

Object-Oriented Analysis, Modeling, and Simulation of a Notional Air Defense System*

Bernard T. Barcio, S. Ramaswamy**, Robert MacFadzean, K. S. Barber

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

ABSTRACT

This paper describes the analysis, modeling, and simulation of a notional air defense system using SMOOCHES (State Machines for Object-Oriented, Concurrent, Hierarchical Engineering Specifications). SMOOCHES is an object-oriented environment based on hierarchical state machines and extensions to Statecharts, specifically developed as an environment to specify, model, simulate and analyze / evaluate distributed, reactive systems.

I. INTRODUCTION

In this paper, we use SMOOCHES, an object-oriented environment developed for the hierarchical state modeling and simulation of distributed, reactive systems, to specify, model, simulate and analyze a notional air defense system. SMOOCHES [1] considers real world systems development as an iterative and interactive process, wherein system requirements and subsystem functionalities are not completely pre-defined as part of the initial requirements specification, but evolve along with the system development process. Moreover, SMOOCHES employs an “event-driven” approach to system specification. Therefore, “events” are assumed to cause a change in object behavior as they trigger the transitions between different states. The example developed in this paper, the air defense system, exemplifies a system with the following characteristics: (i) significant interactions between software, hardware and human users, (ii) requirement of a predictable response from a dynamic, uncertain environment, and, (iii) multiple interacting subsystems and environmental constraints.

The SMOOCHES environment is designed to be a generic object-oriented modeling and simulation environment. The use of object-oriented techniques is supported in SMOOCHES to facilitate ease of use and provide portability across various application domains. SMOOCHES is interactive and provides the following monitoring capabilities: (i) *overall system operations monitoring*: This feature is used in displaying the objects in the system and choosing a particular object for detailed observation, (ii) *communications monitoring*: This feature is used to monitor the communication structure of the selected object, (iii) *observing state hierarchies*: Given a higher level state, this facility is used to observe states belonging to different levels of hierarchy, and, (iv) *observing legal state transitions*: Given a particular state, this facility is used to observe the events and transitions that are legal in the state.

The objective of this paper is to demonstrate the useful-

ness and applicability of the SMOOCHES environment in the object-oriented modeling, simulation and development of complex software systems. SMOOCHES also provides the benefit of allowing the object-oriented development of hierarchical system specifications. This research uses the SMOOCHES environment to specify, model, simulate and analyze an air defense system, which emphasize the applicability of SMOOCHES to address the following issues: (i) the use of a high level specification language to represent hierarchical and object-oriented system characteristics, (ii) the architectural support for using object-oriented techniques and the ease and flexibility with which it can be applied within the SMOOCHES environment, (iii) the use of exit-safe states in specifying and modeling distributed, reactive systems, and, (iv) the specification of system timing characteristics for system modeling and analysis.

The paper is organized as follows: Section II describes the significant features of the air defense system. Section III derives the system model. Section IV presents an overview of the development of the radar tracking system within the SMOOCHES environment. Section V concludes the paper.

II. THE NOTIONAL AIR DEFENSE SYSTEM

The tracking operation involved in radar systems refers to the continuous maintenance of data about the position of a given target. A single target tracking radar implies a dedicated radar used to track a single target. Typically, a com-

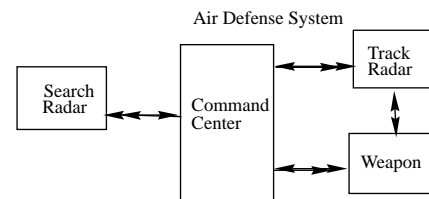


Figure 1. Block Diagram of Air Defense System

mand issued from a Command Center 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 radar consists of a transmitter-receiver pair, a pulse generator, an antenna, and a signal processing unit (SPU). The information from the radar is sent to a display at a command/monitoring center. The antenna is classified by its shape, either rectangular or circular, and the dimensions of the various shapes determine the gain and beam width characteristics. The antenna pattern refers to the lobed shape of the gain as a function of the angle relative to the bore sight. The radar has several attributes contained within the infor-

* This research was supported in part by the National Science Foundation under grant IRI-9409192, and in part by the Texas Higher Education Coordinating Board under Grant ATP-115.

