Deliverable

Title: Test-Case Generation using an Explicit State Model Checker
Final Report

Date: March 7, 2003

Contract

Project Title: Test-Case Generation using an Explicit State Model Checker
Contractor: University of Minnesota
Mailing Address: 200 Union Street SE. 4-192 EE/CS Building.

Principal Investigator

Name: Dr. Mats P.E. Heimdahl
Title: Associate Professor
Phone: (612)-625-2068
Email: heimdahl@cs.umn.edu
Test-Case Generation using an Explicit State Model Checker Final Report

Mats P.E. Heimdahl
Jimin Gao

(612)-625-2068
heimdahl@cs.umn.edu

Department of Computer Science and Engineering
University of Minnesota
4-192 EE/SC Building
200 Union Street S.E.
Minneapolis, Minnesota 55455

Abstract

In the project “Test-Case Generation using an Explicit State Model Checker” we have extended an existing tools infrastructure for formal modeling to export Java code so that we can use the NASA Ames tool JPF for test case generation.
We have completed a translator from our source language RSML to Java and conducted initial studies of how JPF can be used as a testing tool.
In this final report, we provide a detailed description of the translation approach as implemented in our tools.
# Table of Contents

1 Executive Summary ................................................................. 7
  1.1 Translator Status ................................................................. 7
  1.2 Reading This Document ...................................................... 8
  1.3 Getting the Translator ....................................................... 8

2 Translating from RSML-e to Java ........................................... 9
  2.1 Overview ........................................................................... 9
  2.2 Data Types ....................................................................... 10
    2.2.1 Enumerated types ..................................................... 10
  2.3 Expressions ..................................................................... 10
    2.3.1 Boolean Expressions ................................................ 10
    2.3.2 Variable value expressions ....................................... 11
    2.3.3 Variable assignment time expressions ....................... 11
  2.4 Constants ........................................................................ 11
  2.5 Variables ......................................................................... 12
    2.5.1 Input variables .......................................................... 17
    2.5.2 State variables .......................................................... 18
  2.6 Message Definitions ......................................................... 20
  2.7 Functions and macros ....................................................... 20
  2.8 Input Interfaces ................................................................ 21
  2.9 Output Interfaces ............................................................. 23
  2.10 State Machine ................................................................. 24

3 Bibliography ........................................................................... 27

Appendix A - A Flight Guidance Case Example .......................... 28
  A.1 TheToyFGS00 RSML-e Model ............................................. 28
  A.2 TheToyFGS00 Translated Java Code .................................... 36
1 Executive Summary

In a NASA funded project running in parallel with the effort covered in this final report, we are investigating the use of model checking as the means for test case generation from both formal specifications and implementation code. This proposal covered complementary efforts beneficial to the original project.

The central hypothesis of the project is that model checkers can be effectively used to automatically generate test cases from a formal specification to provide test suites that test the required functionality of the software and provide an adequate level of coverage of the specification (for instance, MC/DC coverage). We also hypothesize that we can augment the specification-based test suites to achieve various code coverage criteria by generating additional test cases from the implementation also using model checkers.

During the initial phases of this project, we developed a mapping from the formal specification language RSML-e to the input language of the symbolic model checker SMV. We demonstrated how SMV could be used to generate test cases for smaller systems. Furthermore, we explored how Ames’ Java model checker Java Pathfinder (JPF) could be used to generate test cases for Java code. During this work, we concluded that there might be benefits in using an explicit state model checker, such as JPF, over using a symbolic model checker, such as SMV, for the test case generation efforts. In short, the ability of an explicit state model checker to handle integer (and real valued) variables will be a clear advantage in our problem domain—avionics and space related control systems. Although we can handle these variables in symbolic model checking through aggressive abstractions, we believe using fewer abstractions and relying on various heuristic searches in an explicit state model checker will provide better results. Therefore, we proposed to extend the current project and develop a mapping from RSML-e to the Java programming language for analysis in JPF. This translation is the effort covered in this final report.

1.1 Translator Status

We have implemented a translator from RSML-e to Java to complement our existing capabilities in using SMV. We have demonstrated the translator to NASA as part of a status report. Since this demonstration, we have refined and improved our translation approach and applied it to our case example—a flight guidance system (FGS) from Rockwell Collins Inc. (see below for further details)
1.2 Reading This Document

The discussions in this document assume a working knowledge of the NIMBUS toolset as well as RSML®. For readers unfamiliar with our tools and our language, we refer to the user documentation delivered with the tools. To fully appreciate the proposed translations, the interested reader may want to consult the formal semantics of RSML® [1].

This document is divided into two major sections. First, we present how we translate to Java (Section 2) Finally; we have included an appendix illustrating the artifacts generated by the translator.

1.3 Getting the Translator

The translator is available for download on the web. Should there be a need for any other medium (CD, DVD, Zip, etc.), please contact the CriSys group at the University of Minnesota (see below).

Since the translators are intimately tied to the RSML® execution and simulation environment NIMBUS, we are distributing the complete tool with this deliverable.

The NIMBUS toolset is available to download from

http://www.cs.umn.edu/crisys/nimbus/

Should there be any questions or other requests, please contact

Dr. Mats P.E. Heimdahl
McKnight Presidential Fellow, Associate Professor
(612)-625-2068
heimdahl@cs.umn.edu

Department of Computer Science and Engineering
University of Minnesota
4-192 EE/SC Building
200 Union Street S.E.
Minneapolis, Minnesota 55455
2 Translating from RSML to Java

This section describes our approach to translating specifications in the RSML specification language [1] Java. The objective is to describe in detail a translation scheme that is amenable to automation.

The rest of the section is organized as follows. We first give a broad overview of our translation approach. Then each component of an RSML specification is dealt with in detail with a description of how it is represented in Java, using an example. The appendix provides a complete description of translating an RSML specification to Java.

2.1 Overview

An RSML specification describes a state machine. It consists of input and state variables that can assume values of their respective types, interfaces that act as communication gateways to the external environment, and functions and macros that express computations. The specification describes the changes that occur to the values of the variables and the output produced at the output interfaces when there is some change in the input variables. The input variables in turn are assigned values at the input interfaces, which receive messages from the external environment.

Note: The following notion of a state in RSML is currently undergoing a revision and the translation approach to Java may be affected as a result. A state in RSML is completely described by the assignment history of all the variables and interfaces with their respective timestamps and the current system time. The specification can be thought of as expressing how the history changes with time in response to changes in the environment.

Note: We are currently revising RSML to include a notion of modules. With this new structuring construct, we can move to an underlying semantics where the state is described by the previous and current states only. The general state histories previously available in RSML will now instead be modeled using modules and state variables to record history values. From the users' perspective, the change will be minor and all features of RSML will still be available. From an analysis and proof perspective, however, the job will get considerable simpler.

We now discuss in detail how each construct in RSML is translated. In the descriptions below we adopt the following convention. The annotation above each table gives the BNF grammar for a piece of RSML specification of interest. The top portion of the table gives a concrete example of such a piece of specification and the bottom portion gives the equivalent Java translation for that example.
2.2 Data Types

All RSML\textsuperscript{e} variables and expressions have one of the five associated types: Integer, Real, Boolean, enumerated type and Time. In this translation, the Java primitive types \texttt{integer}, \texttt{double}, \texttt{long} and \texttt{boolean} will be used to represent RSML\textsuperscript{e} Integer, Real, Time and Boolean types, respectively.

