enter" key in order to disarm the system. The user has a limited amount of time to perform this operation; otherwise a time-out will abort it.

According to the number of stimuli, it could be useful to classify activities into:

- *Atomic activities* triggered by a single stimulus. Other stimuli cannot interfere in any way. As an example, let us consider the case study again. Suppose there is one key "ARM": pressing it generates an atomic activity that brings the system to the ARMED state. These activities are fast when compared to the time elapsed between any two stimuli.
- *Non-atomic activities* composed by a succession of stimuli and their corresponding reactions. This kind of activities could be *exclusive* or not. *Exclusive* activities should be considered as atomic activities, since no interference is possible. On the other hand, *non-exclusive* activities could be pre-empted by one or more activities before they get to the final state of the sequence. As an example, consider the case study previously described: if a user types a five-digit password to disarm the system and a sensor is activated in parallel, the alarm should go off and the input activity aborted.

We point out a functional difference between atomic activities and exclusive non-atomic activities: an activity composed by only one stimulus is atomic "per se" whereas a non-atomic activity has to be defined as exclusive or not by the designer, in response to system requirements and specifications.

B. Sequences of bounded states viewed as activities

Consider the case of a system transition between two states, S_0 and S_n, which goes through n-1 bounded states, as illustrated by Fig 1.

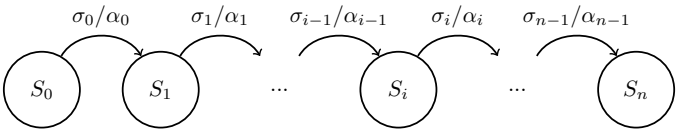

Fig. 1. A sequence of bounded states.

The sequence is triggered by stimulus σ_0 and the system transitions through states S_1; S_2; …; S_{n-1} up to state S_n producing a sequence of actions α_0; α_1; …; α_{n-1} in response to particular stimuli σ_o; …; σ_{n-1} . In the simplest case, when this sequence is atomic, i.e. it cannot be interrupted by any other stimulus, it can be defined as an activity and the state diagram can be simplified as in Fig. 2, by using the statechart style of description [4]. Strictly speaking the activity includes transitions σ_o/α_0, $\sigma_{n-1/}\alpha_{n-1}$ and super state A.

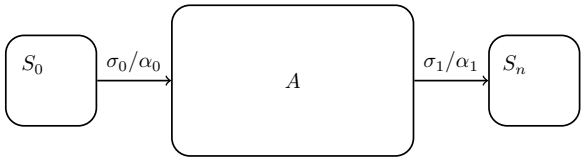

Fig. 2. An atomic activity represented in a statechart.

In many cases the activity is required to be non-atomic. The sequence from state S_i to state S_k may be preempted by stimulus σ_p. This can be modeled by super state S_p (a statechart OR state). The statechart language offers the possibility to restore the state that was active at occurrence of the preempting stimulus, by using a history connector, as illustrated by Fig. 3.

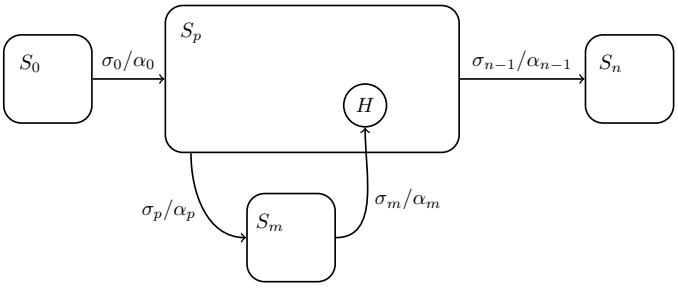

Fig. 3. Preemting stimulus in the statechart diagram.

A simpler approach is shown in Fig. 4: a thick arrow, called activity "A", replaces the sequence of Fig. 1. Stimulus σ_p may be considered as a starting stimulus of a different activity; the problems of interaction among activities are left to a subsequent and separate step of the divide and conquer technique. Thus, it will not interfere with the simplicity of the high level description being introduced.

