AUTHORING BEHAVIORAL MODELS AND GENERATING TESTS WITH .NET

Xiang Li

Winterop Team

Microsoft Corp.

Agenda

- Behavioral Model-based Testing
- Spec Explorer
  - Getting Started: Stop Watch
  - Modeling Asynchronous system: Chat
  - Where We Are
- Q&A

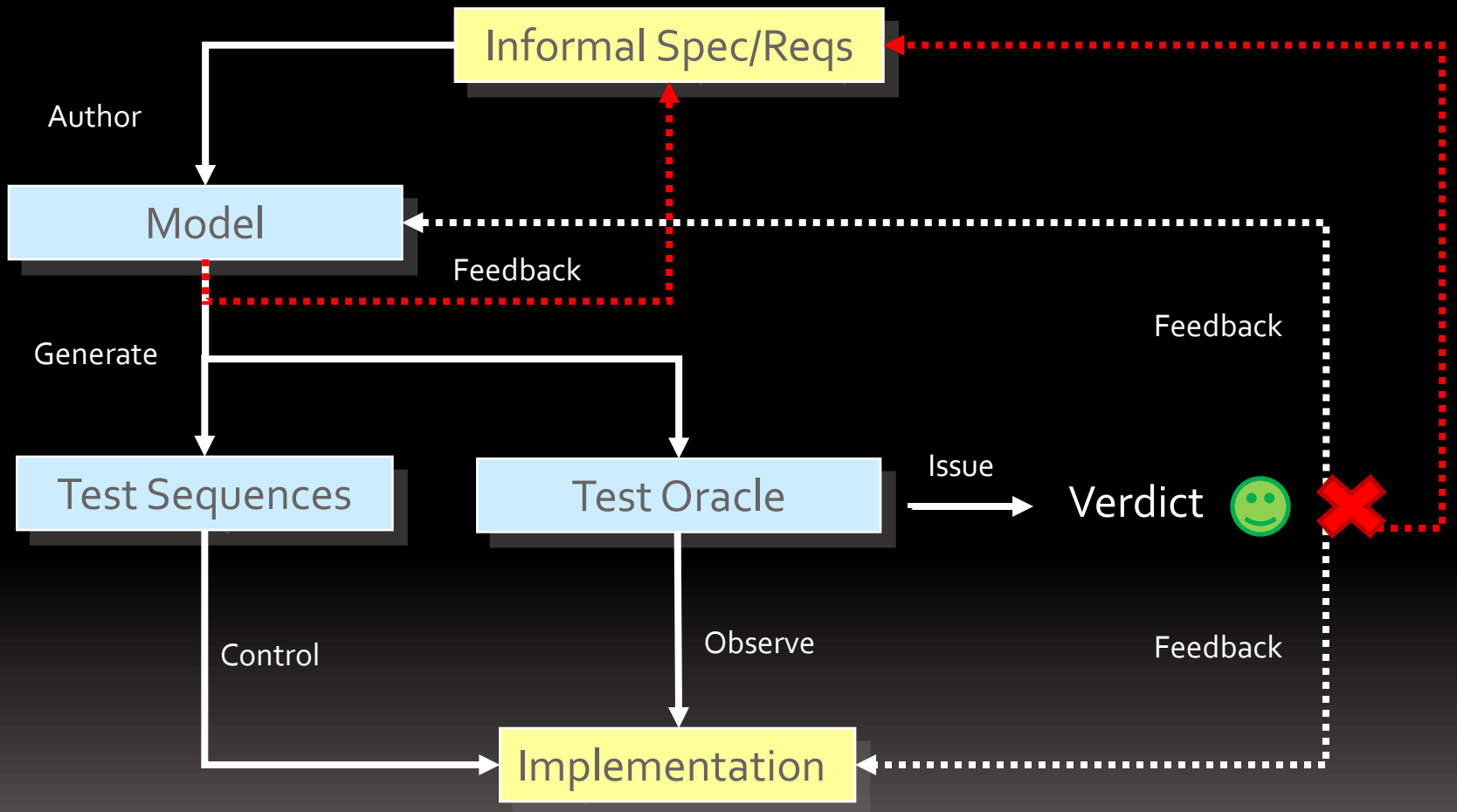
BEHAVIORAL MODEL-BASED TESTING

A Lightweight Formal Method

Behavioral Modeling

- Action
  - A visible action of the system
  - Can be stimulus or response
- Trace
  - A sequence of actions
- Behavior
  - A set of traces describing the allowed or observed behavior of a system

