

# Behavioral Specification of Sensible Agents

**A. Suraj, S. Ramaswamy, K.S. Barber**

The Laboratory for Intelligent Processes and Systems  
Electrical and Computer Engineering  
The University of Texas at Austin  
Austin, TX 78712  
<http://www-lips.ece.utexas.edu>  
[barber@mail.utexas.edu](mailto:barber@mail.utexas.edu)  
phone: (512) 471-6152  
fax: (512) 471-3652

## Abstract

In this paper, Extended Statecharts are used as a comprehensive modeling mechanism for the behavior modeling of Sensible Agents. Extended Statecharts allow for the explicit representation of declarable problem-specific agent soft failures, thereby allowing for failure related information to be incorporated within the high level system design. An example of an air defense system is used to illustrate the modeling capabilities of Extended Statecharts.

## Accepted to

Intelligent Systems: A Semiotic Perspective

## Workshop: Semiotic Modeling for Sensible Agents

Workshop organizers:

**K. Suzanne Barber**

Electrical and Computer Engineering  
The University of Texas at Austin  
24th and Speedway, ENS 240  
Austin, TX 78712  
phone: (512) 471-6152  
fax: (512) 471-3652

**Srini Ramaswamy**

800 Wheatley Street  
School of Computer and Applied Sciences  
Georgia Southwestern State University  
Americus, GA 31709  
Phone: (912) 931-2100  
[srini@gswrs6k1.gsw.peachnet.edu](mailto:srini@gswrs6k1.gsw.peachnet.edu)

**October 23-25, 1996**

# Behavioral Specification of Sensible Agents\*

A. Suraj, S. Ramaswamy\*\* and K. S. Barber

*Laboratory for Intelligent Processes and Systems, The Department of Electrical and Computer Engineering,  
The University of Texas at Austin, Austin, TX 78712-1084*

**Abstract\*\*\*** - In this paper, Extended Statecharts are used as a comprehensive modeling mechanism for the behavior modeling of Sensible Agents. Extended Statecharts allow for the explicit representation of declarable problem-specific agent soft failures, thereby allowing for failure related information to be incorporated within the high level system design. An example of an air defense system is used to illustrate the modeling capabilities of Extended Statecharts.

## I. INTRODUCTION

It is well known that it has been difficult to find a widely acceptable definition of the word *agent* amongst researchers. However, we must be able to define some of the innate attributes that characterize such an agent in a system. Therefore, instead of trying to define what a Sensible Agent (SA) is in a software system, we will define a SA to be a system component that appropriately answers the question "R U A SA?" where, R represents the ability to Represent, U the ability to Understand, and A represents the ability to Act. In greater detail, these attributes mean the following: (i) **Represent**: Represent both behavioral and structural characteristics of the system agents (ii) **Understand**: Understand both local and system goals, prioritize these goals according to knowledge about the other system agents, the environment and itself and drive the agent behavior based on this knowledge, (iii) **Act**: Perform either deliberative or reactive actions with respect to events generated by itself, the environment and other system agents.

Such an agent is able to readily perceive, appreciate, understand and demonstrate cognizant behavior. For this reason, the SA must completely capture "*domain specific intelligence*" through its representation. In this respect, identification and handling of failures is a very important issue. Thus, a modeling mechanism must

be flexible and sufficiently rich to allow a designer to represent in detail the internal structure of SAs in the system. Moreover, these representations must be similar to what an user might observe in the real world operation of the system in question. For this reason, Extended Statecharts (ESC) exploit the XOR state representation in Statecharts to integrate failure information in a high level system representation. Combined with a powerful event and transition structure this modeling technique allows the development of detailed high level specification of agent behaviors. Goals in ESC are defined by a directed path from a start state to an end state (exit-safe state). Each such unique path or subset represents the possible goals and sub-goals that are attainable by the agent.