** Dr. S. Ramaswamy is currently with the Division of Computer and Applied Sciences, Georgia Southwestern College, Americus, GA. .

mation model. These include: (i) power, (ii) gain, (iii) position (iv) pulse width, and (v) pulse repetition frequency (PRF).

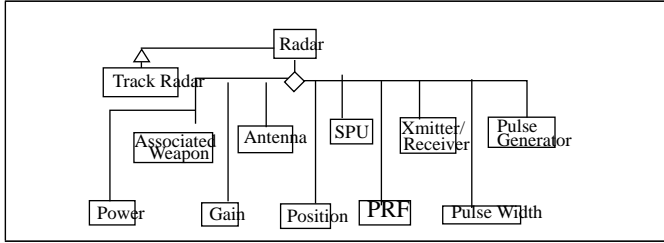


Figure 2. Object Model of Radar

The weapon agent has the following significant attributes: (i) weapon type, (ii) amount of ammunition remaining, and, (iii) position. The generated pulse has an RF carrier frequency of 400 to 18000 MHz with a pulse width of .5 to 5.0 microseconds. After amplification, the radar transmits the pulse in the direction of the antenna bore sight axis. If the generated pulse encounters a target, a fraction of the signal is reflected back to the radar and a fraction of this reflected signal is intercepted by the antenna aperture. The signal arriving at the antenna is amplified, processed, and displayed at the command/monitoring center. Typically, the pulse train consists of equally spaced pulses. The pulse repetition frequency (PRF) is the number of pulses transmitted per unit time. The pulse repetition interval (PRI) represents the time between pulses and is equal to the reciprocal of the PRF. Let δ be the maximum round trip travel time of a pulse (from the radar off the target and back to the radar). In practice, $\delta < \text{PRI}$. Otherwise, there will be range measurement ambiguity among the transmitted pulses.

The range gate is defined as the time when the radar is expecting a pulse from the desired target. The radar “slews” to the appropriate coordinates and then positions a range gate. This process usually takes less than a second. During the range search, the time when the gate turns on and the duration of the gate are varied according to a pre-programmed pattern until a pulse arrives at the gate. This pattern generally differs between different radars. After the target is acquired, the radar begins to track the target, that is, the radar attempts to continually receive the pulse in the center of the gate. The radar adjusts the **GateOn** time in order to maintain target tracking. By integrating the pulse it receives while the gate is on, the radar determines the necessary details to adjust its **GateOn** pulse. For the first half of the **GateOn** time, the radar positively integrates the pulse it receives; and, for the second half, it negatively integrates the pulse. If the result is zero, the target is considered to be centered within the gate. If the result is non-zero the radar adjusts the **GateOn** time.

A dedicated weapon system works in conjunction with every radar system. Similar to the radar, the weapon system will “slew” to desired coordinates when instructed to track a target by the command center. The weapon system will then work closely with an assigned radar system to track the target. Should the need arise, the command center sends a destroy target command to the weapon system. In such a situation, if the target is currently tracked, the weapon system will fire at the target.

Using the Shlaer/Mellor model [3], the block diagram in Figure 1 illustrates the relationship between the Command Center, the Radar, and the Weapon system. A one-to-many (single arrow to double arrow in Figure 1) relationship exists between the Command Center and the Track Radar, and the Command Center and the Weapon. However, there is a one-to-one (single bi-directional arrow in Figure 1) relationship between a Track Radar and a Weapon. This indicates that the Command Center has to typically handle communications with more entities (broadcast), both Track Radar and Weapons, whereas the Track Radar and a Weapon have a dedicated communication structure (point-to-point).

Figure 2 is the object information model of a radar using the Rumbaugh object model [2]. The model contains the constituent parts and attributes of the Radar as described above. Note that the **TrackRadar** inherits all the information from the basic Radar system. The next section describes the behavioral model of the radar system.

