Model-Based Testing

*There is Nothing More Practical than a Good Theory*

Jan Tretmans

*TNO – ESI, Eindhoven, NL and Radboud University, Nijmegen, NL*
Jan Tretmans

TNO
Embedded Systems Innovation
Eindhoven
The Netherlands

Radboud University
Nijmegen
The Netherlands
Research cooperation with leading Dutch high-tech multinational industries & SME’s

Research cooperation with all Dutch universities with embedded systems research

Research cooperation in EU projects
Overview

Model-Based Testing

Theory
Jan Tretmans

- MBT: What and Why
- MBT: A theory with labelled transition systems and ioco
- Variations:
  - Test selection
  - Test-based modelling

Model-Based Testing

Practice
Machiel van der Bijl

- MBT: Practical exercises with Axini Test Manager
- MBT: The difference between theory and practice
Model-Based Testing
(Software) Testing

checking or measuring
some quality characteristics
of an executing object
by performing experiments
in a controlled way
w.r.t. a specification
Testing Complexity

testing effort grows exponentially with system size

x : [0..9]  10 ways that it can go wrong
10 combinations of inputs to check

x : [0..9]  y : [0..9]  100 ways that it can go wrong
100 combinations of inputs to check

y : [0..9]  1000 ways that it can go wrong
1000 combinations of inputs to check

x : [0..9]  y : [0..9]  z : [0..9]

Automation of testing is necessary
Testing Challenges

• Increasing complexity
  • more functions, more interactions, more options and parameters

• Increasing size
  • building new systems from scratch is not possible anymore
  • integration of legacy-, outsourced-, off-the shelf components
  • abstract from details: models

• Blurring boundaries between systems
  • more, and more complex interactions between systems
  • systems dynamically depend on other systems, systems of systems

• Blurring boundaries in time
  • requirements analysis, specification, implementation, testing, installation, maintenance overlap
  • more different versions and configurations
Model-Based Testing: Why

- Mastering increase in complexity, and quest for higher quality
  - testing cannot keep pace with development

- Dealing with models and abstraction
  - model-based development: UML, MDA, Simulink/Matlab

- Promises better, faster, cheaper testing
  - algorithmic generation of tests and test oracles: tools
  - maintenance of tests through model modification

Software bugs / errors cost US economy yearly:
$ 59.500.000.000 (www.nist.gov)
$ 22 billion could be eliminated…
Model-Based Testing (MBT)
MBT : Black-Box Testing of Functionality

- System
- Integration
- Module
- Unit
- Portability
- Maintainability
- Efficiency
- Usability
- Reliability
- Functionality
- Accessibility

Phases:

- White Box
- Black Box

Aspects:

- Portability
- Maintainability
- Efficiency
- Usability
- Reliability
- Functionality

Accessibility
Evolution of Testing

- Manual Testing
- Record & Playback
- Scripted
- Keyword-Driven
- Model-Based Testing
Testing 1: Manual Testing

1. Manual testing
Testing 2: Scripted Testing

1. Manual testing
2. Scripted testing
Testing 3: Keyword-Driven Testing

1. Manual testing
2. Scripted testing
3. Keyword testing

- SUT
- Test execution
- Pass/FAIL
- High-level test notation
Testing 4: Model-Based Testing

1. Manual testing
2. Scripted testing
3. Programmed testing
4. Model-based testing
Model-Based

Verification, Validation, Testing, . . . . .
Validation, Verification, and Testing

validation

(model-based) testing

SUT
Verification and Testing

Model-based verification:
- formal manipulation
- prove properties
- performed on model

Model-based testing:
- experimentation
- show error
- concrete system

Verification is only as good as the validity of the model on which it is based

Testing can only show the presence of errors, not their absence
Models
Models
Models: Labelled Transition Systems

Labelled Transition System: $\langle S, L_i, L_u, T, s_0 \rangle$

- **States:** $S$
- **Input Actions:** $L_i$
- **Output Actions:** $L_u$
- **Transitions:** $T$
- **Initial State:** $s_0$

$? = \text{input}$
$! = \text{output}$
A Theory of Model-Based Testing with Labelled Transition Systems
Model-Based Testing