2.2.1 Enumerated types

Since Java does not have enumerated types, we translate RSML type definition in the following way:

\texttt{type_def: TYPE_DEF IDENTIFIER \{ ' ENUM_ELEMENT_LIST ' \}}

| TYPE_DEF DOIStatusType {on, off} |
| public class RdoistatusType { |
| public static final String on = "on"; |
| public static final String off = "off"; |
|} |

2.3 Expressions

2.3.1 Boolean Expressions

\texttt{condition : TABLE row_list END TABLE | /*boolean*/ expression;}
\texttt{row_list : expression ':' truth_value_list ';'}
\texttt{| row_list expression ':' truth_value_list ';';}
\texttt{truth_value_list : truth_value | truth_value truth_value_list;}
\texttt{truth_value: 'T' | 'F' | 'T' | 'F';}

A simple Boolean expression can be translated in a straightforward manner.

\texttt{Altitude > AltitudeThreshold + Hysteresis}
\texttt{StateMachine. altitude.getValue() > StateMachine. altitudethreshold + StateMachine. hysteresis}
Note: In this example, Altitude is a state variable, while AltitudeThreshold and Hysteresis are constants. For a state variable, we need to use the getValue() to obtain its current value, but we do not need to apply this method to constant values.

The AND-OR table, as a standard form in RSML∗ to represent complex Boolean expressions, can be translated in the following way:

```plaintext
CONDITION:
  TABLE
    ASWOpModes IN_STATE OK : T *;
    ASWOpModes IN_STATE FailureDetected : * T;
  END TABLE
if ((StateMachine._aswopmodes.getValue() == StateMachine._aswopmodes_type.ok) ||
    (StateMachine._aswopmodes.getValue() ==
     StateMachine._aswopmodes_type.faileddetected))
```

2.3.2 Variable value expressions

For variable value expressions that access the historical value of a variable, only PREV_STEP is supported by this translation. The PREV_VALUE expressions and the PREV_ASSIGNMENT expressions that access the value of variable more than one step ago will not be translated. They will be flagged as an error by the translator.

```plaintext
PREV_STEP(AltitudeStatus)
StateMachine. AltitudeStatus.prevStepValue()
```

2.3.3 Variable assignment time expressions

For variable time expressions, only TIME expressions that retrieve current step or previous step time are supported by this translation. The TIME_ASSIGNED and TIME_CHANGED expressions are not supported.

2.4 Constants

All constants in RSML∗ specification can be declared in the StateMachine class (will be discussed later). The UNIT information will not be used.

```plaintext
CONSTANT AltitudeThreshold : INTEGER
  UNITS : ft
  VALUE : 20000
END CONSTANT
Public class StateMachine {
```
2.5 Variables

RSML* variables are represented by the Java classes IntVariable, RealVariable, BoolVariable, EnumVariable and TimeVariable. These classes are not generated by the translator and function as supporting library classes.

IntVariable.java

```java
public abstract class IntVariable {
    protected int expectedMin;
    protected int expectedMax;
    protected int value;
    protected boolean undefined;
    protected int prevStepValue;
    protected boolean prevUndefined;
    protected long timestamp;
    protected long prevTimeStamp;

    // add defined value as the current value of the variable
    public void addNewValue(int newValue) {
        prevTimeStamp = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        value = newValue;
        prevUndefined = undefined;
        undefined = false;
    }

    // add undefined value as the current value of the variable
    public void addNewValue() {
        prevTimeStamp = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        prevUndefined = undefined;
        undefined = true;
    }

    public long getTime() { return timestamp; }

    public int getValue() {
        if (undefined) throw new RuntimeException("Illegal undefined value access");
        else return value;
    }

    public int prevStepValue() {
        if (timestamp == StateMachine.systemTime) {
            if (prevUndefined) throw new RuntimeException("Illegal undefined value access");
            else return prevStepValue;
        }
        else {
            if (undefined) throw new RuntimeException("Illegal undefined value access");
        }
    }
}
```

Test-Case Generation using an
Explicit State Model Checker Final Report
return value;
}

public long prevStepTime() {
    return timestamp == StateMachine.systemTime ? prevTimeStamp : timestamp;
}

public boolean isChanged() {
    if (undefined && prevUndefined) return false;
    else if (undefined || prevUndefined) return true;
    else return value != prevStepValue ? true : false;
}

public boolean isAssigned() {
    return timestamp == StateMachine.systemTime;
}

public boolean isUndefined() {
    return undefined;
}

public boolean prevIsUndefined(int backOffset, boolean forPrevStep) {
    return prevUndefined;
}

--------------------------------------------- RealVariable.java

public abstract class RealVariable {
    protected double expectedMin;
    protected double expectedMax;
    protected double value;
    protected boolean undefined;
    protected double prevStepValue;
    protected boolean prevUndefined;
    protected long timestamp;
    protected long prevTimeStamp;
    // add a new defined value as the current value of the variable
    public void addNewValue(double newValue) {
        prevTimeStamp = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        value = newValue;
        prevUndefined = undefined;
        undefined = false;
    }
    // add a new undefined value as the current value of the variable
    public void addNewValue() {
        prevTimeStamp = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        prevUndefined = undefined;
        undefined = true;
    }
    public long getTime() {return timestamp;}
    public double getValue() {
        if (undefined) throw new RuntimeException("Illegal undefined value access");
        else return value;
    }
    public double prevStepValue() {
        if (timestamp == StateMachine.systemTime) {
            // implementation
        }
    }
}
if (prevUndefined) throw new RuntimeException("Illegal undefined value access");
else return prevStepValue;
}
else {
  if (undefined) throw new RuntimeException("Illegal undefined value access");
  return value;
}

public long prevStepTime() {
  return timestamp == StateMachine.systemTime ? prevTimeStamp : timestamp;
}

public boolean isChanged() {
  if (undefined && prevUndefined) return false;
  else if (undefined || prevUndefined) return true;
  else return value != prevStepValue ? true : false;
}

public boolean isAssigned() {
  return timestamp == StateMachine.systemTime;
}

public boolean isUndefined() {
  return undefined;
}

public boolean prevIsUndefined(int backoffset, boolean forPrevStep) {
  return prevUndefined;
}

public abstract class BoolVariable {
  protected boolean value;
  protected boolean undefined;
  protected boolean prevStepValue;
  protected boolean prevUndefined;
  protected long timestamp;
  protected long prevTimeStamp;

  public void addNewValue(boolean newValue) {
    prevTimeStamp = timestamp;
    timestamp = StateMachine.systemTime;
    prevStepValue = value;
    value = newValue;
    prevUndefined = undefined;
    undefined = false;
  }

  public void addNewValue() {
    prevTimeStamp = timestamp;
    timestamp = StateMachine.systemTime;
    prevStepValue = value;
    prevUndefined = undefined;
    undefined = true;
  }

  public long getTime() {return timestamp;}

  public boolean getValue() {
    if (undefined) throw new RuntimeException("Illegal undefined value access");
    else return value;
  }

  public boolean prevStepValue() {

Test-Case Generation using an
Explicit State Model Checker Final Report Page 14 of 44
if (timestamp == StateMachine.systemTime) {
    if (prevUndefined) throw new RuntimeException("Illegal undefined value access");
    else return prevStepValue;
} else {
    if (undefined) throw new RuntimeException("Illegal undefined value access");
    return value;
}

public long prevStepTime() {
    return timestamp == StateMachine.systemTime ? prevTimeStamp : timestamp;
}

public boolean isChanged() {
    if (undefined && prevUndefined) return false;
    else if (undefined || prevUndefined) return true;
    else return value != prevStepValue ? true : false;
}