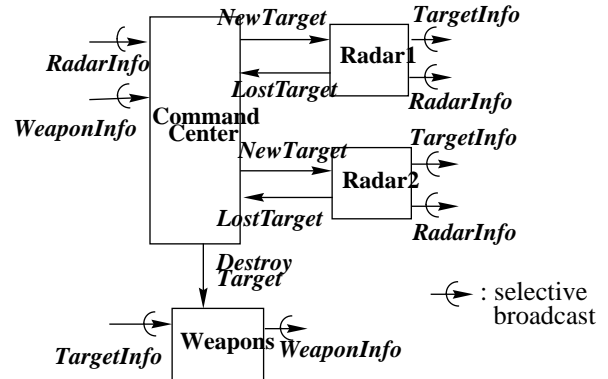


Figure 3. Object Communication Model

III. SYSTEM MODEL

In this section, a general model of the tracking performed by the radar and weapon agents is developed. The underlying assumptions are as follows: (i) the command agent sends out basic initiation commands, and (ii) the command agent is under human control. Figure 3 illustrates the object communication model between the different agents. Events associated



Figure 4. High-level System Specification

with external communication are: (i) *NewTarget*, (ii) *LostTarget*, (iii) *TargetInfo*, (iv) *RadarInfo*, (v) *DestroyTarget*, and (vi) *WeaponInfo*. The *NewTarget* event is sent by the command center to the Radar defining the general area of the target. The *LostTarget* event signifies to the Command Center that the target has been lost. General information like target position information and radar configuration are communicated by the *TargetInfo* and *RadarInfo* events to any agent interested in this type of information. In addition to promoting the use of object-oriented, event-driven, hierarchical state specifications of distributed reactive systems, SMOUCHES also allows the developer to classify events into either *special* or *basic* events, and to specify event hierarchies. These two classes of events

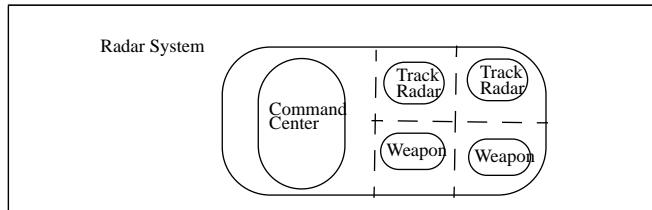


Figure 5. High Level Radar System Description

are processed in different ways. Special events usually refer to events that require immediate attention. These events include error events and events with certain priorities. The handling

Event Name	Event Description
<i>Lock</i>	Command to weapons system to initiate tracking of the target
<i>RSlewDone</i>	Event signifying that the radar is done slewing
<i>WSlewDone</i>	Event signifying the weapons system is done slewing
<i>DestroyTarget</i>	Command to weapons system to initiate destruction of the target
<i>AmmoEqZero</i>	Condition when the weapon is out of ammunition
<i>AmmoGtZero</i>	Condition when the weapon still has ammunition
<i>Refilled</i>	Event signifying that the weapon has been refilled with ammunition
<i>NewTarget</i>	Command telling radar system to monitor a new target
<i>TargetInfo</i>	Class of events which involve information related to the target
<i>LostRange</i>	Event signifying that the radar has lost the range on the target while tracking
<i>HaveRange</i>	Event signifying that the radar has the range on the target while tracking
<i>TargetPos</i>	Information containing the current position of the target
<i>LostTarget</i>	Event signifying that the radar has completely lost the target
<i>DoneWithTarget</i>	Event signifying that the weapons system is done with the target
<i>Pulse</i>	Event signifying the receipt of the returning pulse signal
<i>SendPulse</i>	Command to send a pulse
<i>GateOnEvent</i>	Event signifying the start of the GateOn period for the radar
<i>GateOffEvent</i>	Event signifying the start of the GateOff period for the radar
<i>IniTimeOut</i>	Event signifying the half way point during the GateOn period