- Model-based test generation
- System model
- Test execution
- SUT

Pass / Fail
MBT: Validity

- SUT conforms to model
- SUT passes tests
- System model
- SUT
- Model-based test generation
- Test execution
- Pass fail
Models: Generation of Test Cases

specification model

!coffee

?coin

!alarm

?button

test case model

! coin

! button

?coffee

?alarm

pass

fail

fail
Models: Generation of Test Cases

specification model

!coffee

?coin

!alarm

?button

test case model

! coin

? coffee

fail

pass

fail

? alarm
• Four components in parallel, in any order

\[
\begin{align*}
\text{task}(&\, \text{start}?, \text{ready}!) \\
\text{taskA} &\ := \ \text{task} (\text{startA}?, \text{readyA}!) \\
\text{taskB} &\ := \ \text{task} (\text{startB}?, \text{readyB}!) \\
\text{taskC} &\ := \ \text{task} (\text{startC}?, \text{readyC}!) \\
\text{taskD} &\ := \ \text{task} (\text{startD}?, \text{readyD}!)
\end{align*}
\]

\[
\text{model} := \text{taskA} ||| \text{taskB} ||| \text{taskC} ||| \text{taskD}
\]
MBT : Abstract from Scheduling Details
MBT : Abstract from Scheduling Details
MBT: Nondeterminism, Underspecification

- non-determinism
- under-specification
- specification of properties rather than construction
MBT with LTS and ioco

- SUT ioco model
- sound \(\Downarrow\) exhaustive
- SUT passes tests

\[\text{SUT passes tests} \iff \text{input/output conformance} \iff \text{ioco} \]

- ioco test generation
- set of LTS tests
- LTS test execution
- pass fail
- LTS behaving as input-enabled LTS
MBT: Argue about Validity of Tests

specification model

s

?dime

!tea

!coffee

implementation

i

ioco

s

!choc

!tea

i

fails

t

generated test case

j

?dime

?tea

?coffee

?choc

pass

pass

fail

fail
Model-Based Testing
with Labelled Transition Systems

There is Nothing More Practical
than a Good Theory
Overview

• MBT: Tools
• MBT: Under-specification
• MBT: Test selection
• MBT: Towards test selection for ioco
• Refinement for ioco
• Test-based modelling = Automata learning
Model-Based Testing

Tools
MBT: Off-Line - On-Line

- Model-based test generation
- System model
- Test execution
- SUT
- Pass, Fail
MBT : Off-Line = Batch
MBT: On-Line = On-the-Fly
Model-Based Testing:
Variations for Underspecification
Variations on a Theme

- \( i \ ioco \ s \iff \forall \sigma \in \text{Straces}(s) : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma) \)
- \( i \leq_{\text{ior}} s \iff \forall \sigma \in (L \cup \{\delta\})^* : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma) \)
- \( i \ \text{ioconf} \ s \iff \forall \sigma \in \text{traces}(s) : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma) \)
- \( i \ ioco_F s \iff \forall \sigma \in F : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma) \)
- \( i \ uioco \ s \iff \forall \sigma \in \text{Utraces}(s) : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma) \)
- \( i \ \text{mioco} s \) multi-channel ioco
- \( i \ \text{wioco} s \) non-input-enabled ioco
- \( i \ \text{eco} e \) environmental conformance
- \( i \ \text{sioco} s \) symbolic ioco
- \( i \ (r)\text{tioco} s \) (real) timed tioco (Aalborg, Twente, Grenoble, Bordeaux,..... )
- \( i \ \text{rioco} s \) refinement ioco
- \( i \ \text{hioco} s \) hybrid ioco
- \( i \ \text{qioco} s \) quantified ioco
- \( i \ \text{poco} s \) partially observable game ioco
- \( i \ \text{stioco}_{D} s \) real time and symbolic data
- .......
Underspecification: ioco and uioco

\[ i \text{ ioco } s \quad \overset{\text{def}}{=} \quad \forall \sigma \in \text{Straces}(s) : \quad \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

Implementation \(i\)

\[ i \text{ ioco } s \quad \overset{\text{def}}{=} \quad \forall \sigma \in \text{Straces}(s) : \quad \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

