

Views on Modeling

Modeling the Dynamic Behavior of Systems, Mechanisms, and Classes with Scenarios

Donald G. Firesmith

INTRODUCTION

In my last column, I recommended joint application development (JAD) workshops using whiteboards as an effective way to perform object-oriented modeling. In the next few issues, I will be describing various models that are developed as part of the latest version of the ASTS Development Method (ADM).

ADM is a comprehensive, fourth-generation, object-oriented systems development method that was designed for use on large, complex, distributed, and even real-time distributed applications. It covers both systems- and software-level requirements elicitation, analysis, design, implementation, and testing. Version 3 of ADM was documented in *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach* [1]. Version 4 of ADM is documented in various public domain ASTS standards, procedures, and guidelines [2, 3, 4, 5, 6]. Along with other methods, it will also be summarized in *Object-Oriented Methods, Standards, and Procedures* [7] and in future issues of this magazine.

ADM is a comprehensive method containing the following models:

- Logical Static Architecture Models:
 - Configuration Models
 - Semantic Models
- Logical Dynamic Behavior Models:
 - Interaction Models
 - State Models
 - Timing Models
- Implementation Models:
 - Processor Models
 - Language Models
 - Presentation Models

The remainder of this column will discuss the use of scenarios as part of ADM's interaction model. It will define scenarios, discuss their benefits, provide the requirements for an example that will be used in this and future articles, introduce scenario lifecycles and scenario

lifecycle diagrams, discuss how to document individual scenarios, and discuss the risks involved in using scenarios.

SCENARIOS

ADM defines a **scenario** as a single usage of one or more collaborating classes or objects (i.e., a legal sequence of messages, operation executions, and exception raising and handling). Thus, scenarios are functional abstractions that typically involve several objects or classes and which usually involve multiple threads of control. Unlike Jacobson's *use cases* [8], scenarios need not be initiated by a client terminator (a.k.a., actor). For example, a control loop mechanism in an embedded application may execute automatically without initiation or control from client objects or terminators.

Scenarios often provide a required capability and are used during the requirements elicitation or analysis, design, testing, and integration activities. Scenarios may be used:

- To elicit customer and user requirements
- To document functional requirements
- To communicate with customers and users
- To validate the object-oriented models (e.g., via role playing)
- To prepare integration and functional tests
- As inexpensive manual prototypes
- To capture work flows in enterprise modeling

ADM employs scenarios having different scopes at different points in the development process. Initially, top-level scenarios treating the system to be developed as a blackbox are used to capture the overall behavior of the system from the user or customer viewpoint. Scenarios may also be used to capture mechanisms in individual subsystems. Software scenarios may also be used to capture the behavior of top-level assemblies or mechanisms within individual subassemblies. Scenarios may even be used to capture the interactions of messages, exceptions, operations, and attributes within complex classes.

THE DOOR MASTER EXAMPLE

The next few paragraphs provide a complete set of requirements for Door Master, a system controlling access of employees through a secured door such as might be found in an airport or company facility. These

Donald G. Firesmith provides consulting and training in object technology. He developed and maintains ADM, a fourth-generation object-oriented development method, and OOSDL, a second generation object-oriented specification and design language. He is the author of *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach* (John Wiley and Sons, 1993) and *Object-Oriented Development Methods, Standards, and Procedures* (Prentice-Hall, 1994). He is currently writing *Testing Object-Oriented Software* (SIGS Books, 1994). He can be reached by mail at Advanced Software Technology Specialists, 2910 Webster, Fort Wayne, IN 46807, by phone at (219) 745-7928, and by e-mail at 73664.3515@compuserve.com

