Model-Based Testing

*There is Nothing More Practical than a Good Theory*

Jan Tretmans

*TNO – ESI, Eindhoven, NL
Radboud University, Nijmegen, NL*
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 - State of the Art**
  – Motivation
  – What
  – Theory: *Labelled Transition Systems and ioco*
  – Tools
  – Discussion

• **Model-Based Testing - Next Generation**
  – Motivation
  – Requirements for Model-Based Testing:
    • *State + Data*
    • *Composition and Representation*
  – Example
  – Application
  – Discussion
Systems, Quality, Testing
Embedded Systems

or: What do Dutch Dykes have to do with Model-Based Testing?
Embedded Systems

or: What do Dutch Dykes have to do with Model-Based Testing?
Trend: Software in Embedded Systems

- **1970**
  - SW: 0%
  - electronics: 0%
  - mechanics: 100%
  - physics/chemistry, etc.: 100%

- **2000**
  - SW: 100%
  - electronics: 90%
  - mechanics: 10%
  - physics/chemistry, etc.: 10%

Progress 2006
Quality of Embedded Systems

Software is brain of system

- software controls, connects, monitors
  almost any aspect of ES system behaviour
- majority of innovation is in software

Software determines quality and reliability of Embedded System

- often > 50 % of system defects are software bugs
Quality

Exclusive: Pentagon report faults F-35 on software, reliability

BY ANDREA SHALAL-ESA
WASHINGTON | Thu Jan 23, 2014 3:36pm EST

Software glitch hindered Vettel’s qualifying

2014 Australian Grand Prix

March 15, 2014 at 10:00 am by Keith Collantine

Sebastian Vettel’s qualifying performance was impaired by a software problem on his car, Red Bull have confirmed.

Vettel was knocked out in the second phase of qualifying having been 2.4 seconds off new team mate Daniel Ricciardo’s pace.

A faulty sensor was later found to have contributed to Vettel’s problem. Team principal Christian Horner said: “It was unlucky for Seb.”

“His engine software meant he was down on power with extremely poor drivability and we need to understand that, as it compromised his qualifying.”

Despite his difficulties Vettel said the team have “made a big
Model-Based Testing
Software Testing

Checking or measuring some quality characteristics of an executing software object by performing experiments in a controlled way w.r.t. a specification.
Model-Based Testing

**MBT**

next step in test automation

- test generation
- result analysis
1 : Manual Testing

1. Manual testing

System Under Test

pass fail
2: Scripted Testing

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

1. Manual testing
2. Scripted testing
3. Keyword-driven testing
4 : Model-Based Testing

1. Manual testing
2. Scripted testing
3. Keyword-driven testing
4. Model-based testing

System model

Model-based test generation

Test execution

SUT

pass fail
MBT: Validation, Verification, Testing

- Ideas
- Validation
- Concrete realizations
- Testing
- Abstract models, math
- Verification
- SUT

18
A Theory of Model-Based Testing with Labelled Transition Systems
MBT : Model-Based Testing

SUT conforms to model

SUT passes tests

SUT

test execution

pass fail

model-based test generation

system model

MBT : Model-Based Testing
Models: Labelled Transition Systems

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

- **states**
- **input actions**
- **output actions**
- **transitions**
- **initial state**

\( ? = \text{input} \)
\( ! = \text{output} \)

- ?coin
- !coffee
- ?button
- !alarm

Diagram of a coffee vending machine with transition states.
MBT : Labelled Transitions Systems

SUT ioco model

sound \downarrow exhaustiv \uparrow

SUT passes tests

ioco test generation

set of LTS tests

LTS model

input/output conformance

SUT behaving as input-enabled LTS

ioco

LTS test execution

pass fail

SUT passes tests

ioco model

sound

exhaustive
Conformance: \(ioco\)