Figure 6. List of Events

of these different types of events within the SMOUCHES environment is discussed in detail in [1]. These features are used to create individual grouping of events, associate events to certain states, and specify event sub-groupings. Moreover, the designer can also specify different kinds of event handling mechanisms. For example, the weapons agent is interested in the *TargetInfo* since it will need to follow the target if the target is to be destroyed. The *HaveRange* and *LostRange* events form part of *TargetInfo* event. These events signify whether the target is currently being tracked. An object-oriented, state-based specification language, derived in [1], is used to identify

the main components of the system, the main states of those components, and the areas of concurrency. Figure 4 illustrates a trace of such a high-level specification for the radar tracking system. *Events and states are represented in italics and bold, respectively*. The **TrackRadar** is derived from a radar base class. This radar base class contains basic radar attributes. Since the **Acquiring** and **Tracking** states each transition through a **GateOn** and **GateOff** states and emit pulses. They are derived from a common **Radiate** state which contains sub-states **GateOn** and **GateOff**.

A. Behavioral Description and State Construction

Figure 5 illustrates some agents within an example system. It includes one Command Center and multiple Track Radars and Weapons. This presents the top-level view of the system. Figure 6 provides the definitions of the events involved in the radar system. The events *LostRange*, *HaveRange*, and *TargetPos* all belong to the general class of event *TargetInfo*. Figure 7 through Figure 11 illustrate the state dia-