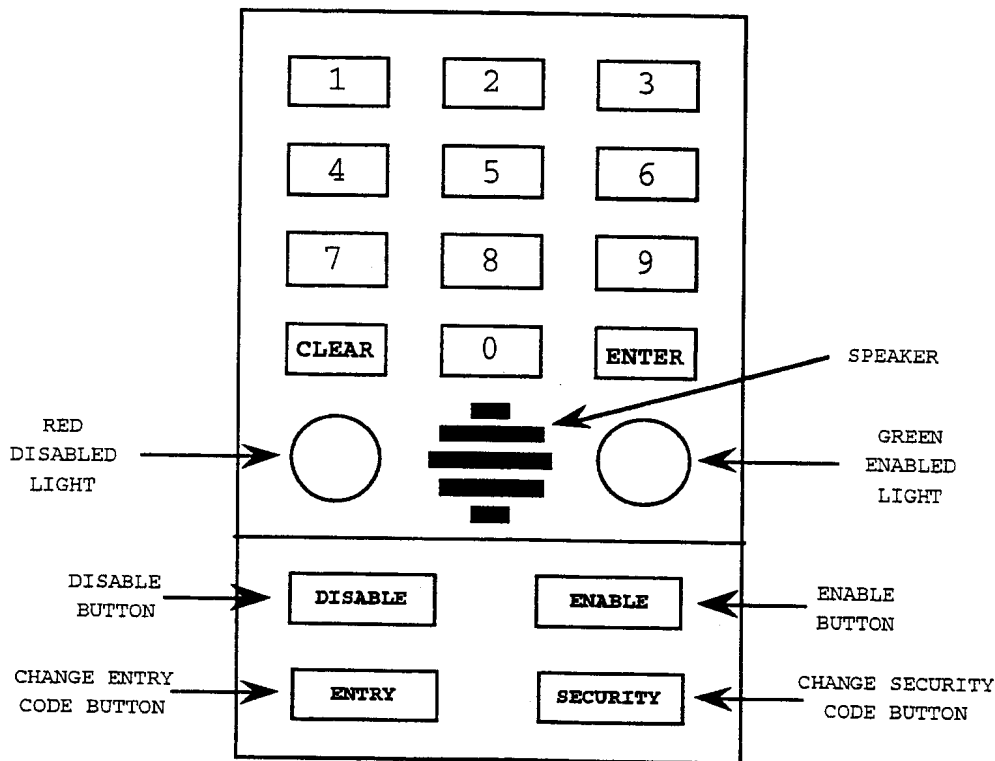


Figure 1: The Control Panel for Door Master

requirements will be used in this column to illustrate the use of scenarios and in future columns to illustrate other modeling techniques of ADM. Door Master consists of a control panel mounted next to the door and a hidden door lock and associated sensor. Figure 1 is a schematic of the control panel.

The following requirements concern *initializing Door Master*. The initial entry code of Door Master shall be "12345." The initial security code of Door Master shall be "1234567." The initial state of Door Master shall be disabled. The initial state of the disabled light shall be on. The initial state of the enabled light shall be off. The initial state of the numeric keypad shall be cleared. The initial state of the speaker shall be off.

The following requirement concerns *allowing free access when Door Master is disabled*. When Door Master is disabled, the door shall be unlocked.

The following requirements concern *controlling access when Door Master is enabled*. When Door Master is enabled, if an employee enters the correct entry code on the numeric keypad on the control panel, then the door shall be unlocked. When Door Master is enabled, if an employee enters an incorrect entry code on the numeric keypad on the control panel, then the numeric keypad shall be reinitialized. If the door is closed within 30 seconds of being unlocked, then the door shall be locked.

The following requirements concern *disabling Door Master*. When Door Master is enabled, if a member of the security personnel presses the disable button on the control panel, then the member of the security personnel has one minute in which to enter the correct security code on the numeric keypad on the control panel. If the correct

security code is entered within one minute, then Door Master shall be disabled, the disabled light on the control panel shall be turned on, and the enabled light on the control panel shall be turned off. Otherwise, Door Master shall rollback to its state prior to the member of the security personnel pressing the disable button on the control panel.

The following requirements concern *enabling Door Master*. When Door Master is disabled, if a member of the security personnel presses the enable button on the control panel, then the member of the security personnel has one minute in which to enter the correct security code on the numeric keypad on the control panel. If the correct security code is entered within one minute, then Door Master shall be enabled, the door shall be locked, the disabled light on the control panel shall be turned off, and the enabled light on the control panel shall be turned on. Otherwise, Door Master shall rollback to its state prior to the member of the security personnel pressing the enable button on the control panel.

The following requirements concern *entering the entry code* on the control panel. When Door Master is enabled, if within one minute an employee enters a five-digit candidate entry code by pressing five digit keys on the numeric keypad followed by the enter key, then the speaker will beep and the candidate entry code may be checked against the current entry code of Door Master. The clear key on the numeric keypad shall reinitialize the numeric keypad. If a candidate entry code is not entered within one minute, then the Door Master shall rollback to its state prior to the employee entering the entry code.

The following requirements concern *entering the security code* on the control panel. When Door Master is

enabled, if within one minute a member of the security personnel enters a seven-digit candidate security code by pressing seven digit keys on the numeric keypad followed by the enter key, then the speaker will beep and the candidate security code may be checked against the current security code of Door Master. The clear key on the numeric keypad shall reinitialize the numeric keypad. If a candidate security code is not entered within one minute, then Door Master shall rollback to its state prior to the member of the security personnel entering the candidate security code.

The following requirements concern *changing the entry code* on the control panel. When Door Master is enabled, if a member of the security personnel presses the change entry code button on the control panel, then the member of the security personnel has two minutes in which to enter the correct security code on the numeric keypad on the control panel, enter the new entry code on the numeric keypad, and confirm the new entry code on the numeric keypad. The clear key on the numeric keypad shall reinitialize the numeric keypad. Otherwise, Door Master shall rollback to its state prior to the member of the security personnel pressing the change entry code button on the control panel.

The following requirements concern *changing the security code* on the control panel. When Door Master is enabled, if a member of the security personnel presses the change security code button on the control panel, then the member of the security personnel has two minutes in which to enter the correct security code on the numeric keypad on the control panel, enter the new security code on the numeric keypad, and confirm the new security code on the numeric keypad. The clear key on the numeric keypad shall reinitialize the numeric keypad. Otherwise, Door Master shall rollback to its state prior to the member of the security personnel pressing the change security code button on the control panel.

The following requirements concern *raising the alarm*. When Door Master is enabled, if the door remains open longer than 30 seconds after being unlocked, then the speaker on the control panel shall raise the alarm. When Door Master is enabled, if a member of the security personnel enters the correct security code on the numeric keypad on the control panel, then the speaker shall stop sounding the alarm.

