USE CASE MAPS FOR THE CAPTURE AND VALIDATION OF DISTRIBUTED SYSTEMS REQUIREMENTS

Daniel Amyot
SITE, University of Ottawa

with

Luigi Logrippo, *SITE, University of Ottawa*
Raymond J.A. Buhr, *SCE Department, Carleton University*
Tom Gray, *Mitel Corporation*

Introduction

- Functional scenarios common for describing distributed systems
- Integration of scenarios and the feature interaction problem
- Capture and integrate functional requirements with Use Case Maps
- Specification and validation of UCMs with LOTOS
- Examples from the 1998 Feature Interaction Contest

Road Map

- Motivation
- Methodology
- Use Case Maps (UCMs)
- Capturing and Integrating features with UCMs
- Specification and Validation of UCMs in LOTOS
- Some Results and Lessons
- Conclusions and Future Work

Motivation

- Interactions between features *are* and *will remain* a challenging problem.
- By definition, features interact, but not always in expected or desired ways.
- Many interactions depend on how features are composed or integrated together.
- Multiple techniques proposed for detection, resolution, and avoidance at design time (static) and run time (dynamic).

Our Proposal:

1) **Avoidance** at design time with visual scenario notation called *Use Case Map*.
2) **Detection** of remaining interactions with a *LOTOS* prototype and scenario-based testing.

- First-hand experience in feature interactions with both UCMs and LOTOS.
- Use some of the best characteristics of UCMs (visual description and integration of features) and LOTOS (theory and tools for formal validation and verification)

Methodology

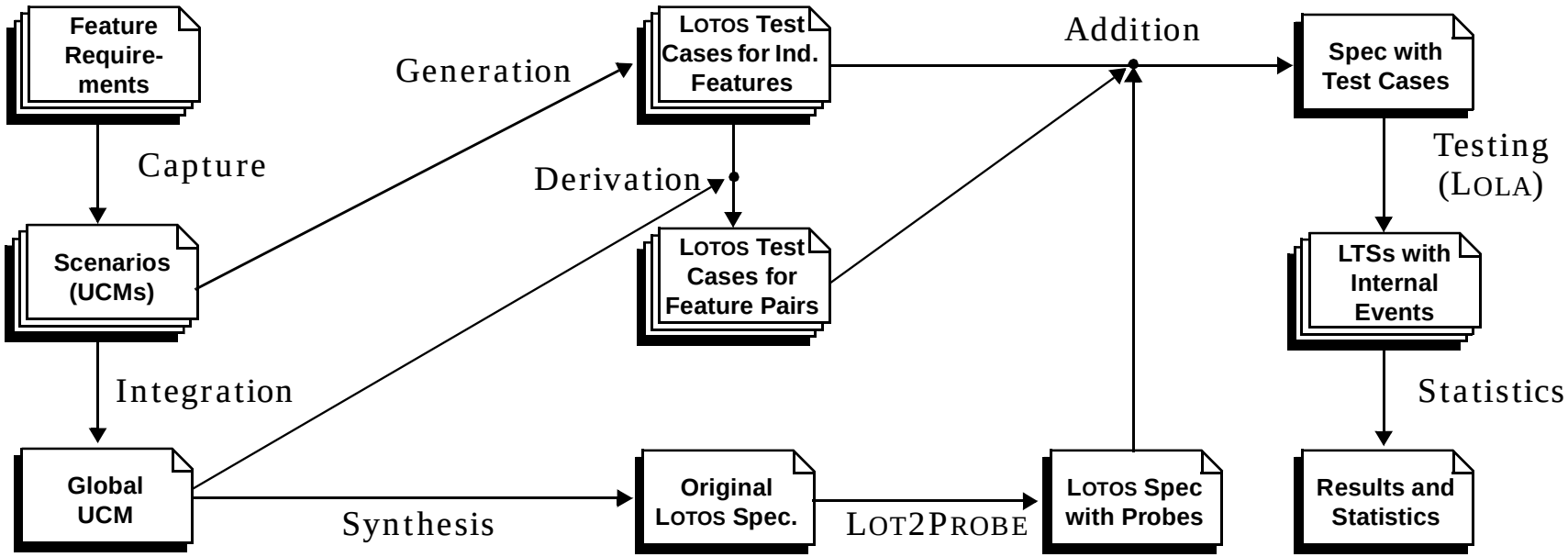


Figure 1. Scenario-Based Approach

Verdicts:

- At least one test case from the individual feature set has failed
- At least one test case from the set for pairs of features has failed (*FI?*)
- At least one probe has not been visited by the entire test suite
- The test suite has passed successfully, and all probes have been covered

About Use Case Maps (UCMs)

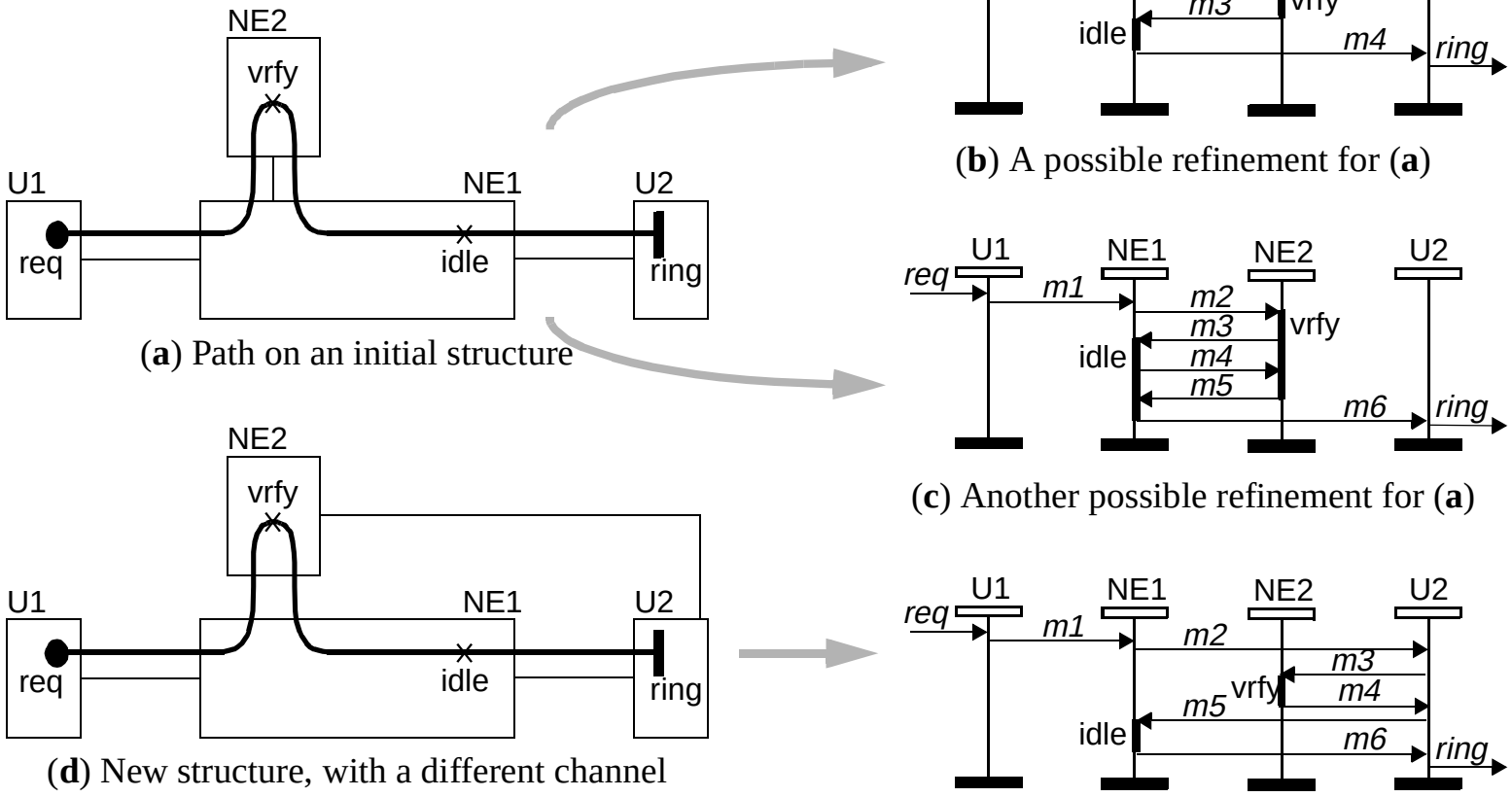


Figure 2. Causal Scenarios and Exchanges of Messages

About the Features

12 telephony features described as UCMs, 4 of which specified in LOTOS:

- *IN Teen Line* (INTL):
  - restricts outgoing calls based on the time of day.
  - can be overridden with an identity code (PIN).
- *Calling Number Delivery* (CND):
  - allows the called telephone to receive a calling party's Directory Number and the date and time.
- *IN Freephone Billing* (INFB):
  - allows the subscriber to pay for incoming calls.
- *Terminating Call Screening* (TCS):
  - restricts incoming calls from lines that appear on a screening list.

Capturing Features with Use Case Maps

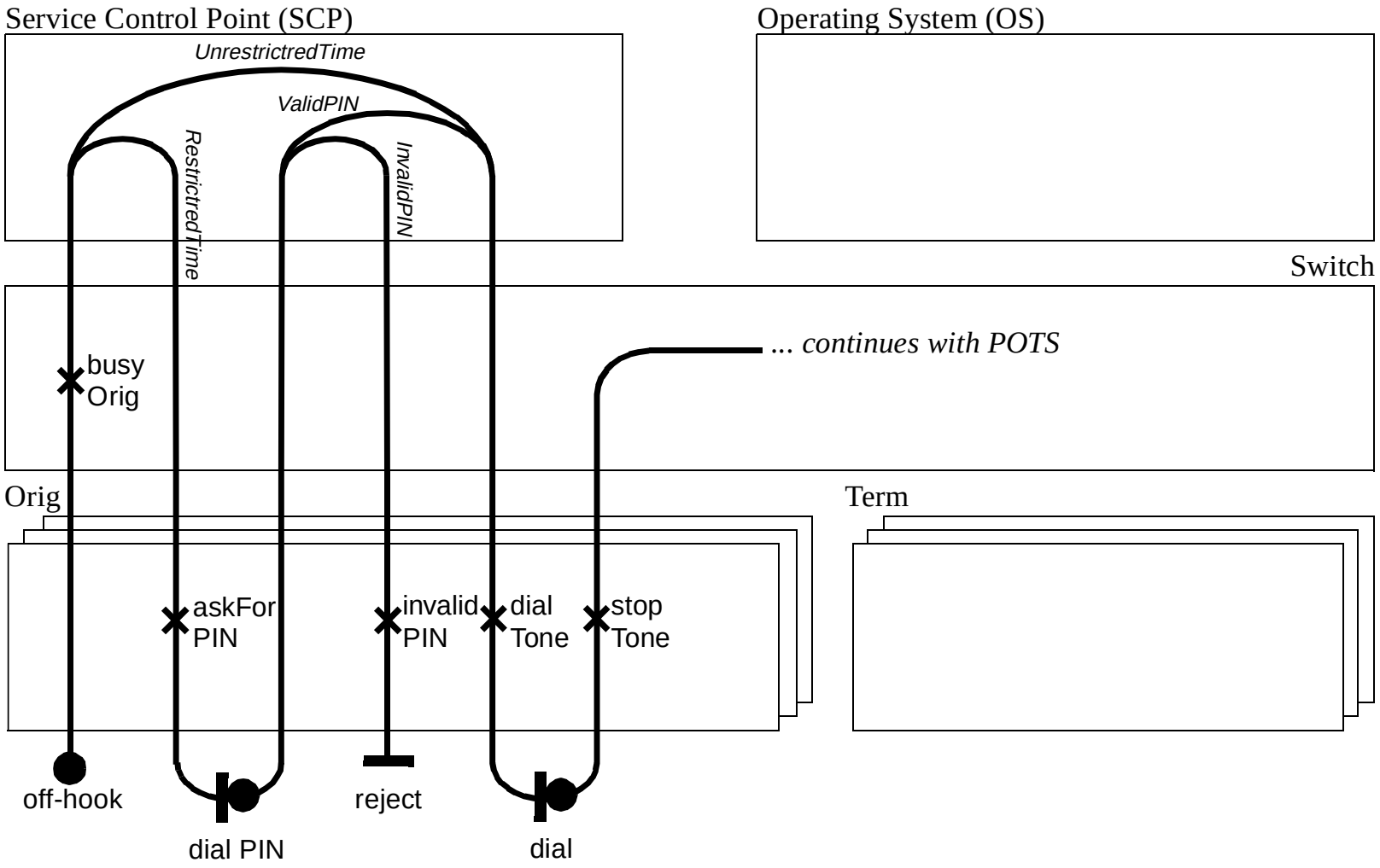