Specification \(s\)
Underspecification: uioco

\[ i \ ioco \ s \quad =_{\text{def}} \quad \forall \sigma \in Straces(s) : \quad \text{out}(i \ \text{after} \ \sigma) \subseteq \text{out}(s \ \text{after} \ \sigma) \]

\[ i \ uioco \ s \quad =_{\text{def}} \quad \forall \sigma \in Utraces(s) : \quad \text{out}(i \ \text{after} \ \sigma) \subseteq \text{out}(s \ \text{after} \ \sigma) \]

\[ Utraces(s) = \]

\[ \{ \sigma \in Straces(s) \mid \forall \sigma_1 \ ?b \ \sigma_2 = \sigma : \quad s \ \text{after} \ \sigma_1 \ \text{must} \ ?b \ \} \]

\[ a \ ?a \in \ Straces(s) \quad \text{ioco} \subseteq \ uioco \]

\[ a \ ?a \notin \ Utraces(s) \]
Test Selection
in Model-Based Testing
Test Selection

• Exhaustiveness never achieved in practice

• Test selection = select subset of exhaustive test suite, to achieve confidence in quality of tested product
  – select best test cases capable of detecting failures
  – measure to what extent testing was exhaustive: coverage

• Optimization problem

*best possible testing* ⇔ *within cost/time constraints*
Test Selection: Approaches

1. random
2. domain / application specific: test purposes, test goals, …
3. model / code based: coverage
   – usually structure based
Towards Test Selection

in the ioco Framework
Test Selection for \texttt{uioco}

\[
i \text{\texttt{uioco}} \ s \ =_{\text{def}} \ \forall \ \sigma \in Utraces(s) : \ \text{out}(i \ \text{after} \ \sigma) \subseteq \ \text{out}(s \ \text{after} \ \sigma)
\]

\textit{Selection of Sub-Set of UTraces}

- Select: \[ M \subseteq Utraces(s) \]
- Test for: \[ i \ \text{\texttt{uioco}}M \ s \]
  \[ \iff \forall \ \sigma \in M : \ \text{out}(i \ \text{after} \ \sigma) \subseteq \ \text{out}(s \ \text{after} \ \sigma) \]
- Coverage: \[ \frac{\# \ M}{\# \ Utraces(s)} \]
Test Selection for $\text{uioco}$

\[
\text{out}(s \text{ after } ?\text{but} \delta \delta ?\text{but}) = \text{out}(s \text{ after } ?\text{but} \delta ?\text{but})
\]

i.e. if already tested for $?\text{but} \delta ?\text{but}$
what does testing for $?\text{but} \delta \delta ?\text{but}$ add?

\[
\text{out}(s \text{ after } ?\text{but}) = \{ \text{!cof}, \text{!tea}, \delta \}
\]

i.e. everything is allowed -
what shall be tested then?

The set $U\text{traces}$ is not minimal, i.e., elements are dependent
Test Selection for uioco

\[ i \text{ uioco } s \quad =_{\text{def}} \quad \forall \sigma \in Utraces(s) : \quad out(i \text{ after } \sigma) \subseteq out(s \text{ after } \sigma) \]

*Take weaker specification* \( s' \)

* = inverse of refinement*

\[ s \leq s' \]

\[ \iff SUT(s) \subseteq SUT(s') \]

\[ \iff \{ i \mid i \text{ uioco } s \} \subseteq \{ i \mid i \text{ uioco } s' \} \]
Test Selection for **uioco**

\[ \text{i uioco } s \text{ } =_{\text{def}} \forall \sigma \in Utraces(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

\[ s \leq s' \iff SUT(s) \subseteq SUT(s') \]

\[ \iff \{ i \mid \text{i uioco } s \} \subseteq \{ i \mid \text{i uioco } s' \} \]

**Coverage:**

\[
\frac{\# SUT(s)}{\# SUT(s')} \leq
\]
Test Selection: Lattice of Specifications

- $S_1$ is stronger than $S_2 \iff S_1 \leq S_2 \iff \{ i \mid i \text{ u/o/co } S_1 \} \subseteq \{ i \mid i \text{ u/o/co } S_2 \}$

- If specs are input-enabled then ioco is preorder then $\leq \equiv \text{ u/o/co}$

- $S_T \equiv \text{ top element} \equiv \text{ allows any impl.} \equiv \text{ chaos } \chi$

- Diagram showing the lattice structure with $S_1$, $S_2$, $S_3$, and $S_T$. 
  - $S_T$ is the top element.
  - $S_1$ allows any implementation.
  - $S_3$ is the chaos element.

- $L_u$ and $L_I$ are input-enabled specifications.