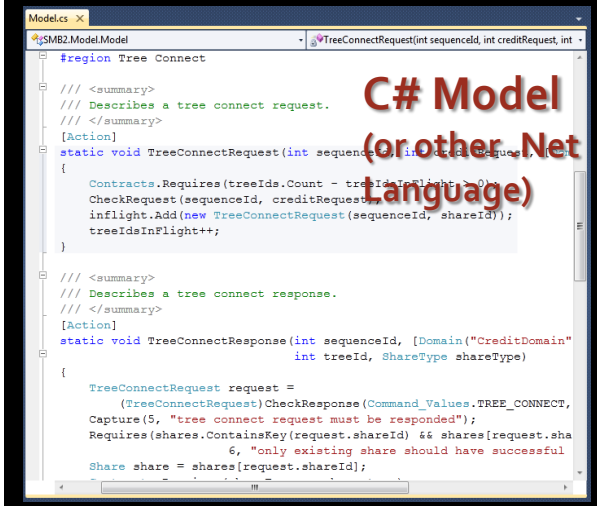
Conformance Testing

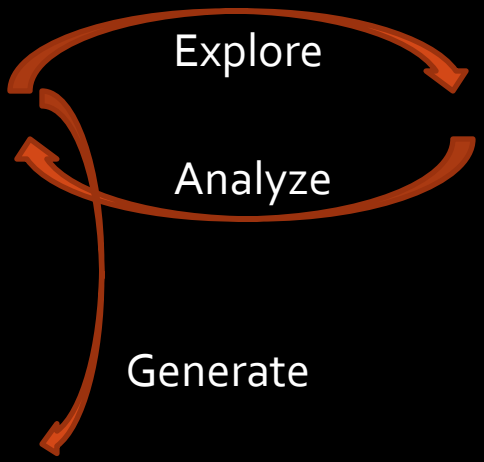


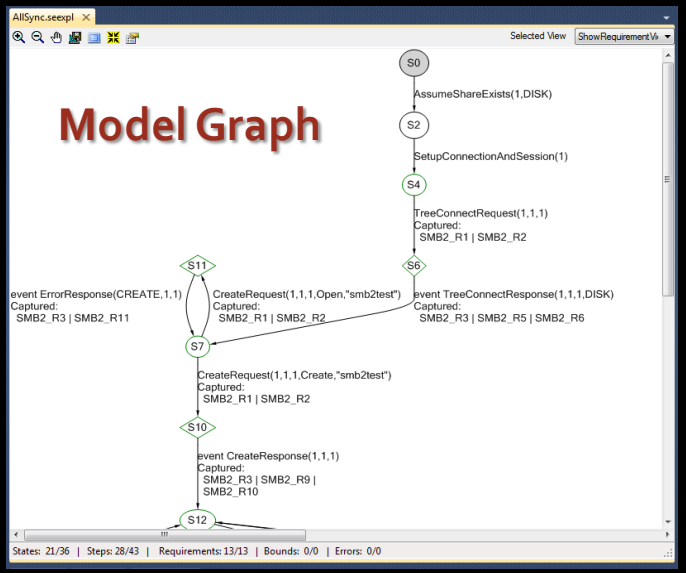
SPEC EXPLORER

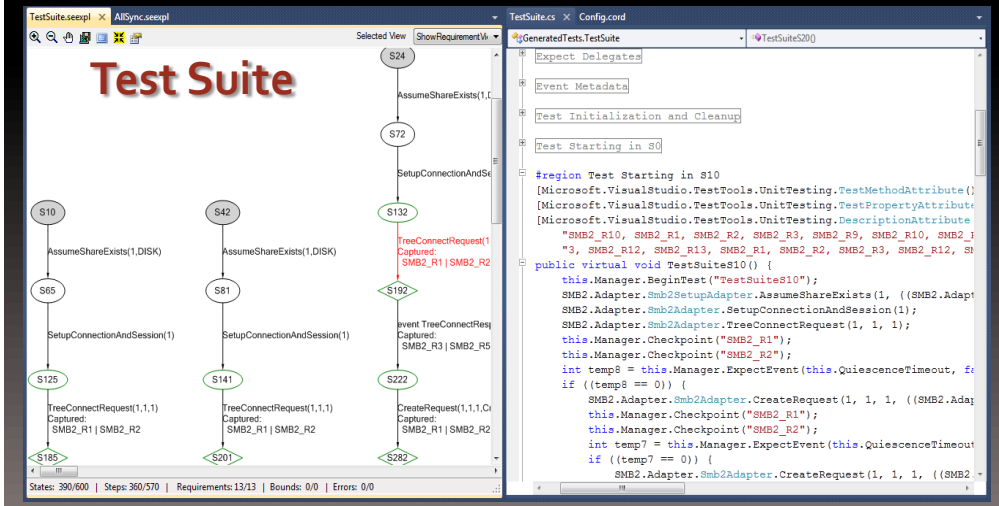
A Model-Based Testing Tool from Microsoft

Spec Explorer 2010 Look & Feel

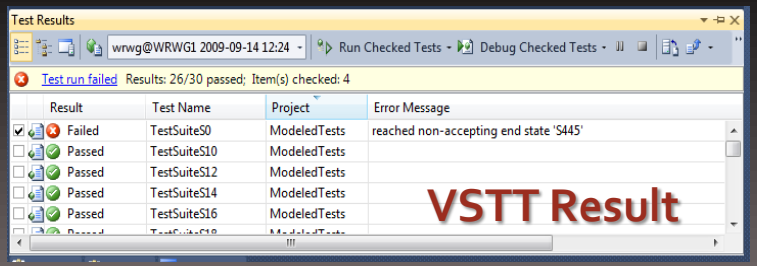








Execute



Spec Explorer 2010 Technology Breakdown

- Model programs
  - Guarded state update rules
  - Rich object-oriented model state (collections, object graphs)
  - Language agnostic (based on .Net intermediate language interpretation)
- Trace patterns
  - Regular style language to represent scenarios
  - Slicing of model program by composition
- Symbolic state exploration and test generation
  - Expands parameters using combinatorial interaction testing
  - Extracts a finite interface automaton (IA) from composed model
  - Traverses IA to generate standalone test code –or-
  - Runs on-the-fly tests from IA
- Integrated into Visual Studio 2010

GETTING STARTED WITH SPEC EXPLORER

STOPWATCH

- Two display modes
  - Date and time
  - Timer
- Three buttons
  - Mode
    - Always enabled
  - Start/stop timer
    - Only in timer mode
    - Starts/stops timer
  - Reset/Lap timer
    - Only in timer mode:
    - Timer running: lap (un)freeze
    - Timer stopped: reset to zero

The rest is abstracted out
(We only describe parts of the UI!)

Actions

- One action per button
- One action allows to check whether the timer is reset
  - I.e. value is 0:00
  - The system has only limited testability!

