Feature Interaction Analysis with Use Case Maps

Simple Phone System

This UCM model of a Simple Phone System (provided in .jucm format) describes how a phone connection is made between an originating user and a terminating user who have their own phone agents. Each agent handles the features of its associated user. Three features are described as plug-ins:

- OCS: *Originating Call Screening* (originating user feature)
  - Filters outgoing calls to phone numbers on a screening list
- TL: *TeenLine* (originating user feature)
  - Filters outgoing calls during a certain time interval (say, from 19:00 to 23:00) from people who don't have an appropriate Personal Identification Number (PIN)
- CND: *Call Number Display* (terminating user feature)
  - Display the originator's information at on the terminating user's terminal while the phone is ringing

Each of these features is optional. Boolean variables are used to specify whether a user has subscribed to a feature or not. These variables are used in the *selection policies* associated with the dynamic stubs (i.e., the Boolean *expressions* attached to each plug-in)

Exploring the Model

1) Open the model and explore the various plug-ins
2) Look at the list of Boolean variables used in the model in the *Scenarios and Strategies* view.
   a. The *Properties* view provides details.
   b. The *sub<u>FeatureName</u>* variables are used to specify which features users are currently subscribed to.
3) Look at the *conditions* that use these variables
   a. On the branches of OR-Forks (e.g., double-click on the exit branches of the OR-fork in the OCS plug-in, and look at the condition in the *Properties* view)
   b. In the dynamic stub selection policies (e.g., right-click the Screening dynamic stub in the OriginatingFeatures plug-in, select *Edit Stub Plug-ins*, and look at the *Expression* field for each plug-in binding in the *Plugin Tree*)
   c. In these Boolean expressions, a Java syntax is used (`!`, `||`, `&&`).
4) Although this model does not show it, it is possible to *change* the content of variables in UCM responsibilities
   a. Double-click a responsibility, and code can be provided to modify any variable using a Java-like or SDL-like syntax (e.g., `x = (y && (! (z==false)));`)
   b. In the *Code Editor*, double-click a responsibility from the right panel to paste it in the code or expression.

Validation of Individual Features

Several scenario definitions are provided (see the groups and scenarios in the *Scenarios and Strategies* view). Some are for the Basic Call (the system without any additional features), some are for the Basic Call composed with one feature at a time (OCS, TeenLine, CND).

1) Explore the two BasicCall scenarios. Look at what initial values were provided to the Boolean variables during initialization. Some variables remain undefined. The definitions also describe which start points are used to trigger each scenario, and the expected end points that should be reached. These definitions can be changed by right-clicking a scenario and selecting the desired modification.
2) Select a scenario and switch to the execution view (yellow softgoal icon in the *Scenarios and Strategies* view). What does each scenario do?
3) Explore the various scenarios of OCS, TeenLine, and CND.
4) Note how some of the OCS scenarios include other scenarios (e.g., OCSbusy includes OCSsuccess and overrides a few variable initializations). Inclusions can help reusing and maintaining scenario definitions.

Feature Interaction Detection

Using many features together in a scenario may lead to unexpected situations, even if no problem was detected while validating the features individually.

1) Explore the scenarios that combine OCS with CND, and TL with CND. When subscribed to these pairs of features, are there noticeable issues?
2) What happens if a user subscribes to both OCS and TL? Add a new scenario to check this case in the FI_OCL_TL group. Do not forget to initialize the relevant variables **AND** add the required start points to trigger it (and optionally add the end points expected to be reached).
3) Make sure you have Eclipse's ***Problems*** view open.
4) If you highlight this scenario, what happens?
5) In *Windows → Preferences → jUCMNav Preferences → UCM Scenario Traversal*, uncheck the *Deterministic algorithm* box. What happens then?
6) Recheck that box.

Feature Interaction Resolution

Looking at this model, one realizes that the selection policy in the call screening stub (*Screening*) is non-deterministic: if a user subscribes to both OCS and TL, then there are *two* possible plug-ins that can be selected. This undesirable **feature interaction** causes your scenario to generate errors when running it with jUCMNav and its deterministic algorithm.

Your task is to modify this UCM model to **resolve** this conflict while leaving the other scenarios (which work) unaffected.

- You can create new responsibilities, new paths, and new variables if necessary, but keep the OCS and TL plug-ins separate. Do not add new stubs.
- You can also modify the selection policies, but not the existing scenario definitions (they should keep working in your solution, like a regression test suite).
- You are allowed to give a specific priority to one feature over another (e.g. OCS could be checked before TL, or vice-versa. Which one makes the most sense?).
- Do not forget that responsibilities can include code to provide specific values to variables (`true` or `false`) or assign the value of one variable (or of any expression of the appropriate type) to another. You can use this facility in your solution.

In the end, your conflicting scenario involving OCS and TL should be working. However, imagine a situation where you would have to resolve such interactions in the presence of hundreds of features… Does your solution scale up? This is a whole area of research!