Triskell team
University of Rennes & INRIA

# Model Based Verification & Validation of Distributed Software

Jean-Marc Jézéquel

http://www.irisa.fr/triskell

IRISA



## 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

IRISA

## 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)

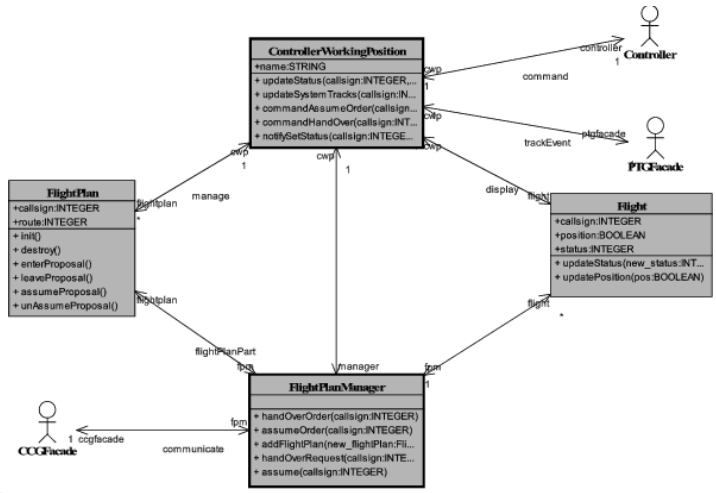
IRISA 3

## 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)

IRISA 4

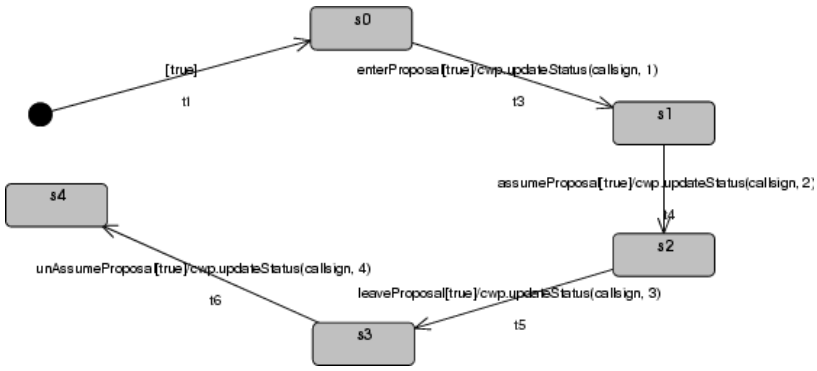
# Case Study: an ATC




IRISA

5

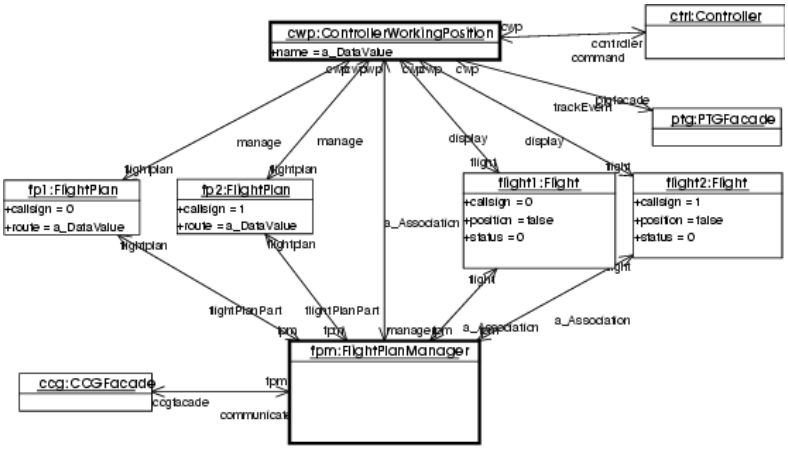
# FlightPlan State Chart




IRISA

6

# Initial Deployment




IRISA

7

# Overview of UMLAUT Simulator

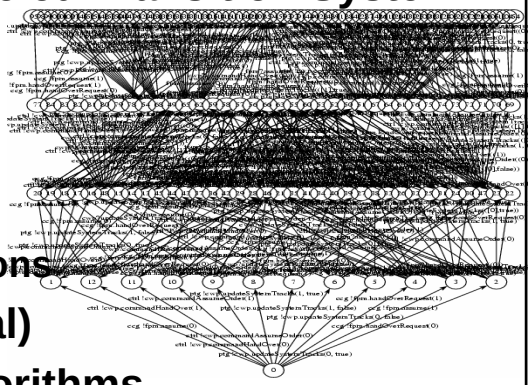
- **Compile the UML model into an IO-LTS**
  - **Input/Output Labeled Transition System**
  - **Aim: weave the various semantics aspects of UML into the subset *{static + atomic actions}***
- **Principle of exhaustive simulation**
  - **Start with *initial state*, explore all possible outcomes (*fireable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