- Action declarations in Spec Explorer

```
action abstract static void Stopwatch.ModeButton();
action abstract static void Stopwatch.StartStopButton();
action abstract static void Stopwatch.ResetLapButton();

action abstract static bool Stopwatch.IsTimerReset();

action abstract static void Stopwatch.Initialize();
```

Traces

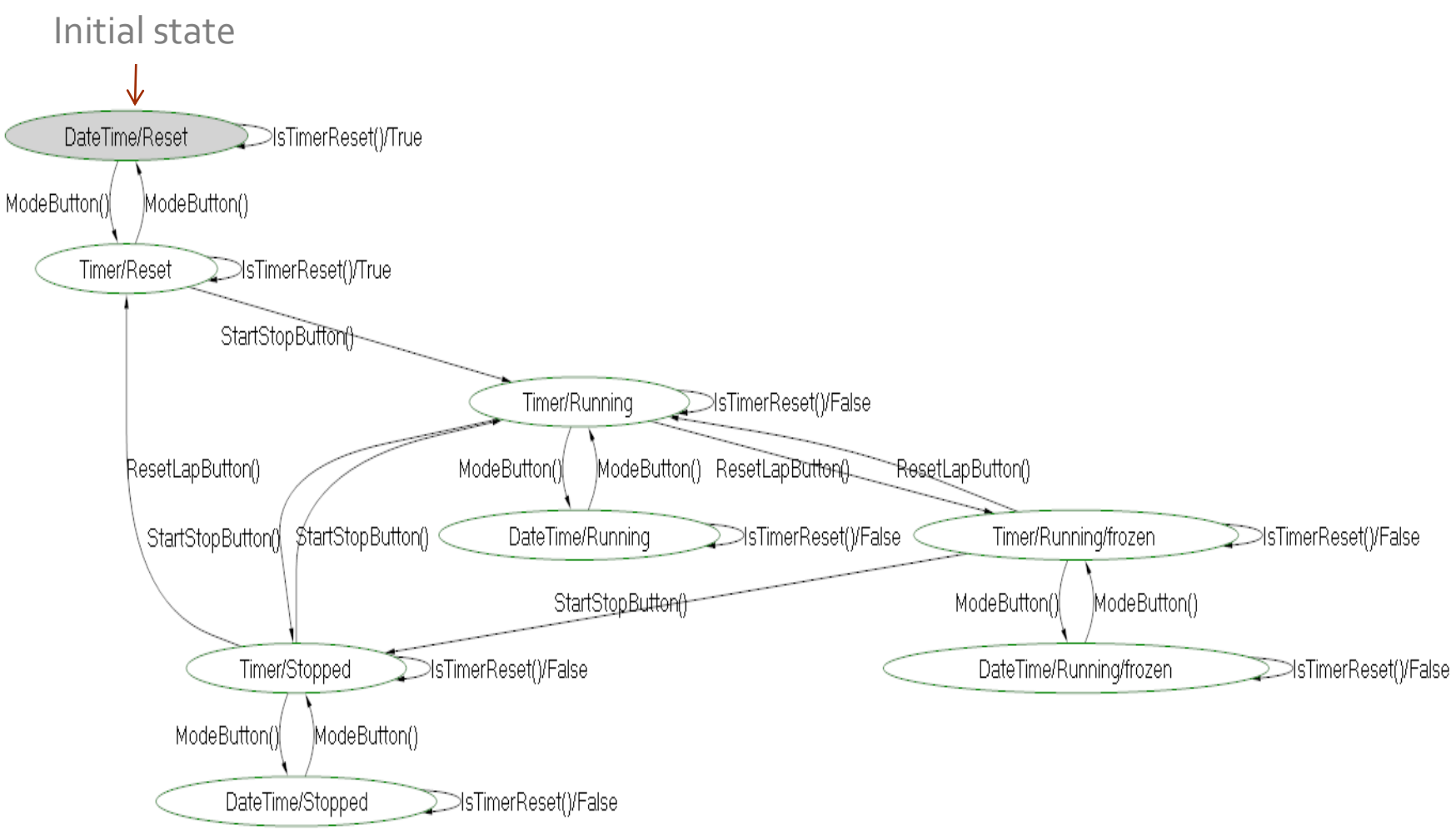
Which of the following traces are valid (is in the behavior)?

Assumption: initially stopwatch is displaying time of the day and the timer is reset

- T1: ModeButton; ModeButton; IsTimerReset/true
- T2: ModeButton; IsTimerReset/true; StartStopButton; IsTimerReset/true
- T3: ModeButton; StartStopButton; ModeButton; ModeButton; IsTimerReset/false
- T4: ModeButton; StartStopButton; ResetLapButton; IsTimerReset/true
- T4: <empty>

Concise representation of all traces

Result of model exploration!



DEVELOPING THE MODEL

Spec Explorer Configuration

```
config StopwatchButtons
{
	action static void Stopwatch.ModeButton();
	action static void Stopwatch.StartStopButton();
	action static void Stopwatch.ResetLapButton();
	action static bool Stopwatch.IsTimerReset();
	action static void Stopwatch.Initialize();
}
machine Model() : Config
{
	construct model program from StopwatchButtons
		where namespace = "StopwatchModel"

}
```

C# Model

```
static class Model
{
  public enum TimerMode
    { Reset, Running, Stopped }

  static bool displayTimer = false;
  static TimerMode timerMode =
    TimerMode.Reset;
  static bool timerFrozen = false;

  [Action]
  static void StartStopButton()
  {
    Contracts.Requires(displayTimer);
    if (timerMode == TimerMode.Running)
    {
      timerMode = TimerMode.Stopped;
      timerFrozen = false;
    }
    else
      timerMode = TimerMode.Running;
  }

  [Action]
  static void ModeButton()
  {
    displayTimer = !displayTimer;
  }

  [Action]
  static void ResetLapButton()
  {
    Contracts.Requires(displayTimer);
    Contracts.Requires
      (timerMode != TimerMode.Reset);
    if (timerMode == TimerMode.Running)
      timerFrozen = !timerFrozen;
    else
      timerMode = TimerMode.Reset;
  }

  [Action]
  static bool IsTimerReset()
  {
   return timerMode == TimerMode.Reset;
  }
}
```

