AGEDIS

Productivity Tools and UML Execution Framework

2nd IBM Software Testing and Verification Seminar

Haifa

December 18 2003

Alan Hartman

IBM Israel

IBM HAIFA LABS

Agenda

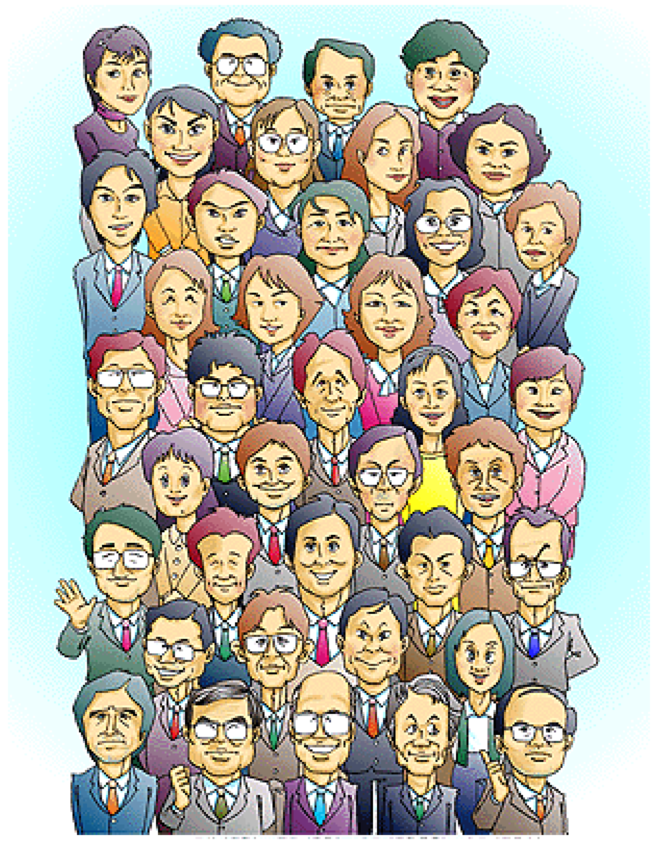
- AGEDIS Project
- Tool Architecture
- UML Execution
- Feedback Tools

AGEDIS Overview

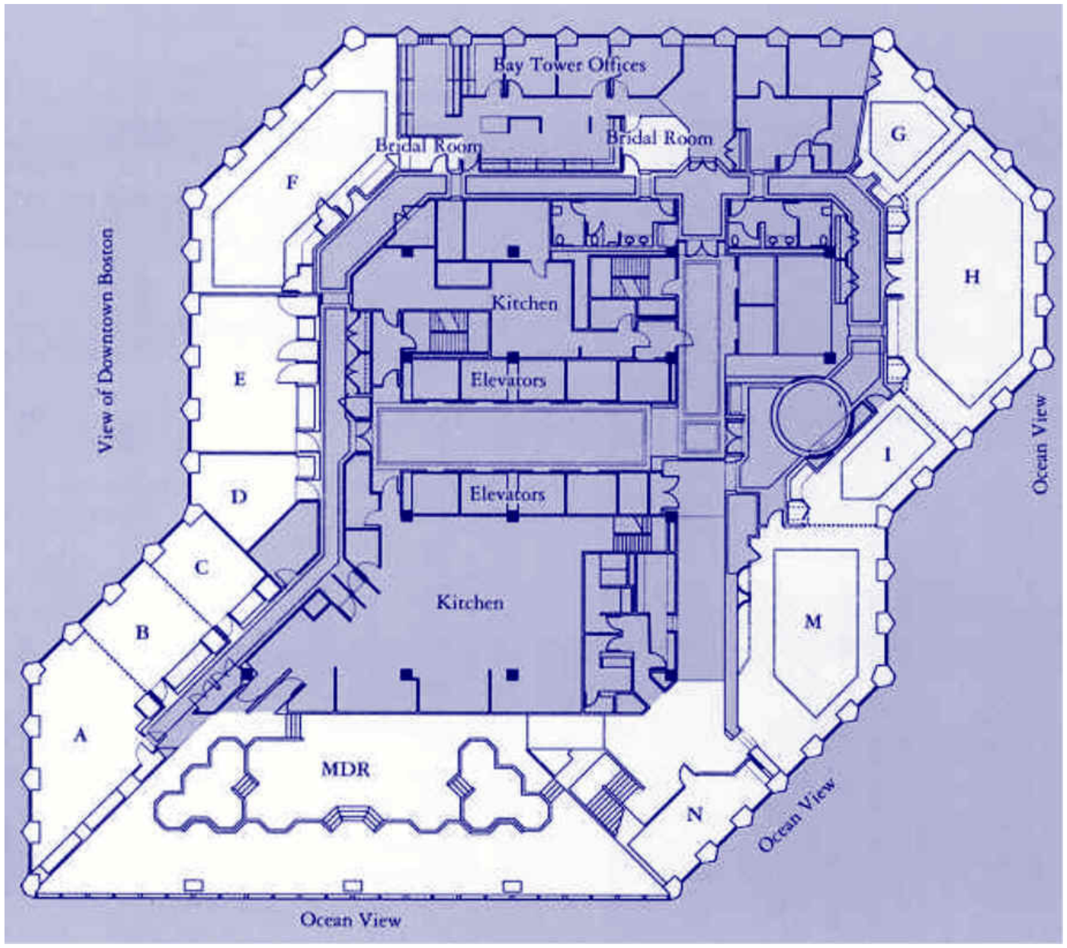
- Automated model-based test Generation and Execution for DIStributed systems
- Methodology and tools for model-based testing
- Open interfaces
- Mixture of academic and industrial partners
- Three phase timetable of experiment and development
- November 2001- February 2004

