AGEDIS

Automated model-based test generation and execution

December 2002

Alan Hartman

Agenda

- Project Overview
- Motivation
- Methodology
- Tools
- Experiments
- Current Status

Project Overview

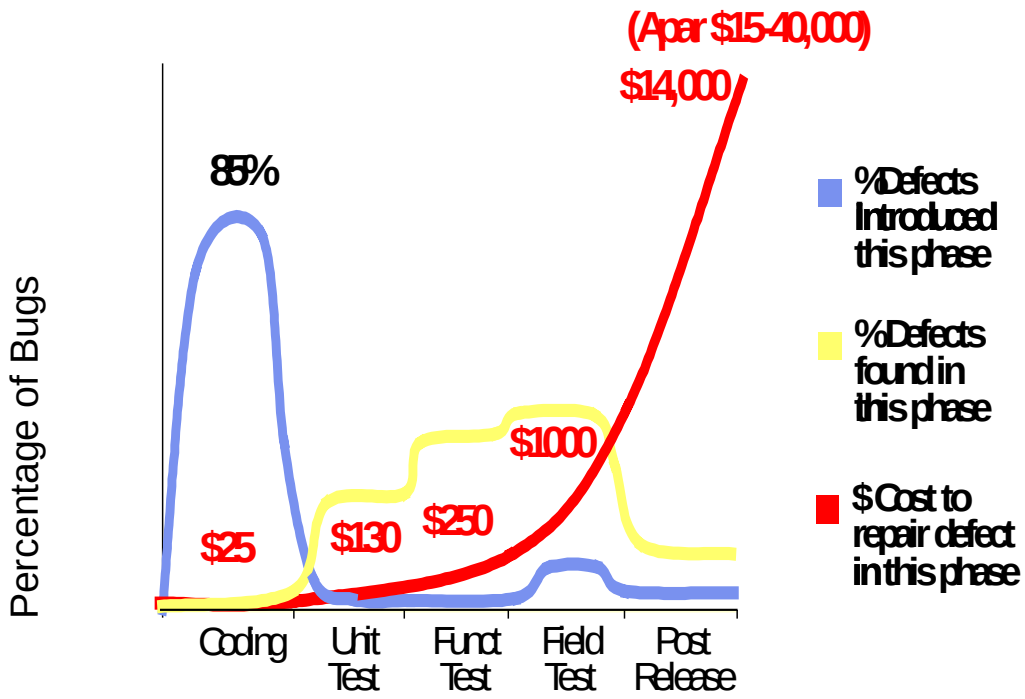
- Automated model-based test Generation and Execution for DIStributed systems
- Methodology and tools for model-based testing
- Open interfaces
- Mixture of academic and industrial partners
- Three phase timetable of experiment and development
- November 2001-2003

Partners

- IBM Haifa Research Lab
- Oxford University
- VERIMAG/IRISA
- Imbus
- France Telecom
- IBM UK
- Intrasoft International