Figure 3. Partial UCM for INTL

Integration of UCM Scenarios

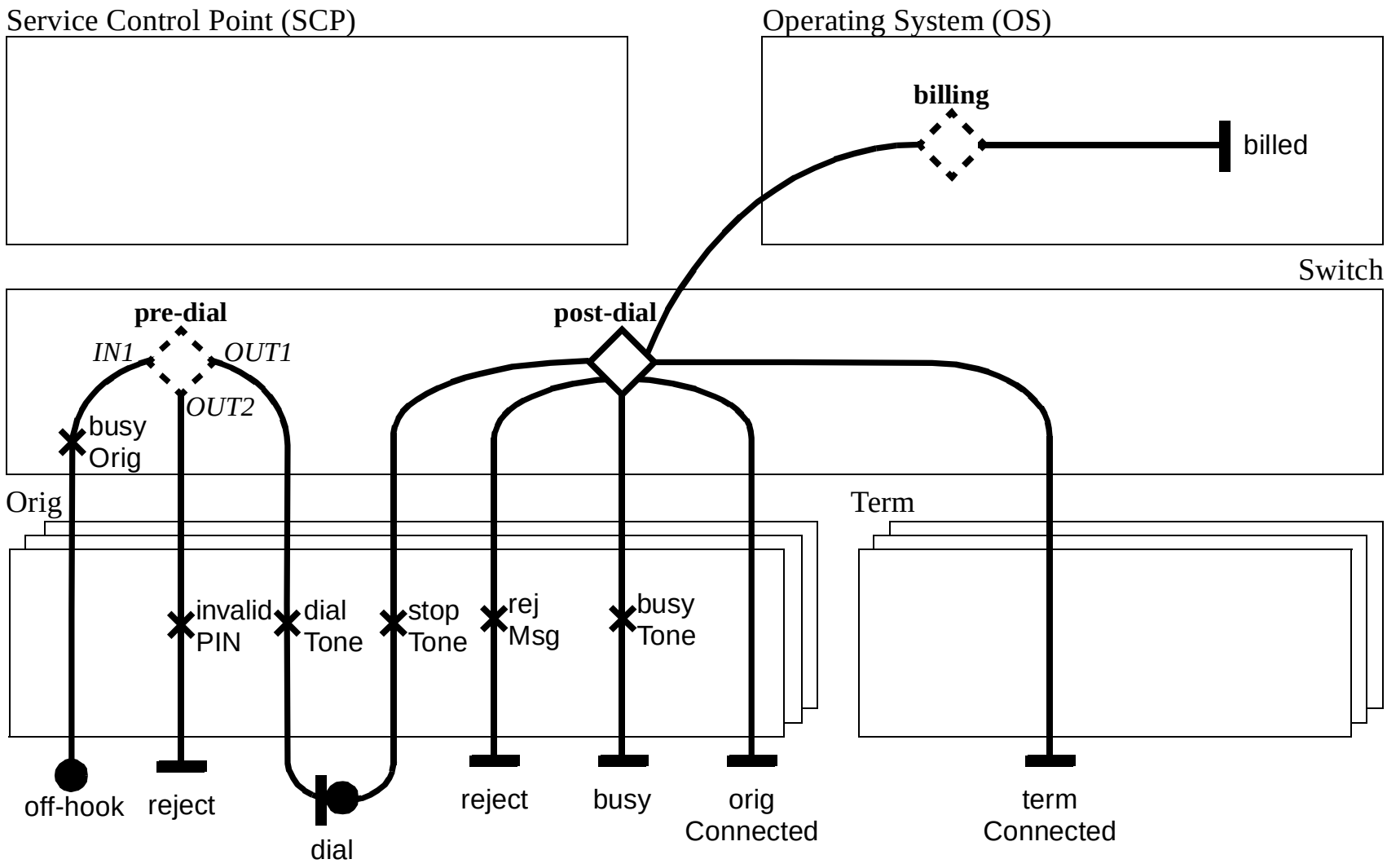


Figure 4. Root Map for the Global UCM

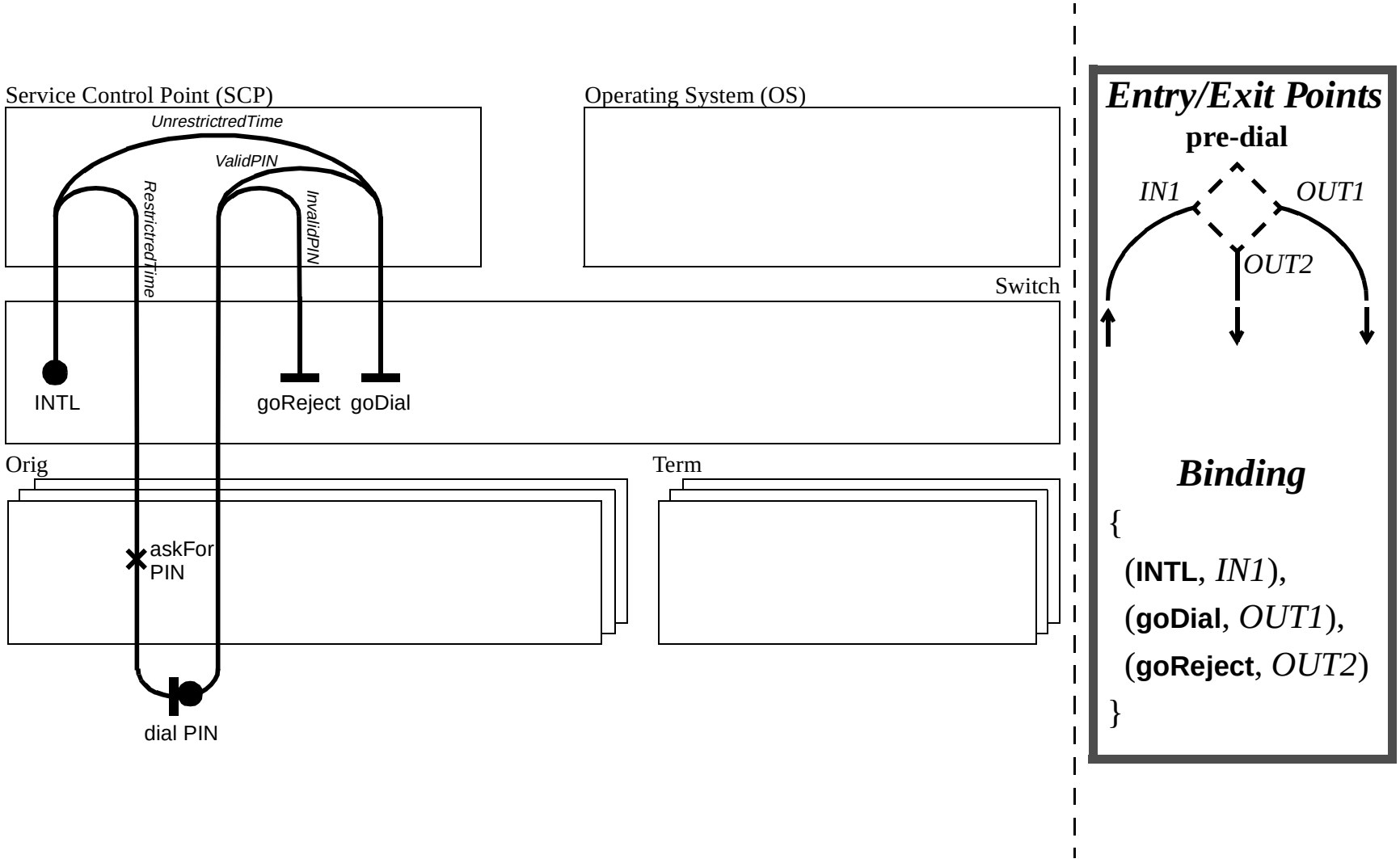


Figure 5. INTL Plugin for Pre-dial Stub in the Root Map

Avoiding Feature Interactions

- *Integration of scenarios at the level of UCMs helps to avoid many interactions between features*.
  - Many interactions between INTL, INFB/TCS, and CND are avoided.
  - Features are integrated using a sequence of three different stubs.
- *Interactions between features in one stub are still possible*.
  - Depends on the selection policy within a stub (e.g., INFB and TCS in stub process-call).
  - However, the impact is much more localized and easier to analyze.
- *Need to distinguish between what should be obliged and what should be permitted or even forbidden in a feature*.
  - CND *obliges* the display and *allows* for the terminator to pay (it is not forbidden), whereas INFB *allows* the display (it is not forbidden) and *obliges* the terminator to pay.