Several software modeling tools and methodologies have been developed for system specification and analysis. The reader is referred to [1] for a detailed overview of such tools and methodologies. Extended Statecharts (ESC) are dynamic models of agent behavior. The ESC representation provides a way to explicitly integrate failure information in the agent design process. It allows data values to be associated with events. State Machines for Object-Oriented Concurrent Hierarchical Engineering Specifications (SMOOCHES) [2] has been developed to derive the system information model and the system behavioral model.

The paper is organized as follows: Extended Statecharts, that further the use of Statecharts for high-level system design representation is presented in Section II. Section III illustrates the use of the ESC based approach in developing an agent-based design of an air defense system. It is to be noted that ESC modeling is to develop a high-level executable behavior of agent based systems. The final section, Section IV, concludes the paper.

## II. EXTENDED STATECHARTS

In this section, Extended Statecharts [1], an extension to Statecharts both with respect to functional and temporal aspects is introduced. ESC is a 4-tuple (S, E, G, T), where, S is a set of states, E is a set of events, G is a set of guards and T is a set of transitions. In this section, the important ESC features are introduced. It is assumed that the reader is familiar with statechart notations [3].

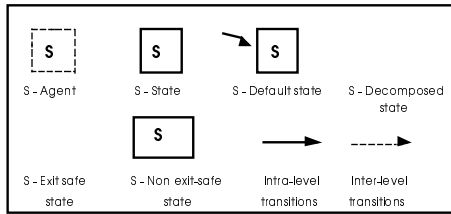
\* This research was supported in part by The Texas Higher Education Coordinating Board Advanced Technology Program (#003656112, #003658452, #003656115), The Office of Naval Research, and The Applied Research Laboratories (Austin, TX) and The Naval Surface Warfare Center in Dahlgren, VA

\*\* Dr. Ramaswamy is with the School of Computer and Applied Sciences, Georgia Southwestern State University, Americus, GA 31709. Between Aug. 94 and June 95 and subsequently in Summer 1996, Dr. Ramaswamy has been a visiting research fellow at the Laboratory for Intelligent Processes and Systems at the University of Texas at Austin.

### A. Basic ESC Notations

The set of states  $S$  is defined as,  $S = \bigcup_{i=1}^n S^i$

where,  $n$  is the number of levels in the model hierarchy. Exit-safe states represent states wherein a substate is in a stable state to process the transition out of a parent state [2]. That is, events causing a transition between higher level (parent) states are handled only if the substate of the parents' state is an exit-safe substate. This extension explicitly allows the developer to specify certain non-interruptible critical operations. Non exit-safe states are states that are not exit-safe and hence do not allow for higher level state transitions.



action(s) can be a simple event or a set of events. This action event(s) is optional. If an action event(s) is specified, it is immediately generated and is regarded as an internal event that may cause transitions in other agents. The input conditions and the event causing the transition are incorporated in the guard itself. This removes the need for having separate queues for events and input conditions during implementation.

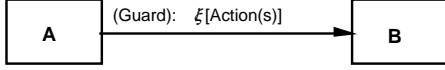


Figure 3: Transition labels in ESC

In Figure 2, the transition from **I** to **J** is caused by a guard  $check(e7)$  (i.e.  $e7$ ).  $check(e7)$  is represented as just  $e7$  for simplicity. This guard does not produce any action. But, the transition from state **I** to state **K** is caused by a guard  $\neg e1$  followed by an action  $e4$ . This means event  $e4$  is generated immediately. The transition from state **B** to state **K** produces two events  $e5$  and  $e6$  which are represented in the set  $\{e5, e6\}$ . Guards and the evaluation of guards to make transitions are discussed in the next section.

#### D. Guards on Transitions

$E$ , the set of all events in the system is defined as  $E = E_{BE} \cup E_{CE}$  where,  $E_{BE}$  is the set of basic events and  $E_{CE}$  is the set of composite events and  $E_{BE} \cap E_{CE} = \phi$ .