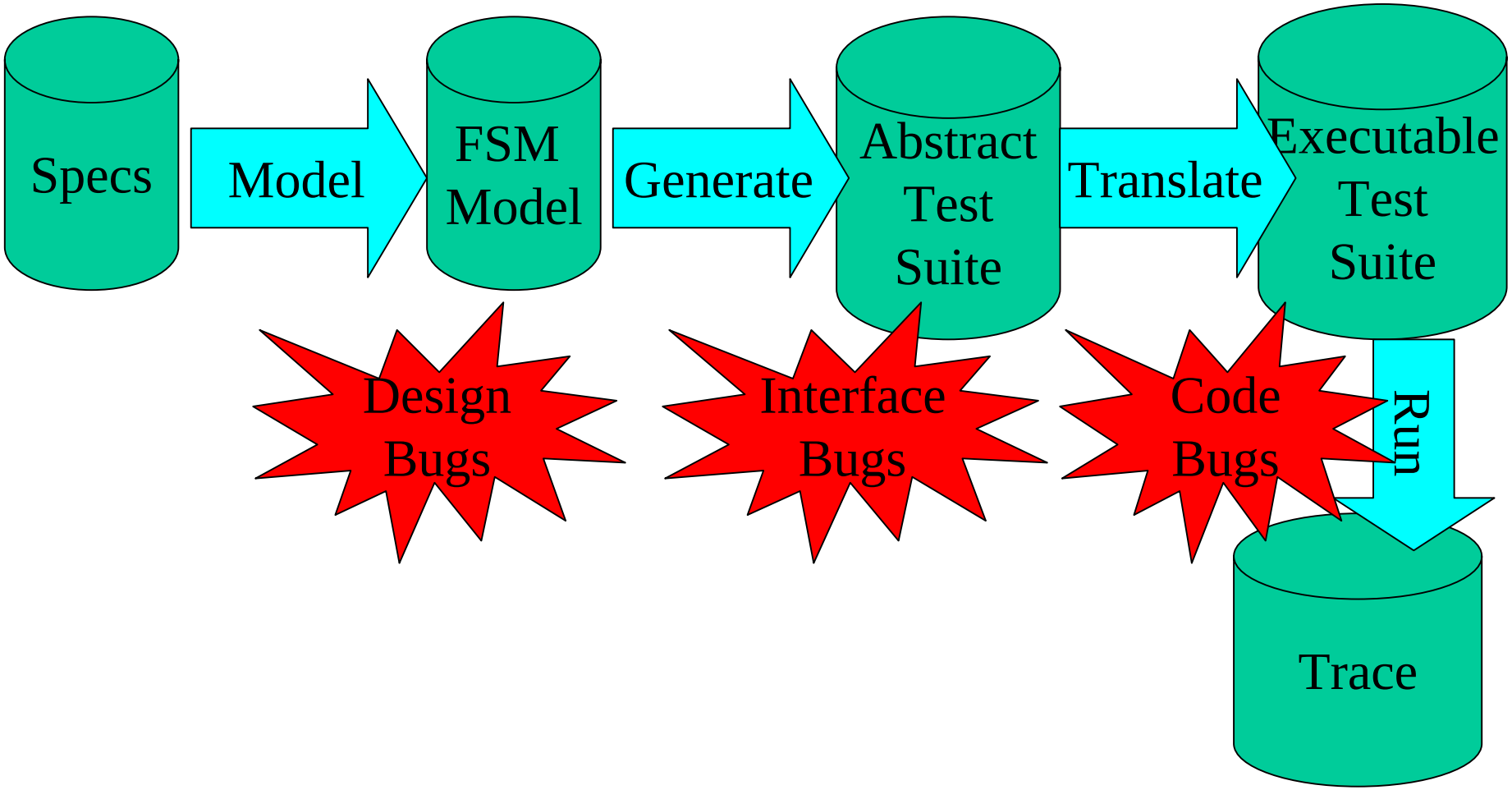
Motivation

- Testing is 40-70% of development cost
- Early bugs cost less than late bugs
- Famous disastrous bugs:
  - Therac-5 radiation therapy controller
  - Ariane 5 spaceship
  - Pentium floating point bug