Consortium Partners

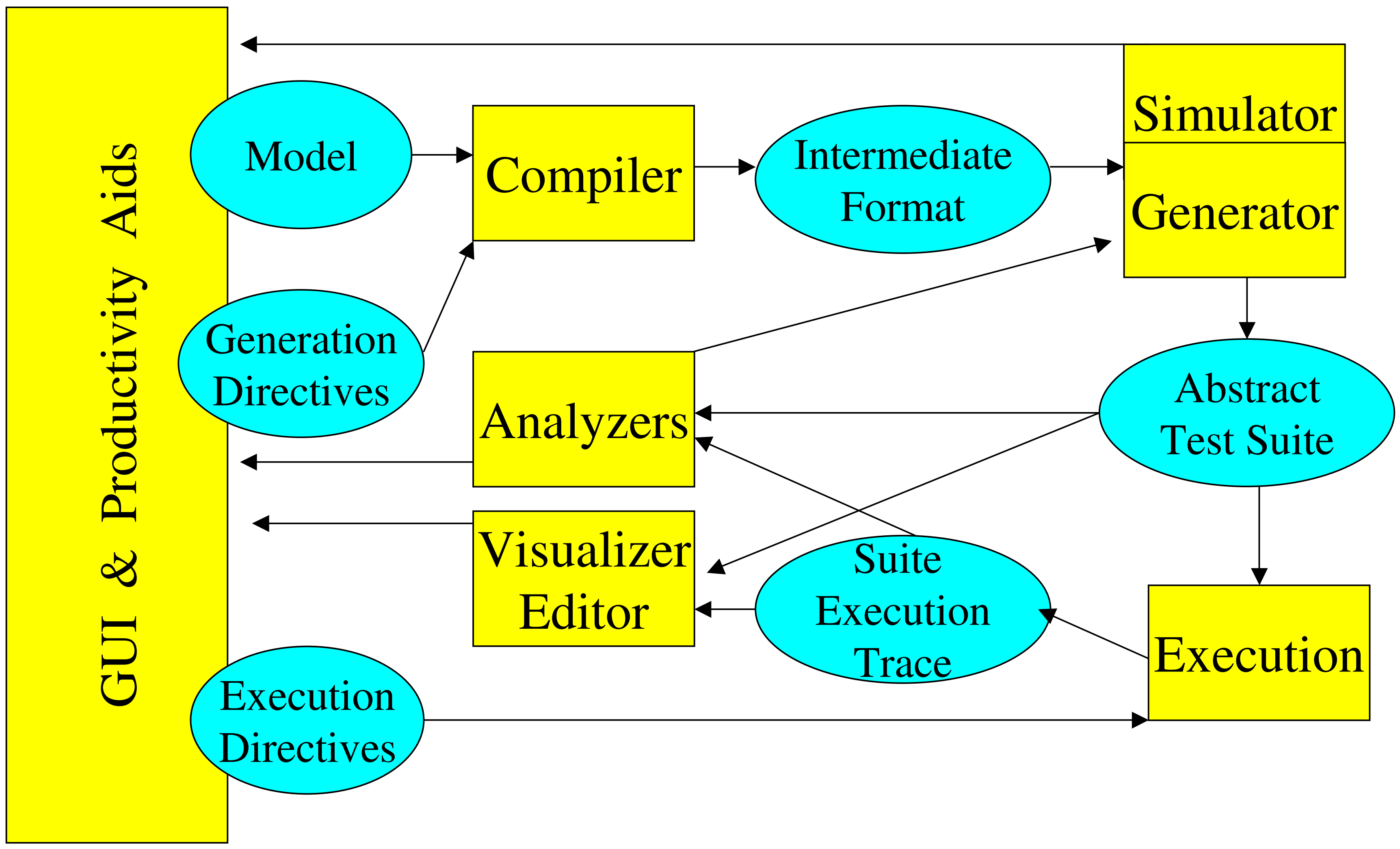
- IBM Haifa Research Lab
- Oxford University
- VERIMAG/IRISA
- Imbus
- France Telecom
- IBM UK
- Intrasoft International