Guards are formed by the combination of four basic functions and different kinds of operators acting on events/states. These operators and functions may be combined to form complex conditions. Henceforth, complex conditions will be called as "complex events". The *functions* include:

- (1)  $in(x)$  which denotes that the system is in state  $x$ .  $in(x:y)$  denotes that the system is in state  $y$  where,  $y$  is the immediate substate of  $x$ .
- (2)  $\tau_m(e, n)$  which denotes a time-out event that is generated if event  $e$  happened  $n$  steps earlier ( $e \in E$  and  $n \in N - \{0\}$ , where  $N$  is the set of Natural numbers).
- (3)  $check(e)$  which returns true if event  $e$  occurs and false if it does not occur.  $check(e)$  is denoted as just  $e$  for simplicity throughout this paper. In Figure 2, the transition from state **I** to **K** is caused by  $\neg check(e1)/e4$  which is represented as  $\neg e1/e4$ .
- (4)  $val(e)$  which returns a data value associated with event  $e$ .

The *operators* used in the construction of guards are classified into three categories:

- (1) *Arithmetic Operators* ( $+, -, <, >, \leq, \geq, =, \neq$ ): These operators act only on events that have associated data values. For example, consider two events  $e_1$  and  $e_2$  that represent temperature values

( $e_1, e_2 \in E$ ); then, the guard  $g \in G$  may be  $g: (val(e_1) + val(e_2)) > 20$ . This means that the guard evaluates to *true* when the total value of the temperatures (i.e.  $val(e_1) + val(e_2)$ ) becomes greater than 20. Similarly, conditions such as 'equal' or 'less than' can be checked.

(2) *Logical Operators* ( $\wedge, \vee, \neg$ ): The logical operators act on both events and states. If  $g \in G$  and  $g = e_1 \vee e_2$  (or  $e_1 \wedge e_2$ ) where  $e_1, e_2 \in E$ ,  $g$  becomes true when either event  $e_1$  or  $e_2$  (or  $e_1$  and  $e_2$ ) occur(s). In the same manner,  $g = \neg e_1$  implies that the guard is true when  $e_1$  does not occur. Some of the rules for the logical operators are summarized as follows: (a)  $g \in G \Rightarrow \neg g \in G$  (b)  $e \in E \Rightarrow e, \neg e \in G$  (c)  $\tau_m(e, n) \in G$  and  $\neg \tau_m(e, n) \in G$  (d) if  $e_1, e_2 \in G$  then  $e_1 \vee e_2, e_1 \wedge e_2 \in G$ .

(3) *Temporal Operators* ( $\square, \diamond, O, U$ ): Within guard functions, the temporal operators act only on states.  $\square$  stands for *always*,  $\diamond$  stands for *eventually*,  $O$  stands for *next* and  $U$  stands for *until*. If  $s_1 \in S$ ,  $g: \square s_1$  (or  $g: \square in(s_1)$ ) is true if and only if  $in(s_1)$  is true at the present step and continues to be true in all the steps from now on.  $g: \diamond s_1$  (or  $g: \diamond in(s_1)$ ) is true if and only if  $in(s_1)$  is true at least once in any step in the future.  $g: O s_1$  (or  $g: O in(s_1)$ ) is true if and only if  $in(s_1)$  is true in the next step (i.e. if the system reaches state  $s_1$  in the next step). Finally, a guard  $g: s_1 U s_2$  (or  $g: in(s_1) U in(s_2)$ ) is true if and only if  $in(s_1)$  is true until  $in(s_2)$  (i.e. the system continues to be in state  $s_1$  till  $s_2$  is reached). The guard becomes false once  $s_2$  is reached.

#### E. Failure Modeling

Potential failures/errors that are identified through sensory inputs are classified into *soft* or *hard* failures. Error classification of previously known errors (and thus incorporated within the ESC design) occurs during the system modeling stage. So, when a system soft failure occurs, it might recover from the error and resume normal operations. System hard failures correspond to fatal failures from which the system cannot recover by normal mechanisms and needs external intervention. The system reaches a stop state when one of these errors occurs. Errors due to hard failures are not explicitly modeled by ESC.

The most important characteristic of ESC is their ability to model failure behavior as part of the agent design. Therefore, the use of ESCs may lead to improved quality. ESC designs integrate failure operations by exploiting the advantages of using XOR configurations

with exit-safe states and the transition extensions proposed earlier in this section.