Source: Applied Software Measurement, Capers Jones, 1996

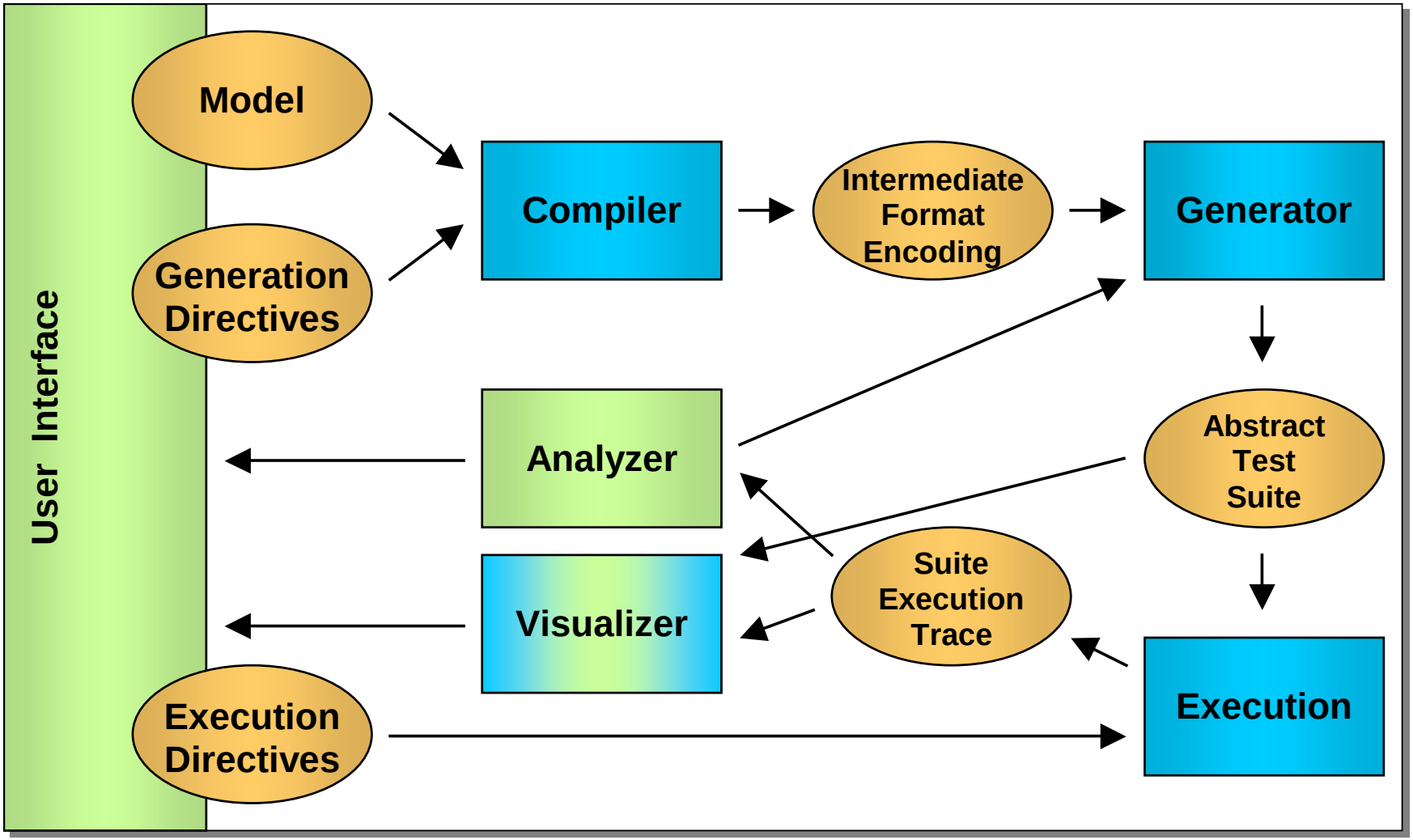
AGEDIS Methodology



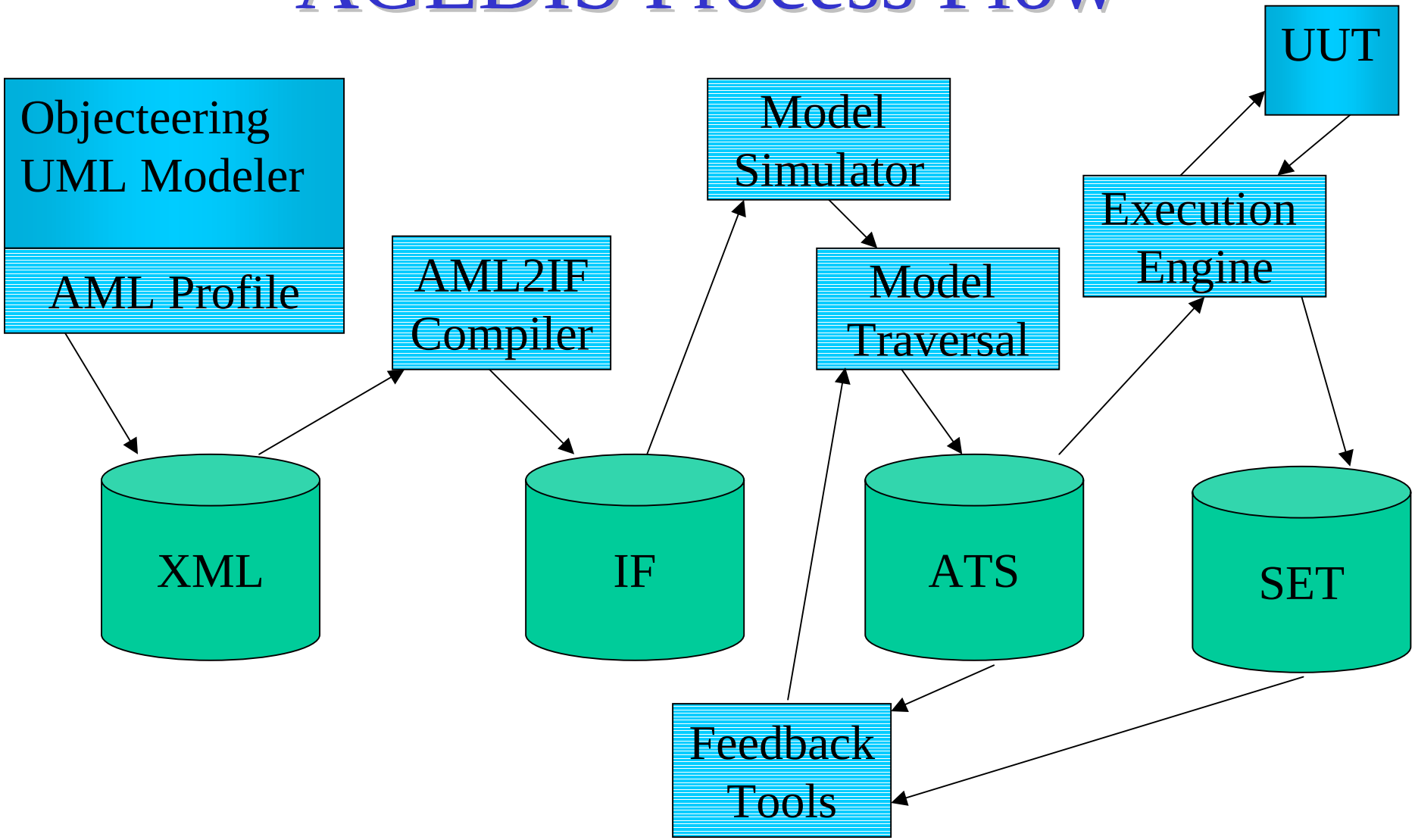
Benefits

- Starting from specification
  - Involves testers early in the development process
  - Teams testers with developers
  - Forces testability into product design
- Building behavioural model and test interface
  - Finds design and specification bugs - before code exists
  - The model is the test plan - and is easily maintained
- Automated test suite generation
  - Coverage is guaranteed - increases testing thoroughness
  - Matches coverage goals to testing budget
  - Zero test suite maintenance costs
- Automated test suite execution
  - Finds code and interface bugs
  - Includes a framework for the testing of distributed applications
  - Reduces test execution costs