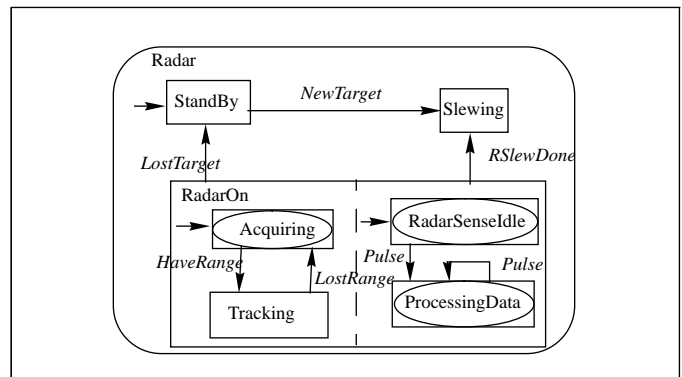


Figure 7. Substate Composition of Radar

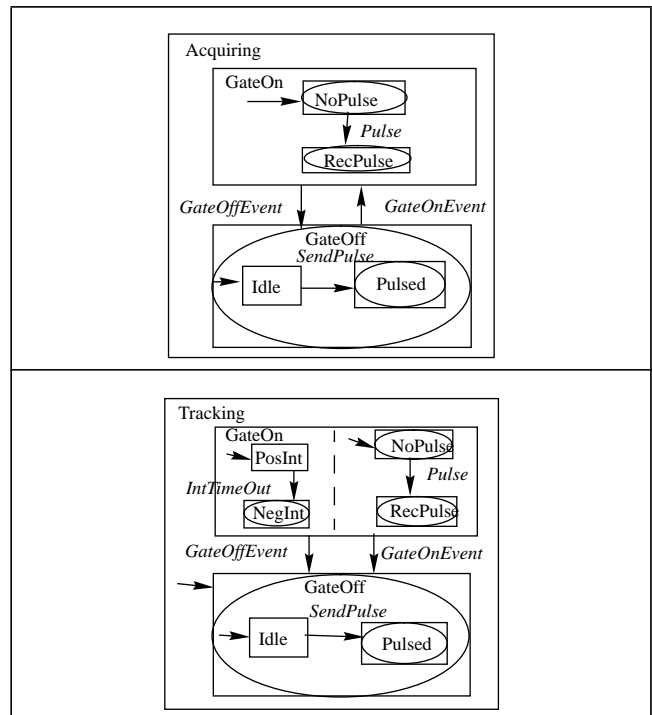


Figure 8. Acquiring and Tracking State Transition Diagrams

grams and transitions for the agents. These state diagrams define the behavioral system model. Figure 7 is the substate descriptions of the **Radar** agent. Note the use of exit-safe states to define the case when an event can cause a transition from a high level state. **StandBy** is the default state. When the radar receives the *NewTarget* event, it transitions to the **Slewing** state. In the **Slewing** state the radar motors engage and move the radar to the general location of the target. The event, *RSlewDone*, signifies that the radar is in the correct position. Upon this event, the radar enters the **RadarOn** state. The radar simultaneously enters the concurrent default states of **Acquiring** and **RadarSenseIdle**. The states **RadarSenseIdle** and **ProcessingData** 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, the radar transitions to the **Tracking** state. If the radar loses the range on the target, signified by the *LostRange* event, the radar returns to the **Acquiring** state. If the radar has lost the target completely, signified by the *LostTarget* event, it transitions back to **StandBy**.

Figure 8 is the substate diagrams of the **Acquiring** and **Tracking** states. For both the **Acquiring** and **Tracking** states, if a pulse is received during the **GateOn** substate, the radar transitions to the corresponding **RecPulse** substate. In the **Acquiring** state, the radar will attempt to modify the gate on time to get a more accurate range on the target. This is accomplished by adjusting the time when the *GateOffEvent* is generated. In the **Tracking** state, the radar uses the information

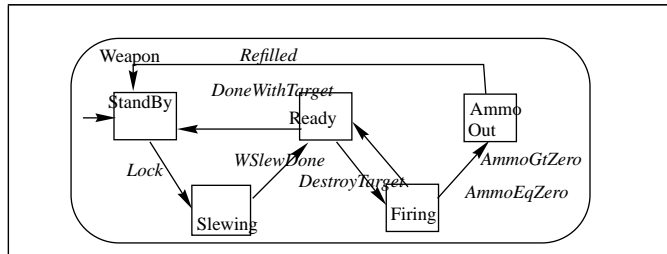


Figure 9. Weapons System States

acquired in the **PosInt** and **NegInt** substates to keep the range on the target. The **PosInt** and **NegInt** substates represent when the radar is integrating the pulse information positively and negatively respectively. To keep the range on the target, the radar adjusts when the *GateOnEvent* is generated. The **GateOff** state has two states representing if an outgoing pulse has been generated or not. The event, *SendPulse*, signifies the start of the **Pulsed** substate.

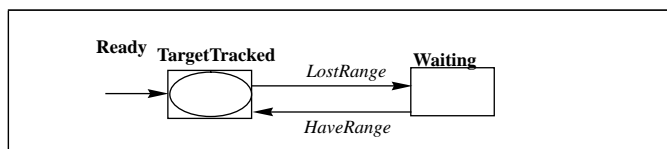


Figure 10. Weapon Ready State Description

Figure 9 is the state diagram for the **Weapon** agent. Like the Radar agent, the Weapon agent defaults to a **StandBy** state. After receiving the *Lock* event, 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 the weapon system then is in the **Ready** state. While in this state, the weapon will follow the target using the information from the corresponding radar system. The **Ready** state is described below. If the *DestroyTarget* event occurs the weapon will move to the **Firing** 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 **StandBy** state upon the occurrence of the *DoneWithTarget* event.

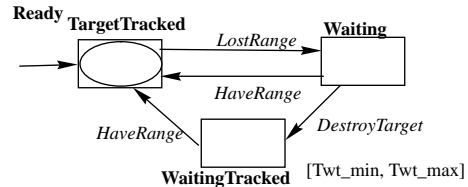


Figure 11. Revised Weapon Ready State Diagram

Figure 10 shows the substates of the **Ready** state. The *DestroyTarget* event will pend on the **Ready** state if the radar loses the track on the target. Once the target is tracked (i.e. the Weapon is in the **TargetTracked** substate of **Ready**) the weapon will transition to the **Firing** state.