For example, in Figure 2, the higher level states **A** and **B** are in an XOR configuration where **A** represents the normal agent operation and **B** represents the state associated with failure operations. In the above example, the agent can be in either state **A** or state **B** which is used to represent a normal state and an error state respectively. In such a representation, the intra-level transition from state **A** to state **B** (caused by the guard  $\square$  in  $G$ ) is never considered possible because it is assumed that there does not occur a normal transition (intra-level transition) from a normal state to an error state. However, the intra-level transition from state **A** to **B** in Figure 2 is used to illustrate the notion of exit-safe states.

### III. AN EXAMPLE AIR DEFENSE SYSTEM

A simple air defense system model [4] includes a search radar, a track radar, a command center, and a weapon. In this paper, the tracking radar is emphasized with the command center and the weapon system adding additional context. The tracking operation involved in air defense systems refers to the continuous measurement, estimation and maintenance of data about the kinematic state (position and velocity) of one or more targets. A single target tracking radar (STTR) implies a dedicated radar used to track a single target. Typically, a command issued from a Command Center to the radar gives the general coordinates of the object to be tracked in the form of azimuth (rotation angle), elevation, and range of the desired target. The STTR then slews to the designated angles, performs a local volumetric search, and then detects and acquires the target.

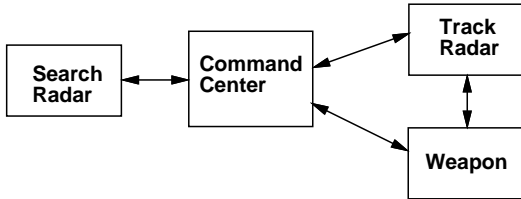


Figure 4: Block Diagram of Air Defense System

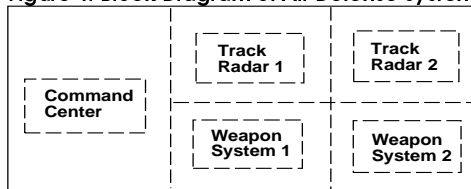


Figure 5: High level agent decomposition

The block diagram in Figure 4 illustrates the relationship between the Command Center, the Radar, and the Weapon system. In this example, a general model of the tracking performed by the radar and weapon agents' is developed. Domain imposed constraints include: (i) the command agent sends out basic initiation commands, and (ii) the command agent is under human control.

Using the decomposition in Figure 5 some agents within the example system are derived. It includes one Command Center and multiple Track Radars and Weapons. This presents the top-level view of the system. Tables 1 and 2 provide the list of events involved in the air defense system model. Figure 6 through Figure 8 illustrate the state diagrams and transitions for the agents. These state diagrams define the behavioral agent model.

Table 1: Basic Events

<b>e1</b>	Assign	<b>e2</b>	RSlewDone
<b>e3</b>	WSlewDone	<b>e4</b>	EngageTarget
<b>e5</b>	AmmoEqZero	<b>e6</b>	AmmoGtZero
<b>e7</b>	Refilled	<b>e8</b>	NewTarget
<b>e9</b>	TargetInfo	<b>e10</b>	LostRange
<b>e11</b>	HaveRange	<b>e12</b>	TargetPos
<b>e13</b>	LostTarget	<b>e14</b>	DoneWtTarget
<b>e15</b>	Pulse	<b>e16</b>	SendPulse
<b>e17</b>	GateOnEvent	<b>e18</b>	GateOffEvent
<b>e19</b>	IntTimeOut	<b>e20</b>	DetectInterf
<b>e21</b>	ControlInterf.	<b>e22</b>	InterfSetLow
<b>e23</b>	Reset	<b>e24</b>	Lock
<b>e25</b>	FiredAtTarget	<b>e26</b>	HaveTarget
<b>e27</b>	SelfDestroyed		