public boolean isAssigned() {
    return timestamp == StateMachine.systemTime;
}

public boolean isUndefined() {
    return undefined;
}

public boolean prevIsUndefined(int backOffset, boolean forPrevStep) {
    return prevUndefined;
}

public abstract class EnumVariable {
    protected String value;
    protected boolean undefined;
    protected String prevStepValue;
    protected boolean prevUndefined;
    protected long timestamp;
    protected long prevTimeStamp;

    public void addNewValue(String newValue) {
        prevTimeStamp = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        value = newValue;
        prevUndefined = undefined;
        undefined = false;
    }

    public void addNewValue() {
        prevTimeStamp = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        prevUndefined = undefined;
        undefined = true;
    }

    public long getTime() {return timestamp;}

    public String getValue() {
        if (undefined) throw new RuntimeException("Illegal undefined value access");
        else return value;
    }
}
public String prevStepValue() {
    if (timestamp == StateMachine.systemTime) {
        if (prevundefined) throw new RuntimeException("Illegal undefined value access");
        else return prevStepValue;
    }
    else {
        if (undefined) throw new RuntimeException("Illegal undefined value access");
        return value;
    }
}

public long prevStepTime() {
    return timeStamp == StateMachine.systemTime ? prevTime : timestamp;
}

public boolean isChanged() {
    if (undefined && prevUndefined) return false;
    else if (undefined || prevUndefined) return true;
    else return value != prevStepValue ? true : false;
}

public boolean isAssigned() {
    return timeStamp == StateMachine.systemTime;
}

public boolean isUndefined() {
    return undefined;
}

public boolean prevIsUndefined() {
    return undefined;
}

public abstract class TimeVariable {
    protected long value;
    protected boolean undefined;
    protected long prevStepValue;
    protected boolean prevUndefined;
    protected long timestamp;
    protected long prevTime;

    public void addNewValue(long newValue) {
        prevTime = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        value = newValue;
        prevUndefined = undefined;
        undefined = false;
    }

    public void addNewValue() {
        prevTime = timestamp;
        timestamp = StateMachine.systemTime;
        prevStepValue = value;
        prevUndefined = undefined;
        undefined = true;
    }

    public long getTime() { return timestamp; }

    public long getValue() {
        if (undefined) throw new RuntimeException("Illegal undefined value access");
        else return value;
    }
}
```java
public long prevStepValue() {
    if (timestamp == StateMachine.systemTime) {
        if (prevundefined) throw new RuntimeException("Illegal undefined value access");
        else return prevStepValue;
    } else {
        if (undefined) throw new RuntimeException("Illegal undefined value access");
        return value;
    }
}

public long prevStepTime() {
    return timestamp == StateMachine.systemTime ? prevTimeStamp : timestamp;
}

public boolean isChanged() {
    if (undefined && prevUndefined) return false;
    else if (undefined || prevUndefined) return true;
    else return value != prevStepValue ? true : false;
}

public boolean isAssigned() {
    return timestamp == StateMachine.systemTime;
}

public boolean isUndefined() {
    return undefined;
}

public boolean prevIsUndefined(int backOffset, boolean forPrevStep) {
    return prevundefined;
}
```

Note that when a variable values is accessed using the `getValue()` method, if the actual value is undefined, an exception will be thrown. Therefore, a variable value access should be properly undefined-guarded, meaning that we should assure that the variable is not undefined (using the `isUndefined()` or `prevIsUndefined()` methods) before accessing its value.

### 2.5.1 Input variables

With the above supporting library classes, RSML* input variables can be translated as the following:

```
in_variable_def: IN VARIABLE IDENTIFIER array_decl ':' type_ref
    INITIAL_VALUE ':' expression
    variable_numeric_decl
    classification_def
END IN VARIABLE;
```

Test-Case Generation using an
Explicit State Model Checker Final Report
public class Raltitude extends IntVariable {
    public Raltitude()
    {
        expectedMin = 0;
        expectedMax = 40000;
        addNewValue(); // initialize the variable to UNDEFINED
    }
}

2.5.2 State variables

For state variable definitions, the major task is to translate the transitions and transition conditions. Below we show the translation of a state variable definition. This state variable does not have a parent state, so this field and related operations are not translated. For state variables that have a parent, we flatten the hierarchy to achieve the same effect. Furthermore, this translation does not support arrays and it is assumed that the variable names must be unique (the path information is ignored when a state variable is referenced). Hierarchy flattening and variable renaming (rename variables if necessary to make them unique) are preprocessing passes that already exist in the NIMBUS framework.

```
state_variable_def:
    STATE_VARIABLE IDENTIFIER array_decl ':' variable_type_decl
    PARENT '::' parent_decl
    INITIAL_VALUE '::' expression
    variable_numeric_decl
    classification_def
    case_list
END STATE_VARIABLE

variable_type_decl: type_ref | VALUES '::' '{' enum_element_list '}'

parent_decl : NONE | parent_name_path;
parent_name_path : IDENTIFIER | parent_name_path '::' IDENTIFIER

case_list : /* EMPTY */ | case_list case;
    case : EQUALS expression IF condition
          | TRANSITION expression TO expression IF condition;

    condition : TABLE row_list END TABLE | /*boolean*/ expression;
    row_list : expression '::' truth_value_list '::';
              | row_list expression '::' truth_value_list '::';

    truth_value_list : truth_value | truth_value truth_value_list;
    truth_value: 'T' | 'F' | '.' | '*' |
```

```
STATE_VARIABLE AltitudeStatus :
    VALUES : { Unknown, Above, Below, AltitudeBad }
```
public class Raltitudestatus extends EnumVariable {
    public Raltitudestatus() {
        addNewValue(StateMachine._altitudestatus_type.unknown);
    }

    public void evaluate() {
        if (StateMachine._ivreset.getValue()) {
            addNewValue(StateMachine._altitudestatus_type.unknown);
            return;
        }

        if (Function._belowthreshold() &&
            Function._altitudequalityok() &&
            !StateMachine._ivreset.getValue()) {
            addNewValue(StateMachine._altitudestatus_type.below);
            return;
        }

        if (!Function._belowthreshold() &&
            Function._altitudequalityok() &&
            !StateMachine._ivreset.getValue()) {
            addNewValue(StateMachine._altitudestatus_type.above);
            return;
        }

        if (!Function._altitudequalityok() &&
            !StateMachine._ivreset.getValue()) {
        }
    }
}
addNewValue(StateMachine._atitudestatus_type.altitudebad);

if isUndefined() addNewValue();
else addNewValue(value);
}

2.6 Message Definitions

Each RSML message type is translated into a Java class, with the message fields as the instance fields. All the instance fields have default access modifier to facilitate access from other classes.

```
message_def: MESSAGE IDENTIFIER '{' field_list '}'
field_list: /* empty */ | IDENTIFIER IS type_ref
  | field_list ',' IDENTIFIER IS type_ref;
```

```
MESSAGE AltitudeMessage {Alt IS INTEGER, aq IS AltitudeQualityType}
public class Raltitudemessage {
  int _alt;
  String _aq;
}
```

2.7 Functions and macros

All functions and macro in an RSML specification can be wrapped in a Function class and defined as static methods. RSML stub functions are not supported by this translation.

```
optional_formal_parms : /* EMPTY */
  | '(' formal_parameter_list ')
macro_def : MACRO IDENTIFIER optional_formal_parms ':'
  condition
  END MACRO
function_def: FUNCTION IDENTIFIER '(' formal_parameter_list ')
  type_ref case_list
  END FUNCTION
  | STUB_FUNCTION IDENTIFIER '(' formal_parameter_list ')
  type_ref optional_expr_list
  END STUB_FUNCTION
```
2.8 Input Interfaces