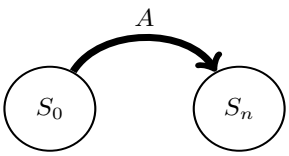

Fig. 4. Activity A hides the complexity of interaction among activities

A tool to analyze interactions among activities consists of an interaction table that will be presented in a follow-up article. Statecharts may be also used in this new step to further refine the description of activities interaction.

C. Differential perception of stimuli

Activities are reactions of the system to one or more stimuli and result in output signals (external reaction) and/or changes of state (internal reaction). Different states may allow different system reactions to the same stimulus.

1) Stimuli perception: when the system transitions from state S_i to state S_j, the set of awaited stimuli changes. Some stimuli in S_i do not affect S_j , hence they are not stimuli of S_j. It is said that in S_j the system has no "perception" of some stimuli that affect S_i.

2) External reaction to stimuli: the system outputs due to the same stimulus may be different depending on the system state.

3) Internal reaction to stimuli: from a state, S_i, the system may transition to different states depending on the next stimulus occurrence.

D. Unifying states

Let a subset of states $\mathcal{S}$ of a system be such that: (1) identical perceived stimulus generates the same external reaction on every state of $\mathcal{S}$, and (2) any transition caused by identical stimulus to different states of $\mathcal{S}$ may bring the system to any state belonging to $\mathcal{S}$ or to a unique state Q out of $\mathcal{S}$. Then $\mathcal{S}$ can be unified in a single state U, yielding a simplified state set. Note that states in $\mathcal{S}$ may not react to the same stimulus but if they do, the external reaction must be equal for each one of them. Different states may only transition to the same state out of $\mathcal{S}$ under identical stimulus.

In order to show that the previous statement is true, it suffices to take into account that transitions among states may be produced only in the following cases:

1) Self-loops on U that change the perception of stimuli, are equivalent to transitions between states belonging to $\mathcal{S}$.

2) Outgoing transitions from states in $\mathcal{S}$ to states not in $\mathcal{S}$, caused by the same stimulus, lead to a unique state Q with the same external reaction by definition. Therefore they can be replaced by a single transition from U to Q.

3) Transitions, caused by the same stimulous, entering to any state of $\mathcal{S}$ from a state W not in $\mathcal{S}$, can be replaced by a unique transition from W to U, since subsequent reactions will be identical following the two previous points.

Therefore, stimuli that cause the same reaction from any state in $\mathcal{S}$ will give the same reaction from state U and the behavior of the system is not modified.

We remark that our approach to simplifying FSMs differ from those related to non-completely specified automata models [7]: modifying system perception ensures that any input

stimuli not specified for state S_i will never be present when this state is active.

Although minimization algorithms [8] are based on the formal definition of state equivalence relations, the simplification showed above is not: states in $\mathcal{S}$ may be unified in U even if they do not react to the same input sequence. Furthermore, one state S_i in $\mathcal{S}$ may react to a given stimulus whereas other State S_j in $\mathcal{S}$ may not "perceive" it.

It follows that the simpler equivalent state diagram contains not only states and activities, but also different perceptions of stimuli.

E. *Modifying System Perception*

To implement the modification of the system perception as a consequence of transitions in the state diagram, a simple stimuli-masking operation is carried out. Each activity leading to a new state of the system must perform this operation when necessary.

Consider, as an example, the case of up to 64 binary stimuli, ordered as a 8x8 matrix Σ. Each position *(i, j)* represents a stimulus. $\Sigma_{i,j} = 1$ means that stimulus *(i, j)* is present, whereas $\Sigma_{i,j} = 0$ means it is not. Let the matrix PM^k be the perception mask of activity k such that $PM^k_{i,j} = 0$ if stimulus (i ,j) is masked out for activity k and $PM^k_{i,j} = 1$ if if it is not. The perceived stimuli matrix for activity k, PS^k, is given by:

$$PS^k = \Sigma \wedge PM^k \tag{1}$$

The input/output subsystem updates matrices Σ and PS^k [9]. The last executed activity updates matrix PM. Perception matrices may be constant and fixed beforehand for each activity or they may vary, depending on the particular system that is being modeled.

