Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features

Daniel Amyot

High Level Design and Prototyping of Agent Systems
Department of Systems and Computer Engineering
Carleton University
email: ***damyot@csi.uottawa.ca***

0. Road Map

1. Introduction

- 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 for detection, resolution, and avoidance at design time (static) and run time (dynamic).

Our Proposal:

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

- Experience in FI with both UCMs and LOTOS.
- Use some of the best features of these complementary approaches: visual description and integration (using stubs), and formal V&V.
- Illustration with the First Feature Interaction Contest example.

2. Methodology

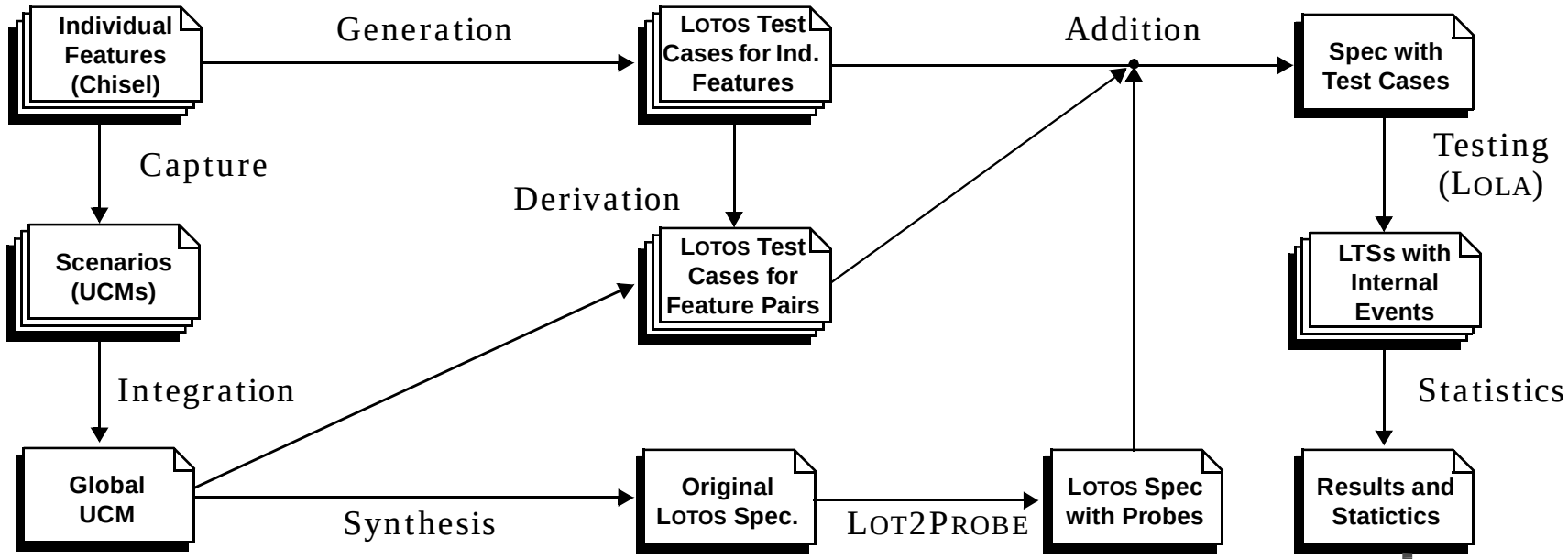


Figure 1. Rigorous Approach Based On Scenarios

Verdicts:

- At least one test case from the individual feature set has failed.
- At least one test case from the feature interaction set has failed.
- 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.

3. Use Case Maps for Features

3.1 Capturing Chisel Diagrams with UCMs

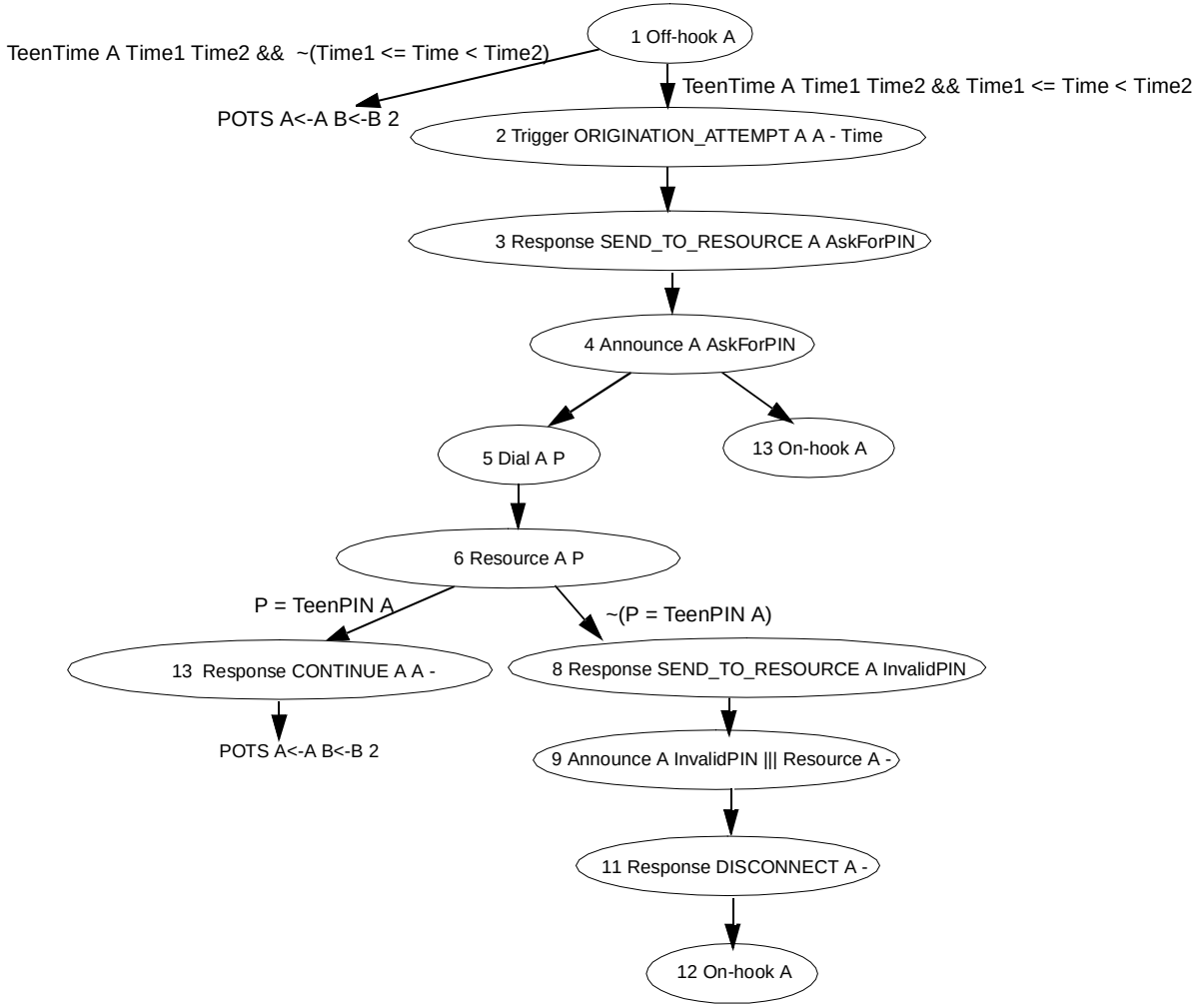


Figure 2. Chisel Diagram for IN Teen Line (INTL)