There are two types of RSML* input interfaces: RECEIVE type and READ type, each having a corresponding method to perform the RECEIVE and READ action in the Java translation. Since the implementation of the `read()` method will depend on the technology we are going to interface with, it is not specified here. Below is an example translation for a RECEIVE type input interface.

```
in_interface def:
   IN_INTERFACE IDENTIFIER ':'
      MIN_SEP '': expression MAX_SEP '': expression
      INPUT_ACTION ':' in_interface_type_spec (' IDENTIFIER ')
   in_handler_list
END IN_INTERFACE
```

```
IN_INTERFACE ResetMessageInterface :
   MIN_SEP : 50 MS
   MAX_SEP : 100 MS
   INPUT_ACTION : RECEIVE(EmptyMessage)

   RECEIVE_HANDLER :
      CONDITION : TRUE
      ASSIGNMENT
         ivReset := TRUE
      END ASSIGNMENT
   END HANDLER
```

MACRO BelowThreshold() :
   TABLE
      Altitude != UNDEFINED : T;
      Altitude < AltitudeThreshold : T;
   END TABLE
END MACRO

public class Function {
   ...
   public static Value _belowthreshold() {
      boolean row1, row2;
      row1 = StateMachine._altitude.getValue().notUndefined().getBoolValue();
      row2 = StateMachine._altitude.getValue().lessThan(StateMachine._altitudethreshold).getBoolValue();
      return new BooleanValue(row1 && row2);
   }
}"
/** an example for RECEIVE IN INTERFACE */

public class Rresetmessageinterface {
    int minSep;
    int maxSep;
    private long timeStamp;
    private Remptymessage message;

    public Rresetmessageinterface() {
        minSep = 50;
        maxSep = 100;
    }

    public void receiveMessage(Remptymessage m) {
        message = m;
        timeStamp =StateMachine.systemTime;
    }

    public boolean isAssigned() {
        return timeStamp == StateMachine.systemTime;
    }

    public long last100() {
        return timeStamp;
    }

    public boolean executeHandlers() {
        boolean flag = false;

        if (isAssigned()) { // for RECEIVE type HANDLERS
            if (receiveHandler1()) flag = true;

            // other RECEIVE handlers
        }
        else {
            // non-RECEIVE handlers
            if (handler1()) flag = true;
        }

        return flag;
    }

    private boolean receiveHandler1() {
        StateMachine._ivreset.addValue(true);
        return true;
    }

    private boolean handler1() {
        // other non-RECEIVE handlers
    }
```java
if (StateMachine._ivreset.prevStepValue() == true) {
   StateMachine._ivreset.addNewValue(false);
    return true;
} else return false;
}
```

For READ type input interfaces, the `receive()` method should be replaced by a `read()` method. In addition, there is no concept of assignment for a READ type handler, thus the `isAssigned()` method should not be present in the translation.

### 2.9 Output Interfaces

There are two types of output interfaces: SEND type and PUBLISH type, each having a corresponding method to perform the SEND and PUBLISH action. Since the implementation of these methods will depend on the technology we are going to interface with, they are left empty by the translation. For testing purpose, we may insert print statements to display the messages to be sent.

```plaintext
out_interface_def:
    OUT_INTERFACE IDENTIFIER ':'
    MIN_SEP ':' expression MAX_SEP ':' expression
    OUTPUT_ACTION ':' out_interface_type_spec '('IDENTIFIER ')' output_handler_list
END OUT INTERFACE
```

```plaintext
OUT INTERFACE DOICommandInterface :
MIN_SEP : 50 MS
MAX_SEP : 100 MS
OUTPUT_ACTION : SEND(DOICommandMessage)
HANDLER :
    CONDITION :
    TABLE
    DOI IN_STATE AttemptingOn : T;
    PREV_STEP(DOI) IN_STATE AttemptingOn : F;
    END TABLE
    ASSIGNMENT
    command := On
    END ASSIGNMENT
    ACTION : SEND
END HANDLER
END OUT INTERFACE
```
public class Rdoicommandinterface {
    int minSep;
    int maxSep;
    private Rdoicommandmessage message;
    private long timeStamp;

    public Rdoicommandinterface() {
        minSep = 50;
        maxSep = 100;
    }

    public void send() {
        // to be filled
        System.out.print("... Sending DOICommandMessage : ");
        System.out.println(message._command);
        timeStamp =StateMachine.systemTime;
    }

    public long lastIO() { return timeStamp; }

    public void executeHandlers() {
        handler1();
    }

    public void handler1() {
        if (StateMachine._doi.getValue() == StateMachine._doi_type.attemptingon &&
            !StateMachine._doi.prevStepValue() ==
            StateMachine._doi_type.attemptingon) {
            message = new Rdoicommandmessage();
            message.command = StateMachine._doistatustype.on;
            send();
        }
    }
}

2.10 State Machine

The State Machine class instantiates all the RSML components (except messages) in a specification as static class variables and has a run() method to increment system time and evaluate the state transitions in every loop. In addition, there is a receive method for each RECEIVE input interface that can be called from outside of the system so that StateMachine is the only class that interfaces with the inputs. The granularity of time is determined by halving the smallest Minimum Separation of all the interfaces. Below is an example State Machine class that should be generated for the ASW example.

```
public class StateMachine {
    // system clock
    static long systemTime;
    static int timeStep = 25; // determined by the minimal minSep
    static long lastSystemTime;

    // constants

    Test-Case Generation using an Explicit State Model Checker Final Report
static final int _altitudethreshold = 20000;
static final int _histeresis = 1000;
static final int _doidelay = 2000;

// user-defined types
static final Rdoistatustype _doistatustype = new Rdoistatustype();
static final R_altitudestatus_type _altitudestatus_type = new R_altitudestatus_type();
static final R doi_type _doi_type = new R doi_type();
static final R_aswopmodes_type _aswopmodes_type = new R_aswopmodes_type();
static final RAltitudequalitytype _altitudequalitytype = new RAltitudequalitytype();
static final Rinhibittype _inhibittype = new Rinhibittype();

// input interfaces
static final Raltitudemessageinterface _altitudemessageinterface = new Raltitudemessageinterface();
static final Rdoistatusmessageinterface _doistatusmessageinterface = new Rdoistatusmessageinterface();
static final Rinhibitmessageinterface _inhibitmessageinterface = new Rinhibitmessageinterface();
static final Rresetmessageinterface _resetmessageinterface = new Rresetmessageinterface();

// output interfaces
static final Rdoicommandinterface _doicommandinterface = new Rdoicommandinterface();
static final Rfaultdetectioninterface _faultdetectioninterface = new Rfaultdetectioninterface();

// input variables
static final Raltitude _altitude = new Raltitude();
static final RAltitudequality _altitudequality = new RAltitudequality();
static final Rdoistatus _doistatus = new Rdoistatus();
static final Rinhibitsignal _inhibitsignal = new Rinhibitsignal();
static final Rivreset _ivreset = new Rivreset();

// state variables
static final Rfaultdetectedvariable _faultdetectedvariable = new Rfaultdetectedvariable();
static final Raltitudestatus _altitudestatus = new Raltitudestatus();
static final Rdoi _doi = new Rdoi();
static final Rdoilastchange _doilastchange = new Rdoilastchange();
static final Raswopmodes _aswopmodes = new Raswopmodes();

public static void run() {
    boolean flag = false;

    // test input interfaces, order is important
    if (_resetmessageinterface.executeHandlers()) flag = true;
    if (_inhibitmessageinterface.executeHandlers()) flag = true;
    if (_doistatusmessageinterface.executeHandlers()) flag = true;
    if (_altitudemessageinterface.executeHandlers()) flag = true;

    if (flag) {
        // evaluate state variables in order
        _altitudestatus.evaluate();
        _doi.evaluate();
        _doilastchange.evaluate();
        _aswopmodes.evaluate();
        _faultdetectedvariable.evaluate();

        // execute output interfaces
        _faultdetectioninterface.executeHandlers();
        _doicommandinterface.executeHandlers();
    }

    lastSystemTime = systemTime;
    systemTime += timeStep;
}
public static void altitudemessageinterfaceReceive(Raltitudemessage message) {
    _altitudemessageinterface.receiveMessage(message);
    run();
}

public static void doistatusmessageinterfaceReceive(Rdoistatusmessage message) {
    _doistatusmessageinterface.receiveMessage(message);
    run();
}

public static void inhibitmessageinterfaceReceive(Rinhibitmessage message) {
    _inhibitmessageinterface.receiveMessage(message);
    run();
}

public static void resetmessageinterfaceReceive(Remptymessage message) {
    _resetmessageinterface.receiveMessage(message);
    run();
}

---------------------------------------------------------------
3 Bibliography

Appendix A  - A Flight Guidance Case Example

Below we show the RSML-e model ToyFGS00 and its complete Java translation generated automatically by the Java translator.

A.1 The ToyFGS00 RSML-e Model