IRISA

8

# 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



IRISA

9

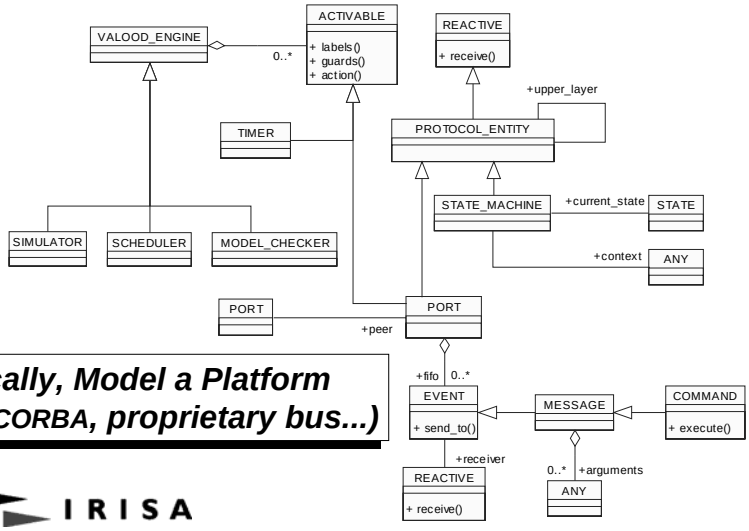
# General Idea for Dynamic Semantics: 2-layers Approach

- 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

IRISA

10

# Make Distribution & Async. Communications Explicit




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

IRISA

11

# 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***

IRISA

12

## From UML to IO-LTS

- The Simulator give 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

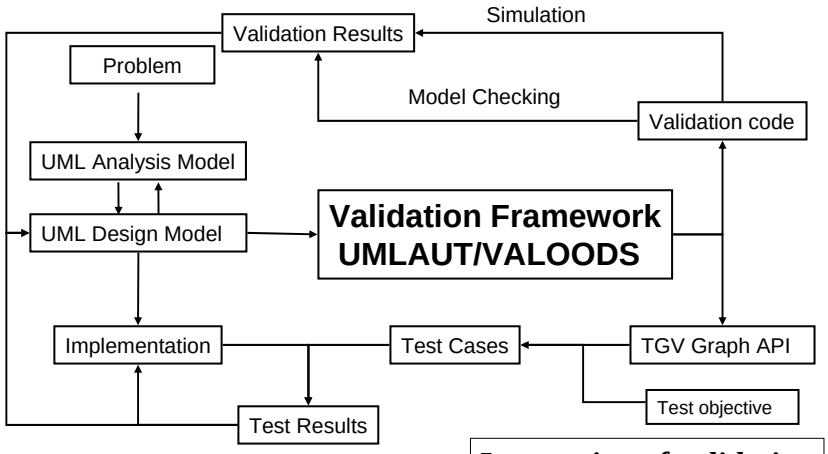
IRISA 13

## 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

IRISA 14

# Application to Validation of UML Models

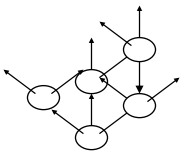



**Integration of validation in the OO life-cycle**

IRISA

# Driving the simulation code

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

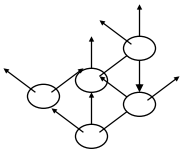


IRISA

16

## Driving the simulation code

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

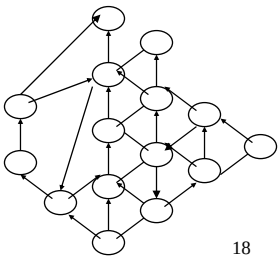


IRISA

17

## Driving the simulation code

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

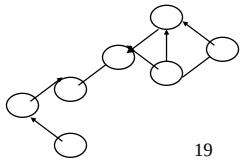


IRISA

18

## 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



IRISA

19

## 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

IRISA

20

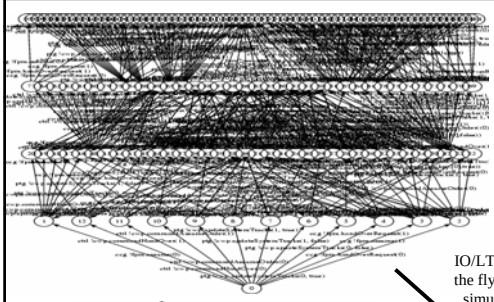
# Test Cases Synthesis

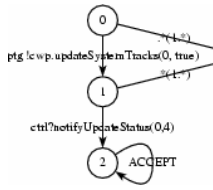
- Based on the (lazy) synchronous product between the model and a test objective
- A test objective allows the construction of test cases by guiding the graph exploration according to some criterion
- Test objectives and test cases are represented in UML as collaborations / interactions

IRISA

21

# Generating test cases



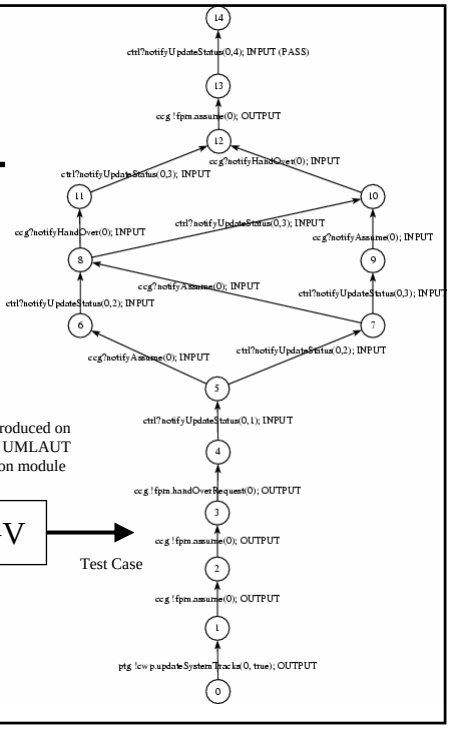



IO/LTS produced on the fly by UMLAUT simulation module

Test purpose as extracted from Use Case/Sequence diag

TGV

Test Case




IRISA

# 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**

IRISA

23