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

\[p \xrightarrow{\delta} p \quad = \quad \forall \ \!x \in L_U \cup \{\tau\} \cdot p \xrightarrow{\!x}\]

\[Straces(s) \quad = \quad \{ \ \sigma \in (L \cup \{\delta\})^* \mid s \xrightarrow{\sigma}\}\]

\[p \ \text{after} \ \sigma \quad = \quad \{ \ p' \mid p \xrightarrow{\sigma} p' \} \]

\[out(P) \quad = \quad \{ \ \!x \in L_U \mid p \xrightarrow{\!x}, p \in P \} \cup \{ \ \delta \mid p \xrightarrow{\delta} p, \ p \in P \} \]

\(s\) is a Labelled Transition System

\(i\) is (assumed to be) an input-enabled LTS
Conformance: \textit{ioco}

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

Intuition:

\textit{i \ ioco}-conforms to \textit{s}, iff

- if \textit{i} produces output \textit{x} after trace \textit{\sigma}, then \textit{s} can produce \textit{x} after \textit{\sigma}

- if \textit{i} cannot produce any output after trace \textit{\sigma}, then \textit{s} cannot produce any output after \textit{\sigma} \ (\textit{quiescence} \ \delta)
Example: \textit{ioco}

\begin{itemize}
  \item \textbf{specification model}
  \item \textbf{non-determinism uncertainty under-specification}
\end{itemize}
Example: \( \text{ioco} \)

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

\[
\text{out}(i \text{ after } ?\text{dub}.?\text{dub}) = \text{out}(s \text{ after } ?\text{dub}.?\text{dub}) = \{ !\text{tea}, !\text{coffee} \}
\]

\[
\text{out}(i \text{ after } ?\text{dub}.\delta.?\text{dub}) = \{ !\text{coffee} \} \neq \text{out}(s \text{ after } ?\text{dub}.\delta.?\text{dub}) = \{ !\text{tea}, !\text{coffee} \}
\]
MBT: Labelled Transitions Systems

SUT ioco model

sound \downarrow \uparrow \hspace{1cm} exhaustive

SUT passes tests

LTS test execution

ioco test generation

set of LTS tests

LTS model

input/output conformance

SUT behaving as input-enabled LTS

pass fail
Test Case

test case = labelled transition system

- ‘quiescence’ / ‘time-out’ label $\theta$
- tree-structured
- finite, deterministic
- final states pass and fail
- from each state $\neq$ pass, fail:
  - either one input !a
  - or all outputs ?$x$ and $\theta$
Test Generation Algorithm: \textit{ioco}

Algorithm to generate a test case \( t(S) \) from a transition system state set \( S \), with \( S \neq \emptyset \) (initially \( S = s_0 \) after \( \varepsilon \)).

Apply the following steps recursively, non-deterministically:

1. end test case
   - pass

2. supply input !a

3. observe all outputs

\begin{itemize}
  \item \text{allowed outputs (or } \delta \text{): } !x \in \text{out}(S)
  \item \text{forbidden outputs (or } \delta \text{): } !y \notin \text{out}(S)
\end{itemize}
Example: *ioco* Test Generation

**specification model**

![Diagram of specification model with nodes and edges]

**generated test case**

![Diagram of generated test case with nodes and edges]

**implementation**

![Diagram of implementation with nodes and edges]

\[
i \text{ioco } s \overset{=}{=} \text{def} \forall \sigma \in \text{Straces} (s) : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma)
\]
MBT with \(ioco\) is Sound and Exhaustive

Test assumption:
\[
\forall SUT \in \text{IMP} . \exists m_{SUT} \in \text{IOTS} . \\
\forall t \in \text{TESTS} . \\
sut \text{ passes } t \iff m_{SUT} \text{ passes } t
\]

Prove soundness and exhaustiveness:
\[
\forall m \in \text{IOTS} . \\
( \forall t \in \text{gen}(s) . m \text{ passes } t ) \\
\iff m \text{ ioco } s
\]

SUT conforms to \(s\)

\begin{align*}
\text{exhaustive} & \uparrow \downarrow \\
\text{sound} & \\
\text{SUT passes gen(s)} & \\
\end{align*}
Model-Based Testing
with Labelled Transition Systems
Background Theory
Testing Equivalences

\[
S_1 \approx S_2 \iff \forall e \in E. \ obs(e, S_1) = obs(e, S_2)
\]

?  ?
MBT: Test Assumption

Test assumption:

∀ SUT. ∃ m_{SUT} ∈ MODELS.

