Feature Interaction Analysis with Use Case Maps

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?

There are no noticeable issues. These feature pairs work together.

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).

Name:	OCL_TL_ActiveNotOnListSuccess
Start Points:	req (SimpleConnection), enterPIN (TeenLine)
Variable Initializations:	SubOCS = true, SubTL = true, SubCND = false, Busy = false, OnOCSlist = false, TLactive = true, PINvalid = true
End Points:	ring and ringing (SimpleConnection)

3) Make sure you have Eclipse's ***Problems*** view open.
4) If you highlight this scenario, what happens?

The scenario generates an error at the dynamic stub *Screening* because the selection policy of the stub is non-deterministic: if a user subscribes to both OCS and TL, then there are two alternative plug-ins that can be selected. This is an undesirable feature interaction.

5) In *Windows* → *Preferences* → *jUCMNav Preferences* → *UCM Scenario Traversal*, uncheck the *Deterministic algorithm* box. What happens then?

No error, but the Problems view reports that one random option was taken to solve the non-deterministic choice. If you repeat this test many times you will realize that the tool may or may not complain about an additional timer issue in TeenLine (enterPIN even not handled when OCS is selected instead of TeenLine).

6) Recheck that box.

Feature Interaction Resolution

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.
- …

There are different ways of resolving this conflict. One solution requires the following changes to the UCM model:

- Add two new Boolean variables: `ChkOCS` and `ChkTL`.
- Add responsibility initFeatures just before the Screening stub (initializes the new variables: `ChkOCS = SubOCS; ChkTL = SubTL;`
  - The new variables act like local variables.
- Change the selection policy for the Screening stub:
  - OCS plug-in: `ChkOCS`
  - TeenLine plug-in: `ChkTL && (!ChkOCS)`
  - Default plug-in: `!(ChkOCS || ChkTL)`
- Loop back to the Screening stub if `(ChkOCS || ChkTL)`, continue otherwise.
- Add variable assignments to checkOCS (`ChkOCS = false;`) on the OCS plug-in and to checkTime (`ChkTL = false;`) on the TeenLine plug-in.

This solution gives priority to OCS over TeenLine because OCS does not require user interaction. It is not worth asking the originating user for a PIN if the call is going to be blocked by OCS anyway…