AGEDIS Architecture



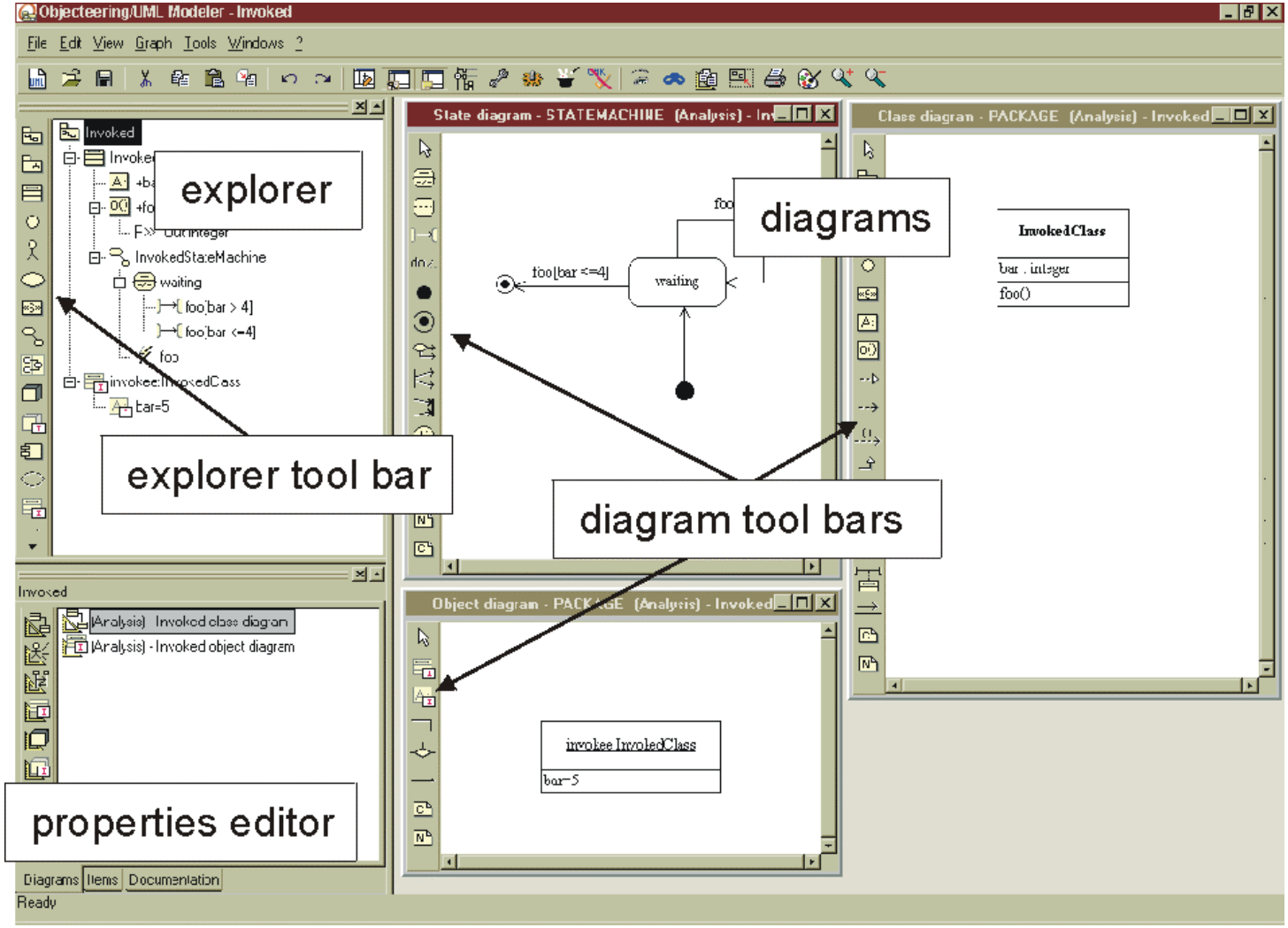
AGEDIS Process Flow



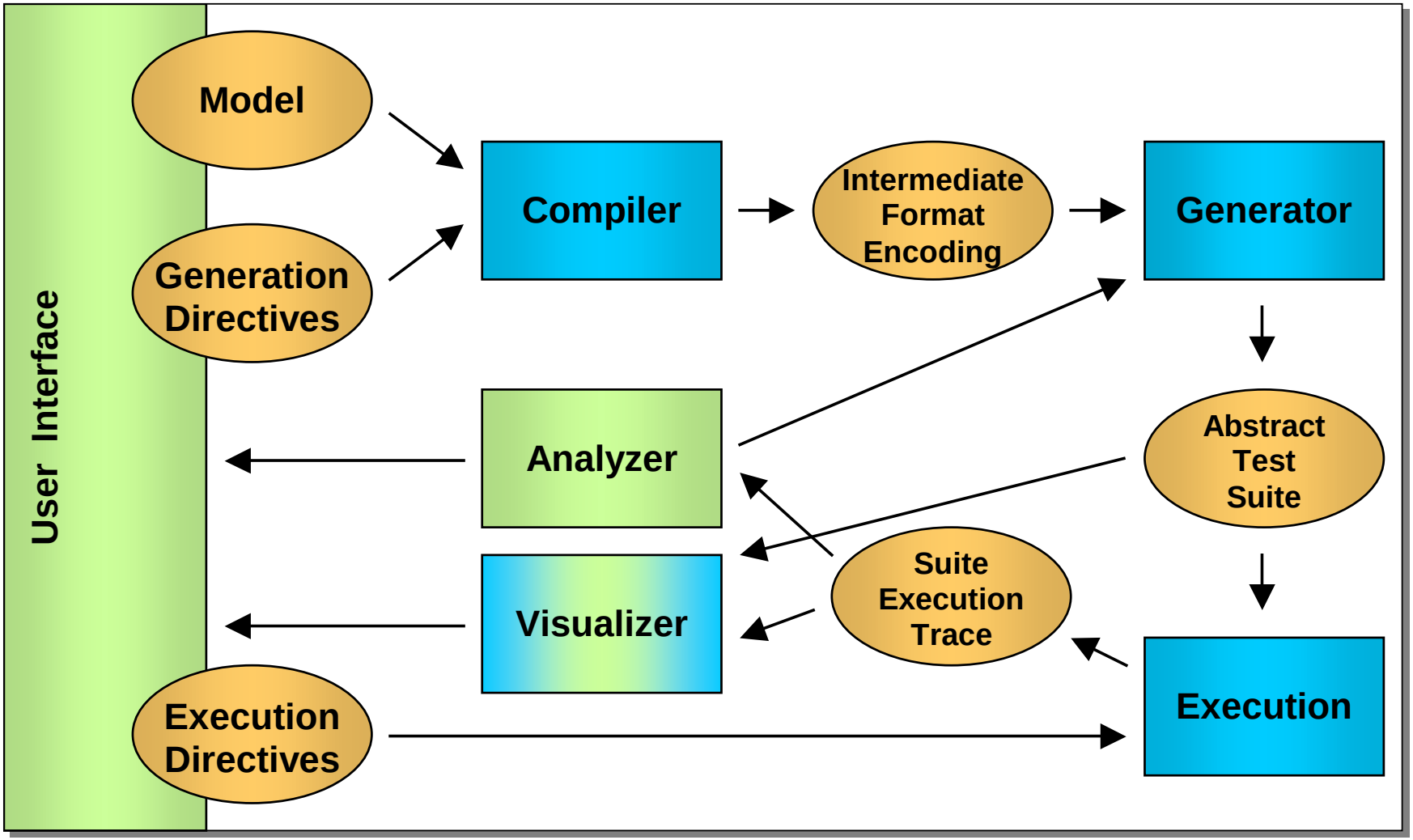
Modeling Environment

- The AGEDIS Modeling Language:
  - UML Class diagrams - structure
  - UML Object diagrams - snapshots
  - UML State diagrams – behaviour & test purposes
- Annotated with an action language – IF
- Currently use Objecteering UML modeling tool
- Tool profile to convert to XML
- General purpose XML to IF compiler