TESTING FROM THE MODEL

- The goal of Model-Based Testing
  - To check whether an implementation conforms to the modeled behavior (set of traces)
- How big is the set of valid traces for Stopwatch?
  - Infinite!
- How many tests do we need for Stopwatch?
  - The "test selection" problem
  - Test selection is not complete (testing never is)
- Strategies for test selection
  - Select a coverage criterion for the model graph
    - "transition coverage" is a good criteria
  - Slice the model to extract interesting cases

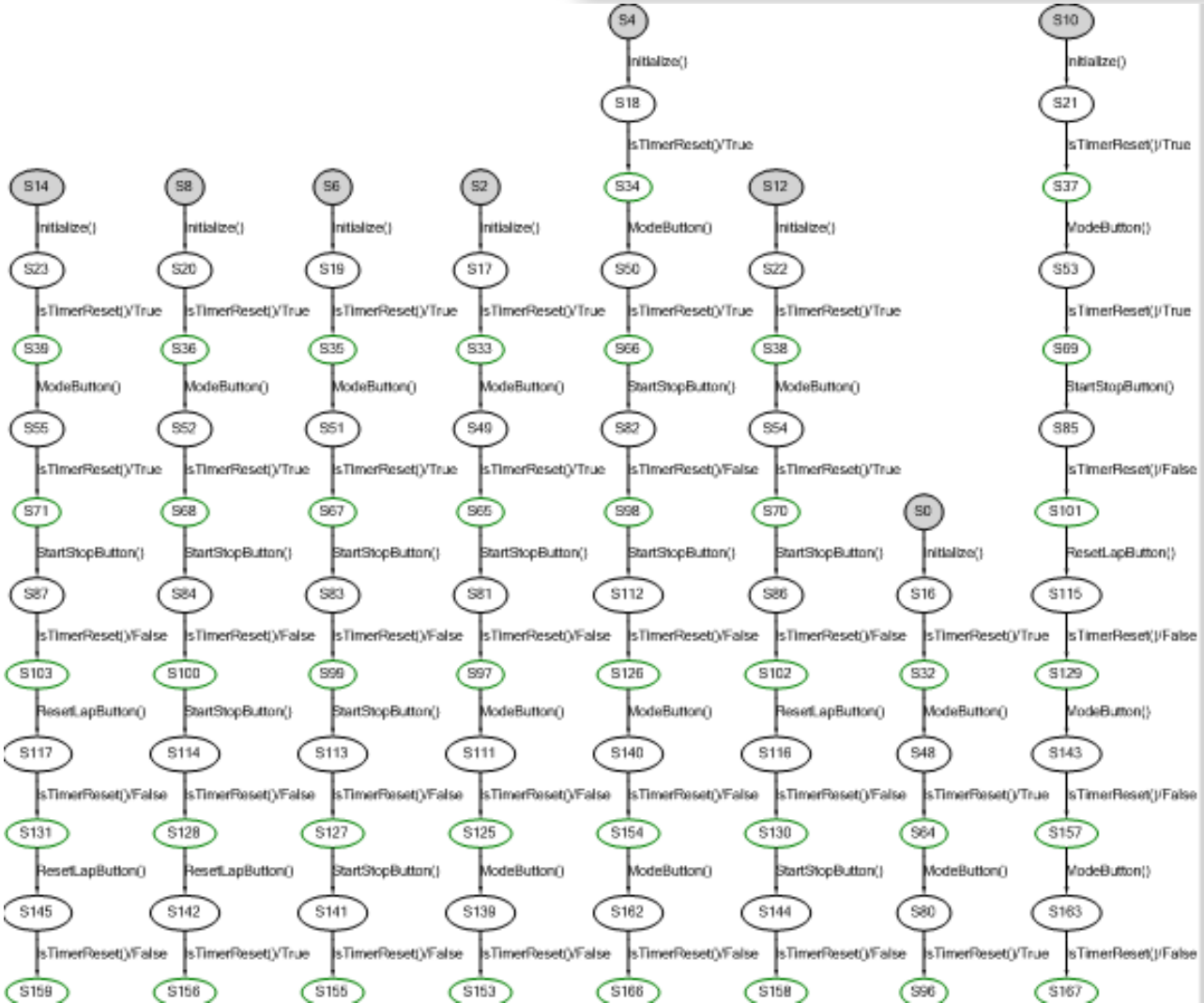
Approaches for Testing

- Test code generation with static traversal
  - Statically traverses the explored transition system and extracts test traces with configurable strategies
    - Transition coverage
    - State coverage
    - …
- Dynamic traversal
  - Dynamically traverses explored transition system at test execution time with configurable strategies
    - Traversal strategy may adapt to runtime behavior of SUT to get better coverage
    - Good for highly non-deterministic system
- On-the-fly testing
  - Testing is combined with model exploration (alternating simulation)
  - A pre-explored transition system is NOT needed, so on-the-fly testing can test against an infinite model without slicing
  - Good for highly non-deterministic systems