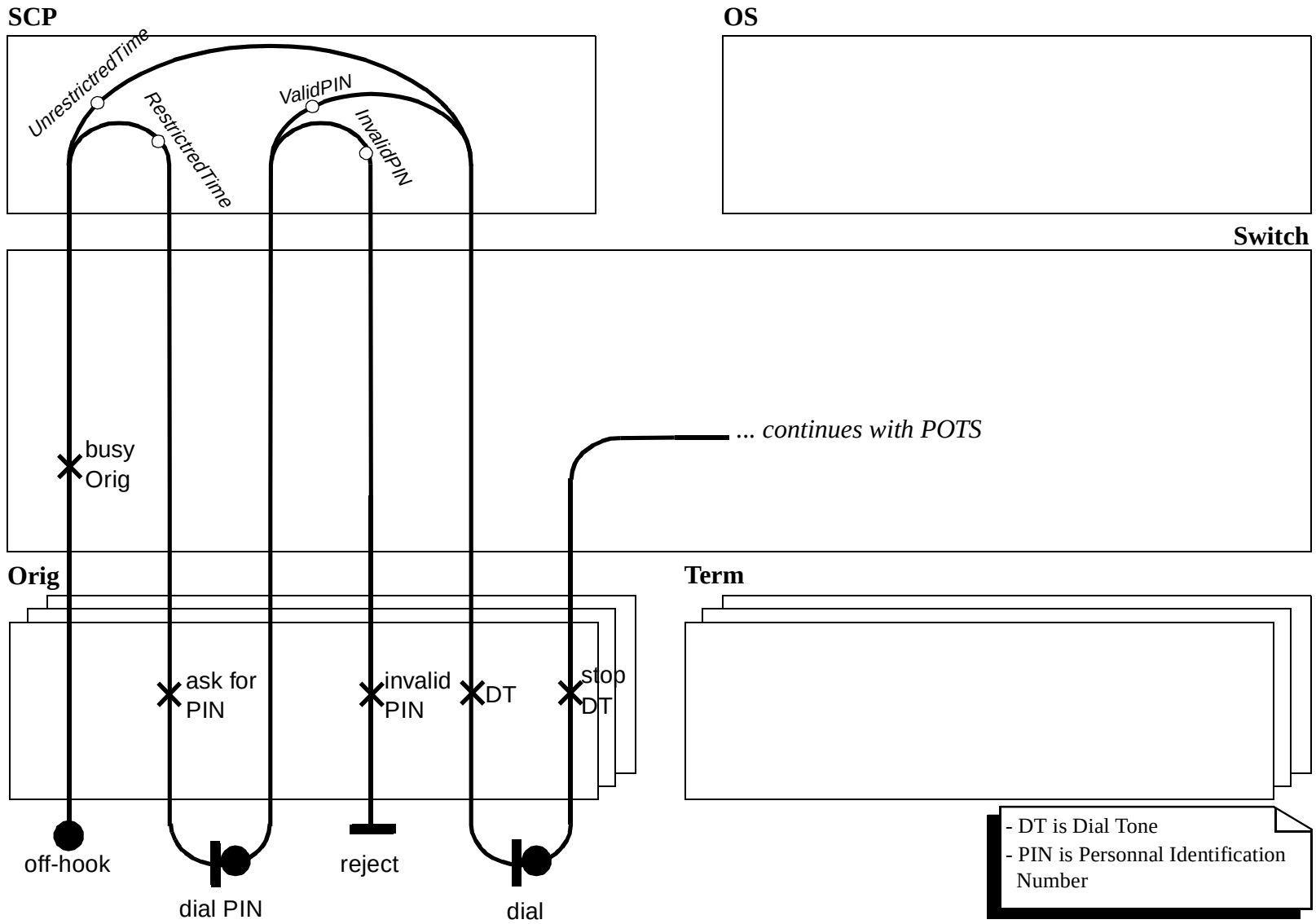


Figure 3. Partial UCM for INTL

3.2 Integration of UCM Scenarios

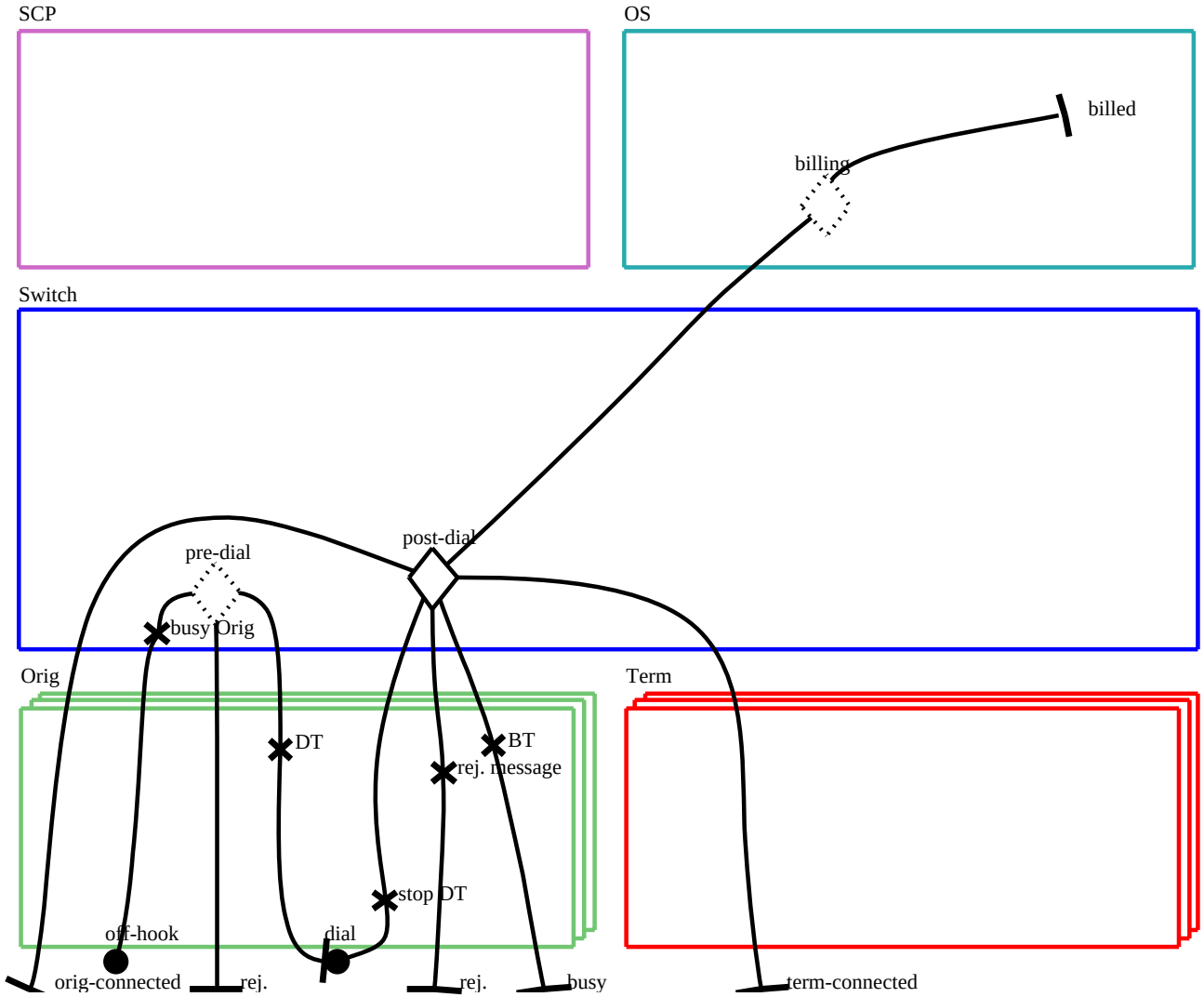


Figure 4. Root Map for the Global UCM