Table 2: Composite Events

<b>c1</b>	$val(intf) > t_1$
<b>c2</b>	$\neg c1 = val(intf) \leq t_1$
<b>c3</b>	$val(intf) < t_2$
<b>c4</b>	$e13 / e23$
<b>c5</b>	$e4 \wedge in(RADAR: RadarOn)$
<b>c6</b>	$val(RADAR: RadarOff. \phi) \leq t$
<b>c7</b>	$val(RADAR: RadarOff. \phi) > t$
<b>c8</b>	$in(RADAR: RadarOff)$
<b>c9</b>	$e13 \wedge e27$
<b>c10</b>	$e26 \wedge e27$

Figure 6 illustrates the substate descriptions of the Radar agent. The Radar agent has substates 'Normal' and 'Error' in the XOR configuration, which represent the normal and failure operations respectively. 'Normal' is the default state of the Radar agent. Exit-safe states are used to define legal transitions between high level states. 'Radar Off' is the default state of 'Normal'. 'Stand by' is the default state of 'Radar Off'. When the radar receives the NewTarget event (event e8, see Table 1), it transitions to the 'Slewing' state from the 'Stand by' state in 'Radar Off'. In the 'Slewing' state the radar motors engage and move the radar to the general location of the target. The event, RSlewDone (event e2 in Table 1), signifies that the radar is in the correct position. Upon this event, the radar enters the 'Radar On' state (Notice that 'Slewing' is an

exit-safe state of the 'Radar Off' state. The event e2 transitions 'Radar Off' state to 'Radar On' state only when the substate 'Slewing' is reached in the 'Radar Off' state). In the 'Radar On' state, the radar simultaneously enters the concurrent default states of 'Acquiring' and 'Radar Sense Idle'. The states 'Radar Sense Idle' and 'Processing Data' show the activity of the signal processing unit of the radar; when a pulse arrives, it is processed by the signal processing unit. When the radar has the range on the target, signified by the HaveRange event (event e11), the radar transitions to the 'Tracking' state. If the radar loses the range on the target, signified by the LostRange event (event e10), the radar returns to the 'Acquiring' state. If the radar has lost the target completely, signified by the composite event c4, it transitions to the 'Target Lost' exit-safe state and produces the event e23. Event e23 then allows the transition from 'Target Lost' to the 'Stand By' state of the 'Radar Off' state.

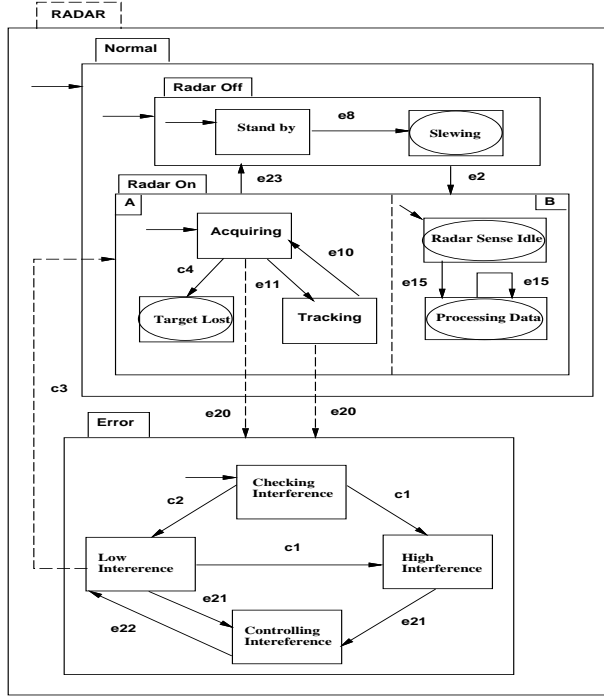


Figure 6: ESC decomposition of the Radar Agent and its substates

The radar agent can fail when it gets an interference from other emitters/radars. Interference is normally measured in wattage. The radar can function normally if the interference is low (soft failure) and fails to perform well if the interference is high (hard failure). In Figure 6, whenever the radar detects any interference (event e20), in the 'Acquiring' and the 'Tracking' states, it transitions to the 'Error' state and defaults to the 'Checking Interference' state (Notice that these transitions are caused by failures and therefore represented as inter-level transitions). If the value of the interference is greater than a certain variable  $t_1$  (event c1, see Table 2), it transitions to the 'High Interference' state. Otherwise, it