Based on the preceding requirements, Door Master has the following eight top-level scenarios:

- 1) ALLOW_FREE_ACCESS_WHEN_DISABLED
- 2) CHANGE_THE_ENTRY_CODE
- 3) CHANGE_THE_SECURITY_CODE
- 4) CONTROL_ACCESS_WHEN_ENABLED
- 5) DISABLE_DOOR_MASTER
- 6) ENABLE_DOOR_MASTER
- 7) INITIALIZATION
- 8) RAISE_THE_ALARM

Door Master also has the following two subscenarios that are used in several of the previous top-level scenarios:

- 1) ENTER_THE_ENTRY_CODE
- 2) ENTER_THE_SECURITY_CODE

MODELING SCENARIO LIFECYCLES

In most applications, the ordering of its scenarios is not arbitrary. One or more **start scenarios** may need to occur before any other scenarios may execute. Similarly, one or more **stop scenarios** may terminate the application. Some scenarios may only execute when a certain condition holds or the condition may determine which of two or more subsequent scenarios executes. Other scenarios may occur a number of times in a row, possibly with the associated loop based on some condition. Several scenarios often execute concurrently and must be synchronized. Other times, only one of several scenarios that may execute must be selected. Thus, the possible sequencing of scenarios forms a usage lifecycle for a system or assembly. A **scenario lifecycle** is the specification of ordering of the top-level scenarios of a system, an assembly of software objects or classes, or an individual object or class. Scenario lifecycles are similar to the lifecycles of FUSION [9], but address scenarios rather than events.

A **Scenario Lifecycle Diagram (SLD)** is used to document the valid interactions among the top-level scenarios of a system or assembly. SLDs document all valid orders of these scenarios during the life cycle of the system or assembly.

Figure 2 documents ADM's icons for scenario lifecycle diagrams. Using part of the notation of Jacobson et al. [8], ellipses are used as the icons for scenarios. Consistent with ADM's state transition diagram notation, start scenarios are drawn with dashed outlines whereas stop scenarios are drawn with thick outlines.

The sequencing of scenarios is indicated by transition arrows between the scenarios. Normal transitions are drawn with solid arrows, whereas exceptional transitions are drawn with dashed arrows labeled with the associated exception. If a trigger must fire a normal transition between two scenarios, the trigger is indicated by the label on the transition. Otherwise, no trigger is indicated and the subsequent scenario is assumed to execute as soon as the prior scenario is completed.

As with flow charts, the sequence of scenarios may include branching. Traditional "if" and "loop" branches based on some condition are provided. For the handling of concurrency, the "and" branch indicates that several scenarios may fire simultaneously, whereas the "select" branch indicates that only one scenario of a set of ready scenarios is selected to fire. The selection of the scenario to fire is primarily based on priority, but assumed to be random within each priority.

An aggregate scenario is a scenario containing one or more subscenarios as parts. It may be documented using ADM's standard notation for aggregation relationships.

Finally, the timing of scenarios is often important, especially in real-time applications. ADM's timing probes are used to capture such timing requirements as minimum, average, and maximum time between a first and second measurement. Timing arcs are drawn from the timing probe to one or two transition arcs for such purposes as capturing the timing of loops, the duration of scenarios, and the delta time between two transitions.