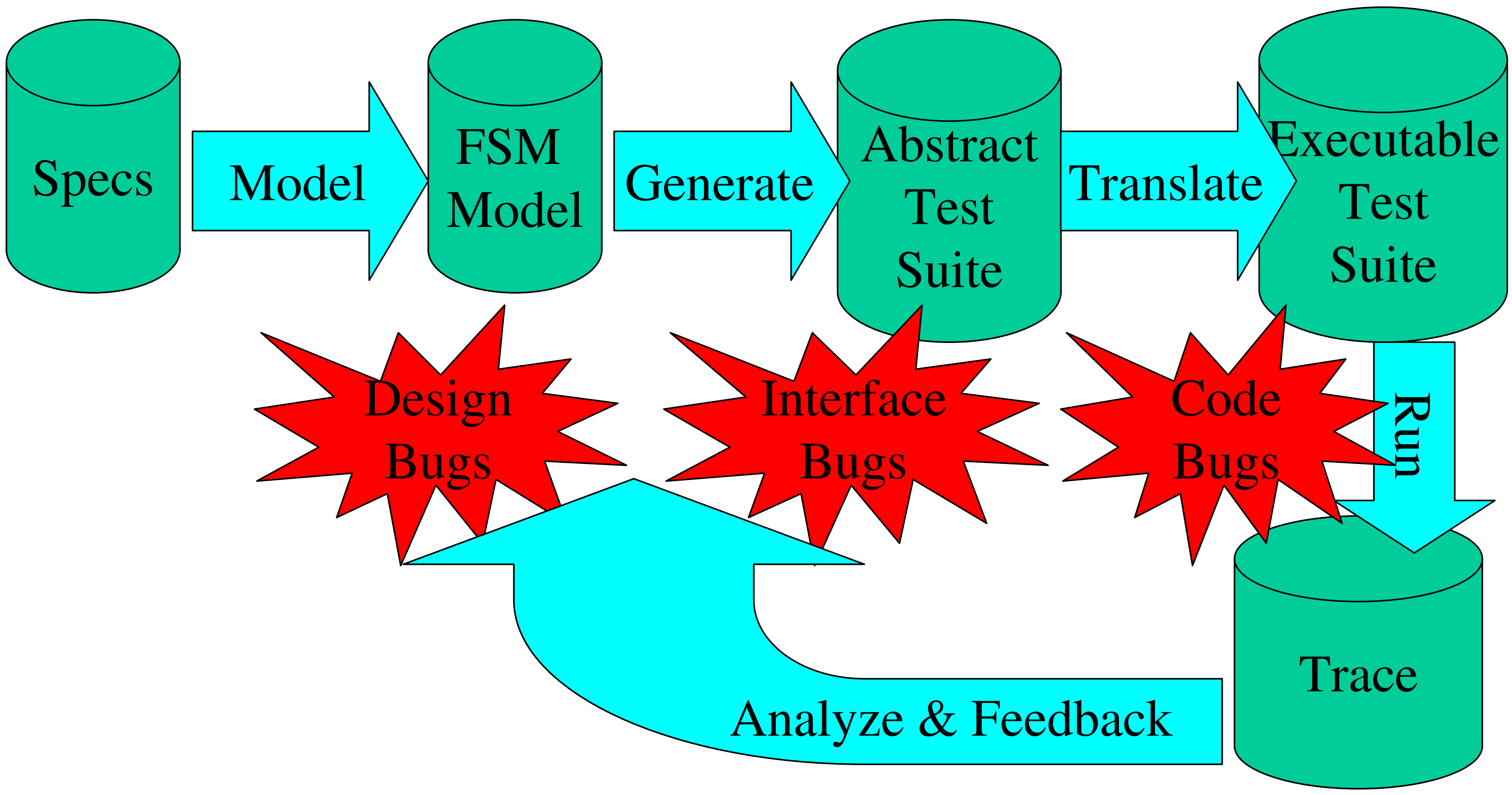
Architecture & Methodology



AGEDIS Architecture



AGEDIS Methodology

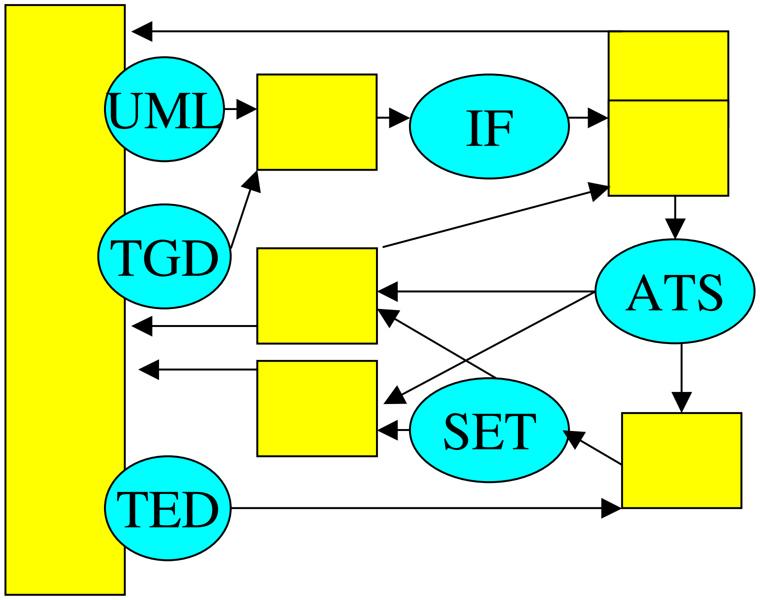


Benefits

- 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

Interfaces

- UML Profile for AGEDIS
- Test Generation Directives
- Test Execution Directives
- IF Model Execution Interface