transitions to the 'Low Interference' state (event c2). Event ControlInterf (event e21) transitions both the 'High Interference' state and the 'Low Interference' state to the 'Controlling Interference' state. In this state, the radar tries to control the interference by reducing it. After the radar sets the interference to a low value (event e22), the agent transitions from the 'Controlling Interference' state to the 'Low Interference' state. If the radar can perform normally with an interference less than a variable  $t_2$  (event c3), the agent transitions to the 'Acquiring' state of the 'Radar On' state. Otherwise, the agent transitions back to the 'Controlling Interference' state to reduce the interference further (event e21).

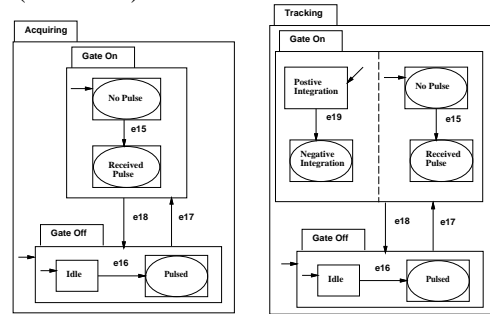


Figure 7: ESC decomposition of Acquiring and Tracking states of the Radar agent

Figure 7 illustrates the substate diagrams of the 'Acquiring' and 'Tracking' states. For both the 'Acquiring' and 'Tracking' states, if a pulse is received during the 'Gate On' substate (i.e. if event e15 occurs), the radar transitions to the corresponding 'Received Pulse' substate. In the 'Acquiring' state, the radar will modify the gate on and gate off time to get a more accurate range on the target. In the 'Tracking' state, the radar uses the information acquired in the 'Positive Integration' and 'Negative Integration' substates to keep the range gate approximately centered on the target. The 'Positive Integration' and 'Negative Integration' substates represent when the radar is integrating the pulse information positively and negatively respectively. To keep the range gate centered on the target, the radar adjusts when the GateOnEvent (event e17) is generated. The 'Gate Off' state has two states representing if an outgoing pulse has been generated or not. The event SendPulse (event e16), signifies the start of the 'Pulsed' substate.

Figure 8 is the state diagram for the Weapon agent. Like the Radar agent, the Weapon agent defaults to a 'Stand By' state of the 'Normal' state. After receiving the Lock event (event e24), it transitions to the 'Slewing' state where the motors are engaged to turn the weapon to the appropriate initial location. When finished slewing, as signified by the WSlewDone event (event e3), the weapon agent then is in the 'Ready' state. While in this state, the weapon will follow the target using the information from the corresponding radar agent. If the EngageTarget event (event e4) occurs and if the Radar

agent is in the 'Radar On' substate (composite event c5), the weapon will move to the 'Guided Flight' state. The weapon is launched in the 'Guided Flight' state. After it fires at the target (see event e25), the weapon agent transitions to the 'Missile Firing Succeeded' state. If the weapon is out of ammunition it will transition to a state where it will need to be reloaded, otherwise it returns to the 'Ready' state. If the target is no longer a concern (i.e. if the target leaves or it has been destroyed), the weapon returns to the 'Stand by' state upon the occurrence of the DoneWtTarget event (event e14) from the 'Ready' state.