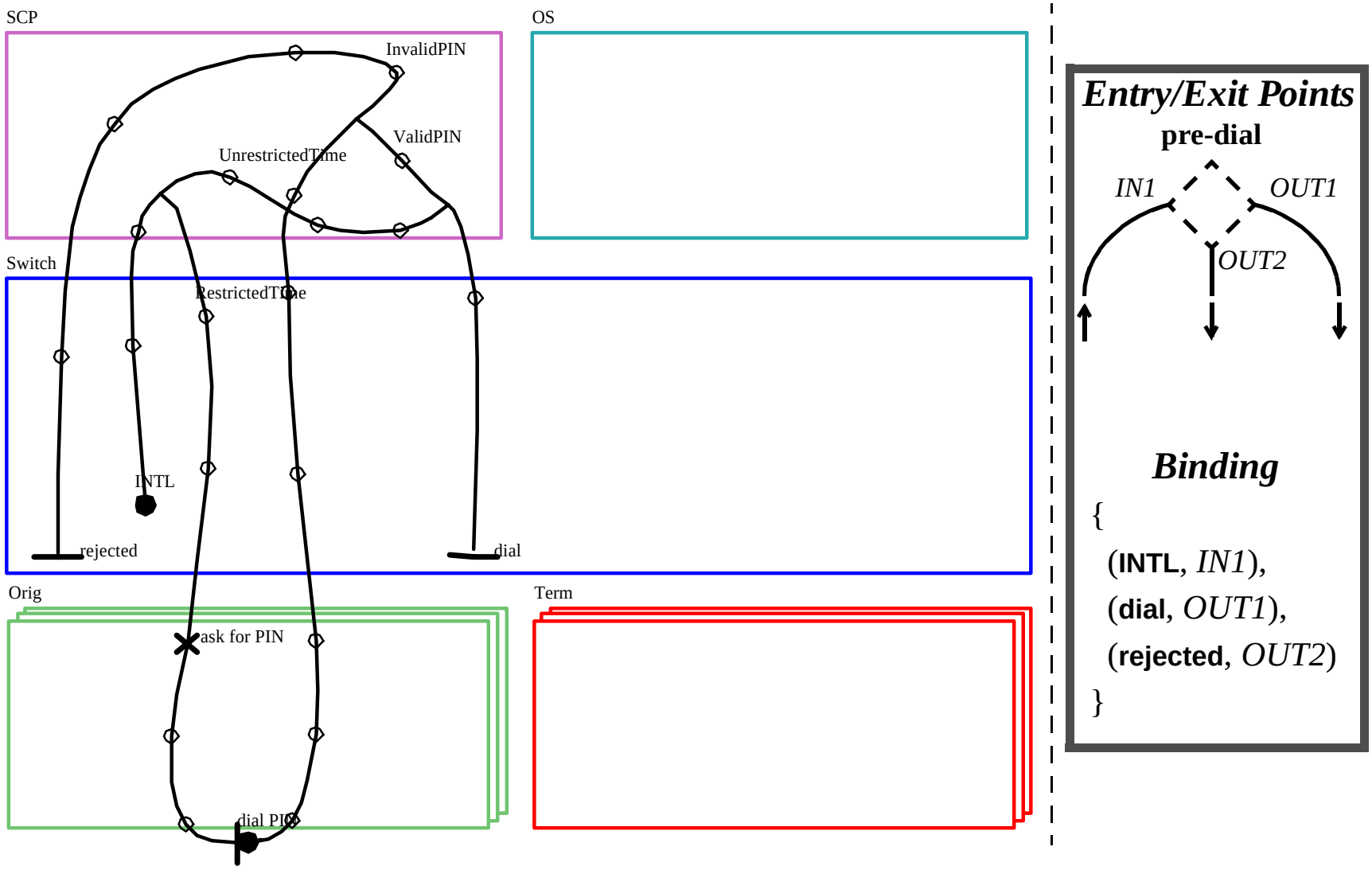


Figure 5. INTL Plugin for Pre-dial Stub in the Root Map

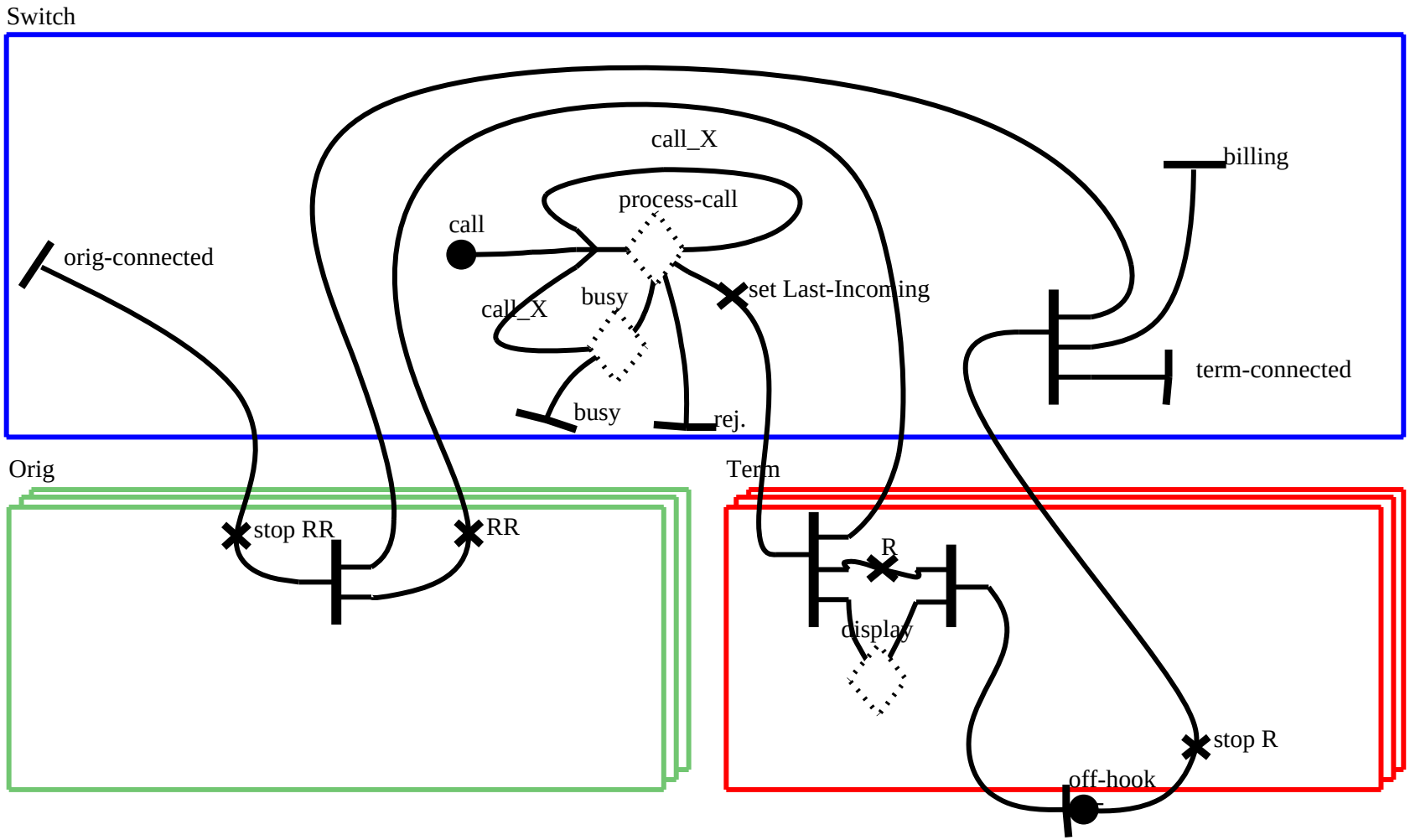
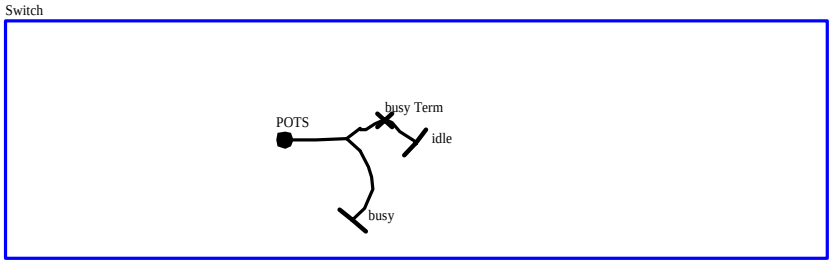
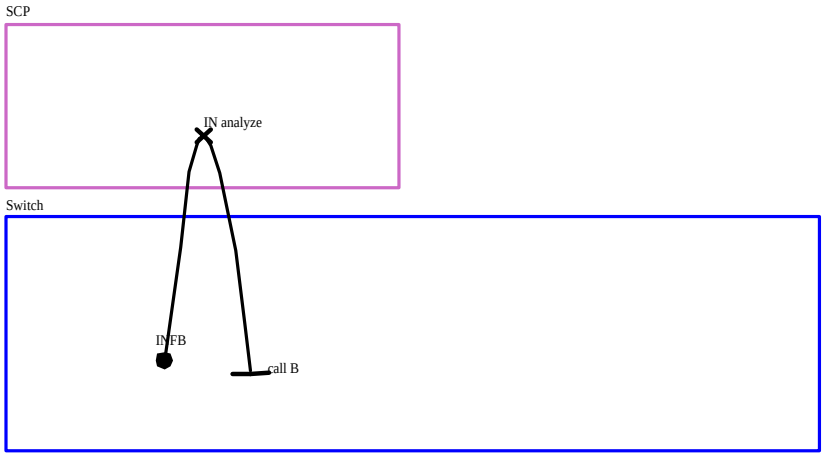