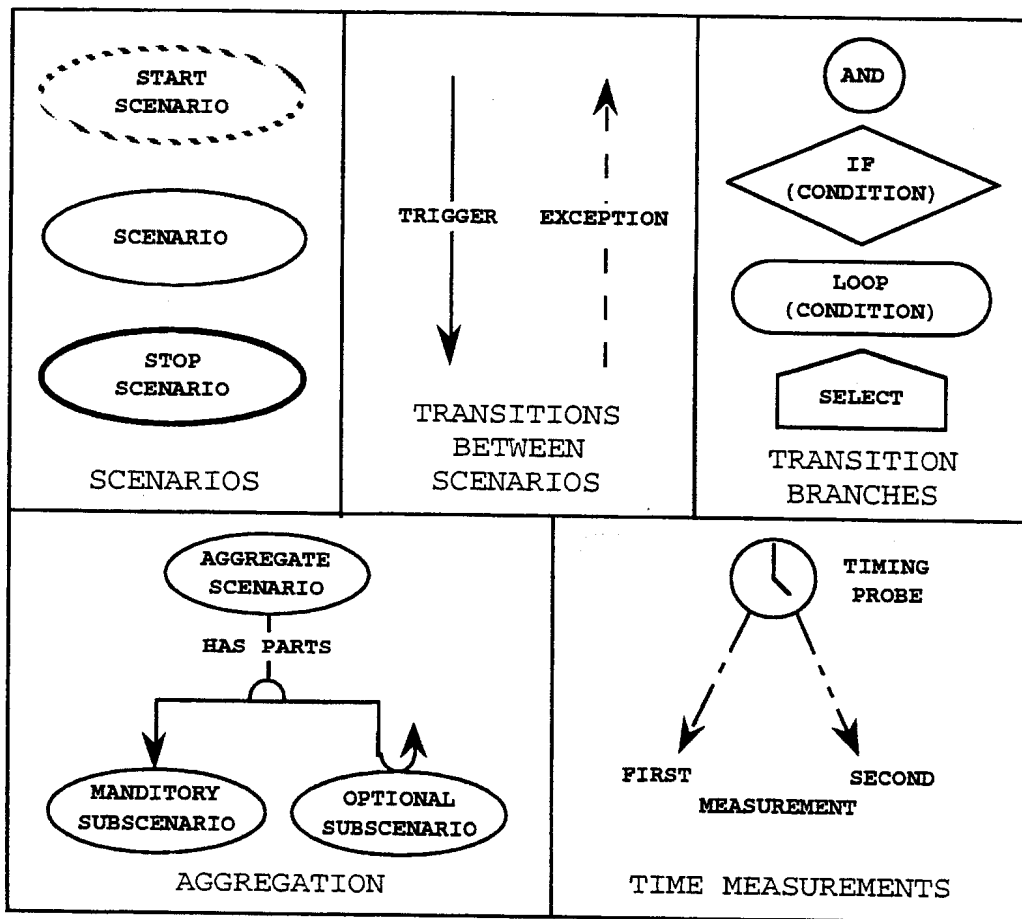


Figure 2: Icons for Scenario Lifecycle Diagrams (SLDs)

Figure 3 illustrates the SLD for the Door Master System. The start scenario, `INITIALIZATION`, executes automatically first. Then an outer loop executes zero or more times. The `ALLOW_FREE_ACCESS_WHEN_DISABLED` scenario executes until a member of the security personnel presses the enable button, triggering the `ENABLE_DOOR_MASTER` scenario. Next, an inner loop executes zero or more times. Within this inner loop, one of three equal priority scenarios is chosen. If a member of the security personnel presses the change entry code button, then the `CHANGE_THE_ENTRY_CODE` scenario executes, and if a member of the security personnel presses the change security code button, then the `CHANGE_THE_SECURITY_CODE` scenario executes. Otherwise, the `CONTROL_ACCESS_WHEN_ENABLED` scenario executes. If the door is left open more than 30 seconds, then the `RAISE_THE_ALARM` scenario executes. The inner loop is exited when a member of the security personnel presses the disable button and the `DISABLE_DOOR_MASTER` scenario executes and the outer loop repeats.

ADM's object-oriented specification and design language, OOSDL, can also be used to model the scenario lifecycle:

```

system DOOR_MASTER
  lifecycle is
    scenario INITIALIZATION;
    loop
      scenario ALLOW_FREE_ACCESS_WHEN_DISABLED;
      scenario ENABLE_DOOR_MASTER;
      until (instance(SEcurity_PERSONNEL) -> Press.The_Disable_Button) loop
        select
          scenario CHANGE_THE_ENTRY_CODE;
        or
          scenario CHANGE_THE_SECURITY_CODE;
        or
          scenario CONTROL_ACCESS_WHEN_ENABLED;
          if exception DOOR_LEFT_OPEN then scenario RAISE_THE_ALARM;
        end if;
      end select;
    end loop;
  end lifecycle;

```

```

end loop;
scenario DISABLE_DOOR_MASTER;
end loop;
end lifecycle;