Generated test cases

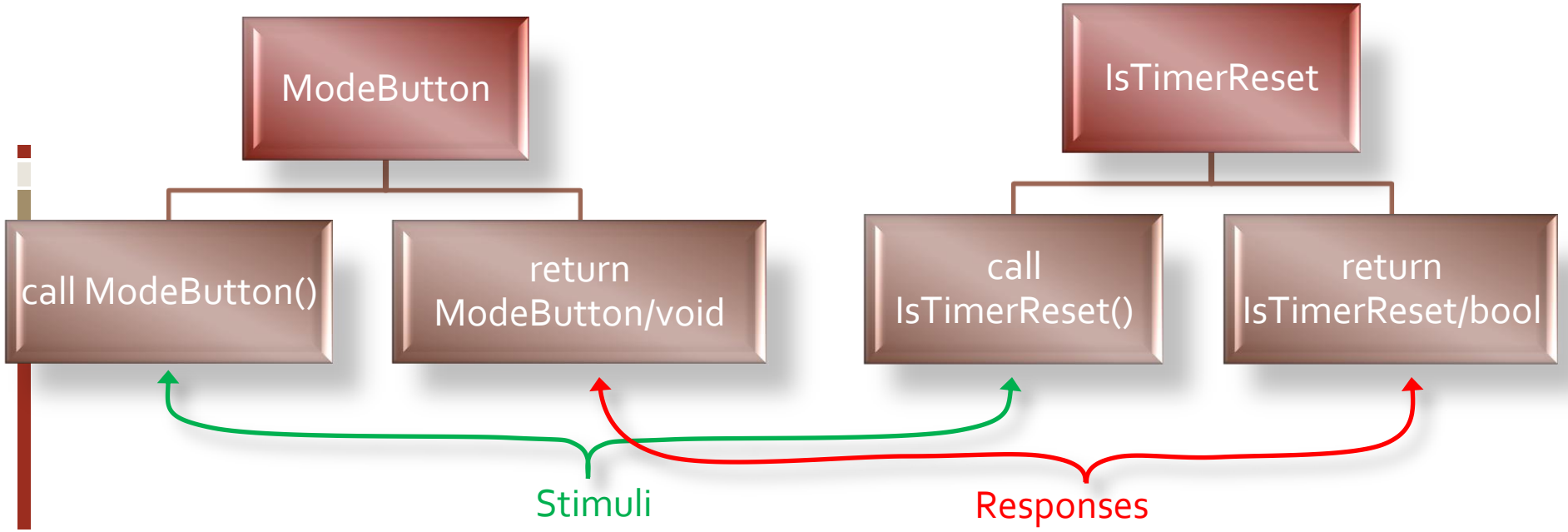
(static traversal with short tests strategy)

```
machine TestSuite() : Config
{
  construct test cases
    where strategy = "shorttests"
      for Initialize;Model
}
```

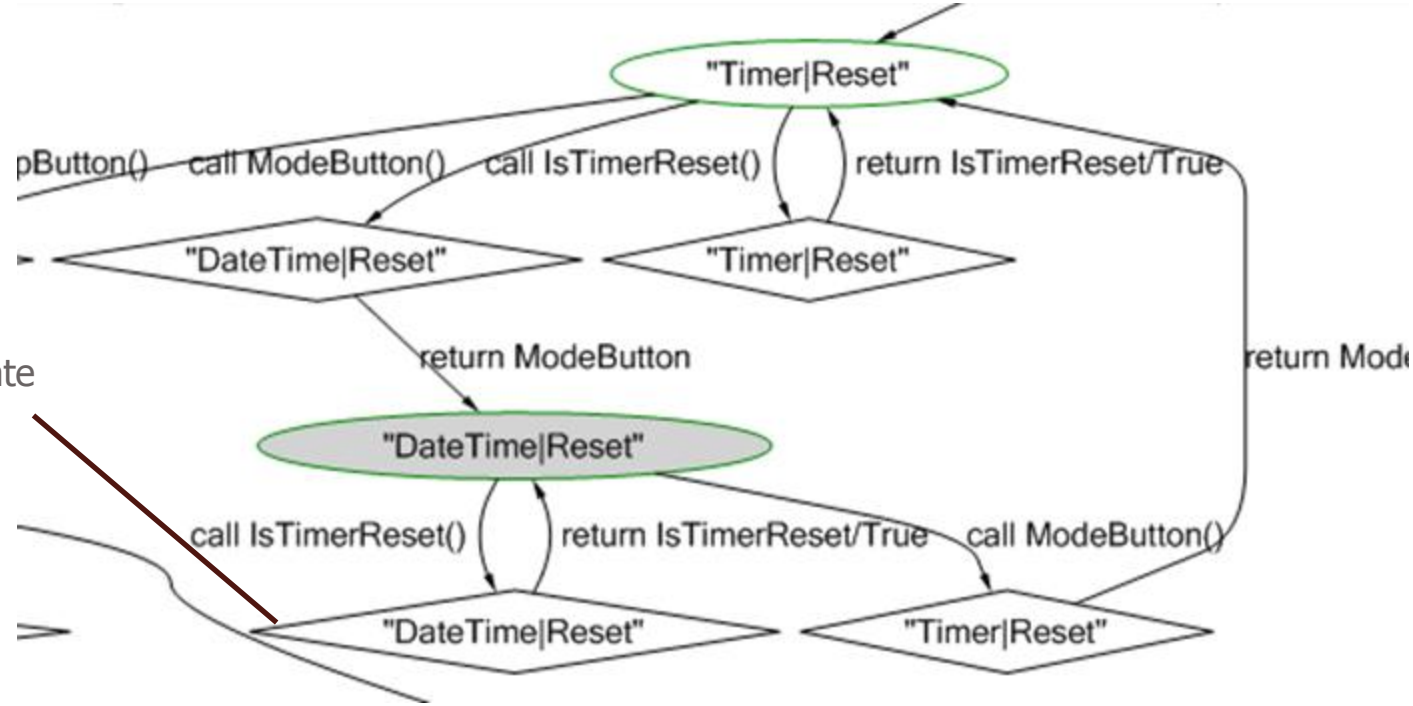


Stimuli and Responses

- Test cases can
  - Provide stimuli to the tested system
  - Observe the correctness of the systems by looking at system responses
- Actions like ModeButton and IsTimerReset actually stand for a pair of actions each