```
/* Copyright © 2001 Rockwell Collins, Inc. All rights reserved. */

/* Toy FGS Requirements Specification Version 0 */
/* Version 0 consists of a simple Flight Director and the lateral */
/* modes of Roll Hold (ROLL) and Heading Hold (HDG). */

/* */

/* Section (Basic Definitions) */
/* This section defines types and constants that are used throughout the specification. */

/* The following types are the states of the hierarchical modes defined in the specification. */

TYPE_DEF On_Off {Off, On}
TYPE_DEF Base_State {Cleared, Selected}
TYPE_DEF Selected_State {Armed, Active}

/* */

/* Flight Director (FD) */
The Flight Director (FD) displays the pitch and roll guidance commands to the pilot and copilot on the Primary Flight Display. This component defines when the Flight Director guidance cues are turned on and off.

/* */

/* */

MACRO When_Turn_FD_On():
    TABLE
        When_FD_Switch_Pressed_Seen(): T *;
        When_Lateral_Mode_Manually_Selected(): T;
    END TABLE
    Purpose: *L This event defines when the onside FD is to be turned on (i.e., displayed on the PFD). *L*
END MACRO
```
MACRO When_Turn_FD_Off(): When_FD_Switch_Pressed_Seen()

    Purpose: This event defines when the onside FD is to be turned off (i.e., removed from the PFD).
END MACRO

MACRO When_Lateral_Mode_Manually_Selected():
    When_HDG_Switch_Pressed_Seen()

    Purpose: This event defines when a lateral mode is manually selected.
END MACRO

STATE_VARIABLE Onside_FD: On-Off
    PARENT: None
    INITIAL_VALUE: Off
    CLASSIFICATION: State

    Transition Off TO On IF When_Turn_FD_On()
    Transition On TO Off IF When_Turn_FD_Off()

    Purpose: This variable maintains the current state of the onside Flight Director.
END STATE_VARIABLE

/*L \sectionp{Flight Modes}
The flight modes determine which modes of operation of the FGS are active and armed at any given moment. These in term determine which flight control laws are generating the commands directing the aircraft along the lateral (roll) and vertical (pitch) axes. This component encapsulates the definitions of the lateral and vertical modes and defines how they are synchronized.
L*/

MACRO When_Turn_Modes_On(): Onside_FD = On

    Purpose: This event defines when the flight modes are to be turned on and displayed on the PFD.
END MACRO

MACRO When_Turn_Modes_Off(): Onside_FD = Off

    Purpose: This event defines when the flight modes are to be turned off and removed from the PFD.
END MACRO

STATE_VARIABLE FD_Cues_On: Boolean
    PARENT: None
    INITIAL_VALUE: FALSE
    CLASSIFICATION: CONTROLLED

    EQUALS Onside_FD = On IF TRUE

Test-Case Generation using an
Explicit State Model Checker Final Report
Purpose: Indicates if the FD Guidance cues should be displayed on the PFD.

END STATE_VARIABLE

STATE_VARIABLE Mode_Annunciations_On: Boolean
PARENT: NONE
INITIAL_VALUE: FALSE
CLASSIFICATION: CONTROLLED

EQUALS Modes = On IF TRUE

Purpose: Indicates if the mode annunciations should be displayed on the PFD.

END STATE_VARIABLE

READ STATE_VARIABLE Modes: On_Off
PARENT: None
INITIAL_VALUE: Off
CLASSIFICATION: State

TRANSITION Off TO On IF When_Turn_Modes_On()
TRANSITION On TO Off IF When_Turn_Modes_Off()

Purpose: This variable maintains the current state of whether the mode annunciations are turned on or off.

END STATE_VARIABLE

---

The lateral modes select the control laws generating commands directing the aircraft along the lateral, or roll, axis. This component encapsulates the specific lateral modes present in this aircraft and defines how they are synchronized.

---

MACRO When-Nonbasic-Lateral-Mode-Activated(): When_HDG_Activated()

Purpose: This event occurs when a new lateral mode other than the basic mode becomes active. It is used to deselect active or armed modes.

Comment: Basic mode is excluded to avoid a cyclic dependency in the definition of this macro.

END MACRO

MACRO Is_No_Nonbasic_Lateral_Mode_Active(): NOT Is_HDG_Active

Purpose: This condition indicates if no lateral mode except basic mode is active. It is used to trigger the activation of the basic lateral mode.

Comment: Basic mode is excluded to avoid a cyclic dependency in the definition of this macro.

END MACRO

---

Test-Case Generation using an Explicit State Model Checker Final Report
In Roll Hold mode the FGS generates guidance commands to hold the aircraft at a fixed bank angle. Roll Hold mode is the basic lateral mode and is always active when the modes are displayed and no other lateral mode is active.

MACRO Select_Roll():
  TABLE
    Is_No_Nonbasic_Lateral_Mode_Active(): T;
    Modes = On: T;
  END TABLE

  Purpose: &*L This event defines when Roll Hold mode is to be selected. Roll Hold mode is the basic, or default, mode and is selected whenever the mode annunciations are on and no other lateral mode is active. L*6

  Comment: &*L To avoid cyclic dependencies, the only way to select Roll Hold mode is to deselect the active lateral mode, which will automatically activate Roll Hold. L*6

END MACRO

MACRO Deselect_Roll():
  TABLE
    When_Nonbasic_Lateral_Mode_Activated(): T *;
    When(Modes = Off): * T;
  END TABLE

  Purpose: &*L The event defines when Roll Hold mode is to be deselected. This occurs when a new lateral mode is activated or the modes are turned off. L*6

END MACRO

STATE_VARIABLE Is_Roll_Selected: Boolean
  PARENT: NONE
  INITIAL_VALUE: FALSE
  CLASSIFICATION: CONTROLLED

  EQUALS ..ROLL = Selected IF TRUE

  Purpose: &*L Indicates if Roll Mode is selected. L*6

END STATE_VARIABLE

STATE_VARIABLE Is_Roll_Active: Boolean