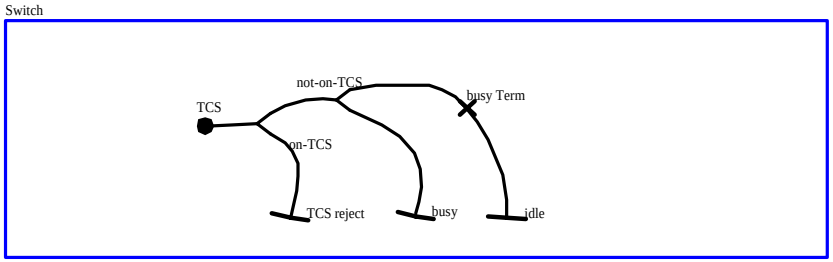
Figure 6. Plugin for Post-dial Static Stub in the Root Map



(a) Default Plugin

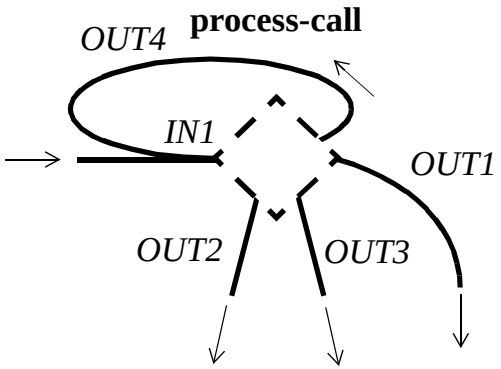


(b) INFB Plugin



(c) TCS Plugin

Entry/Exit Points



Bindings

(a): { (**POTS**, *IN1*),
(**idle**, *OUT1*),
(**busy**, *OUT2*) }

(b): { (**INFB**, *IN1*),
(**callB**, *OUT4*) }

(c): { (**TCS**, *IN1*),
(**idle**, *OUT1*),
(**busy**, *OUT2*),
(**TCS-reject**, *OUT3*) }

Figure 7. Plugins for Process-call Stub in Figure 6

3.3 Avoiding Feature Interactions

- *Integration of scenarios at the level of UCMs helps to avoid some trivial or artificial interactions between features*. For instance, many potential interactions between INTL, INFB/TCS, and CND are avoided because each pair is allowed to proceed independently in the map. They are integrated using a sequence of three different stubs.
- *Interactions between features in one stub (INFB and TCS) are still possible*, depending on the composition/decision mechanism within the (Process-call) stub. Important design decisions need to be made, but the impact is much more localized and easier to analyze (mutually exclusive but complete preconditions to avoid non-determinism and unspecified behaviour, priorities that need to be established, etc.). This is not done at the UCM level, but with LOTOS or with agent meta-models.
- *Chisel diagrams do not distinguish between what should be obliged and what should be permitted or even forbidden in a feature*. For instance, we need to infer that 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. This would help to determine what stubs are required and how the default behaviour (POTS) is overridden.
- *A notation such as the OPI model (Obligation-Permission-Interdiction) would make this distinction in the description of a feature*. Supplemented with OPI concepts, UCMs could be used to capture the *intent* of features in terms of scenarios, not properties.

4. LOTOS Specification

4.1 LOTOS, Validation, and Testing

- LOTOS is an algebraic specification language, standardized by ISO, used for system description by defining the temporal relations along the interactions that constitute the system's externally observable behavior. Data abstractions can also be described by using *Abstract Data Types* (ADTs).
- Prototyping of distributed systems at many levels of abstraction through the use of *processes*, *hiding*, *parallel composition* and *multiway 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

Functionality-Based Testing

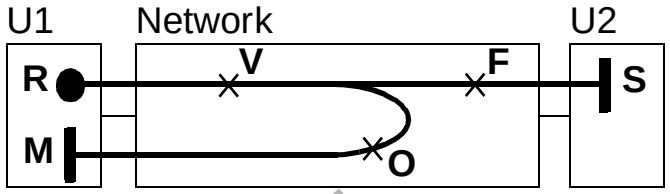
- Concerned with the existence (or the absence) of traces, use cases, or scenarios in the specification.
- (Black-box) test cases are often more manageable and understandable than properties, and they relate more closely to informal requirements. They are also more reusable for the next stages of the development process.
- Favorite approach for the validation of the features and the detection of interactions:
    - Simulation: too many global sequences of events possible.
    - Equivalence checking: we aim to produce a **first** high-level specification from the scenarios...
    - Model checking: when requirements are expressed operationally, UCMs and test cases are easier to extract than properties.
- Test cases are synchronized with the specification. **All** possible evolutions (due to non-determinism or interleaving) are analyzed by this composition.
- Three verdicts for tests with LOLA:
    - **Must pass**: all the possible executions were successful.
    - **May pass**: some executions were successful, some unsuccessful.
    - **Reject**: all executions failed to reach the *Success* event (deadlocks).