```

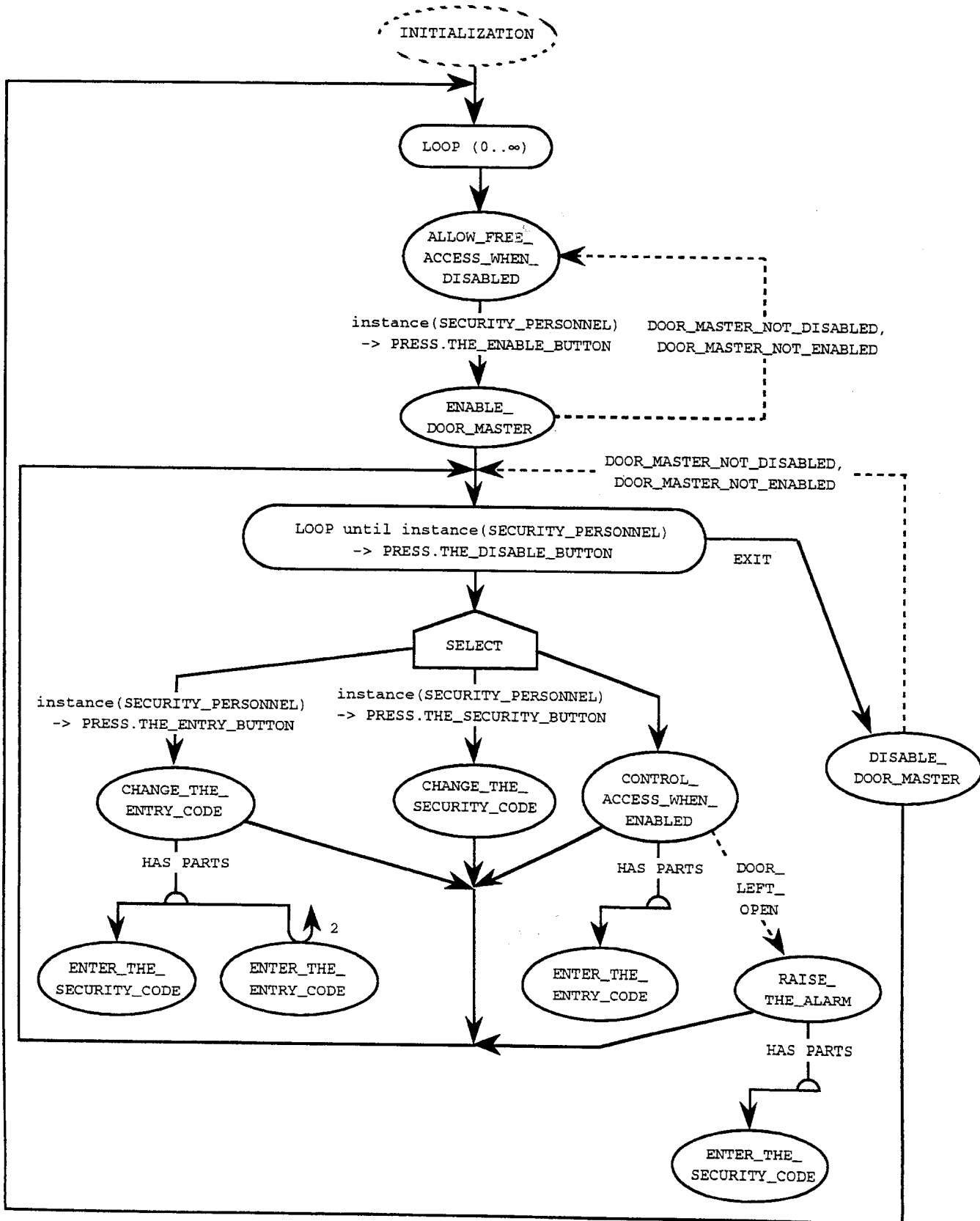


Figure 3: Scenario Lifecycle Diagram (SLD) for the Door Master System

There may be a close similarity between state models and scenario lifecycles. State transition diagrams capture the transitions between the states of a single object or class in terms of its states, whereas scenario lifecycle diagrams capture the transitions between the relevant scenarios. Both may be used to capture how an object (e.g., the system) or class behaves. Figure 4 shows the state model of the instances of the class bank accounts, and Figure 5 shows the corresponding scenario lifecycle diagram for using an instance of this class. Although each diagram contains different information, their contents are closely related and must remain consistent.

MODELING INDIVIDUAL SCENARIOS

ADM also uses OOSDL, its object-oriented specification and design language, to textually document individual scenarios. The following is the syntax for the OOSDL specification (i.e., definition, interface, and implementation) of a scenario:

```
scenario SCENARIO_IDENTIFIER
definition is
  abstraction
    <statement of abstraction>
  end abstraction;
  responsibilities
    <list of responsibilities>
  end responsibilities;
end definition;

scenario SCENARIO_IDENTIFIER
interface is
  protocol
    <list of triggers>
    <list of exceptions>
  end protocol;
end interface;

scenario SCENARIO_IDENTIFIER
implementation is
  invariants
    <list of invariants>
  end invariants;
  preconditions
    <list of preconditions>
  end preconditions;
  body
    <sequence_of_statements>
  end body;
  postconditions
    <list of postconditions>
  end postconditions;
end implementation;
```

The following is the specification of the ENABLE_DOOR_MASTER scenario:

```
scenario ENABLE_DOOR_MASTER
definition is
  abstraction
    Enable the Door Master system.
  end abstraction;
  responsibilities
    R01: Check the security code;
    R02: Lock the door;
```

```
    R03: Turn off the disabled light;
    R04: Turn on the enabled light;
  end responsibilities;
end definition;

scenario ENABLE_DOOR_MASTER
interface is
  protocol
    trigger instance (SECURITY_PERSONNEL)
      -> Press.The_Enable_Button;
    exception DOOR_MASTER_NOT_DISABLED;
    exception DOOR_MASTER_NOT_ENABLED;
  end protocol;
end interface;

scenario ENABLE_DOOR_MASTER
implementation is
  preconditions
    require
      (The_State_Of.The_Door_Master =
        Disabled and
        The_State_Of.The_Door = Unlocked
        and
        The_State_Of.The_Disabled_Light =
        On and
        The_State_Of.The_Enabled_Light =
        Off)
    else
      raise DOOR_MASTER_NOT_DISABLED;
    end require;
  end preconditions;
  body
    within 1 : MINUTES.TIME
      scenario ENTER_THE_SECURITY_CODE;
    else
      rollback;
    end within;
  if
    The_Current_Value_Of.
    The_Numeric_Keypad =
    The_Security_Code_Of.
    The_Door_Master
  then
    The_State_Of.The_Door_Master
      := Enabled;
    Turn_off.The_Disabled_Light;
    Turn_on.The_Enabled_Light;
    Lock.The_Door;
  end if;
  end body;
  postconditions
    require
      (The_State_Of.The_Door_Master =
        Enabled and
        The_State_Of.The_Door = Locked and
        The_State_Of.The_Disabled_Light =
        Off and
        The_State_Of.The_Enabled_Light =
        On)
    else
      raise DOOR_MASTER_NOT_ENABLED;
    end require;
  end postconditions;
end implementation;
```