  PARENT: NONE
  INITIAL_VALUE: FALSE
  CLASSIFICATION: CONTROLLED

  EQUALS ..ROLL = Selected IF TRUE

  Purpose: &*L Indicates if Roll Mode is active. L*6

  Comment: &*L Even though ROLL Selected and ROLL Active are the same thing, this variable is introduced to maintain a common interface across modes. L*6

END STATE_VARIABLE

Test-Case Generation using an
Explicit State Model Checker Final Report
STATE-VARIABLE ROLL: Base_State
  PARENT: Modes.On
  INITIAL_VALUE: UNDEFINED
  CLASSIFICATION: State

  TRANSITION UNDEFINED TO Cleared IF NOT Select_ROLL()
  TRANSITION UNDEFINED TO Selected IF Select_ROLL()
  TRANSITION Cleared TO Selected IF Select_ROLL()
  TRANSITION Selected TO Cleared IF Deselect_ROLL()

  Purpose: &*L This variable maintains the current base state of Roll Hold mode, i.e., whether it is cleared or selected. L*

END STATE-VARIABLE

MACRO Select_HDG(): When_HDG_Switch_Pressed_Seen()

  Purpose: &*L This event defines when Heading Select mode is to be selected. L*
END MACRO

MACRO Deselect_HDG():
  TABLE
    When_HDG_Switch_Pressed_Seen()           : T * *
    When_NonBasic_Lateral_Mode_Activated()  : * T *
    When(Modes = Off)                       : * * T

  Purpose: &*L This event defines when Heading Select mode is to be deselected. L*
END MACRO

STATE-VARIABLE Is_HDG_Selected: Boolean
  PARENT: NONE
  INITIAL_VALUE: FALSE
  CLASSIFICATION: CONTROLLED

  EQUALS ..HDG = Selected IF TRUE

  Purpose: &*L Indicates if HDG Mode is selected. L*
END STATE-VARIABLE

STATE-VARIABLE Is_HDG_Active: Boolean
  PARENT: NONE
  INITIAL_VALUE: FALSE
  CLASSIFICATION: CONTROLLED

  EQUALS ..HDG = Selected IF TRUE

  Purpose: &*L Indicates if HDG Mode is active. L*
END STATE-VARIABLE

Test-Case Generation using
Explicit State Model Checker Final Report
Comment: &*L Even though HDG Selected and HDG Active are
the same thing, this variable is introduced to maintain a
common interface across modes. L*

END STATE_VARIABLE

MACRO When_HDG_Activated():

TABLE

Select_HDG() : T;
PREV_STEP(. . .HDG) = Selected : F;
END TABLE

Purpose: &L This signal occurs when Heading Select mode
is activated. L*

Comment: &L This event is defined this way to avoid
circular dependencies. It would be preferable to define
it as When(HDG = Selected). L*

END MACRO

PEND MACRO

STATE_VARIABLE HDG : Base_State
PARENT : Modes.On
INITIAL_VALUE : UNDEFINED
CLASSIFICATION : State

Purpose: &L This variable maintains the current base
state of Heading Select mode, i.e., whether it is
cleared or selected. L*

TRANSITION UNDEFINED TO Cleared IF NOT Select_HDG()
TRANSITION UNDEFINED TO Selected IF Select_HDG()
TRANSITION Cleared TO Selected IF Select_HDG()
TRANSITION Selected TO Cleared IF Deselect_HDG()

END STATE_VARIABLE

PEND MACRO

PEND MACRO

PEND MACRO

PEND MACRO
MACRO No_Higher_Event_Than_FD_Switch_Pressed():
    TABLE
        When_HDG_Switch_Pressed() : F;
        No_Higher_Event_Than_HDG_Switch_Pressed() : T;
    END TABLE

Purpose: "This event occurs when no event with a priority higher than pressing the FD switch has occurred."

END MACRO

MACRO When_HDG_Switch_Pressed(): When(HDG_Switch = ON)

Purpose: "This event indicates when the HDG switch is pressed."

Comment: "This is redefined as a macro to simplify verification."

END MACRO

MACRO When_HDG_Switch_Pressed_Seen():
    TABLE
        When_HDG_Switch_Pressed() : T;
        No_Higher_Event_Than_HDG_Switch_Pressed() : T;
    END TABLE

Purpose: "This event indicates when the HDG switch pressed and no higher priority event has occurred."

END MACRO

MACRO No_Higher_Event_Than_HDG_Switch_Pressed(): TRUE

Purpose: "This event occurs when no event with a priority higher than pressing the HDG switch has occurred."

END MACRO

/*L \encapsulated L*/

TYPE_DEF Switch {OFF, ON}
TYPE_DEF Lamp {OFF, ON}

/*L Holds the last sensed position of the FD switch associated with this FGS."

END IN_VARIABLE

/*L Holds the last sensed position of the HDG switch."

END IN_VARIABLE

/*L Holds the last sensed position of the HDG switch."

END IN_VARIABLE

/*L Holds the last sensed position of the HDG Lamp."

END IN_VARIABLE
STATE_VARIABLE HDG_Lamp: Lamp
  PARENT : NONE
  INITIAL VALUE : OFF
  CLASSIFICATION: CONTROLLED
  
  EQUALS ON IF Is_HDG_Selected
  EQUALS OFF IF NOT Is_HDG_Selected

  Purpose: #L Indicates if the HDG switch lamp on the FCP should be on or off. L#

END STATE_VARIABLE

This section defines the physical interface for all inputs to the FGS. The input variables associated with these fields are defined in the part of the specification to which they are logically related.

MESSAGE This_Input_Msg (FdSwi IS Switch, HdgSwi IS Switch)

IN_INTERFACE This_Input:
  MIN_SEP : UNDEFINED
  MAX_SEP : UNDEFINED

  INPUT_ACTION : READ(This_Input_Msg)
  HANDLER:
    CONDITION : TRUE
    ASSIGNMENT
      FD_Switch := FdSwi,
      HDG_Switch := HdgSwi
    END ASSIGNMENT
  END HANDLER
END IN_INTERFACE

This section defines the physical interface for all outputs from the FGS. The output variables associated with these fields are defined in the part of the specification to which they are logically related.

MESSAGE This_Output_Msg (FdOn IS Boolean, FGSActive IS Boolean, HdgLamp IS Lamp, HdgSel IS Boolean, ModesOn IS Boolean, RollSel IS Boolean)

OUT_INTERFACE This_Output:
  MIN_SEP : UNDEFINED
  MAX_SEP : UNDEFINED

  OUTPUT_ACTION : PUBLISH(This_Output_Msg)
  HANDLER:
    CONDITION : TABLE
      CHANGED(FD_Cues_On) : T * * *;

Test-Case Generation using an Explicit State Model Checker Final Report
A.2 TheToyFGS00 Translated Java Code