Expanded Model Graph



Observation state

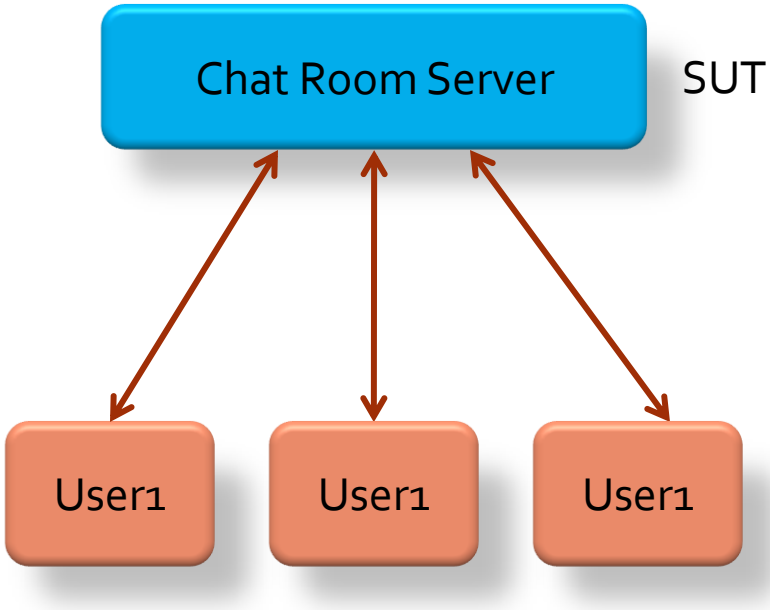
MODELING ASYNCHRONOUS SYSTEMS

CHAT SERVER

Sync - Async

- Synchronous system
  - Each response immediately follows the associated stimulus
- Asynchronous system
  - No such restriction
  - Responses may be out-of-order
  - Responses may be spontaneous
  - The presence and ordering of responses may be unknown at model-design time

- Users can
  - Enter the session
  - Exit the session
  - List all session users
  - Broadcast a message
    - Received by all session users
    - (with action BroadcastAck)



What happens if two users broadcast a message at virtually the same time?

Chat Requirements

- R1: User MUST receive response for logon request
- R2: User MUST receive response for logoff request
- R3: User MUST receive response for list request
- R4: List response MUST contain the list of logged-on users if successful
- R5: All logged-on users MUST receive broadcast message
- R6: Messages from one sender MUST be received in order

Chat traces

Is the trace correct?

(Assume two users (user1, user2) are logged on)

T1	**T2**	**T3**
Broadcast(user1,"1")	BroadcastAck(user1,"1")	BroadcastAck(user1,"1")
Broadcast(user2,"2")	BroadcastAck(user2,"2")	BroadcastAck(user2,"2")
BroadcastAck(user2,"1")	BroadcastAck(user1,"2")	BroadcastAck(user1,"2")
BroadcastAck(user1,"2")	BroadcastAck(user1,"1")	BroadcastAck(user2,"2")
BroadcastAck(user1,"1")	BroadcastAck(user2,"1")	BroadcastAck(user1,"1")
BroadcastAck(user2,"2")	BroadcastAck(user2,"2")	BroadcastAck(user2,"1")

If each user sends one message, any receiving order is correct!

Chat traces

Is the trace correct?

(Assume two users (user1, user2) are logged on)

T_4

Broadcast(user1,"1a")
Broadcast(user1,"1b")
BroadcastAck(user2,"1a")
BroadcastAck(user1,"1a")
BroadcastAck(user1,"1b")
BroadcastAck(user2,"1b")

T_5

Broadcast(user1,"1a")
Broadcast(user1,"1b")
BroadcastAck(user1,"1a")
BroadcastAck(user2,"1b")
BroadcastAck(user1,"1b")
BroadcastAck(user2,"1a")

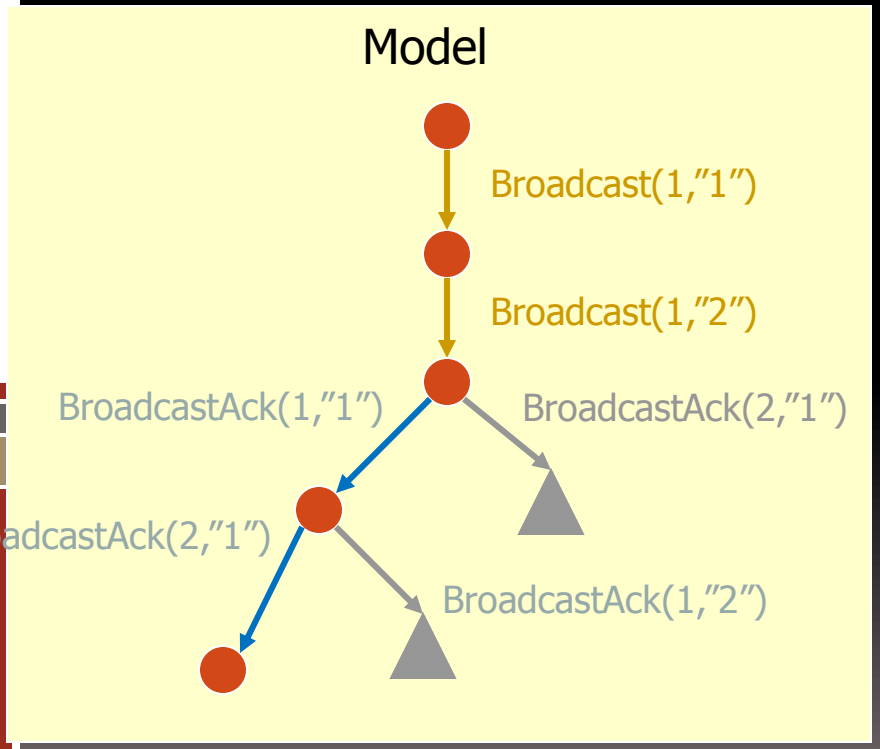
Local order consistency:

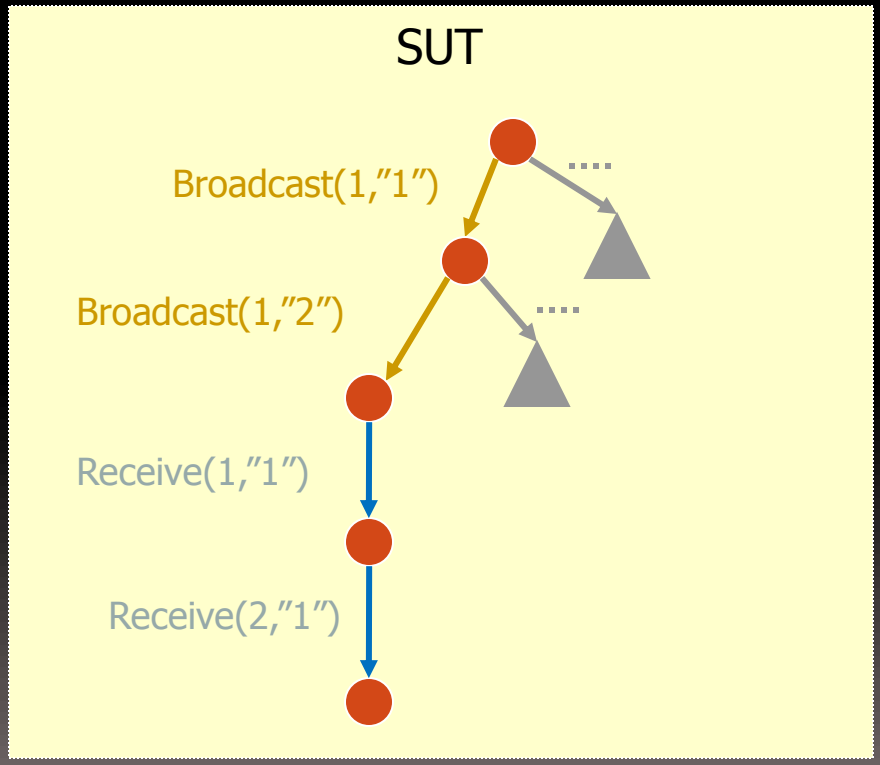
messages sent by one user must be received in order

Conformance and Events

Alternating simulation:

- SUT must "simulate" all stimuli of model
- Model must "simulate" all responses of SUT





Slicing and Conformance

- Can we slice events out?

```
machine MySlice()
{
  Broadcast(1,"1"); Broadcast(1,"2"); BroacastAck(1,"1"); …
}
```

- No!
  - Events are like return values (responses)
  - We can slice stimuli (its part of test selection)
  - We can't slice responses (its part of test oracle)

Where a Trace can end: Accepting State Condition

- Is the following a valid word?
  - Micr
- Does the following trace represents a useful test?
  - Broadcast(1,"1"); Broadcast(1,"1")
- An *Accepting state condition* characterizes those states in which a trace can end
  - Used to ensure that a trace does not stop at arbitrary points
  - Used to ensure that a test leaves the system in a good state
- Accepting state condition for Chat:
  - All messages have been delivered to recipients

DEVELOPING THE MODEL

Modeling with objects and non-determinism

Chat State

```
enum UserState
{
  WaitingForLogon, LoggedOn, WaitingForList,  WatingForLogoff,
}

// A class representing a user
partial class User
{
  // The state in which user currently is.
  internal UserState state;
  // Broadcast messages which are waiting for delivery to this user.
  // This is a map indexed by the user which broadcasted the message,
  // mapping into a sequence of broadcast messages from this same user.
  internal MapContainer<int, Sequence<string>> waitingForDelivery;
}

// A mapping from logged-on users to their associated data.
static MapContainer<int, User> users = new MapContainer<int,User>();
```

User Helper Methods

```
partial class User
{
  internal bool HasPendingDeliveryFrom(int sId)
  { return waitingForDelivery.ContainsKey(sId); }

  internal bool HasPendingDeliveryFrom(int sId, string message)
  { return HasPendingDeliveryFrom(sId) &&
              waitingForDelivery[sId].Contains(message); }

  internal string FirstPendingMessageFrom(int sId)
  { return waitingForDelivery[sId][0]; }

  internal void AddLastMessageFrom(int sId, string message)
  {
    if (!HasPendingDeliveryFrom(sId))
      waitingForDelivery[sId] = new Sequence<string>();
    waitingForDelivery[sId] = waitingForDelivery[sId].Add(message);
  }

   internal void ConsumeFirstMessageFrom(int sId)
   {
     if (waitingForDelivery[sId].Count == 1)
       waitingForDelivery.Remove(sId);
     else
       waitingForDelivery[sId] = waitingForDelivery[sId].RemoveAt(0);
   }
}
```

Actions: Logon/Logoff

```
[Action]
static void LogonRequest(int userId)
{
  Requires(!users.ContainsKey(userId));
  User user = new User();
  user.state = UserState.WaitingForLogon;
  user.waitingForDelivery = new MapContainer<int, Sequence<string>>();
  users[userId] = user;
}

[Action]
static void LogonResponse(int userId)
{
  Requires(users.ContainsKey(userId));
  User user = users[userId];
  Requires(user.state == UserState.WaitingForLogon,
      1, "User MUST receive response for logon request");
  user.state = UserState.LoggedOn;
}
```