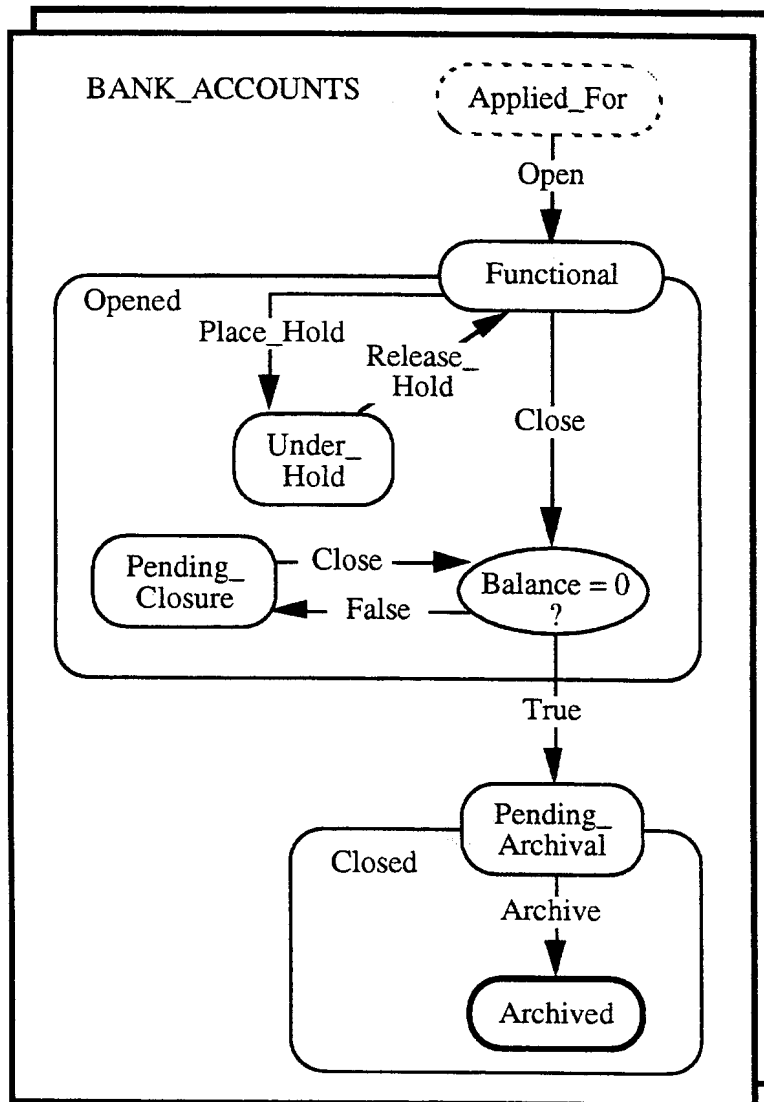


Figure 4: State Transition Diagram (STD) for the Bank Accounts Class

ADM uses blackbox interaction diagrams (BIDs) and event timing diagrams (ETDs) to graphically document individual scenarios. Figure 6 is the BID for the Enable Door Master Scenario and Figure 7 is the corresponding ETD.

RISKS WITH SCENARIO MISUSE

Although scenarios are important tools, there are also significant inherent risks involved when using them, especially when they are misused. Scenarios typically involve multiple collaborating objects and classes, and each object or class is often involved in several scenarios. There is thus no one-to-one mapping between classes and scenarios. Because scenarios are functional abstractions, there is often a strong tendency to functionally decompose them, thereby scattering the bits and pieces of the associated objects and classes (e.g., attributes and operations) throughout the resulting hierarchy of scenarios. When scenarios are used early on large projects (e.g., to elicit requirements and to identify objects

and classes), there is a tendency for different scenarios to be allocated to different teams to analyze and design and to different builds or releases for development. These different teams may then independently identify and develop different partial variants of the same object or class, reducing reuse and productivity while greatly increasing source code size as well as development and maintenance costs. Great care should therefore be exercised to ensure that scenarios are not functionally decomposed to any significant depth and that scenarios do not totally drive the identification of objects and classes on large projects.

Because a single scenario typically threads through the parts of several objects and classes, there is usually no one-to-one mapping between scenarios, scenario lifecycles, and their OOSDL specifications to the code, especially if the popular recommendation for distributing the intelligence of a system among its classes is followed. Although the OOSDL may specify an operation of a controller object, it is more likely that it will specify