Agent	State	Timing Specs
Radar	Slew	[RadarSlew_min, RadarSlew_max]
Radar	GateOn	[GateOn_min, GateOn_max]
Radar	GateOff	[GateOff_min, GateOff_max]
Radar	TrackedOffIdle	[Idle_min, Idle_max]
Radar	TrackedOffPulsed	[Pulsed_min, Pulsed_max]
Radar	Acquire	[Acquire_min, Acquire_max]
Weapon	Slew	[WeaponSlew_min, WeaponSlew_max]
Weapon	WaitingTracked	[WaitingTracked_min, WaitingTracked_max]

TABLE 1. State Timing

B. Timing Attributes

Specific timing requirements are necessary for the **GateOn** and **GateOff** substates of the **RadiateState**. Specification of timing requirements for the **Slewing** are specified to ensure that the positioning system is sufficiently fast enough to position either the radar or the weapons system. A violation of the maximum time generates an exception which leads to a state in order to determine if there is a problem with the motor control of the radar or weapons base. The radar has a meaningful upper time bound on the **Acquire** state. If the radar remains in the **Acquire** state too long, an exception occurs which causes the *LostTarget* event to be generated. For the weapons system state of **Ready** and **Waiting**, a strict upper limit could be set on how long the weapons system will have to wait before proceeding to tracking the target.

Figure 11 shows a new substate description of the **Ready** state that contains a timing specification which denotes the

maximum amount of time the weapons system should wait before proceeding to the **TargetTracked** state after receiving the *DestroyTarget* command. This substate diagram shows a

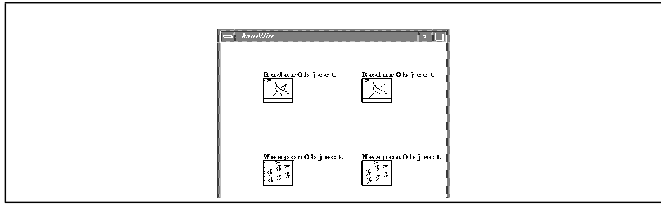


Figure 12. Air Defense System

state called **WaitingTracked** which signifies the state when a *DestroyTarget* event has occurred, but the radar has lost the range on the target. Thus, if the *DestroyTarget* event occurs while the target is not being tracked, the **WaitingTracked** state is entered and the *HaveRange* event must occur within *Twt_max* amount of time, or an exception will be generated. If the *DestroyTarget* event occurs while in the **TargetTracked**, the weapon will transition to the **Firing** state since **TargetTracked** is exit-safe. The *DestroyTarget* event will be pending to the **Ready** event if the weapons agent receives the *DestroyTarget* command while the **Ready** state is not in an exit-safe substate. This procedure prevents the weapons system from firing unless the radar is actually tracking a target.

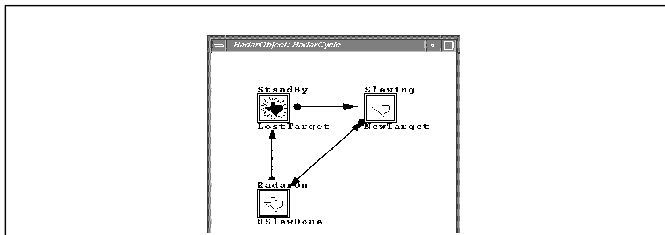


Figure 13. Naval Radar Main State Diagram

If a timing requirement of the system fails, an exception is generated. The exception handling might be different depending on the state in which the exception is generated. The exception generated due to the radar losing the target might be to simply send the radar back to standby and announce that the target is lost or out of range. However, if a state in the weapons control violates its timing requirements, the exception might require more serious attention.

Table 1 is a listing of some states within the system that have associated hard time values. The other states within the system can be considered to have timing specifications of $[0, \infty]$ which effectively mean that they exhibit soft time characteristics. Some timing characteristics about certain states that are not listed in the table can be easily derived. The timing characteristics of the **TrackedGateOff** state is $[Idle_min + Pulsed_min, Idle_max + Pulsed_max]$. One can also determine that the GateOn - GateOff cycle of the **Acquire** state will cycle at least $Acquire_min / (GateOn_max + GateOff_max)$ times and at most $Acquire_max / (GateOn_min + GateOff_min)$. These timing characteristics are derived from the rules defined in [1].