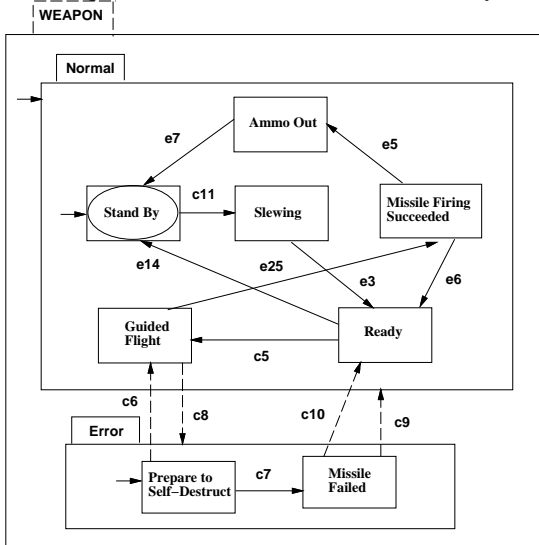


Figure 8: ESC decomposition of the Weapon agent and its substates

The weapon agent relies on the radar agent for illuminating the target. Therefore, if the radar agent transitions to the 'Radar off' state while the weapon agent has launched a missile (i.e. in the 'Guided Flight' state), the weapon agent enters the 'Prepare to Self-Destruct' substate of the 'Error' state (event c8). When 'Prepare to Self-Destruct' state is entered, a timer is started internally in the missile. The timer runs as long as the radar agent is in the 'Radar Off' state. If the timer times out (i.e. if the system stays in the 'Radar Off' state of the radar agent for more than  $t$  seconds), the missile self destructs and enters the 'Missile Failed' state.

The transition from the 'Prepare to Self-Destruct' state to the 'Missile Failed' state occurs when the composite event  $c7 = val(RADAR: RadarOff. \phi) > t$ <sup>1</sup> is true. If the radar comes back ON before the timer times out (i.e. if the radar agent transitions to the 'Radar On' state), the weapon agent transitions to the 'Guided Flight' state (composite event c6), and resumes normal operation. If the target is lost and the missile is self-destructed (event c9), the weapon agent transitions to the 'Stand by'

state of 'Normal' from the 'Missile Failed' state. If the system has the target after the missile self-destructed (event c10), the weapon agent transitions to the 'Ready' state and resumes normal operation. In Figure 8, the states 'Prepare to Self-Destruct' and 'Guided Flight' are at same hierarchical level even though the transition between these states is represented using an inter-level transition. This is because, the states 'Prepare to Self-Destruct' and 'Guided Flight' belong to two different parent states and therefore the transition between these states is erroneous. The same case is true for the transition (event c10) from state 'Missile Failed' to state 'Ready' in Figure 8.

#### IV. CONCLUSIONS

In this paper, ESC have been used as a modeling mechanism for the behavioral specification of SAs. An ESC design of an example air defense system has been presented. The notion of exit-safe states in ESC is shown to be useful for representing reactive systems. The ESC extensions were used to explicitly capture failure information within the high level agent design. The different functions and operators defined on the events were shown to provide a complete event structure.

Current research is focused on combining Petri Net extensions with ESC to derive a nested dual approach for SA modeling where Petri nets are used for representing lower level system details and for studying the issue of agent stability.

#### ACKNOWLEDGMENTS

We acknowledge all our colleagues in this lab for helping us crystallize the notion of a SA. We would like to thank Anuj Goel, Cheryl Martin, Tse-Hsin (Jason) Liu and Tom Graser in particular, for their thoughtful insights in revising this paper. We also thank Bob MacFadzean for the detailed information he provided on the operations of the radar tracking system.

#### References

1. Suraj, A., *Extended Statecharts for Systems Modeling, Specification and Verification*, August 1996, Master's thesis, Dept. of Electrical and Computer Engg., The University of Texas at Austin.
2. Barcio, B., *SMOOCHES: State Machines for Object-Oriented Concurrent, Hierarchical Engineering Specifications*, December 1994, Master's thesis, Dept. of Electrical and Computer Engg., The University of Texas at Austin.
3. Harel, D. *Statecharts: A Visual Formalism for Complex Systems*. in *Science of Computer Programming*. June 1987. 8(3): p 231-274.
4. Barcio, B., et al., *Object-Oriented Analysis, Modeling, and Simulation of a Notional Air Defense System*. SIMULATION, January 1996. 66(1): p. 5-21.

<sup>1</sup> When the system enters a state  $i$ , a timer event  $i.\phi$  maintains the time spent in  $i$ . This event is reset when the system leaves state  $i$ . Therefore,  $val(x: y. \phi)$  gives the time spent in substate  $y$  of  $x$ .