The context

- Component Based Developments:
  - OO Analysis & Design based on UML
  - Conformance Testing
- Asynchronous communications in distributed software:
  - arbitrary latency,
  - loss of messages,
  - dead locks, etc...
  => non-deterministic behavior
Operational Semantics?

- Problem with UML
  - no well-defined operational semantics
  - ... and more than one is needed!
    - depending on application domain...
- UML is really a family of languages
  - cf the notion of Profiles
  - Need for versatile tools that extract relevant information from a UML Model
    - the semantics is in the tool (as usual really)

Several approaches for Operational Semantics

- Based on either:
  - Class Diagram + StateCharts + Elementary Actions (program fragments) + initial deployment
  - Class Diagram + OMG’s own Action Semantics
  - Class Diagram + High Level MSCs (as available in UML2.0)
Initial Deployment

Overview of UMLAUT Simulator

- Compile the UML model into an IO-LTS
  - Input/Output Labeled Transition System
  - Aim: weave the various semantics aspects of UML into the subset \(\{\text{static + atomic actions}\}\)
- Principle of exhaustive simulation
  - Start with \textit{initial state}, explore all possible outcomes (\textit{fireable transitions}) recursively
  - storage and comparison of global states = (dynamic) unions of local states
Transformation of a UML model into an (API to an) IO-LTS

- Input/Output Labeled Transition System
  - Simulation
  - Model-checking
  - Test Case Gen.
- Dynamism:
  - creations/deletions
- Infinite (in general)
- => On-the-fly algorithms

General Idea for Dynamic Semantics: 2-layers Approach

- Use (UML static subset + atomic actions) to model the semantic domain of both
  - UML statecharts
  - and target platform
- Transpose to the mathematical domain of Labeled Transitions Systems (LTS)
  - by compilation to Eiffel
    - to deal with efficient OO compilation (static analysis)
- Impact:
  - Semantics of concurrency: interleaving
  - System evolution -> transitions of LTS, labels = actions
Make Distribution & Async. Communications Explicit

Basically, Model a Platform (e.g. CORBA, proprietary bus...)

Compiling the Dynamic Semantics

- **Goal:**
  - Integration of “uninterpreted” actions
    - written in Eiffel, C++, Java ...

- **Impact:**
  - Transformations by specialization of semantic domain with a mapping to the target OO language
  - Construction of accessibility graph as side effects caused by actions
    - implies duplication of object graph at each step
From UML to IO-LTS

- The Simulator gives the list of objects with non-empty files.
- When a transition is selected:
  - the current (global) state of the system is duplicated.
  - « dispatch » is called on the first message, and run to completion (=> side effects).
- Compute the label of the transition.
- Make it possible to store the new state.

Duplications & Comparisons

- Duplication:
  - extent of side effect unknown (non-computable statically).
  - all of the global state must then be saved.
- Comparison:
  - allow cycles to be detected (for exhaustive simulation, aka model checking).
  - Importance of information held by associations.
- Working implementation for GNU Eiffel.
Application to Validation of UML Models

Driving the simulation code

- UMLAUT generates a model-specific simulation API (Simulator)
- This API is used for either:
  - Interactive Simulation
Driving the simulation code

- UMLAUT generates a model-specific simulation API (Simulator)
- This API is used for either:
  - Interactive Simulation
  - Random simulation
  - Model checking
Driving the simulation code

- UMLAUT generates a model-specific simulation API (Simulator)
- This API is used for either:
  - Interactive Simulation
  - Random simulation
  - Model checking
  - Test case synthesis

Verifying properties

- Deadlock = state with no outgoing transition
- Case of classical OCL assertions
  - pre- and post-conditions of routines as well as class invariants are compiled into special “fail” transitions
- Possibility to use temporal logic for more sophisticated properties
Test Cases Synthesis

- Based on the (lazy) synchronous product between the model and a test objective
- A test objective allows the construction of test cases by guiding the graph exploration according to some criterion
- Test objectives and test cases are represented in UML as collaborations / interactions

Generating test cases

![Diagram showing test case generation process]
Conclusion

- UML Models useful as input for V&V
  - Early computation of integration plans
  - Simulation, test case synthesis
    - All building on complex model transformations

- Huge industrial stake (V&V>50% dvp effort)
  - Prototype versions: modelware.inria.fr
  - Industrial modules for
    - Telelogic TauG2
    - Softeam Objecteering