IV. EXAMPLE: CASE STUDY REVISITED

To illustrate the case of a state diagram simplified by means of activities, consider the alarm system introduced in section 2. Fig. 5 shows a high level state diagram that we will describe now.

Initially, all sensors are disabled and the system may react only to input keys, as the only allowed activity consists in decoding a special password. The present state is INACTIVE.

Once the special password is entered, the perception of permanent sensors (fire and anti-tampering) is enabled. A stimulus coming from any perceived sensor would trigger the alarm. The system stays in state DISARMED.

The arm-present activity adds the perception of external sensors, allowing for the movement of people inside the house. Perceived stimuli would trigger the alarm. The system gets to ARMED PRESENT state.

The reader may easily check that the ARMED PRESENT FORCED state is similar to the ARMED PRESENT state, except that the transition activity does not enable the perception of sensors, which, due to malfunctioning or other reasons, remain active. A similar case occurs with other ways to arm the system, i.e the ARMED ABSENT state.

Every different way of arming the system would require a new different armed state (i.e. ARMED PRESENT, ARMED ABSENT, …). The fact that each arming activity modifies the system perception allows unifying them in a unique state ARMED, as shown in Fig. 5. Since not all of the systems arming activities are represented in Fig 5 for simplicity's sake, every activity and the way it affects perceptions of the system is presented in Table I.

The system admits two different passwords: one to transition between INACTIVE and DISARMED states, and another one for switching between DISARMED and ARMED states. To define which of the many arming activities will take place, the user has to type a function key before the password.

In order to disarm the system, the user needs to get into the house through a path of delayed-sensors and enter a password. The first activation of a delayed-sensor will enable a time-out action (TIMER2) that consists of masking out the whole set of these sensors for a short time. If TIMER2 expires before typing the right password, the system stays in the ARMED state and the alarm is triggered.

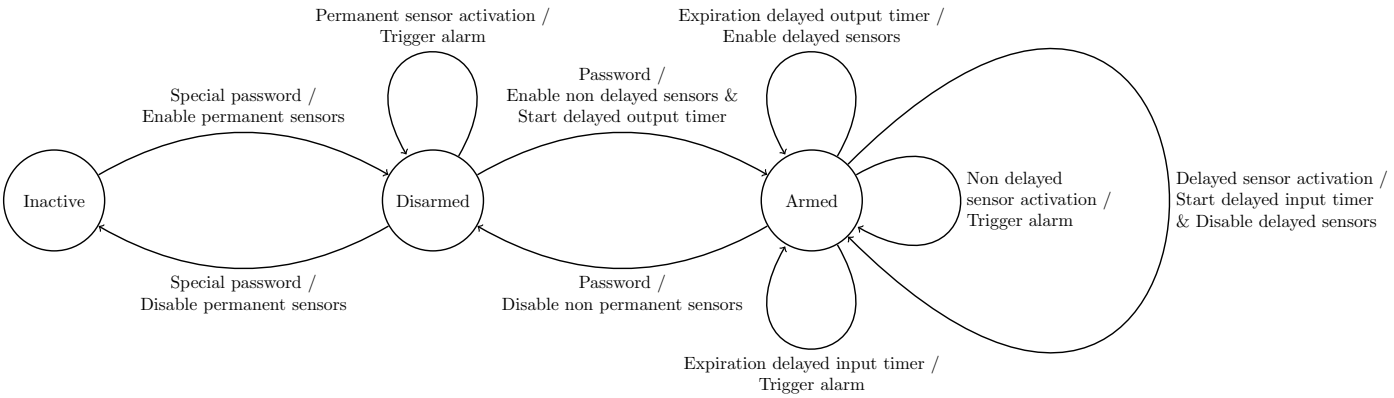

Fig. 5. High-level state diagram of the alarm system.