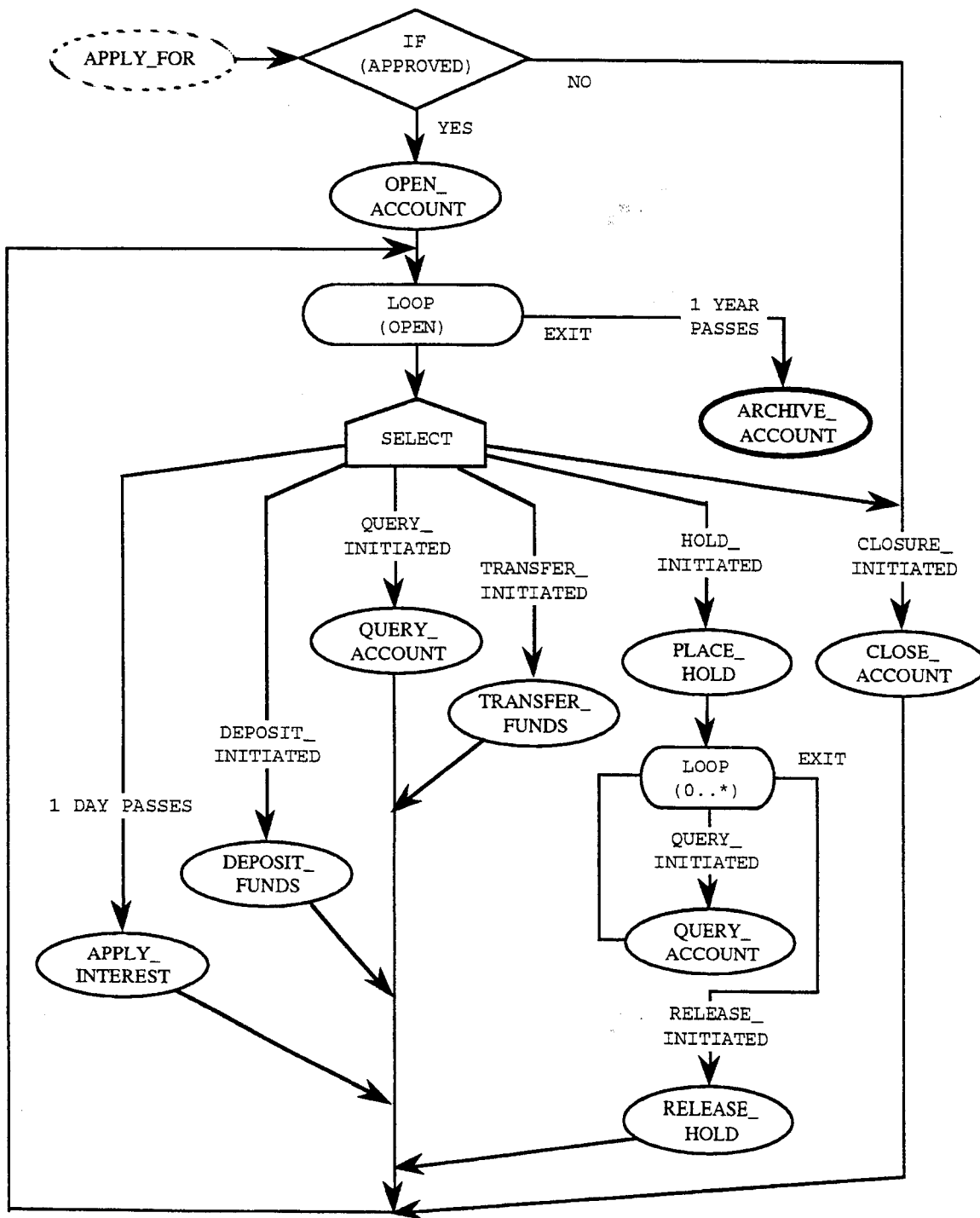


Figure 5: Scenario Lifecycle Diagram (SLD) for the Bank Accounts Class

different lines in different classes. Care must therefore be taken to ensure that scenarios and scenario lifecycles are properly captured in the resulting code.

The requirements and design captured by scenario lifecycles and certain state transition diagrams overlap significantly. Care should therefore be taken to ensure

that they remain consistent with one another and that the development of both is justified if both are used.

CONCLUSION

Scenarios and scenario lifecycles are powerful weapons that should be in the arsenal of all software

analysts, designers, and testers. They can be used to communicate with domain experts and customers, elicit and analyze requirements concerning dynamic behavior and timing, capture related design decisions, document mechanisms and other patterns of collaborating objects and classes, and prepare integration and blackbox system tests.

Conversely, scenarios can promote a functional decomposition mindset and lead to the redundant identification and development of partial variants of objects and classes involved in different scenarios, especially on large projects.