4.2 Synthesis

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 8. Synthesis of a LOTOS Specification from a UCM

Guiding Rules:

- Components implemented as processes synchronized on their common channels/gates.
- Hidden gates used for what is not observable by the user.
- Path segments in one component are integrated together, often as alternatives (could also integrated as concurrent multi-sequences, depending on the UCM context).
- UCM activities implemented as gates or as messages exchanged between components.
- Composition with the disconnection phase applied to specific points in the global UCM.
- ADTs used to represent databases and operations, and to evaluate conditions.
- Symmetry enforced in synchronized actions (actions in one process must be mirrored in the other synchronized processes, unless locally hidden).
- Chisel states with the ||| operator refined into a simpler sequence, for the reduction of the state space.
- Implicit recursive behaviour in components

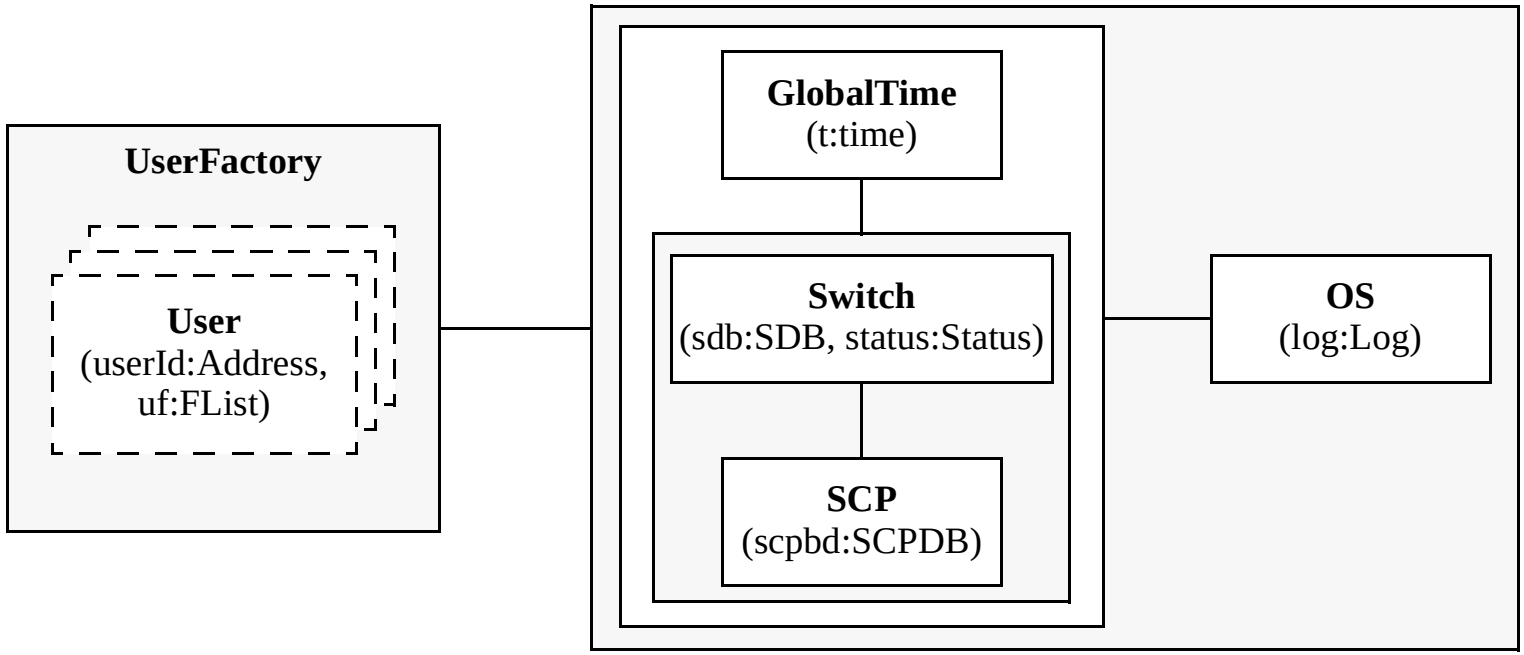


Figure 9. Top-Level Process Structure

More rules for specifying stubs:

- Components with stubs have sub-processes, one for each stub.
- Dynamic stubs may have multiple sub-processes, one for each plugin.
- The stub process is used to *specify the type of composition* between the possible plugins.
- Each stub process receives a list of entry/exit points as input and then outputs another such list upon termination.

4.3 Testing

Intuitive Example:

- 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.
- Acceptance and/or rejection test cases for abstract sequences.

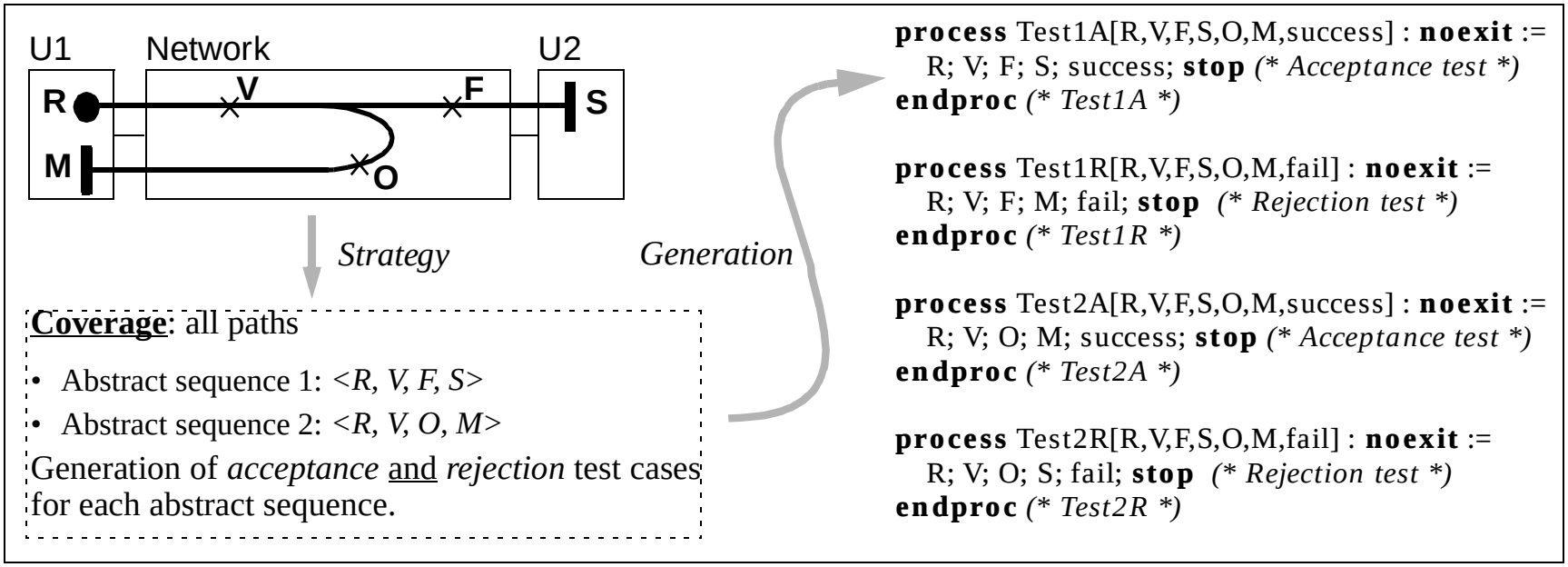
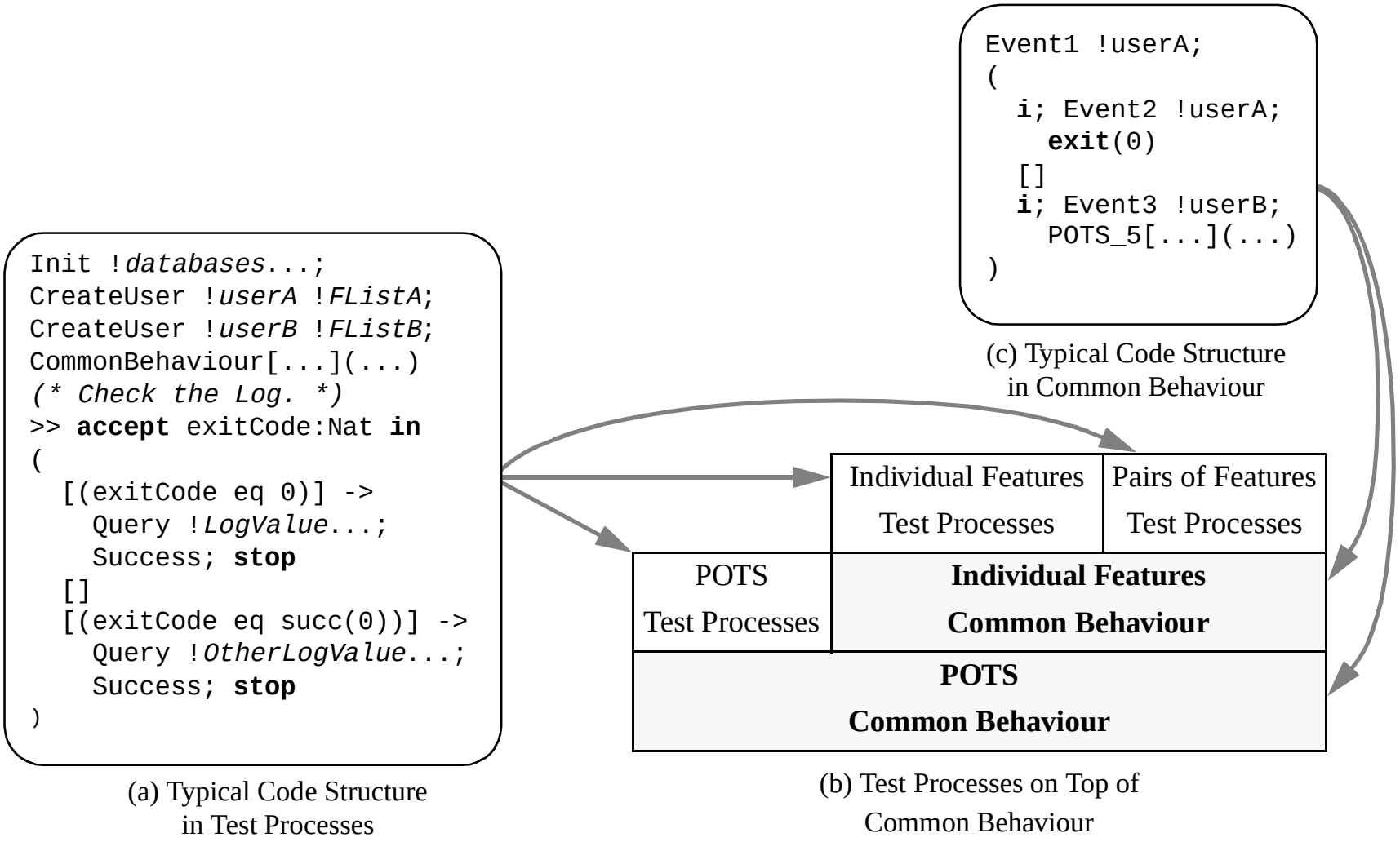