Modeling Environment



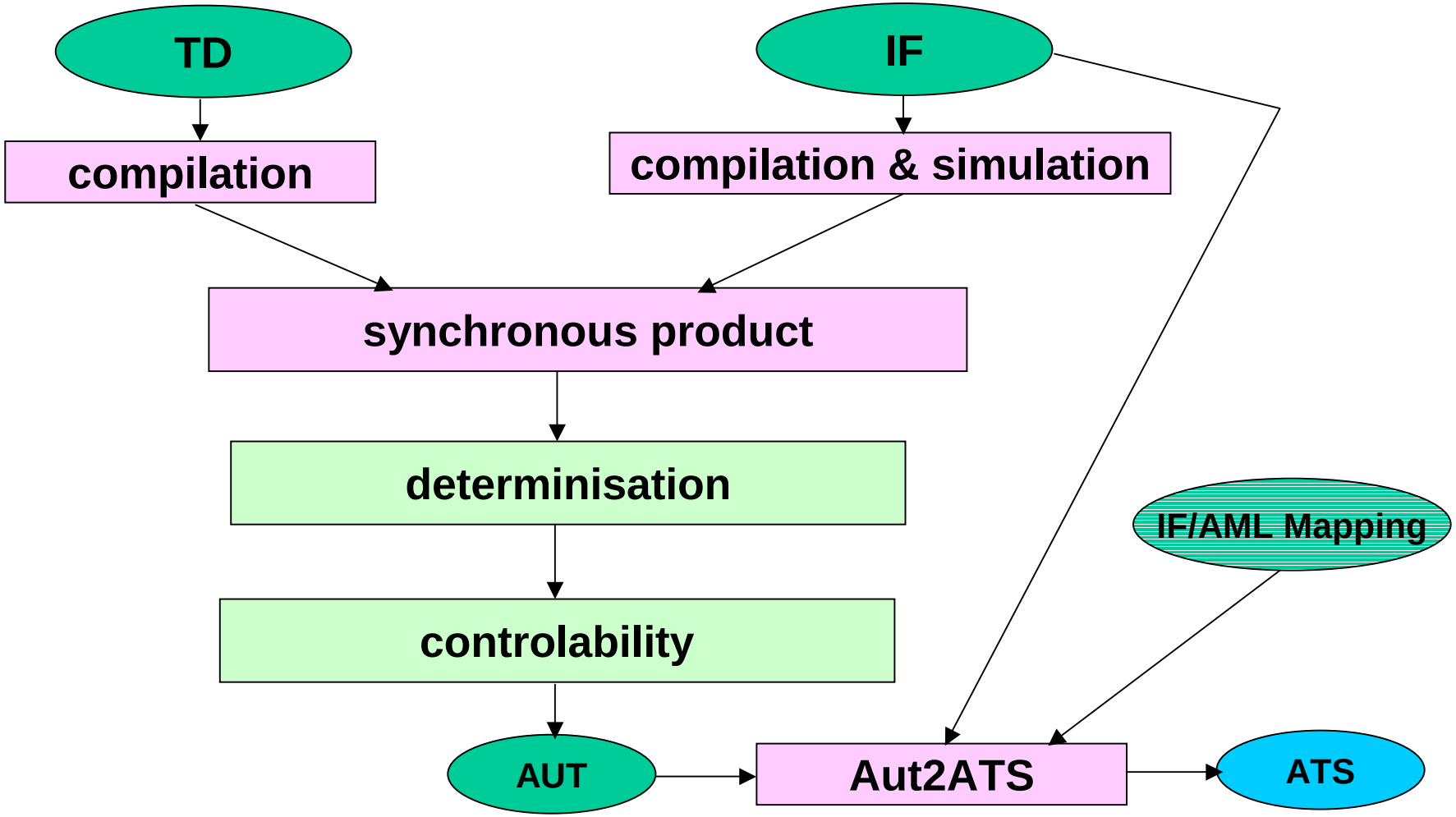
AGEDIS Architecture



Test Generator Background

- Based on GOTCHA and TGV
- GOTCHA
  - uses Murphi specification language
  - explicit traversal of state space
  - extensive coverage criteria
- TGV
  - language independent simulator
  - focus on distributed applications
  - explicit test purposes as sequences of interactions