∀ t ∈ TEST. SUT passes t ⇔ m_{SUT} passes t
MBT : Completeness

\[ \text{SUT passes } T_S \iff \text{SUT conforms to } s \]

\[
\begin{align*}
\text{SUT passes } T_S & \iff \text{SUT passes } T_S \\
& \iff \forall t \in T_S . \text{SUT passes } t \\
& \iff \text{test hypothesis: } \forall t \in \text{TEST} . \text{SUT passes } t \iff \text{m}_{\text{SUT}} \text{ passes } t \\
& \iff \forall t \in T_S . \text{m}_{\text{SUT}} \text{ passes } t \\
& \iff \text{prove: } \forall m \in \text{MOD}. ( \forall t \in T_S . \text{m passes } t ) \iff \text{m imp s} \\
& \iff \text{m}_{\text{SUT}} \text{ imp } s \\
& \iff \text{define: } \text{SUT conforms to } s \iff \text{m}_{\text{SUT}} \text{ imp } s \\
& \iff \text{SUT conforms to } s
\end{align*}
\]
Genealogy of ioco

Labelled Transition Systems

IOTS (IOA, IA, IOLTS)

Trace Preorder

Testing Equivalences (Preorders)

Quiescent Trace Preorder

Canonical Tester

Conf

Refusal Equivalence (Preorder)

Repetitive Quiescent Trace Preorder (Suspension Preorder)

ioco
Model-Based Testing
with Labelled Transition Systems
Variations
model with data and time and hybrid and action refinement

\[
\begin{align*}
\text{[ } n \geq 35 \text{ ] } & \rightarrow \text{ ? button1 } \\
\text{[ } n \geq 50 \text{ ] } & \rightarrow \text{ ? button2 }
\end{align*}
\]

\[
\begin{align*}
V_t & := 0 \\
V_c & := 0
\end{align*}
\]

\[
\begin{align*}
\text{d}V_t / \text{d}t & = 3 \\
\text{d}V_c / \text{d}t & = 2
\end{align*}
\]

\[
\begin{align*}
\text{[ } V_t = 15 \text{ ] } & \rightarrow \text{ ! tea } \\
\text{[ } V_c \geq 5 \text{ ] } & \rightarrow \text{ ! coffee }
\end{align*}
\]
Variations on a Theme

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

Tools
MBT: Ingredients

off-line MBT

model-based test generation

system model

requirements

ideas

SUT

test harness

test cases

model-based test generation

test execution

verdict test result analysis

pass fail

MBT: Ingredients

off-line MBT

model-based test generation

system model

requirements

ideas

SUT

test harness

test cases

model-based test generation

test execution

verdict test result analysis

pass fail

MBT: Ingredients

off-line MBT

model-based test generation

system model

requirements

ideas

SUT

test harness

test cases

model-based test generation

test execution

verdict test result analysis

pass fail
on-the-fly MBT

model based
test generation
+ execution

SUT

test harness

verdict
test result
analysis

pass fail

requirements

ideas

system model
MBT : Many Tools

- AETG
- Agatha
- Agedis
- Autolink
- Axini Test Manager
- Conformiq
- Cooper
- Cover
- DTM
- G∀st
- Gotcha
- Graphwalker
- JTorX
- MaTeLo
- MBT suite
- M-Frame
- MISTA
- NModel
- OSMO
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- Recover
- RT-Tester
- SaMsTaG
- Smartesting CertifyIt
- Spec Explorer
- Statemate
- STG
- Temppo
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TestOptimal
- TGV
- Tigris
- TorX
- TorXakis
- T-Vec
- Uppaal-Cover
- Uppaal-Tron
- Tveda
- .............
# MBT: Many Tools