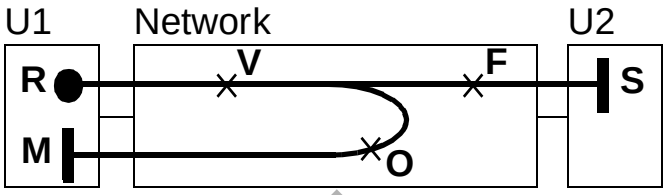
About LOTOS

- LOTOS is an algebraic specification language, standardized by ISO.
- Prototyping of distributed systems at many levels of abstraction through the use of *processes*, *hiding*, *parallel composition* and *multi-way synchronization.*
- Integration of behavior and structure in a unique executable model.
- Many validation and verification techniques such as:
    - step-by-step execution (simulation)
    - random walks
    - equivalence checking
    - **testing**
    - expansion (symbolic or not)
    - model checking
    - goal-oriented execution

Synthesis of Specifications from UCMs

- ***Intuitive example:*** Connection request (**R**) sent, availability of other party verified (**V**), ring signal (**S**) when free (**F**), message (**M**) when occupied (**O**).

The structure is mapped onto a set of processes composed through channels or shared events.



Each component becomes a process that implements all the paths that cross it (possibly from multiple scenarios).

```
specification Connection[R,V,F,S,O,M] : noexit
(* Abstract Data Types *)
behaviour
   hide Chan1, Chan2 in
      ( U1[R,M,Chan1] ||| U2[S,Chan2]
      |[ Chan1, Chan2 ]|
      Network[V,F,O,Chan1, Chan2]
where
   (* Component processes ... *)
endspec (* Connection *)
```

```
process Network[V,F,O,Chan1, Chan2] : noexit :=
   Chan1 !U1ToNetwork ! m1;  V;
   (
      F; Chan2 !NetworkToU2 ! m2;
      Network[V,F,O,Chan1, Chan2]
      []
      O; Chan1 !NetworkToU1 ! m3;
      Network[V,F,O,Chan1, Chan2]
   )
endproc (* Network *)
```

Figure 6. Synthesis of a LOTOS Specification from a UCM

Testing

- Test selection strategies based on the coverage of UCM paths:
  - Some paths
  - All paths and their combinations
  - All the temporal sequences in concurrent paths, etc.
- Test cases built on top of others.
- FI test cases express how two features are expected to interact.