Test Generator Structure

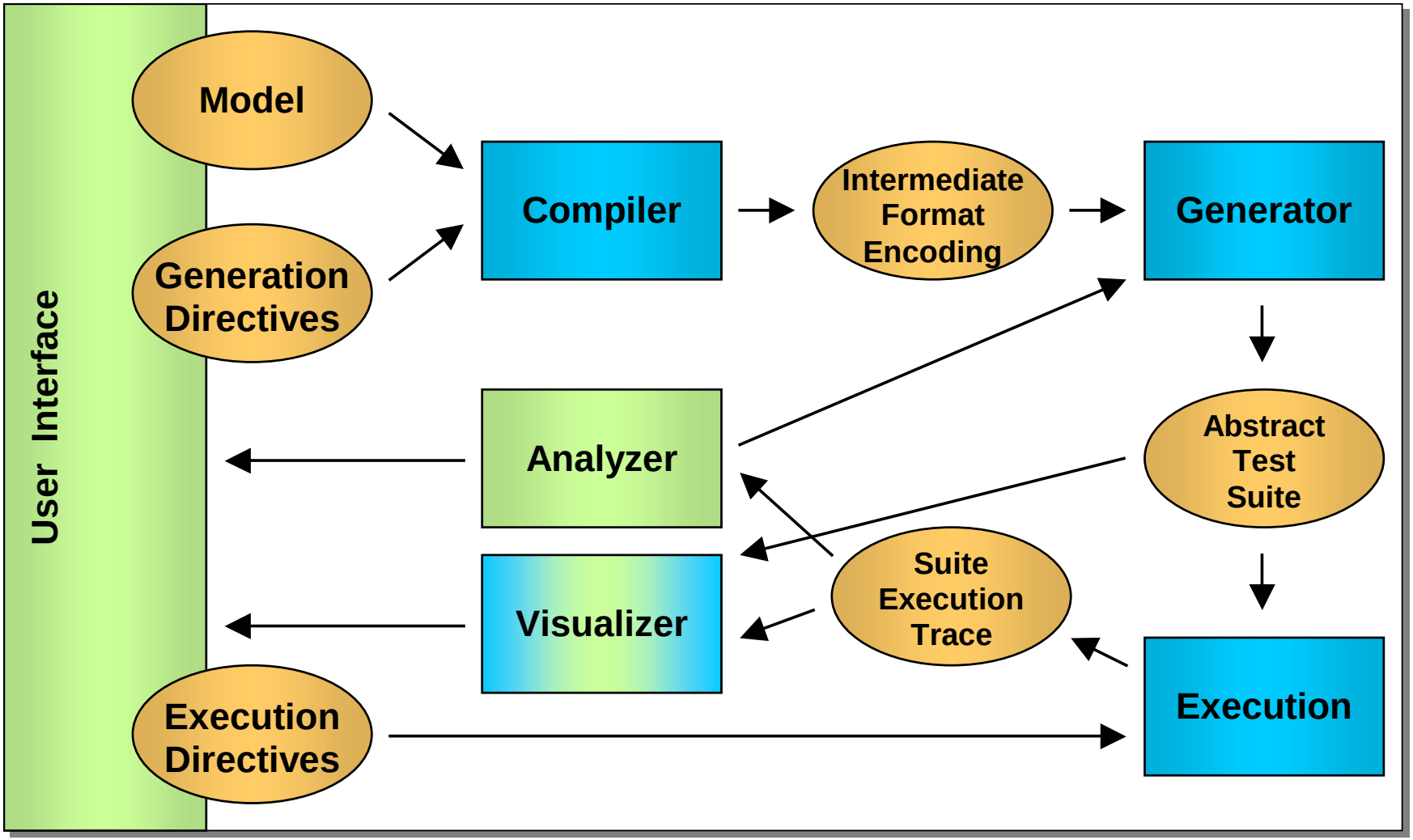


The ATS Format

- XML format – contains all the information needed for execution
  - Can be produced manually or automatically
- Model description
  - classes : constants, types, control & observable signatures

    (a special class is defined for the *tester*)
  - object identities
- Test Suite = set of test cases
  - « interaction graphs » between the tester and the SUT
  - associated verdicts (Pass, Fail, Inconclusive)