- AETG
- Agatha
- Agedis
- Autolink
- Axini Test Manager
- Conformiq
- Cooper
- Cover
- DTM
- G∀st
- Gotcha
- Graphwalker
- JTorX
- MaTeLo
- MBT suite
- M-Frame
- MISTA
- NModel
- OSMO
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- Recover
- RT-Tester
- SaMsTaG
- Smartesting CertifyIt
- Spec Explorer
- Statemate
- STG
- Temppo
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TestOptimal
- TGV
- Tigris
- TorX
- TorXakis
- T-Vec
- Uppaal-Cover
- Uppaal-Tron
- Tveda
- ............
Model-Based Testing
Discussion
MBT: Benefits!

- **Automatic test generation**
  - *automation of test generation* + *execution* + *result analysis*
  - *test fast, test often*

- **More, longer, and more diversified test cases**
  - *more variation in test flow and in test data*

- **Model is precise and consistent test basis**
  - *unambiguous analysis of test results*
  - *early error detection during model construction and analysis*
  - *link to model-based system development*

- **Test maintenance by maintaining models**
  - *improved regression testing*

- **Expressing test coverage by model coverage**
MBT : Benefits ???

Promising, emerging technology .... but

*If doing MBT is so smart, why ain’t you rich?*

- Many companies experiment, but who’s really doing it?
- Application of MBT still disappointing
- Organizational issues : Education, adoption, shift in the development process

*Technical Issues : Theory, Algorithms, Tools*
Testing: Trends & Challenges

- Model-driven development, continuous *
- Complexity
- Size
- Quest for quality
- Connectivity, systems-of-systems
- Uncertainty
- Heterogeneous components
MBT: Next Generation Challenges

- abstraction
- concurrency parallelism
- usage profiles for testing
- scalability
- link to MBSD
- model composition
- test selection criteria
- multiple paradigms integration
- uncertainty nondeterminism
- state + complex data
- heterogeneous components
- uncertainty
- complexity
- size
- quality
- model based testing
- next generation challenges
- size
- continuous
MBT: Next Generation Challenges

State of the Art MBT Tools

- abstraction
- concurrency parallelism
- state + complex data
- usage profiles for testing
- model composition
- scalability
- test selection criteria
- link to MBSD
- multiple paradigms integration
- uncertainty nondeterminism
Model-Based Testing

Next Generation

TorXakis
Next Generation MBT: TorXakis

- abstraction
- concurrency parallelism
- state + complex data
- model composition
- test selection criteria
- multiple paradigms integration
- uncertainty nondeterminism
- usage profiles for testing
- scalability
- link to MBSD
Yet Another MBT Tool: TorXakis
Yet Another MBT Tool: TorXakis

- Testing Equivalences
- LTS: Labelled Transition Systems
  - ioco
- ADT: Algebraic Data Types
- STS: Symbolic Transition Systems
- Equational Reasoning
- Process Algebra
- Soundness Completeness
- Uncertainty Non-determinism Abstraction Under-specification
- Uncertainty Nondeterminism
- Concurrency Parallelism
- Abstraction
- Model Composition
- State + Complex Data
Model-Based Testing

State + Data
**TorXakis: A Black-Box View on Systems**

- **Modelled as state-transition system**
- **SUT**
- **Black-box system view**

Diagram:
- Input: `a`
- Output: `x`
- Transition labels:
  - `a?n` to `x!n+1`
  - `b?m` to `y!`no`'
  - `a?k` to `x!42`
  - `y!`yes`' to `a?n`

- `SUT` is modelled as a state-transition system.