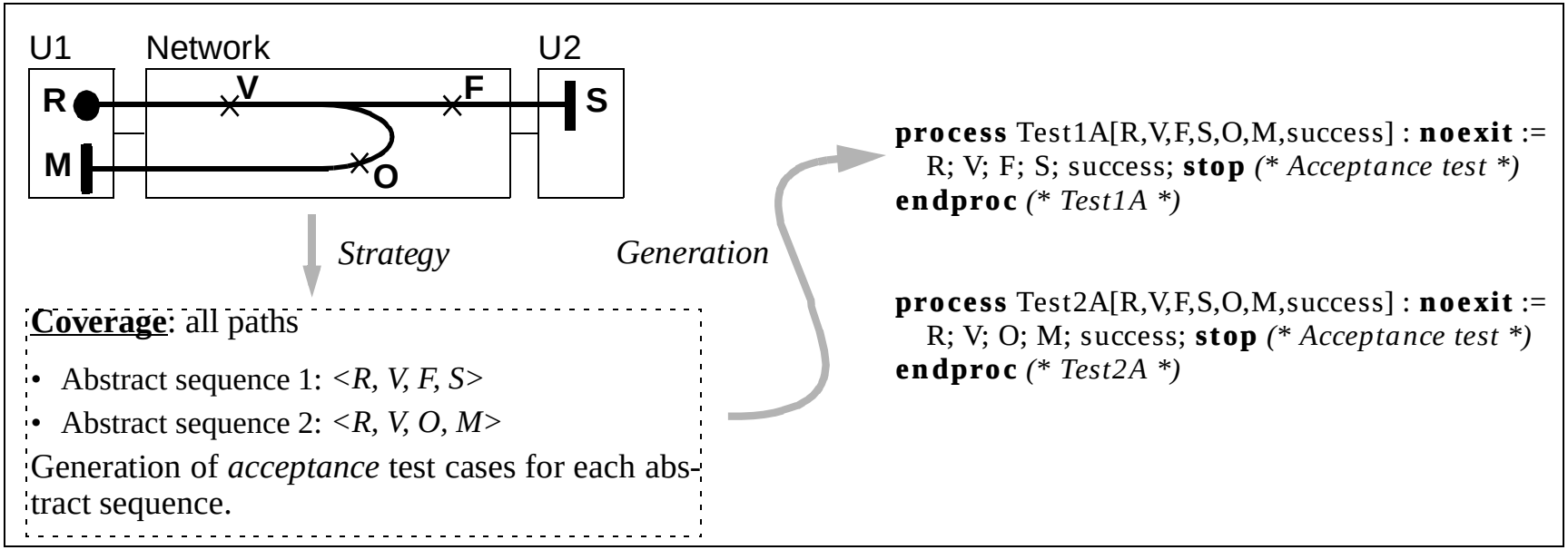


Figure 7. Derivation of Validation Test Cases from UCMs

Unexpected Interaction

- B has INFB and TCS, A is not on B's screening list, yet A is billed.
- LOTOS test deadlocks when querying the OS for the billing log.