IV. IMPLEMENTATION

Figure 12 shows the top-level states of the defense system and Figure 13 shows the top-level states of the radar: **StandBy**, **Slewing**, and **RadarOn**. The radar transitions from **StandBy** to **Slewing** when it receives the event *NewTarget*. *NewTarget* contains the position information necessary to tell the radar where to set its position. When the radar is done slewing it transitions to the **RadarOn** state. If the radar loses the target it transitions back to **StandBy**. The command con-

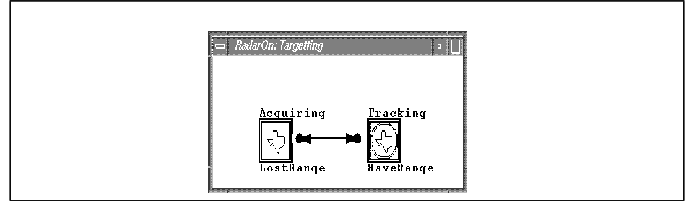


Figure 14. RadarOn State Diagram

troller agent would send new commands in response to the *LostTarget* event. Figure 14 presents the two substates of the

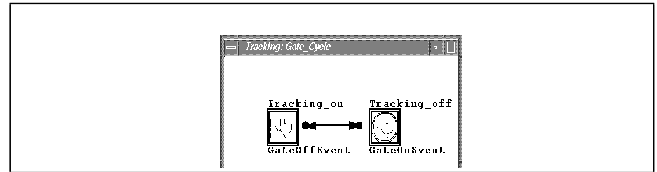


Figure 15. Gate Cycle State Diagram

RadarOn state: **Acquiring** and **Tracking**. Both substates are derived from common base state, **Radiate**. Since the **GateOn** and **GateOff** cycle is common to both the **Acquire** state and **Tracking** state, the base state **Radiate** is abstracted. This state contains the common state lifecycle of **GateOn** and **GateOff**. **Acquire** and **Tracking** are then derived from **Radiate**. Additional states and activities are then added to the derived classes. When in the **GateOn** state, the radar is looking for a return pulse. When in the **GateOff** state the radar will generate the pulse but not be expecting the return pulse. Figure 15

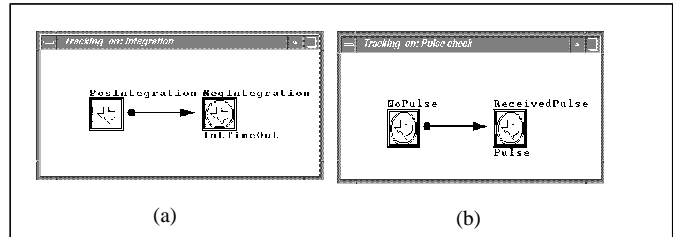


Figure 16. Concurrent State Diagrams for Gate On SubState of Track State

shows the gate cycle for the **Tracking** state. The gate cycle state diagram for the **Acquiring** state is similar, but the substates for the **GateOn** state are different. Whereas the **Acquiring** state is simply looking for a pulse to occur within the **GateOn** substate, the **Tracking** state needs to keep the pulse centered. Thus, it performs an integration on the pulse data it receives. For the first half of the **GateOn** state the radar performs a positive integration, and for the second half, it performs a negative integration. Thus, the **GateOn** state contains the **PosIntegration** to **NegIntegration** state diagram

illustrated in Figure 16 (a). Concurrently, the radar starts in a state in which it has not received a pulse and then transitions to a state denoting it has received a pulse if the pulse returns. This is useful in determining if the range on the target has been lost. Figure 16 (b) illustrates this state diagram.