- Input: `y`
- Output: `b`
- Transition labels:
  - `y` to `b`
  - `b` to `x`
TorXakis: A Black-Box View on Systems

MODEL
labelled transition system
with parameterized actions on
input- and output channels

Not (yet) in TorXakis:
• real-time
• probabilities
• derivatives (hybrid)
STS: Symbolic Transition Systems

STS: model with data

money ? n :: int

[[ n ≥ 35 ]] -> button1

[[ n ≥ 50 ]] -> button2

tea

coffee

return ! n – 35

return ! n – 50

STS: Symbolic Transition Systems
MBT: Nondeterminism, Underspecification

- non-determinism
- under-specification
- specification of properties rather than construction
STS : Symbolic Transition Systems

\[
\text{in } ? \ n :: \text{int} \\
[[ n \neq 0 ]] \\
\text{out } ! \ m :: \text{int} \\
[[ 0 < m < -n ]] \\
\text{out } ! \ m :: \text{int} \\
[[ 0 < m < n ]] \\
\text{semantics}
\]
TorXakis: Data Types

- Standard types: Int, Bool, String
- Algebraic data types

```plaintext
TYPEDEF Colour ::= Red | Yellow | Blue

TYPEDEF IntList ::= Nil
| Cons { hd :: Int, tl :: IntList }
```
TorXakis: Functions

- Functions: name, parameters, type, (recursive) value expression
- Overloading
- Standard functions

```plaintext
TYPEDEF Colour ::= Red | Yellow | Blue

FUNCTION add (x, y :: Int) :: Int ::= x + y

FUNCTION ++ (s :: IntList; x :: Int) :: IntList ::=  
  IF isNil(s) 
  THEN Cons(x, Nil) 
  ELSE Cons(hd(s), tl(s) ++ x) 
  FI
```
**TorXakis**: Process with Data

- **Value**: `! hd(buf)`
  
  ```
  buf := tl(buf)
  ```

- **Deq**: `[[ not (isNil(buf)) ]]`

- **Enq**: `? x`
  
  ```
  buf := buf ++ x
  ```

- **Deq**: `[[ isNil(buf) ]]`

Diagram: A process with data input and output operations.
More Complex Data

Test data generation from XSD (XML) descriptions with constraints

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name='Root'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='Customers'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name='Customer' type='CustomerType' minOccurs='0' maxOccurs='unbounded' />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name='Orders'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name='Order' type='OrderType' minOccurs='0' maxOccurs='unbounded' />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name='complex'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name='CustomerIDKey'>
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name='CustomerID'>
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name='xPath' type='xs:string' />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name='CompanyID'>
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name='xPath' type='xs:string' />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name='customerIDKey'>
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name='xPath' type='xs:string' />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
TorXakis: Lift Test Generation

symbolic test generation

semantics

ioco test generation

==

semantics
TorXakis: Lift Test Generation

- STS (symbolic-concrete test generation)
- LTS (ioco test generation)
- TEST
Model-Based Testing

Composition and Representation
Compositionality – Representing LTS
Compositionality – Representing LTS

- Explicit:
  \[ \langle \{ S0, S1, S2, S3 \}, \{10c,coffee,tea\}, \{ (S0,10c,S1), (S1, coffee, S2), (S1,tea,S3) \} , S0 \rangle \]

- Transition tree / graph

- Language:
  \[ S ::= 10c \rightarrow\rightarrow ( coffee \#\# tea ) \]
Compositionality – Representing LTS

a \rightarrow (b \#\# c)

a \rightarrow b \#\# a \rightarrow c

a \rightarrow b \mid \mid c \rightarrow d
Representation of LTS

where

\[ P ::= a \rightarrow\!
ot\rightarrow P \]

\[ Q ::= a \rightarrow\!
ot\rightarrow (b \mid \mid \mid Q) \]
TorXakis: Defining Behaviour - LTS