Figure 10. Derivation of Validation Test Cases from UCMs

Test Cases Built on Top of Others:

- *Common behaviour* processes improve consistency, simplicity, and reuse.



(a) Typical Code Structure in Test Processes

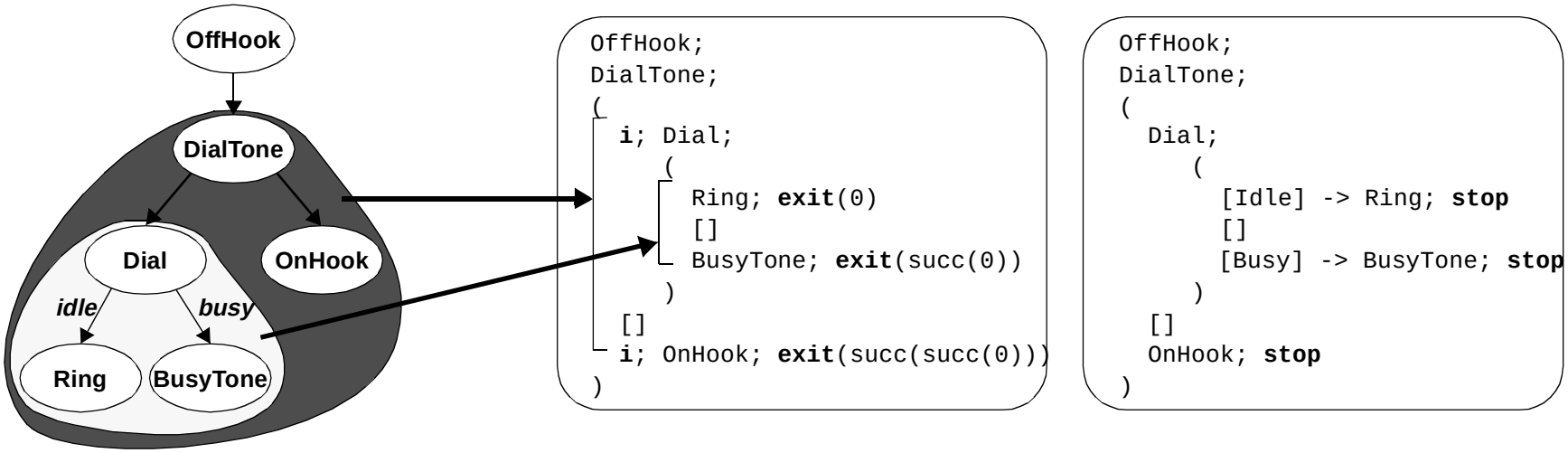
(c) Typical Code Structure in Common Behaviour

(b) Test Processes on Top of Common Behaviour

Figure 11. Construction of our Test Suite

Canonical Testers for Common Behaviour Processes

- Test all traces in a Chisel diagram.
- Several initial states and configurations (*test processes*) might be needed when data and conditions are present. For instances, testing POTS requires two test cases (terminator busy, terminator idle).



(a) Chisel Diagram (b) Test Process (c) System

Figure 12. Example of a Canonical Tester for a Chisel Diagram

Purposes Consistent with UCMs and Chisel Diagrams

- Use paths and alternatives to obtain the partitioning of the domain.

Feature	Test Process	Purpose	Used Common Behaviour	Number of Global Sequences
INTL	tINTL1	TeenTime not restricted: allow call.	POTS_1	29
	tINTL2	TeenTime restricted, valid PIN: allow call.	cINTL1	30
	tINTL3	TeenTime restricted, invalid PIN: do not allow call.	cINTL2	2
CND	tCND1	Terminator idle: display.	cCND1	84
	tCND2	Terminator busy: do not display.	POTS_1	2
INFB	tINFB1	Terminator idle: affect billing.	POTS_1	29
	tINFB2	Terminator busy: do not affect billing.	POTS_1	2
TCS	tTCS1	Terminator idle, A not on Screened B: allow call.	cTCS1	29
	tTCS2	Terminator busy, A not on Screened B: busy tone.	cTCS2	2
	tTCS3	A on Screened B: announce screened message.	cTCS3	2

Figure 13. Description of Test Processes for Individual Features

5. Detecting Feature Interactions

5.1 Test Cases for Detecting FI

- FI test cases express how two features are expected to interact, for different initial states and configurations.
- Consistent with UCM integration and composition in stubs.
- Number of test cases can be reduced w.r.t. the Cartesian product of the previous table. For instance, in INTL-CND (3x2=6 cases), the cases where the terminator is busy is not interesting (and covered in INTL-POTS).

FI Test Process	Number of Test Cases	Used Common Behaviour	Number of Global Sequences
fiINTL_CND	3	cCND1, cINTL2	170
fiINTL_INFB	3	POTS_1, cINTL1, cINTL2	61
fiCND_INFB	2	cCND1, POTS_1	86
fiINTL_TCS	9	cTCS1, cTCS2, cTCS3, cINTL1, cINTL2	74
fiCND_TCS	4	cCND1, cTCS2, cTCS3	90
fiINFB_TCS	4	cTCS1, cTCS2, cTCS3	35

Figure 14. Description of Test Processes for Pairs of Features

5.2 Unexpected Interactions

- 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 log.

