Unified Behavior Modeling

Peter H. Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-1092

Objective

To investigate a common approach to modeling flow graphs, behavior specifications, and error behavior specifications

To simplify the three layered approach of EMV2 specifications

To simplify these specifications as annex declarations

To leverage V3 unified type system and expression language

Proposal:

Common syntax with variation in expression notation

Token semantics for use in Meta behavior (error, security)

Existing and Desired Mechanisms

AADL mode state machine

- One per component in hierarchy

AADL flow specifications

- Currently linear paths with sources and sinks
- Desired: branch/merge points

BA

- extension of mode states
- condition – transition – actions model

EMV2

- Token propagation & transformation with sources/sinks
- State-full behavior: state transformation, state-sensitive output

Agree

- Functional input to output mapping

Product line constraints

- Configuration to satisfy feature label constraint

Behavior Rules

Input to output relationships

- Stateless and stateful
- General computational expressions & token logic

Stateless

- Input condition -> <output> actions

Stateful

- Current state –[input condition]-> target state { actions }

In the general case some actions can be computational with intermediate results and some actions determine output.

Stateless behavior rules on input and output features only are the equivalent of flow specifications.

Stateful behavior rules on input/output feature only are the equivalent of mode specific flow specifications

Sources and Sinks

Source

- Generator concept to represent component internal data/Meta data sources, e.g., error events
- Identified in behavior rule condition as trigger for out action

Sink

- Behavior rule without output action
- Currently we use explicit **sink** keyword to indicate no output action

Representation of State

Use of enumeration type to define states of a state machine

- Reusable definition of state machine states
- Transitions tend to be context specific

State machine instance

- Effectively state variable holding current state
  - Currently represented as unnamed state variable in particular behavior context, e.g., @EM behavior
  - Allows for identification of state value in behavior rule without also naming a state variable
  - Could be represented explicitly to allow mix of state machines in behavior rules
    - as property, subcomponent, or expression language variable

Token System Behavior

Data or Meta data associated with input and output

- Simplified condition logic and output actions
- Context determines whether data or kind of Meta data
  - @EM for error type tokens
- Alternative: explicit identification of Meta data in input/output references
  - Inport1#Etoken or Input1@EM

Types of tokens

- Any literal: type reference, enumeration literal, numeric, string, Boolean

Token Expressions

Input constraint as condition element

- input token contained in specified set
- Input1 **in** <tokenset>

Condition expression

- Multi-literal operation of condition elements
- Operators: any, all, one of, <k> of, <k> ormore, <k> orless

Output action

- Assign token to output
- Output1 (ServiceOmission);

Currently we do not use computational actions. Condition evaluation drives output value assignment.

Named token sets

We could use inputx? and output! to align with current BA syntax

Representing Error Behavior

Separate interface consistency from behavior specification

- Interface consistency based on in/out propagation
  - Between components
  - Behavior rules cover propagation
  - Interface error propagation specification can be derived from behavior rules
- Here we focus on behavior specification

Error Behavior Rules

Currently specified in @EM { } context

- For stateful includes state machine identification
- We use type references as tokens
  - Types can have subtypes
  - Enumeration literals act as types and instances

Stateless rules

- Current error flows (source/sink/path)
- With condition logic
- Token propagation: output feature without assigned value
- Token transformation: output feature with assigned value

Stateful rules

- State transition: behavior rule without output action
- State sensitive token propagation/transformation: behavior rule possibly without target state
- State propagation: current state and action (no condition, target state)

Partial Token Behavior Specification

If no specific token behavior rule interpret general (“flow”) behavior rules

- Input to output mapping with possible input logic (multi-literal operators) and multiple output actions

If no behavior rules

- Assume any input to all output mapping

Semantics of Token Propagation

- Transformation: matching input condition to specified output value -> new token value
- Propagation: unspecified output value
  - Set of incoming token values as output token value
  - Any, one of: single output token value
  - Same for “flow” behavior rules and default behavior

Use Cases

EMV2 like behavior

- Including FTA and fault impact analysis

System models with unhandled token values or missing behavior rules

- EMV2 FTA and impact analysis assume complete specifications
  - Indicate but do not propagate unhandled token values

System models with token sources and sinks only

- This is the John Hatcliff virus/dirty word example
- Can be done via property only
- Supported by token system

Implementation Prototype

Instance model with behavior rule and generator instantiation

JGraphT based graph representation superimposed on instance model for processing

- Basic component interaction
  - Sufficient for property based token source/sink analysis and default any to all propagation
- Feature specific component interaction
  - Takes into account behavior flow logic
- Behavior rule specific component interaction
  - Takes into account token condition logic

Forward and backward token trace representation

Backward trace representation optimization transformation into fault tree, minimal cut set

Product Line Constraints

A prototype implementation

- FeatureLabels property that takes a collection of literals
  - Literals can be type references, enumeration literals, string
  - Applied to classifiers

Product line constraint on feature labels of classifiers

- Defined and associated with an instantiation root
- Disjunction of conjunctions
- All classifiers[cl| productlineconstraint.exists[discjunction| cl#FeatureLabels.contains(disjunction)]]

Usage

- Verification that configured system meets constraint
- Filter of candidate choices in interactive configuration tool