**basic behaviour**  
= transition system

**complex behaviour**  
= combining transition systems

- name
- named behaviour definition
- behaviour use
- choice
- parallel
- communication
- exception
- interrupt
- hiding

*old but solid theory*
Model-Based Testing

Example:

Dispatcher-Processing System
Example: Dispatcher-Processing System
Example: Dispatcher-Processing System

![Diagram showing the state transition system of a processor. The diagram includes states: Idle, Start, Processing, and Finish. Arrows indicate the transition from Idle to Start to Processing to Finish.]
Example: Dispatcher-Processing System

processors ::= processor(1) ||| processor(2) ||| processor(3) ||| processor(4)
Example: Dispatcher-Processing System

```
dispatch_procs ::= HIDE [ Start ]
    IN processors |[ Start ]| dispatcher
    NI
```
Example: Dispatcher Processing System
**Example: Dispatcher-Processing System**

**FUNCDEF**

```
gcd ( a, b :: Int ) :: Int ::=
    IF a == b THEN a ELSE IF a > b THEN gcd ( a - b, b ) ELSE gcd ( a, b - a ) FI FI
```

**TYPEDEF**

```
JobData ::= JobData {
    jobld :: Int ;
    jobDescr :: String ;
    x, y :: Int
}
```

**FUNCDEF**

```
isValidJob ( jobdata :: JobData ) :: Bool ::=
    jobdata.jobld > 0 ∧ strinre ( jobdata.jobDescr, REGEX('[A-Z][0-9]{2}[a-z]+') ) ∧ . . . . .
```
Running TorXakis and SUT

TorXakis

```
SPECDEF Spec ::= CHAN IN Stim CHAN OUT Resp
ENDDEF
BEHAVIOUR Stimulus >> Response
ADAPDEF Adap ::= CHAN IN Stim CHAN OUT Resp
ENDDEF
MAP IN Stimu -> In!
MAP OUT Out ? s ->
ENDDEF
SUTDEF Sut ::= SUT IN In :: String
SUT OUT Out :: String
SOCK IN In HOST "localhost" PORT 7890
SOCK OUT Out HOST "localhost" PORT 7890
ENDDEF
```

adapter

model.txs

```
client server
```

```
c:> torxakis.exe model.txs
```

```
c:> sut.exe
```

host portnumber
Demo: Dispatcher-Processing System
Next Generation MBT : Status

Testing Equivalences

Soundness Completeness

Uncertainty Non-determinism Abstraction Under-specification

LTS : Labelled Transition Systems \( \text{ioco} \)

STS : Symbolic Transition Systems

ADT : Algebraic Data Types

Process Algebra

Equational Reasoning

SMT : Satisfiability Modulo Theories

uncertainty nondeterminism concurrency parallelism abstraction model composition state + complex data
Next Generation MBT : Status

- abstraction
- state + complex data
- concurrency parallelism
- scalability
- usage profiles
- test selection criteria
- multiple paradigm integration
- model composition
- link to MBSD
- uncertainty nondeterminism
- Next Generation MBT: Status
Discussion

Perspective
MBT: Benefits

- **Automatic test generation**
  - automation of test generation + execution + result analysis
  - test fast, test often

- More, longer, and more diversified test cases
  - more variation in test flow and in test data

- **Model is precise and consistent test basis**
  - unambiguous analysis of test results
  - early error detection during model construction and analysis
  - link to model-based system development

- **Test maintenance by maintaining models**
  - improved regression testing

- **Expressing test coverage by model coverage**
Model-Based Testing

There is Nothing More Practical than a Good Theory

- Arguing about validity of test cases and correctness of test generation algorithms
- Explicit insight in what has been tested, and what not
- Use of complementary validation techniques: model checking, theorem proving, static analysis, runtime verification, . . . . .
- Implementation relations for nondeterministic, concurrent, partially specified, loose specifications
- Comparison of MBT approaches and error detection capabilities
Model-Based Testing

There is Nothing More Practical than a Good Theory

Questions?