- $CS$ is a subset of $S_1$.
Test Selection for $uioco$

\[
i \ u i o c o \ s \ =_{\text{def}} \ \forall \ \sigma \in Utraces(s) : \ \text{out}(i \ \text{after} \ \sigma) \subseteq \ \text{out}(s \ \text{after} \ \sigma)
\]

\[
s \leq s' \ \iff \ SUT(s) \subseteq SUT(s')
\]

\[
\iff \ \{ \ i \ \mid \ i \ u i o c o \ s \} \subseteq \ \{ \ i \ \mid \ i \ u i o c o \ s' \}
\]

Requires refinement preorder $\leq$ on specifications.

$ioco / uioco$ are not refinement preorders and are only defined for input-enabled implementations.
Set of Required Traces

\[ Rtraces(s) \overset{\text{def}}{=} \{ \sigma \in Utraces(s) \mid \delta \text{ is not a substring of } \sigma, \]

\[ \sigma \text{ does not end with } \delta, \]

\[ out(s \text{ after } \sigma) \neq L_U \cup \{ \delta \} \} \]

\[ \varepsilon \notin Rtraces(s) \]

\[ \delta \overset{\text{def}}{=} Utraces(s) \]

\[ \delta \overset{\text{def}}{=} Utraces(s) \]
Set of Required Traces

**Rtraces** throw away superfluous traces, and only those

1. For input enabled implementations:

   \[ i \text{ uioco } s =_{\text{def}} \ \forall \sigma \in Utraces(s) : \ out(i \text{ after } \sigma) \subseteq out(s \text{ after } \sigma) \]

   \( \iff \forall \sigma \in Rtraces(s) : \ out(i \text{ after } \sigma) \subseteq out(s \text{ after } \sigma) \)

2. **Rtraces** is “minimal”: For \( A \subset Rtraces(s) \) and \( A \neq Rtraces(s) \), there exists an input-enabled \( i \) such that

   \( \forall \sigma \in A : \ out(i \text{ after } \sigma) \subseteq out(s \text{ after } \sigma) \)

   and \( i \text{ uioco } s \)
From Required Traces to $\text{wioco}$

Refinement preorder $\leq$ is given by $\text{wioco}$, considering superfluous traces and non-input enabledness.

$s \ \text{wioco} \ s' =_{\text{def}} \ \forall \sigma \in Rtraces(s') :$

1. $\text{out}(s \ \text{after} \ \sigma) \subseteq \text{out}(s' \ \text{after} \ \sigma)$

2. $\forall \sigma_1 \leq \sigma : \ \text{in}(s \ \text{after} \ \sigma_1) \supseteq \text{Rin}(s' \ \text{after} \ \sigma_1)$

\[
\text{in}(s \ \text{after} \ \sigma_1) =_{\text{def}} \left\{ a? \in L_I \mid s \ \text{after} \ \sigma_1 \ \text{must} \ a? \right\}
\]

\[
\text{Rin}(s' \ \text{after} \ \sigma_1) =_{\text{def}} \left\{ a? \in \text{in}(s \ \text{after} \ \sigma_1) \mid \exists \sigma_2 \in Rtraces(s') : \sigma_1 \ a? \leq \sigma_2 \right\}
\]
A Weaker Specification through \textit{wioco}

\[ s \xrightarrow{\text{wioco}} s' \iff SUT(s) \subseteq SUT(s') \]

\( s' \) is a weaker than \( s \):
- remove inputs
- add outputs
Required Traces Automaton

\( \sigma \in R\text{traces}(s) \) 

\( \iff \sigma \) accepted by \( R\text{TA}(s) \)