- Abstract Test Suite
- Suite Execution Trace



User Modeling Interface

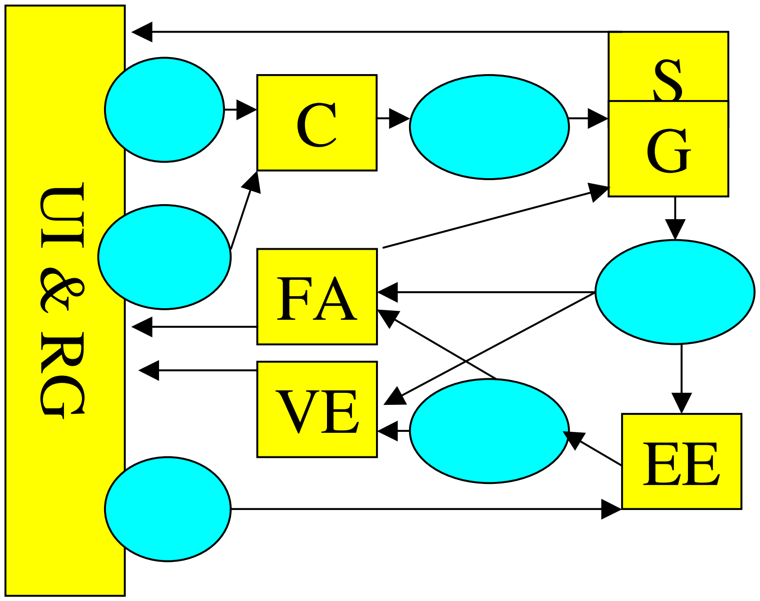
- The AGEDIS Modeling Language is a profile for UML 1.4:
  - UML Class diagrams - structure
  - UML Object diagrams - snapshots
  - UML State diagrams – behaviour & test purposes
- Annotated with an action language – IF