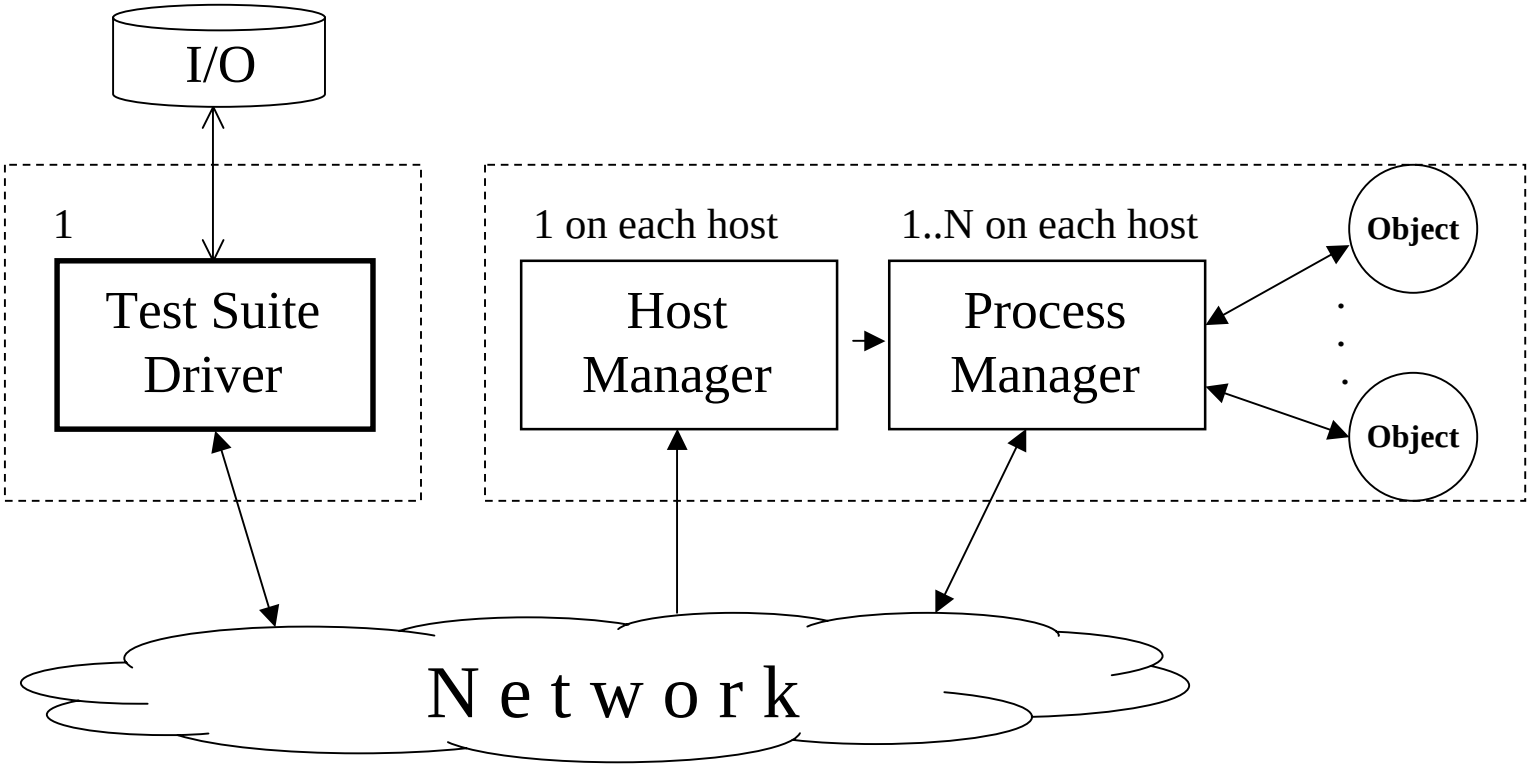
AGEDIS Architecture



Test Execution Engine

- Input: ATS and Test Execution Directives
- Output: Suite Execution Trace (XML)
- Multiple platform (e.g. Linux, Windows) support
- Multiple external interface (Java,C++,C) support
- Test distribution + centralized logging
- Interactive test case execution
- GUIs to view execution progress, inputs, and outputs

Execution Engine Architecture



Experiments with the Methodology

- File System
  - Duplicate testing with and without tools
  - 20% less resources, same bug detection quality
- Non-standard GUI Application
  - Failure
- Java garbage collector
- Automated GUI testing

Current Status

- It works!
- Objecteering profile gives easy access to AML
- Compiler is well-structured for future developments
- Test Generator creates multiple test paths with fewer inconclusive test cases
- Execution Engine demonstrates the benefits of abstract models for versatile testing of distributed SW

Future Plans

- MQSeries Experiment
- Productivity tools
- Integrated working environment
- Finished Tools
- France Telecom & Intrasoft Experiments
- Exploitation Activity