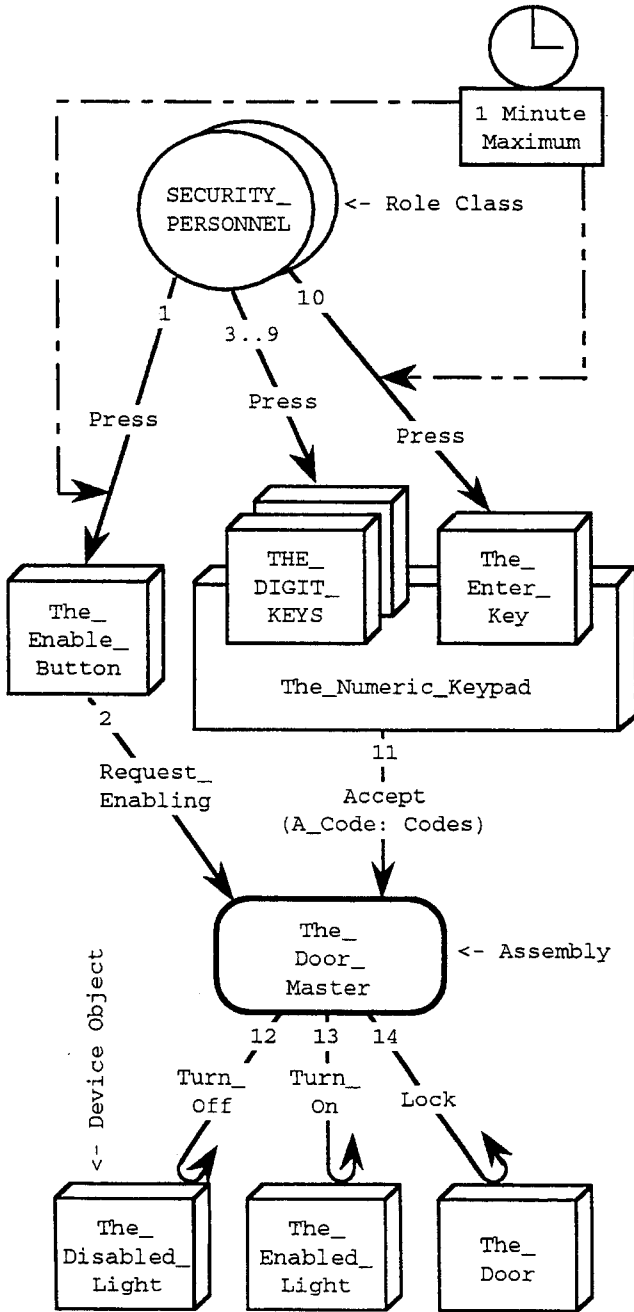


Figure 6: Blackbox System Interaction Diagram (BSID) for the Enable Door Master Scenario

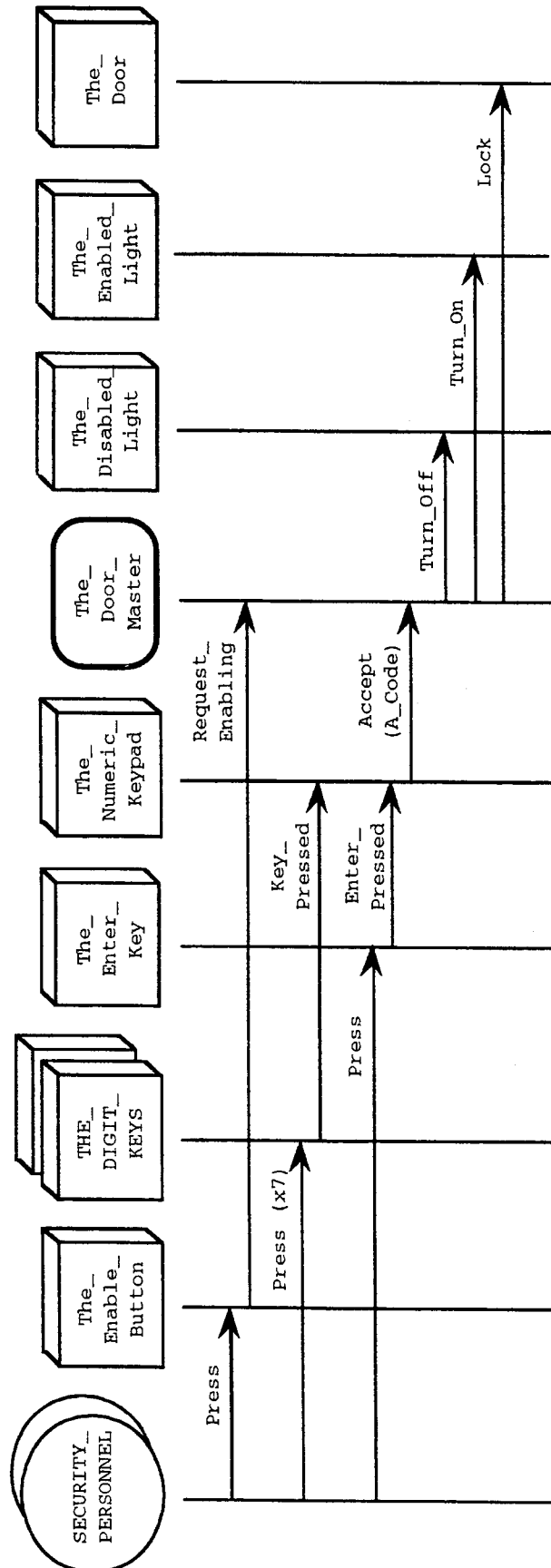


Figure 7: Event Timing Diagram (ETD) for the Enable Door Master Scenario

BIBLIOGRAPHY

- [1] Donald G. Firesmith, *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*, John Wiley and Sons, New York, New York, 1993.
- [2] Donald G. Firesmith, *ASTS Object-Oriented Requirements Elicitation Procedure*, ASTS-P02-V1.1, Advanced Software Technology Specialists, Fort Wayne, Indiana, 17 March 1994.
- [3] Donald G. Firesmith, *ASTS Object-Oriented Requirements Analysis and Logical Design Procedure*, ASTS-P03-V1.2, Advanced Software Technology Specialists, Fort Wayne, Indiana, 19 March 1994.
- [4] Donald G. Firesmith, *ASTS Object-Oriented Software Requirements Specification and Design Document Standard*, ASTS-S02-V2.2 Advanced Software Technology Specialists, Fort Wayne, Indiana, 16 March 1994.
- [5] Donald G. Firesmith, *ASTS Glossary*, ASTS-G01-V1.5, Advanced Software Technology Specialists, Fort Wayne, Indiana, 5 March 1994.
- [6] Donald G. Firesmith, *ASTS Object-Oriented Guidelines*, ASTS-G02-V1.1, Advanced Software Technology Specialists, Fort Wayne, Indiana, 20 March 1994.
- [7] Donald G. Firesmith, *Object-Oriented Methods, Standards, and Procedures*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.
- [8] Ivar Jacobson et al., *Object-Oriented Software Engineering: A Use Case Approach*, Addison-Wesley, Reading, Massachusetts, 1992.
- [9] Derek Coleman et al., *Object-Oriented Development: The FUSION Method*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.