```java
public class Function {
    public static boolean _When_Turn_FD_On()
    {
        return (Function._When_FD_Switch_Pressed_Seen()) ||
            (Function._When_Lateral_Mode_Manually_Selected());
    }

    public static boolean _When_Turn_FD_Off()
    {
        return (Function._When_FD_Switch_Pressed_Seen());
    }

    public static boolean _When_Turn_Modes_On()
    {
        return ((StateMachine._Onside_FD.getValue()).equals((ROn_0ff.0n)));
    }

    public static boolean _When_Lateral_Mode_Manually_Selected()
    {
        return (Function._When_HDG_Switch_Pressed_Seen());
    }

    public static boolean _When_Turn_Modes_Off()
    {
        return ((StateMachine._Onside_FD.getValue()).equals((ROn_0ff.0ff)));
    }

    public static boolean _When_Nonbasic_Lateral_Mode_Activated()
    {
        return (Function._When_HDG_Activated());
    }

    public static boolean _Select_ROLL()
    {
        return (Function._Is_No_Nonbasic_Lateral_Mode_Active()) &&
            ((StateMachine._Modes.getValue()).equals((ROn_0ff.0n)));
    }

    public static boolean _Deselect_ROLL()
    {
        return (Function._When_Nonbasic_Lateral_Mode_Activated()) ||
            !((StateMachine._Modes.prevStepValue()).equals((ROn_0ff.0ff)) &&
            ((StateMachine._Modes.getValue()).equals((ROn_0ff.0ff))));
    }

    public static boolean _Select_HDG()
    {
        return (Function._When_HDG_Switch_Pressed_Seen());
    }
}
```

Test-Case Generation using an
Explicit State Model Checker Final Report
public static boolean -Deselect_HDG() {
    return (Function._When_HDG_Switch_Pressed_Seen()) ||
    (Function._When_HDG_Modes_Activated()) ||
    (!(StateMachine._HDG_Switch.prevStepValue()).equals((RBase_State.Selected))) &&
    (StateMachine._HDG_Switch.getValue()).equals((RBase_State.Selected));
}

public static boolean -Is_No_Nonbasic_Lateral_Mode_Active() {
    return (!StateMachine._Is_HDG_Active.getValue());
}

public static boolean -When_FD_Switch_Pressed() {
    return (!(((StateMachine._FD_Switch.prevStepValue()).equals((RSwitch.ON))) &&
        (StateMachine._FD_Switch.getValue()).equals((RSwitch.ON))));
}

public static boolean -When_FD_Switch_Pressed_Seen() {
    return (Function._When_FD_Switch_Pressed()) &&
        (Function._No_Higher_Event_Than_FD_Switch_Pressed());
}

public static boolean -No_Higher_Event_Than_FD_Switch_Pressed() {
    return (!((Function._When_HDG_Switch_Pressed()) &&
        (Function._No_Higher_Event_Than_HDG_Switch_Pressed())));
}

public static boolean -When_HDG_Switch_Pressed() {
    return (!(((StateMachine._HDG_Switch.prevStepValue()).equals((RBase_State.Selected))) &&
        (StateMachine._HDG_Switch.getValue()).equals((RBase_State.Selected))));
}

public static boolean -When_HDG_Switch_Pressed_Seen() {
    return (Function._When_HDG_Switch_Pressed()) &&
        (Function._No_Higher_Event_Than_HDG_Switch_Pressed());
}

public static boolean -No_Higher_Event_Than_HDG_Switch_Pressed() {
    return true;
}

public static boolean -When_HDG_Activated() {
    return (Function._Select_HDG()) ||
        (StateMachine._HDG.prevStepValue()).equals((RBase_State.Selected)));
}

RbaseState.java

// RSML user-defined enumerated type Base_State
public class RbaseState {
    public static final String Cleared = "Cleared";
    public static final String Selected = "Selected";
}

RFD_Cues_On.java

// RSML state variable FD_Cues_On
public class RFD_Cues_On extends BoolVariable {
    public RFD_Cues_On() {
        addNewValue(false);
    }

    public void evaluate() {
        if (true) {
            // Your code here
        }
    }

Test-Case Generation using an Explicit State Model Checker Final Report
public class RHDG extends EnumVariable {
    public RHDG() {
        addNewValue();
    }
    public void evaluate() {
        if (!StateMachine._Mode.getValue().equals((ROn_Off.On))) {
            addNewValue();
        } else addNewValue(value);
    }
}

public class RFD_Switch extends EnumVariable {
    public RFD_Switch() {
        addNewValue();
    }
    public void evaluate() {
        if (!StateMachine._Mode.getValue().equals((ROn_Off.On))) {
            addNewValue();
        } else addNewValue(value);
    }
}

public class RHDG_Lamp extends EnumVariable {
    public RHDG_Lamp() {
        addNewValue((RLamp.OFF));
    }
    public void evaluate() {
        if (!StateMachine._Is_HDG_Selected.getValue()) {
            addNewValue((RLamp.OFF));
        } else addNewValue((RLamp.ON));
    }
}
```java
public void evaluate0 {
    if (true) {
        addNewValue(((StateMachine._HDG.getValue()).equals((RBase-State.Selected)));
        return;
    }
    if (undefined) addNewValue();
    else addNewValue(value);
}
```

---

```java
// RSM input variable HDG_Switch
public class HDG_Switch extends EnumVariable {
    public HDG_Switch() {
        addNewValue();
    }
}
```

---

```java
// RSM state variable Is_HDG_Active
public class Is_HDG_Active extends BoolVariable {
    public Is_HDG_Active() {
        addNewValue(false);
    }
    public void evaluate() {
        if (true) {
            addNewValue(((StateMachine._HDG.getValue()).equals((RBase_State.Selected))));
            return;
        }
        if (undefined) addNewValue();
        else addNewValue(value);
    }
}
```

---

```java
// RSM state variable Is_HDG_Selected
public class Is_HDG_Selected extends BoolVariable {
    public Is_HDG_Selected() {
        addNewValue(false);
    }
    public void evaluate() {
        if (true) {
            addNewValue(((StateMachine._HDG.getValue()).equals((RBase_State.Selected))));
            return;
        }
        if (undefined) addNewValue();
        else addNewValue(value);
    }
}
```

---