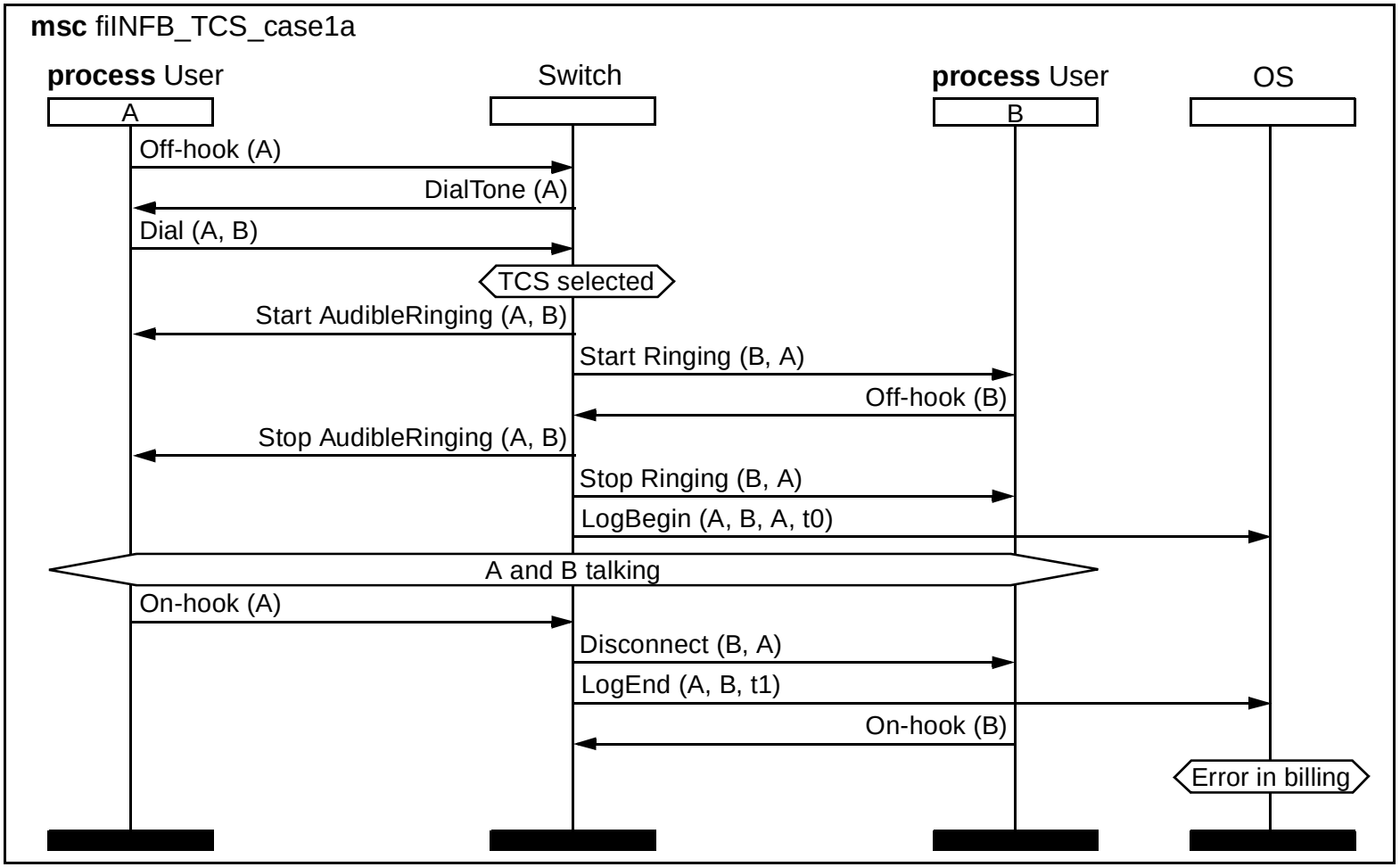


Figure 8. FI: Originator Billed Instead of the Terminator

Diagnostics

Source of the Problem

- Composition of INFB and TCS plug-ins in stub **process-call**.
- When both subscribed, the choice is non-deterministic.

Fixing the Specification and the UCMs

- Need to be more constrained: priority of TCS over INFB (and other features in stub **process-call**).
- Solution at the LOTOS level resulted in all test cases to pass successfully.

Insuring Coverage with Probes

Structural Coverage

- Generation of test cases from scenarios is an *a priori* approach to validation.
- Assumption: functional coverage achieved when all tests pass successfully.
- Quality of test suite enhanced by using structural coverage (syntactic approach).
- New tests can be added *a posteriori*.

Probe Insertion

- Well-known white-box technique for structural coverage:
    - Identify portions of code not yet exercised.
    - Measure efficiency and completeness of test suites.
- Program instrumented with *probes* (hidden gate **Probe** for an equivalent specification).
- Structural coverage achieved when all probes are visited by the tests.
- Added value: valid specification *and* test suite.
- Some probes missed because of features not yet implemented, as expected.
- Same coverage for both test suites (individual features and pairs of features).

Discussion

Performances

- 2800 lines of LOTOS code.
- 30 seconds for compilation and batch execution of all test cases (Cyrix 150).
- 7 minutes for measuring structural coverage (used at the end only).
- Good for iterative and incremental processes where numerous modifications, additions, debugging sessions, and executions of regression test suites need to be supported.

Adding New Features

- Impact on the Global UCM: new plugins, few new stubs and exit paths.
- Impact on the Specification: reflect global UCM, some new types and gates.
- Impact on the Test Suite:
    - Major impact caused by combinations of conditions (*exponential*) and features (n^2).
    - Structuring with common behaviours helps a lot.
    - Number of tests reduced by using UCM-based domain partitioning.

Conclusions

- Approach for the avoidance and detection of feature interactions at design time.
- Features are captured as UCM scenarios, integrated in one global map with stubs and plugins, and then transformed into a LOTOS specification.
- Test selection techniques based on UCMs (and their integration) help us generate reduced sets of test cases for features.
- Test suites for detecting interactions between pairs of features are constructed on top of existing test cases, hence promoting reuse and consistency among tests.
- The quality of the specification and of the validation test suite is finally assured by measuring the structural coverage through probe insertion.
- Good tool support for the UCM integration (**UCM Navigator**) and for the validation and coverage measurement of the LOTOS specification (LOLA) suggests that this approach can be used in an iterative and incremental design process.

Future Work

- *Use of other LOTOS validation techniques for further FI detection:*
    - Experiment with other methods for FI detection such as model checking and observer processes.
- *Linkage of the OPI model to the UCM notation*:
    - The intent of a feature-UCM would be better described by indicating which events or paths should be obliged, permitted, or forbidden.
    - Forbidden paths would also allow for a better way of generating rejection test cases.
- *Generation of UCM-based functional test suites in TTCN.*
- *Improvements on the UCM Navigator.*