Test Suite and Trace Interface

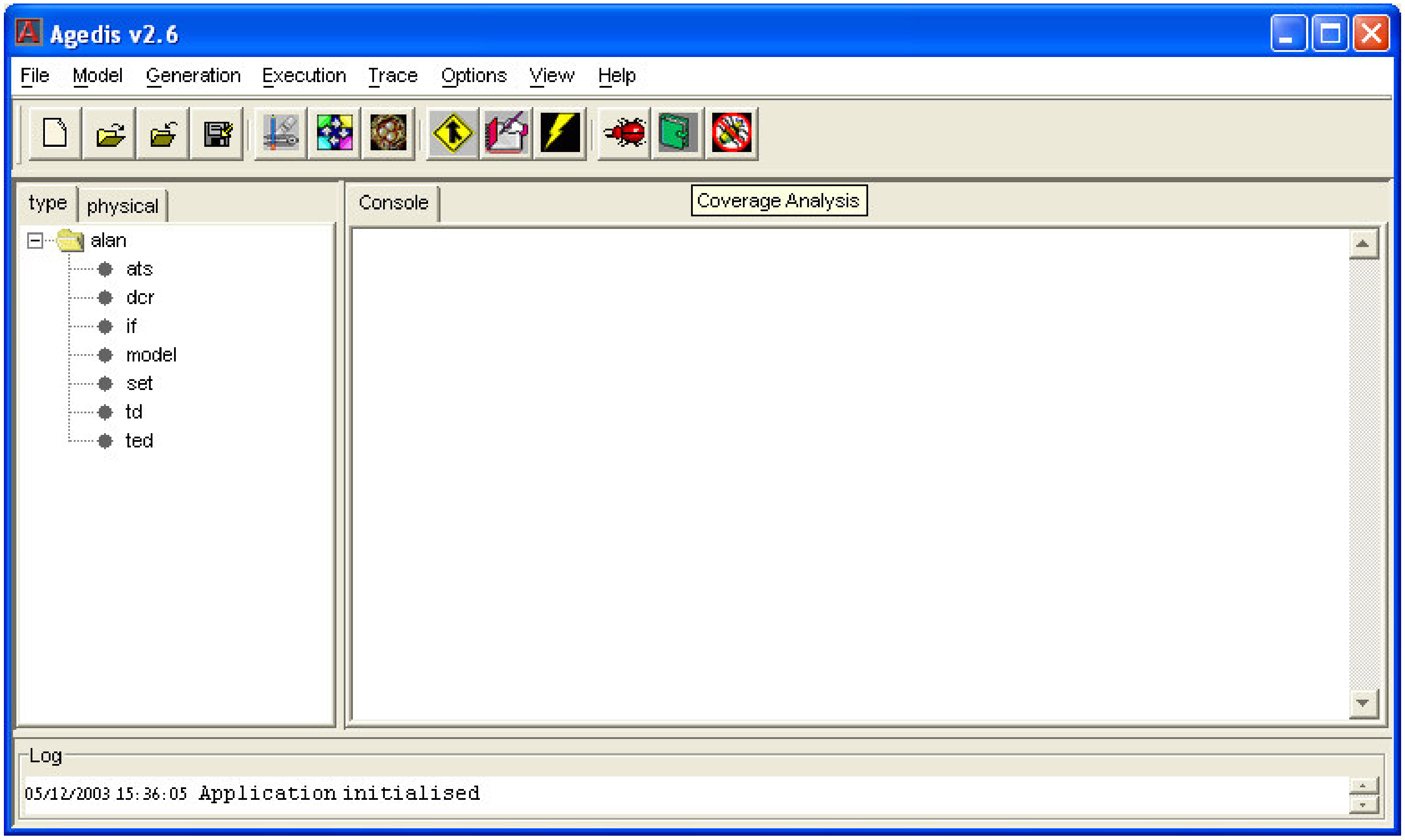
- XML schema – for test execution and tracing
- Model description
  - classes : constants, types, control & observable signatures
  - a special class is defined for the tester
  - object identities
- Test Suite - set of test cases
- Test Trace – record of executed test cases

Tools

- User Interface
- Modeler & Model Compiler
- Model Simulator
- Test Generator
- Test Execution Engine
- Test Suite/Trace Viewer/Editor
- Feedback & Analysis
- Bug Reporter
- Report Generator



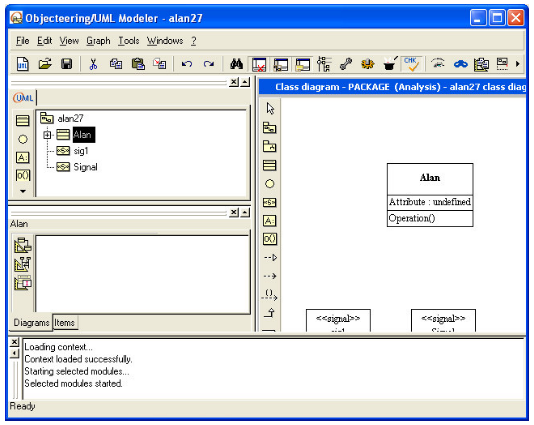
GUI



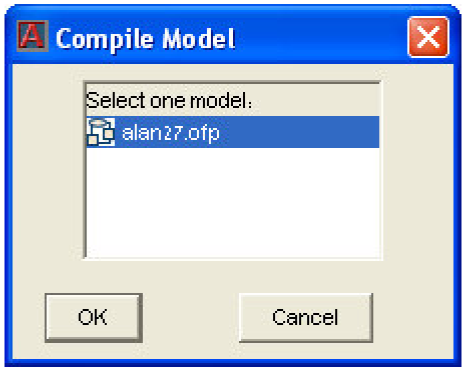
```
05/12/2003 15:36:05 Application initialised
```