Actions: Broadcast

```
[Action]
static void BroadcastRequest
          (int id, string m)
{
  GetLoggedOnUser(id);
  foreach (User user in users.Values)
    user.AddLastMessageFrom(id, m);
}

[Action]
static void BroadcastAck
          (int id, int sId, string m)
{
  User u = GetLoggedOnUser(id);
  Requires
    (u.HasPendingDeliveryFrom(sId));
  Requires
    (u.FirstPendingMessageFrom(sId) == m);
  Capture(6,
    "Messages from one sender " +
     "MUST be received in order");
  u.ConsumeFirstMessageFrom(sId);
  if (EveryoneReceived(sId, m))
    Capture(5, "All logged-on users " +
      "MUST receive broadcasted message");
}
```

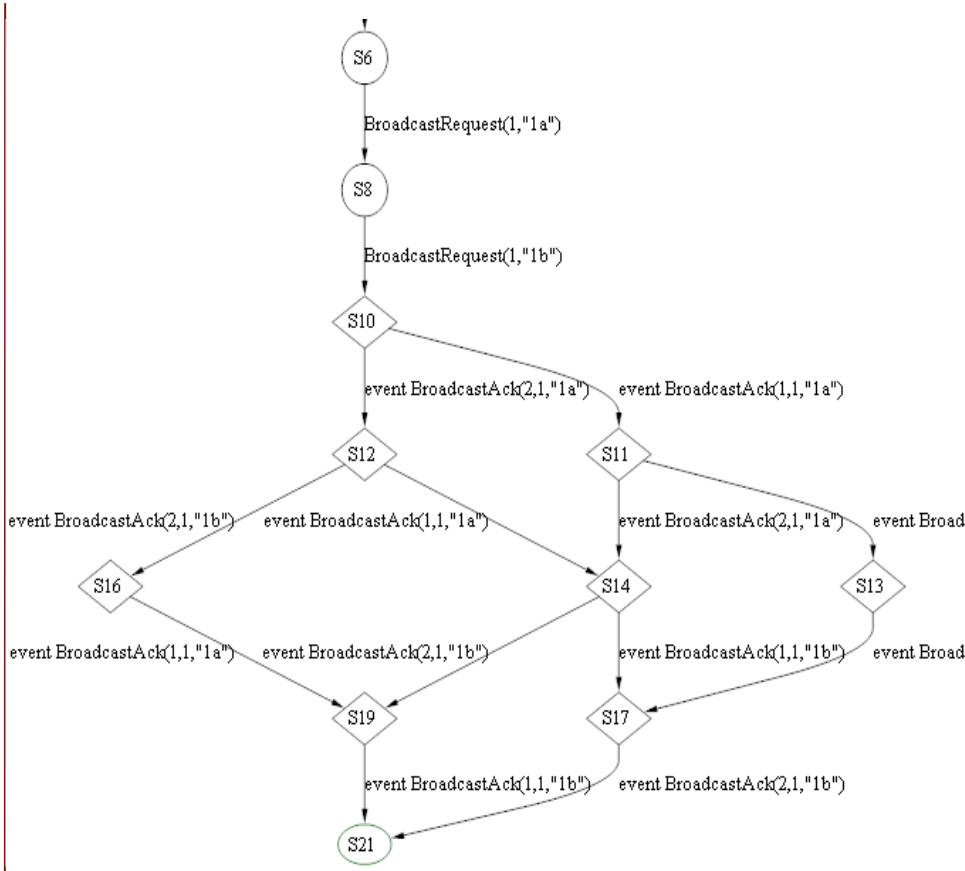
```
static User GetLoggedOnUser(int userId)
{
  Requires(users.ContainsKey(userId));
  User user = users[userId];
  Requires
    (user.state == UserState.LoggedOn);
  return user;
}

static bool EveryoneReceived
          (int sId, string m)
{
  return !users.Exists(u =>
    u.Value.state == UserState.LoggedOn &&
    u.Value.HasPendingDeliveryFrom(sId, m));
}
```

Event Queues

Recall one of the Chat slices

- What is the assumption?
  - Hint: event BroadcastAck can come in very fast!
- Events are buffered!
- Why buffer events?
  - They can occur asynchronously
    - Even within a call-return frame
  - Some state-space explosion avoided



WHERE WE ARE

Adoption

Where we are

- Shipped as Visual Studio Power Tool
  - Home Page
    - http://go.microsoft.com/fwlink/?LinkID=166911
  - Online MSDN Doc
    - http://msdn.microsoft.com/library/ee620411.aspx
  - Blog
    - http://blogs.msdn.com/b/specexplorer/
  - Online Forum
    - http://social.msdn.microsoft.com/Forums/en-US/specexplorer

Where we are

- Microsoft Product Groups (20+ teams, 40000+ test cases regularly executed)

Team Name
Dynamic AX
Windows Phone Communication
Forefront for Office
Winterop windows TSD/MIP
SharePoint Developer Experience
WSC
IEB Paris Reach Pillar
ISS can Media Discovery
DAIP AD
Office Market place
ECM
IEB Paris Media Pillar
Zune Services Commerce
USB Core
Windows Azure Marketplace
Dynamics NAV
Office TWC Security
Could Directory
CRM Dynamics IDC
SQL DMG Montego
Lync Client

Where we are

- External Customers

Corp/Team Name
GMAPCE, Adam Opel GmbH
Verizon
Ace Automation, Intel
Hengtian Ltd. China
Sensing and Inspection technologies, General Electric
Service and Developer Experience, Nokia, China
Wiscape Ltd. China

Questions?