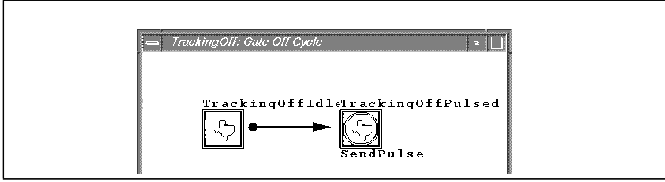


Figure 17. Gate Off State Diagram

Figure 17 shows the state transition of the **GateOff** state. At first it is idle and then after a period of time transitions to an exit-safe state representing that the radar has transmitted a pulse. Figure 18 describes the state diagram of the weapons

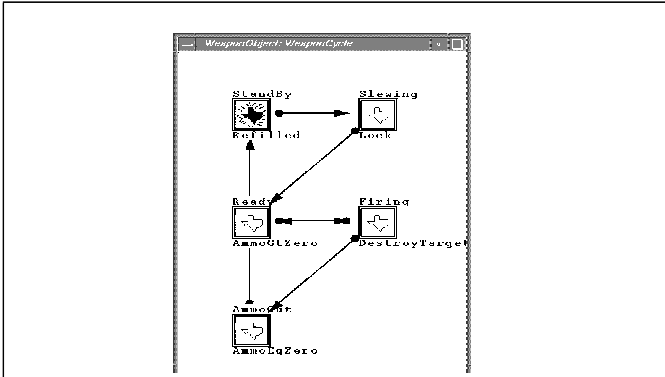


Figure 18. Weapon State Diagram

system. The weapon system starts in a **StandBy** state. After receiving a *Lock* command event the weapon slews to the appropriate position. During this time the weapon system is in a **Slewing** state. When the weapon system is done slewing it transitions to a **Ready** state. If a *DestroyTarget* event is received the weapon system will transition to the **Firing** state and then back to ready state if it still has ammunition. If the weapons system is out of ammunition it will transition to the **AmmoOut** state.

Figure 19 shows the substates involved with the **Ready** state. While in the **Ready** state, the weapon system is communicating with the radar, that is, the weapon system is interested in specific events of the radar system. The first substate is the

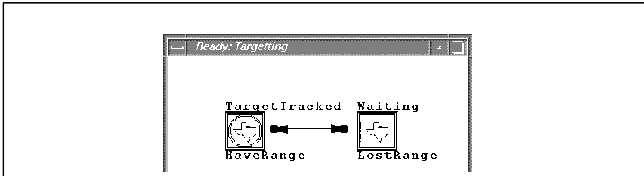


Figure 19. Weapon Ready State Diagram

TrackedTarget state. Within this state, the target is being tracked. If the radar loses the range on the target, the weapon system transitions to the **Waiting** state. A *DestroyTarget* event can occur while in the **Ready** state. The weapon system will not transition to the **Firing** state until the substate is in the exit-safe state, **TargetTracked**. Therefore the *DestroyTarget* event will be pended until the **TargetTracked** state is current. This behavior definition is to ensure that the weapon system

does not fire at a target that is not tracked.

V. CONCLUSIONS

The operation of a notional air defense system has been modeled and analyzed using the SMOOCHES environment. The example demonstrates the application of SMOOCHES as an useful tool in behavioral modeling: (i) it provides a graphical means of specifying the system under development, (ii) it maintains underlying formal behavioral definitions, and (iii) it simulates behavior models for verification purposes. The SMOOCHES environment is currently being used as a test bed to study problems related to manufacturing applications. Design verification and validation within the SMOOCHES environment is a topic of ongoing research.

References

1. B. Barcio, S. Ramaswamy, K. S. Barber, "An Object-Oriented Modeling and Simulation Environment for Reactive Systems Development", *Submitted to IEEE Transactions on Software Engineering*, Feb 1995. *Preliminary version appeared in the 1995 IEEE Intl. conf. on Robotics and Automation, Japan.*
2. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design," Englewood Cliffs, New Jersey: Prentice Hall, 1991.
3. Sally Shlaer, Stephen Mellor, "Object Lifecycles: Modeling the World in States," Englewood Cliffs, New Jersey: Yourdon Press, 1992.