Modeling Tool & Compiler

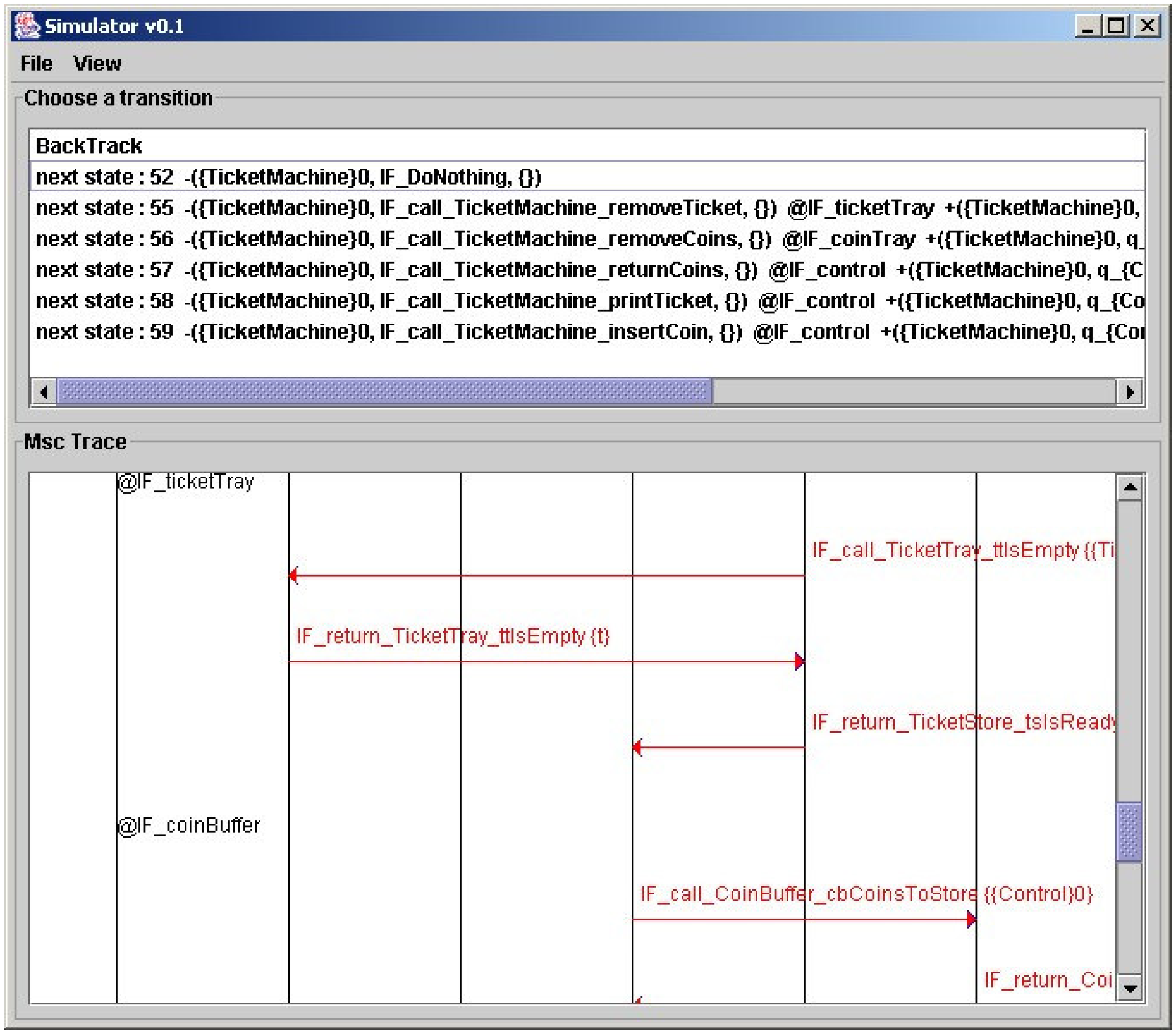
- Objecteering UML modeling tool
- Tool profile to convert to XML
- General purpose XML to IF compiler
  - Written in Java, with XMI in mind as a future input format


Objecteering/UML Modeler - alan27
File Edit View Graph Tools Windows ?
Class diagram - PACKAGE (Analysis) - alan27 class diag
alan27
Alan
sig1
Signal
Alan
Diagrams
Items
Alan
Attribute : undefined
Operation()
<<signal>>
<<signal>>

```
Loading context...
Context loaded successfully.
Starting selected modules...
Selected modules started.
Ready
```