TABLE I.

Activity	Perception Changes		
	Initial State	***Final State***	***Effect on Perception***
Special password	1	2	Enable permanent sensors
	2	1	Disable all sensors
Password	2	3	Enable non delayed sensors
	3	2	Disable non permanent sensors
Expiration delayed output timer	3	3	Enable delayed sensors
Delayed Sensor Activation	3	3	Disable delayed sensors

[a.] Effect of Activities into the system perception

A future version of the system may require arming a particular zone Z of the house whereas the rest remain unarmed. It is straightforward to define a new perception matrix that would allow only permanent sensors and Z sensors to be active. A new activity ARM ZONE would add this new perception matrix when bringing the system to the ARMED state. This example shows that the simplifying tools provide an important contribution to the maintainability of the system.

V. CONCLUSION

Activities are the main tool to build a simpler state representation of FSMs mainly due to the following reasons: (1) they change the perception of a system, thus allowing many equivalent states to be unified, (2) they contain bounded states, providing a way to further reduce the number of states, and (3) they provide an upper level of abstraction. Activities interactions do not need to be considered. This can be left for a following step. Statecharts and many tools from real time systems [10] may be used at this stage.

The example revisited in section IV shows that the tools introduced in this article contribute to better system maintainability. We believe that there are many systems for which these tools could bring the same advantage.

Complexity and memory requirements of this simple representation are transferred to activities and perception masks. This new approach may be considered as a way to apply the divide and conquer strategy, yielding a simpler higher-level hierarchical representation.

The activities approach allows the definition of software threads for their implementation in a programmed logic environment, such as a microcontroller, for an embedded system.

Bounded states, belonging to activities, usually differ in functionality from unbounded ones. The former are often related to exceptional cases, usually not present in the early phases of user specification. They may represent unusual and complex conditions, which could affect important features of the system such as security, reliability and disponibility. Since activities may be carried out as software threads, their design can be carried out with well-known programming languages (C, for instance).

REFERENCES

[1] F. Balarin et al., Hardware-Software Co-Design of Embedded Systems: The Polis Approach, Kluwer Academic Publishers, ISBN 0-7923-9936-6, 1997.

[2] Valvano J, Real-Time Interfacing to Arm® Cortex(TM)-M Microcontrollers, 5th Edition, Createspace, SC, USA, 2015.

[3] White E., Making Embedded Systems, 2nd Edition, CA, USA, 2012.

[4] Maier M. & Rechtin E., The Art of System Architecting, 2nd edition, CRC Press, Inc. Boca Raton, FL, USA, 2000.

[5] Keating, M., The Simple Art of SoC Design, Springer Science+Business Media, New York, USA, 2011.

[6] Harel D., "Statecharts: a visual formalism for complex systems", Journal Science of Computer Programming, Vol. 8, Issue 3, Elsevier, Philadelphia, USA, 1987, pp. 231-274.

[7] Higuchi H. and Matsunaga Y., "A fast state reduction algorithm for incompletely specified fihnite state machines", Proc. Design Automation Conference, ACM Press, New York, USA 1996. pp. 463-466.

[8] Hopcroft, J. E., "An n log n algorithm for minimizing the states in a finite automaton" in Z. Kohavi, editor, *The Theory of Machines and Computations*, pp. 189–196. Academic Press, New York, USA, 1971.

[9] Cohen, E. D., Volentini, E. & Giori, M., "Múltiples Entradas y Salidas en Sistemas Embebidos", Memoria – Investigaciones en Ingeniería, Nº 13, Montevideo, Uruguay, 2015, pp. 49-62.

[10] Laplante, P. A. & Ovaska, S. J., Real-Time Systems Design and Analysis, 4th edition, IEEE Press, John Willey & Sons, New Jersey, USA,2012.