```java
// RSM state variable Is_Roll_Active
public class Is_Roll_Active extends BoolVariable {
    public Is_Roll_Active() {
        addNewValue(false);
    }
}
```
public void evaluate() {
    if (true) {
        addNewValue(((StateMachine._ROLL.getValue()).equals((RBase_State.Selected))));
        return;
    }
    if (undefined) addNewValue();
    else addNewValue(value);
}

public class RIs_ROLL_Selection extends BoolVariable {
    public RIs_ROLL_Selection() {
        addNewValue(false);
    }
    public void evaluate() {
        if (true) {
            addNewValue(((StateMachine._ROLL.getValue()).equals((RBase_State.Selected))));
            return;
        }
        if (undefined) addNewValue();
        else addNewValue(value);
    }
}

// RSML state variable Is_ROLL_Selected
public class RIs_ROLL_Selection extends BoolVariable {
    public RIs_ROLL_Selection() {
        addNewValue(false);
    }
    public void evaluate() {
        if (true) {
            addNewValue(((StateMachine._ROLL.getValue()).equals((RBase_State.Selected))));
            return;
        }
        if (undefined) addNewValue();
        else addNewValue(value);
    }
}

public class RLamp {
    public static final String OFF = "OFF";
    public static final String ON = "ON";
}

// RSML state variable Mode_Annunciations_On
public class RMode_Annunciations_On extends BoolVariable {
    public RMode_Annunciations_On() {
        addNewValue(false);
    }
    public void evaluate() {
        if (true) {
            addNewValue(((StateMachine._Modes.getValue()).equals((ROn_Off.On))));
            return;
        }
        if (undefined) addNewValue();
        else addNewValue(value);
    }
}

// RSML state variable Modes
public class RModes extends EnumVariable {
    public RModes() {
        addNewValue((ROn_Off.Off));
    }
    public void evaluate() {

if ((Function._When_Turn_Modes_On()) &&
(StateMachine._Modes.prevStepValue()).equals((RON_Off.Off))) {
    addNewValue((RON_Off.On));
    return;
}
if ((Function._When_Turn_Modes_Off()) &&
(StateMachine._Modes.prevStepValue()).equals((RON_Off.On))) {
    addNewValue((RON_Off.Off));
    return;
}
if (undefined) addNewValue();
else addNewValue(value);

-- ROon_Off.java

// RSML user-defined enumerated type On_Off
public class ROon_Off {
    public static final String Off = "Off";
    public static final String On = "On";
}

-- ROnside_FD.java

// RSML state variable Onside_FD
public class ROnside_FD extends EnumVariable {
    public ROnside_FD() {
        addNewValue((RON_Off.Off));
    }
    public void evaluate() {
        if ((Function._When_Turn_FD_On()) &&
        (StateMachine._Onside_FD.prevStepValue()).equals((RON_Off.Off))) {
            addNewValue((RON_Off.On));
            return;
        }
        if ((Function._When_Turn_FD_Off()) &&
        (StateMachine._Onside_FD.prevStepValue()).equals((RON_Off.On))) {
            addNewValue((RON_Off.Off));
            return;
        }
        if (undefined) addNewValue();
        else addNewValue(value);
    }
}

-- RROLL.java

// RSML state variable ROLL
public class RROLL extends EnumVariable {
    public RROLL() {
        addNewValue();
    }
    public void evaluate() {
        if (!StateMachine._Modes.isDefined() ||
        (!StateMachine._Modes.getValue()).equals((RON_Off.On))) {
            addNewValue();
            return;
        }
        if (!Function._Select_ROLL() && (StateMachine._ROLL.prevIsUndefined())) {
            addNewValue((RBase_State.Cleared));
            return;
        }
        if (Function._Select_ROLL() && (StateMachine._ROLL.prevIsUndefined())) {
            Test-Case Generation using an
Explicit State Model Checker Final Report  Page 41 of 44
```java
addNewValue((RBase_State.Selected));
return;
}
if ((Function._Select_ROLL()) &&
(StateMachine._ROLL.prevStepValue()).equals((RBase.State.Cleared))) {
    addNewValue((RBase.State.Selected));
    return;
}
if ((Function._Deselect_ROLL()) &&
(StateMachine._ROLL.prevStepValue()).equals((RBase.State.Selected))) {
    addNewValue((RBase.State.Cleared));
    return;
}
if (undefined) addNewValue();
else addNewValue(value);
}
```

---

**RSelected_State.java**

// RSML user-defined enumerated type Selected_State
public class RSelected_State {
    public static final String Armed = "Armed";
    public static final String Active = "Active";
}

---

**Rswitch.java**

// RSML user-defined enumerated type Switch
public class RSwitch {
    public static final String OFF = "OFF";
    public static final String ON = "ON";
}

---

**RThis_input.java**

// RSML input interface This_INPUT
public class RThis_input {
    public int minSep;
    public int maxSep;
    private long timestamp;
    private RThis_input_Msg message;
    public void readMessage() {
    }
    public long lastIO() (return timestamp);
    public boolean executeHandlers() {
        boolean flag = false;
        if (handler1()) flag = true;
        return flag;
    }
    private boolean handler1() {
        if (true) {
            StateMachine._FD_Switch.addNewValue(message.FdSwi);
            StateMachine._HDG_Switch.addNewValue(message.HdgSwi);
            return true;
        }
        else return false;
    }
}
```
This-Output.java

public class This-Output
{
    private RThis-Output_Msg message;
    private long timestamp;

    public void publish()
    {
        timestamp = StateMachine.systemTime;
    }

    public long lastIO() {return timestamp;)
    public void executeHandlers()
    {
        handler1();
    }

    public void handler1()
    {
        if (((StateMachine._FD_Cues_On.prevStepValue()) !=
            (StateMachine._FD_Cues_On.getValue())) ||
            ((StateMachine._HDG_Lamp.prevStepValue()) !=
                (StateMachine._HDG_Lamp.getValue())) ||
            ((StateMachine._Is_HDG_Selected.prevStepValue()) !=
                (StateMachine._Is_HDG_Selected.getValue())) ||
            ((StateMachine._Mode_Annunciations_On.prevStepValue()) !=
                (StateMachine._Mode_Annunciations_On.getValue())) ||
            ((StateMachine._Is_ROLL_Selected.prevStepValue()) !=
                (StateMachine._Is_ROLL_Selected.getValue())))
        {
            message = new RThis-Output_Msg();
            message.RollSel = (StateMachine._Is_ROLL_Selected.getValue());
            message.ModesOn = (StateMachine._Mode_Annunciations_On.getValue());
            message.HdgSel = (StateMachine._Is_HDG_Selected.getValue());
            message.HdgLamp = (StateMachine._HDG_Lamp.getValue());
            message.FGSActive = true;
            message.FdOn = (StateMachine._FD_Cues_On.getValue());
            publish();
        }
    }

    public class RThis-Input_Msg
    {
        String FdSwi;
        String HdgSwi;
    }

    public class RThis-Output_Msg
    {
        boolean FdOn;
        boolean FGSActive;
        String HdgLamp;
        boolean HdgSel;
        boolean ModesOn;
        boolean RollSel;
    }

    public static final R NimbusSystemClockReceiver NimbusSystemClockReceiver = new
    RNimbusSystemClockReceiver();
    public static final RThis-Input _This_Input = new RThis-Input();
public static final RThis_Output _This_Output = new RThis_Output();

public static final RFD_Switch _FD_Switch = new RFD_Switch();
public static final RHDG_Switch _HDG_Switch = new RHDG_Switch();

public static final ROnside_FD _Onside_FD = new ROnside_FD();
public static final RMode_Annunciations_On _Mode_Annunciations_On = new RMode_Annunciations_On();

public static final RIs_ROLL_Selected _Is_ROLL_Selected = new RIs_ROLL_Selected();
public static final RModes _Modes = new RModes();
public static final RROLL _ROLL = new RROLL();

public static final RIs_HDG_Selected _Is_HDG_Selected = new RIs_HDG_Selected();
public static final RIs_ROLL_Active _Is_ROLL_Active = new RIs_ROLL_Active();
public static final RIs_HDG_Active _Is_HDG_Active = new RIs_HDG_Active();

public static final RHDG_Lamp _HDG_Lamp = new RHDG_Lamp();

static int timestep = 0;
static long systemTime = 0;
static long lastSystemTime;

public static final void run() {
    boolean flag = false;
    if (_This_Input.executeHandlers()) flag = true;
    if (_NimbusSystemClockReceiver.executeHandlers()) flag = true;
    if (flag) {
        _Onside_FD.evaluate();
        _Modes.evaluate();
        _HDG.evaluate();
        _Is_HDG_Active.evaluate();
        _FD_Cues_On.evaluate();
        _Mode_Annunciations_On.evaluate();
        _ROLL.evaluate();
        _Is_ROLL_Selected.evaluate();
        _Is_HDG_Selected.evaluate();
        _Is_ROLL_Active.evaluate();
        _HDG_Lamp.evaluate();
        _This_Output.executeHandlers();
    }
    lastSystemTime = systemTime;
    systemTime += timestep;
}

public static final void _NimbusSystemClockReceiverReceive(RNimbusSystemClockMessageType message) {
    _NimbusSystemClockReceiver.receiveMessage(message);
    run();
}

public static final void _This_InputReceive(RThis_Input_Msg message) {
    _This_Input.readMessage(message);
    run();
}