UML Model Execution Dialog



Simulator v0.1

File View

Choose a transition

```
BackTrack
next state : 52  -({TicketMachine}0, IF_DoNothing, {})
next state : 55  -({TicketMachine}0, IF_call_TicketMachine_removeTicket, {}) @IF_ticketTray  +({TicketMachine}0,
next state : 56  -({TicketMachine}0, IF_call_TicketMachine_removeCoins, {}) @IF_coinTray  +({TicketMachine}0, q_
next state : 57  -({TicketMachine}0, IF_call_TicketMachine_returnCoins, {}) @IF_control  +({TicketMachine}0, q_{C
next state : 58  -({TicketMachine}0, IF_call_TicketMachine_printTicket, {}) @IF_control  +({TicketMachine}0, q_{Co
next state : 59  -({TicketMachine}0, IF_call_TicketMachine_insertCoin, {}) @IF_control  +({TicketMachine}0, q_{Coi
```

Msc Trace

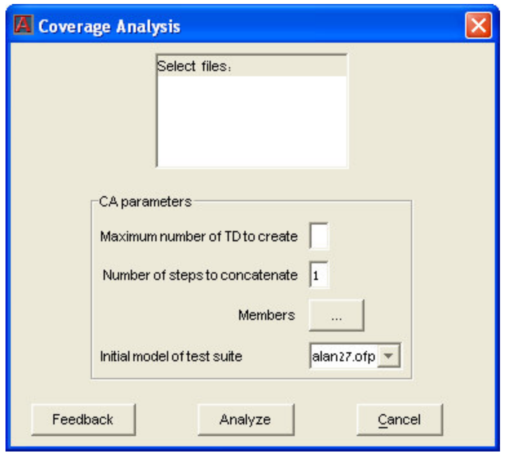
@IF_ticketTray
IF_call_TicketTray_ttIsEmpty ({Ti
IF_return_TicketTray_ttIsEmpty (t)
IF_return_TicketStore_tsIsReady
@IF_coinBuffer
IF_call_CoinBuffer_cbCoinsToStore ({Control}0)
IF_return_Coi

UML Execution Framework

- Upper window lists all available actions
- The tick symbol indicates that input is required from the environment (tester).
- Tester chooses the appropriate input
- Model responds with actions (user chooses from the non-deterministic alternatives)
- Until next tick point
- Outputs message sequence chart in lower window

Feedback & Analysis Tools

- Coverage analysis
  - Detect uncovered areas of the model in either test suite or test trace
  - Create test purposes to reach them
  - Invokes FoCus, a functional coverage tool from www.alphaworks.com
- Defect analysis
  - Clustering of defects
  - Feature extraction from clusters
  - Create test purposes to reproduce the bug