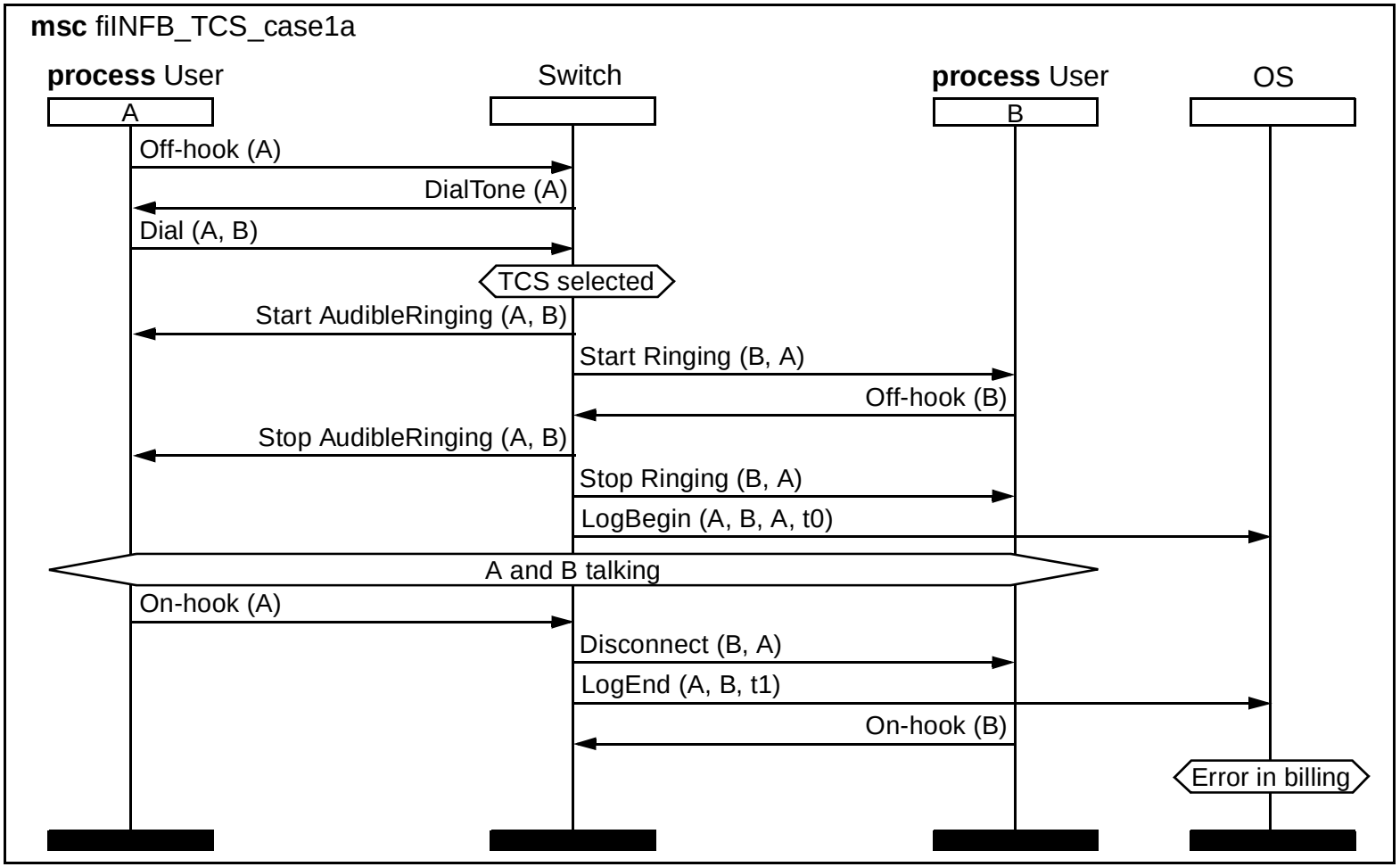


Figure 15. FI: Originator Billed Instead of the Terminator

Same Pair, Other Interaction

- B is idle and has INFB and TCS. A *is* on B's screening list.
- LOTOS test deadlocks when it expects a *ScreenMessage* announcement while the switch offers a *Start AudibleRigning*.

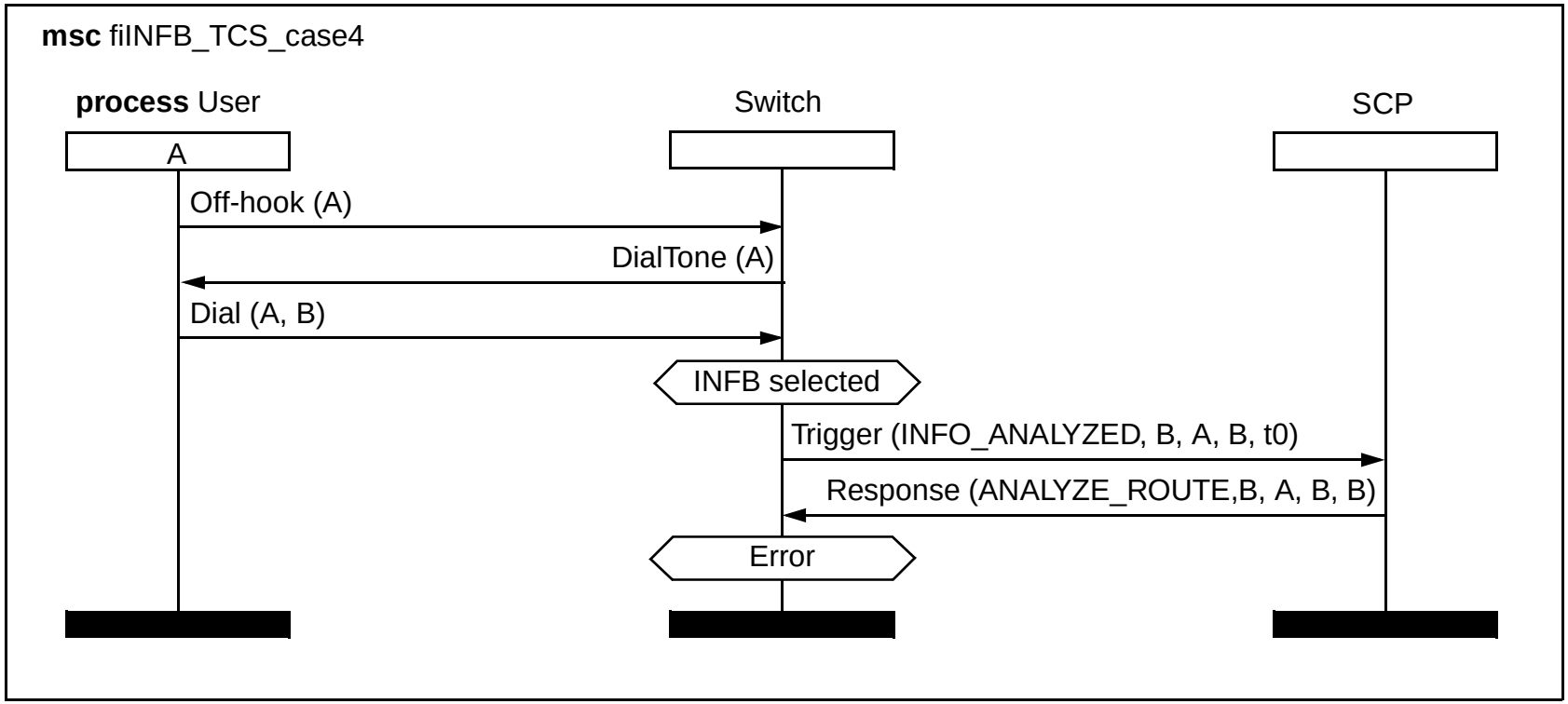


Figure 16. FI: Call Should Be Blocked but Is Not

Source of the Problem

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

Fixing the Specification and UCM

- 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.
- In UCM terms, a similar solution would be to move the TCS checking at a higher level that what it used to be in process-call:

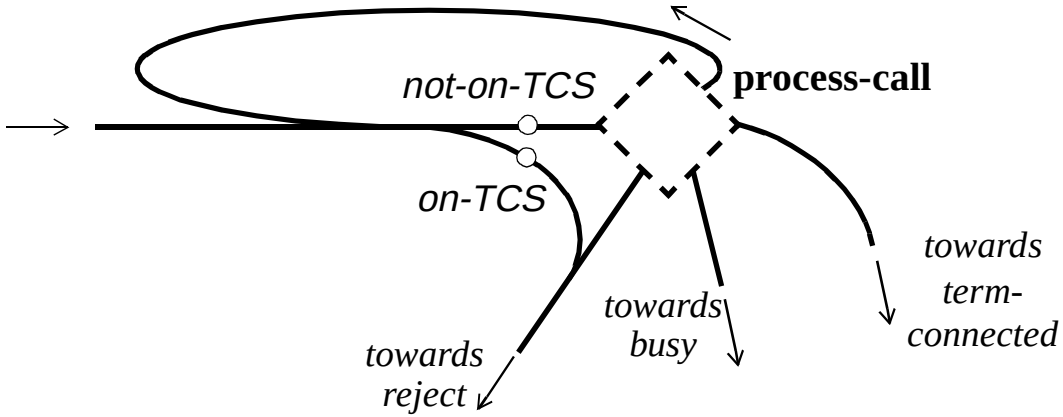


Figure 17. New Surroundings of Process-call Stub in Figure 6

5.3 Insuring Coverage with Probes

Structural Coverage

- Generation of test cases from scenarios is an *a priori* approach to validation.
- Assumption:
    - functional coverage is achieved when all tests execute 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 (event) coverage is 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).

6. Discussion

Performances

- 2864 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.

Improved Call Structure

- For features that involve 3 parties (INBL, 3WC, CW, INCF, INFR).
- Dynamic creation of call sessions in the switch, and new status database.
- Current specification reflect UCM structure, but it is too simple.

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 ADTs 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.

Feature	Number of Conditions (*c*)	Theoretical Upper Bound ($u = 2^c$)	Actual Number of Test Cases (*t*)	Gain ($g = u - t$)
INTL	2	4	3	1
CND	1	2	2	0
INFB	1	2	2	0
TCS	2	4	3	1

Figure 18. Number of Test Cases for Individual Features

Pair of Features	Number of Distinct Conditions (*c*)	Theoretical Upper Bound ($u = 2^c$)	Product of Number of Cases (*p*)	Better Upper Bound $b = MIN(u, p)$	Actual Number of Test Cases (*t*)	Gain ($g = b - t$)
INTL-CND	3	8	3*2 = 6	6	3	3
INTL-INFB	3	8	3*2 = 6	6	3	3
CND-INFB	1	2	2*2 = 4	2	2	0
INTL-TCS	4	16	3*3 = 9	9	9	0
CND-TCS	2	4	2*3 = 6	4	4	0
INFB-TCS	2	4	2*3 = 6	4	4	0

Figure 19. Number of Test Cases for Pairs of Features

6.1 Comparison with Other Techniques

Agent Systems

- About the opportunistic avoidance of interactions at *run time*. The contest was about detection...
- Mapping from UCMs needs to be improved (with OPI).
- Detection/validation techniques are still ad hoc.

GCS and GPRS

- Integration of multiple UCMs done at the LOTOS level. No global map, no stub, no plugin.
- Test cases generated solely from UCMs (no Chisel diagrams).
- Required more expertise in LOTOS.
- Integration more difficult to understand for readers.
- GCS example had rejection test cases. Need a OPI-like reject concept at the Chisel/UCM level to help in that context.

Faci's Approach (LOTOS and testing)

- Composition (f_1 |[]| f_2) used to generate test cases (manually).
- Features integrated at the Labeled Transition System level: more complex and less scalable/modular than UCMs and stubs/plugins.
- Interaction if the integration does not *conform* to the composition.
- Detection only, with many interactions (deadlocks) as soon as the integration is not f_1 |[]| f_2. Tests do not consider the integration.

7. 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.
- Some interactions can be avoided with deterministic and complete preconditions and by composing plugins in stubs according to the intent of the features.
- Many features can be considered in a global UCM. Further design decisions are necessary when synthesizing the specification (composition in stubs), although the burden of the integration is mostly taken care of at the UCM level.
- Canonical testers and 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, which promotes reuse and consistency among tests.
- Two interactions were detected, and then fixed at the LOTOS and UCM levels.
- 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

- Improvement of the call process within the Switch for the support of features involving more than two users.
- Completion of the specification by integrating the remaining nine features. By observing the impact on the specification and on the number of test cases required for validation, it might be possible to learn new lessons.
- Comparison with other LOTOS-based techniques applied to the same set of features, by detecting interactions in our specification with their approaches (if the tools allow it) and by applying our test cases to their specifications. We could also observe how “trivial and artificial” interactions detected with their techniques have been avoided by our UCMs.
- Linkage of the OPI model to the UCM notation. The intent of a feature would be better described by indicating which events or paths are obliged, permitted, or forbidden to be in the implementation. This would also allow for an easy way of generating rejection test cases.
- Finally, we could look at the best way of integrating this approach in a design process that generates agent prototypes (where interactions would not need to be statically solved in advance) from use case maps.

8. References

[1] Aho, A., Gallagher, S., Griffeth, N., Scheel, C., and Swayne, D. (1998) “Sculptor with Chisel: Requirements Engineering for Communications Services”, *Fifth International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press,45-63. http://www-db.research.bell-labs.com/user/nancyg/sculptor.ps

[2] Amyot D., Use Case Maps for the Design and the Validation of Interaction-Free Telephony Features. CITO report, Ottawa, Canada. To appear. http://www.csi.uottawa.ca/~damyot/FI/

[3] Buhr, R.J.A., Amyot, D., Elammari, M., Quesnel, D., Gray, T., and Mankovski, S. (1998) “Feature-Interaction Visualization and Resolution in an Agent Environment”. In: K. Kimbler and W. Bouma (eds.), *Fifth International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press, 135-149. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/fiw98.pdf

[4] Buhr, R.J.A. (1998) “Use Case Maps as Architectural Entities for Complex Systems”. To appear in: *Transactions on Software Engineering*, IEEE, 1998. http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/tse98final.pdf

[5] Griffeth, N.D., Tadashi, O., Grégoire, J.-C. and Blumenthal, R. (1998) “First Feature Interaction Detection Contest”. In: K. Kimbler and W. Bouma (eds.), *Fifth International Workshop on Feature Interactions in Telecommunications Software Systems*, IOS Press, 327-359. http://www.tts.lth.se:80/FIW98/contest.html

[6] Miga A. (1998), *Application of Use Case Maps to System Design with Tool Support*. Masters Thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada.

[7] Petriu, D. (1998) *Feature Interaction Detection and Avoidance — Smart Design of Telephony System with Use Case Maps*. CITO report, Ottawa, Canada. To appear.

[8] Quemada, J., Pavón, S. and Fernández, A. (1988) “Transforming LOTOS Specifications with LOLA: The Parametrized Expansion”. In: K. J. Turner (Ed), *Formal Description Techniques, I*, IFIP/North-Holland, 45-54.