**Algorithm 1** Find the required traces automaton of \( s \in L\text{TS}(L_I, L_U) \)

1. add loops \( s \overset{\delta}{\to} s \) for all quiescent states
2. build \( DC(s) \), the demonic completion of \( s \)
3. determinize \( DC(s) \) obtaining a new input-output transition system \( DC^d(s) = (Q, L_I, L_U, T, q_0) \)
4. for each \( q \in Q \) do
5. \( \text{if } (\text{out}(q) \neq L_U \cup \delta) \land (\exists p \mid p \overset{\lambda}{\to} q \land \lambda \neq \delta \land \lambda \neq \tau) \text{ then mark } q \text{ as accepting} \)
6. for each \( q \in Q \) do
7. \( \text{if } q \text{ is trace equivalent to chaos state } \chi \text{ then mark } q \text{ as } \text{chaotic} \)
8. remove all \text{chaotic} states from \( Q \)
9. for each \( (q, \delta, q_1) \in T \) do
10. add a new state \( p \) to \( Q \)
11. add \( (q, \delta, p) \) to \( T \)
12. for each \( a \in L_I \) such that \( q_1 \overset{a}{\to} q_2 \) do
13. add \( (p, a, q_2) \) to \( T \)
14. remove \( (q, \delta, q_1) \) from \( T \)
## MBT: Some Tools - ioco

- AETG
- Agatha
- Agedis
- All4Tec MaTeLo
- Autolink
- Axini Test Manager
- Conformiq Qtronic
- Cooper
- G\textsuperscript{\textregistered}st
- Gotcha
- JTorX
- NModel
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- RT-Tester
- SaMst\textsuperscript{a}G
- SeppMed MBTsuite
- Smartesting Certif\textsuperscript{y}lt
- Spec Explorer
- Statemate

- STG
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TGV
- TorX
- TorXakis
- T-Vec
- Uppaal Cover
- Uppaal-Tron
- Tveda
- .......
MBT: Some Tools - commercial

- AETG
- Agatha
- Agedis
- All4Tec MaTeLo
- Autolink
- Axini Test Manager
- Conformiq Qtronic
- Cooper
- G∀st
- Gotcha
- JTorX
- NModel
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- RT-tester
- SaMsTaG
- SeppMed MBTsuite
- Smartesting Certify!
- Spec Explorer
- Statemate
- STG
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TGV
- TorX
- TorXakis
- T-Vec
- Uppaal-Cover
- Uppaal-Tron
- Tveda

...
Learning

Test-Based Modelling
Models

- model-based testing
- simulation
- model-driven architecture
- model-driven design
- model-based control
- model-based monitoring
- model checking
- model-based analysis
- code generation

(Klaas Smit)
Models

- Everybody wants models
- Doing nice things with models
  - model checking,
    simulation, ..... 
- How to get these models?
  - in particular for:
    legacy, third-party, out-sourced, off-the-shelf, ..... components
- Does the model correspond with the real system?
Testing: Model-Based Testing

model-based test generation

model

system

test execution

pass fail
Test-Based Modeling

Model Learner

Model

test execution

system

pass fail
Test-Based Modeling

Automatically learning a model of the behavior of a system from observations made with testing

- test-based modeling
- automata learning
- black-box reverse engineering
- observation-based modeling
- behavior capture and test
- grammatical inference
Active learning is an active research area:

- LearnLib: Tool for FSM learning
- ...

Learning Finite Automata with $L^*$:

![Diagram showing active learning process with Teacher and Learner communicating through Membership Queries, Equivalence Queries, Yes/No, Yes/No + Counterexample.]
Learning Models of Reactive Systems

- Tool for active learning of Finite State Machines: **LearnLib**
- Developed by group B. Steffen (U. Dortmund)
- Able to learn models with up to 10,000 states

```
Teacher

MBT tool

SUT

Learner

Validity queries

Output queries

• Learner: formulate a hypothesis FSM
• Equivalence query replaced by model-based testing of hypothesized model
```
Application: Banking Cards: Learning the EMV protocol
Fides Aarts, Erik Poll, and Joeri de Ruiter

- EMV = Europay, Mastercard and Visa
- Models from black-box implementations
- Learn behaviour blindly
- Security: absence of unwanted functionality
- Correctness/conformance: presence of required functionality
Model of Maestro app on Dutch banking card
Model of Maestro app on German banking card

- Dutch vs. German banking card: different handling of errors
Learned Model of OCE Printer Module
Model-Based Testing & Test-Based Modeling

- **model**
  - learn model
  - refine model
  - MBT
    - conforming
      - yes
      - satisfied
      - yes
      - model world
      - no
      - no
    - no
    - system
      - repair system
      - more tests
      - yes
      - physical world
      - no
      - no

Test Coverage = Learning Precision

$\chi \chi \chi$

$\chi : \text{not precise, cheap}$

$S : \text{precise, expensive}$

$SUTs$
Model-Based Testing

There is Nothing More Practical

than a Good Theory