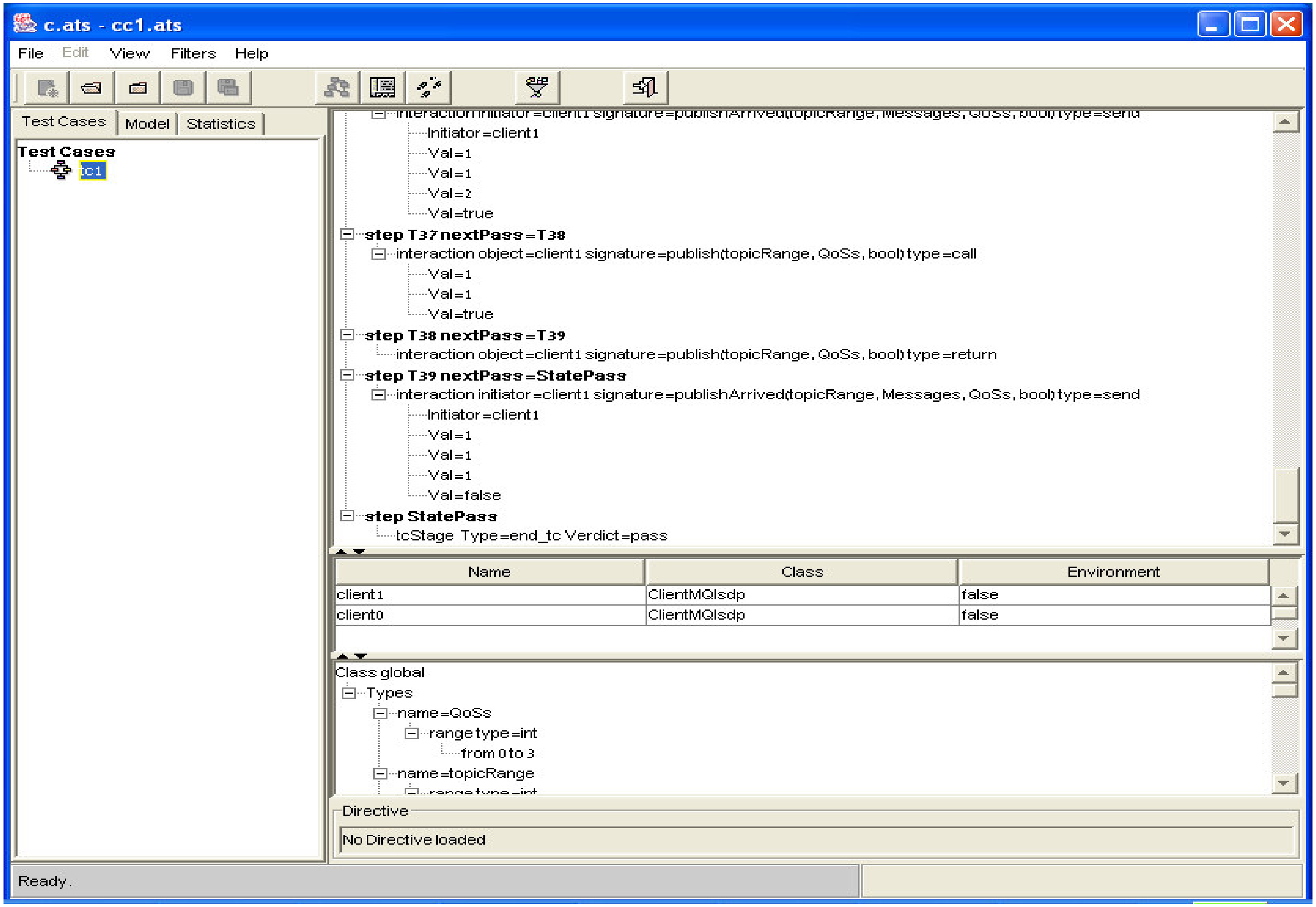
Maximum number of TD to create

Number of steps to concatenate 1

Members ...

Initial model of test suite alan27.ofp

Abstract Test Suite



```
    interaction initiator=client1 signature=publishArrived(topicRange, Messages, QoSs, bool) type=send
        Initiator=client1
        Val=1
        Val=1
        Val=2
        Val=true
step T37 nextPass=T38
    interaction object=client1 signature=publish(topicRange, QoSs, bool) type=call
        Val=1
        Val=1
        Val=true
step T38 nextPass=T39
    interaction object=client1 signature=publish(topicRange, QoSs, bool) type=return
step T39 nextPass=StatePass
    interaction initiator=client1 signature=publishArrived(topicRange, Messages, QoSs, bool) type=send
        Initiator=client1
        Val=1
        Val=1
        Val=1
        Val=false
step StatePass
    tcStage Type=end_tc Verdict=pass
```

Name	Class	Environment
client1	ClientMQIsdp	false
client0	ClientMQIsdp	false

```
Class global
  Types
    name=QoSs
      range type=int
        from 0 to 3
    name=topicRange
      range type=int
```

Directive

No Directive loaded

Ready.

Coverage Analysis

- Collects statistics on
  - Methods called, including parameter values
  - Observable variable values
  - Return values
  - Exceptions
- Also on sequences of the above
- Creates test purposes on least frequently covered sequences

Defect Analysis

- A defect trace is seen as a sequence of stimuli and observations which culminate in an exception or an observation conflict with the model
- Distance between sequences is defined by weighted measures depending on distance from the defect and equal stimuli
- E.g. s1o1s2o2e1 is different from s1o1s3o2e1, but close, whereas s1o3s4o4e1 is more different
- Clusters are formed from the distance matrix
- Experimental work still ongoing to determine good distance measures

AGEDIS' Future Plans

- Finishing Touches
- Exploitation Activity
- Incorporation in wider Model Driven SE Effort

Thanks to:

- K. Nagin, O. Edelstein, A. Kirshin, S. Olvovsky, M. Berg, L. Raskin, T. Shiran, C. Sacharen, M. Barshay, D. Neimer
- L. Mounier, M. Bozga, Y. Lakhnech
- T. Jeron, E. Demairy, V. Tschaen
- J. Davies, A. Cavarra, C. Crichton, J. Woodcock, M. Field
- A. Ramfos., S. Liapis, N. Giannelos, A. Hondouridakis, M. Sardis, K. Bechrakis, V. Akousi-Krivki
- K. Dussa-Zieger, J. Trost, B. Nossem, T. Linz, B. Mattern, J. Hofer, H. Raessler, T. Rossner,
- I. Craggs, I. Griffiths
- Y.-M. Quemener, D. Vincent, T. Heuillard, N. Moteau
- A.Bertolino, A. Wills, S. Reid