Pattern-Based Development of TTCN-3 Test Suites

Alain Vouffo, Ina Schieferdecker, E. Noumedem
(<vouffo,schieferdecker>@fokus.fraunhofer.de)

Roadmap

- What are (Test) Patterns?
- Why TTCN-3 Test Patterns?
- Pattern Based TTCN-3 Test Development
- Evaluation of the Approach based on case study (OSA-Parlay API)
- Conclusions and Outlook
- Questions
Patterns Characteristics

- **Identify and specify abstractions** above level of single instances or components in a software system.
- **Document** existing well proven design experiences, software architectures and guidelines.
- Provide a **common vocabulary** and understanding for design principles.
- Address **functional** as well as **non-functional requirements** for software systems.
- Can provide **support for building software** with defined properties.
- Always **come from practical use**, although they are themselves abstract.

Example Software Design Patterns

- **Software Design Patterns**
  - Model-View-Controller
  - Proxy
  - Proactor
  - Visitor
  - Adapter
  - Singleton
  - Observer
  - Facade
What are Test Patterns?

- Test Patterns are an attempt to apply the pattern-based approach known in general software design in developing Test Systems.
- Goals are the same as for general software patterns:
  - Documenting sound solutions
  - Provide means for test system developers to focus more on what to test and less on the notation itself
  - Enhance reuse of test artifacts
  - Common Vocabulary
  - Simplify and fasten the test development process
  - Increase level of automation through pattern-based test generation
- Test Patterns are the first step towards test libraries.
- (Test) Patterns can be implemented in tools (e.g. Wizards, Type completion, Code Generation).

Motivations: Why Test Patterns with TTCN-3? (I)

- TTCN-3 Test Systems are increasingly complex:
  - Difficult to maintain
  - Documentation Problem, because the test intents get lost in the complexity
- Provide Test/System Developers a mean to express key aspects of testing in a more abstract manner than TTCN-3
- Facilitate Model transformation from SUT Model to Test Model
- Abstracting from too complex test specifications, without losing the powerful features of the abstract test notation being used
- Back to the essentials:
  - Without necessarily having to navigate through the TTCN-3 source, it should be possible to understand rapidly what actually happens in a test case.
Motivations: Why Test Patterns With TTCN-3? (II)

- Growing complexity of the SUTs => growing complexity of the ATS
  - *E.g.* Middleware or Telecommunication Systems with several different components providing and using *several interfaces* at the same time (IMS, OSA-Parlay)
  - The complexity of the Test System rises dramatically for performance and load testing involving *concurrent behaviour* among the test components
  - Less Readability of ATS
  - Less Reusability => Maintenability => Costs

- **Question:** How to ensure that the test system, while coping with the SUT’s complexity, does not also turn into a programming nightmare? (Avoiding the „Who test the tester” dilemma).

- **The Answer:** Focus on the essential aspects => Test Patterns

---

Motivations: TTCN-3 Test Development Process

1. Requirements Analysis
2. System Design using Design Patterns (e.g. UML)
3. Coding in target language (C, C++, Java etc.)

Software System Development

TTCN-3 Test System Development
Classification of TTCN-3 Patterns

- Test data patterns
  - Test Data (TTCN-3 Templates) are defined for certain test purposes or to fulfill certain constraints
  - Incoming/Outgoing data (wildcards, optional fields)
- Behavioral test patterns
  - Send — Receive
  - Send — Discard
  - Trigger — Receive
  - Exception handling
  - …
- Architectural test patterns
  - Configurations
  - Coordination and synchronization of Test Components

Test Data Patterns: Examples

- Data Pattern Kinds
  - Domain Partition
  - Default Value
  - Boundary Value
  - Random Value

```tcl
template EchoRequest m_echoRequest_extHdr_data (...) := {}

template EchoRequest mw_echoRequest (...) := {
  ipv6Hdr := mw_ipHdr_nextHdr_srcDst(c_icmpHdr, p_src, p_dst),
  extHdrList := *,
  icmpType:= c_echoRequest,
  icmpCode:= c_icmpCode0,
  checksum:= ?,
  identifier:= ?,
  sequenceNumber:= ?,
  data:= *
}
```
Architectural Patterns: Examples

- Mesh Configuration Pattern

- PMP Pattern

TTCN-3 Behavior Patterns: Examples

function `<SendAndReceive>` (template `<T>`) to_send, (template `<T>`) to_receive) {
    <send to_send>
    <activate defaults>
    <start guard timer>
    <receive to_receive>
    <stop guard timer>
    <deactivate defaults>
    }

Test Patterns: Methodology

SUT Model
- Data Model
- Arch. Model
- Behavior Model

TTCN-3 ATS
- TTCN-3 Libs
- Hand-Written TTCN-3 Code
  - Test Data
  - Test System Arch
  - Test Behavior

TTCN-3 User Conference, Stockholm, 2007/05/31

Case Study: OSA-Parlay API Testing
- Pattern-Based Approach was used to write a test suite for an operation-based system (OSA-Parlay API)

- Challenges
  - System specification in IDL or XML (TTCN-3 Mapping issues)
  - Very complex SUT involving several hundreds interfaces
  - Test System plays both client and server roles
  - Operation-based system => No default behaviors => Exceptional behaviors must be handled explicitly

- Chances
  - Good opportunity to demonstrate usage of TTCN-3 for such systems
Background and work presentation (I)

- **Background: OSA-Parlay Architecture**
  - Open API specified by 3GPP, ETSI and Parlay Group.
  - The aims are to enable Operator and 3rd party Applications developer to use networks functionality independently of the underlying networks.
  - Parlay/OSA APIs are specified in CORBA/IDL and XML/SOAP and define three types of component: Framework, services capability server and Applications Interface
  - Current version of parlay is 6.0 whereas release 4 of 3rd version has been use in this work. It is grouped on the following parts.

```
From SUT Specification to ATS (I): Test Configuration
```

```
Generated TTCN-3 Component Types & Configuration
```

```
Test Component Type
```

```
MTC
```

```
IpInitial IpAPILevelAuthentication
```

```
IpAccess
```

```
Framework
```

```
IpClientAPILevelAuthentication
```

```
1. initiateAuthentication ()
2. selectEncryptionMethod ()
3. authenticate ()
4. authenticationSucceeded ()
5. authenticate ()
6. authenticationSucceeded ()
7. requestAccess ()
8. obtainInterface ()
```

**Parlay/OSA Applications**

```
Enterprise Domain
```

```
Network Operator Domain
```

```
Parlay/OSA Applications
```

```
Internet
```

```
IP Network
```

```
Hosted Application Server
```

```
Intranet
```

```
Firewall
```

```
Parlay Gateway
```

```
PSTN
```

```
Parlay/OSA Applications
```

```
Mobile Network
```

```
Enterprise Domain
```

```
3rd Party Domain
```

```
Account Management
```

```
Generic Messaging
```

```
Different part of the Parlay 3.1 API specification
```

```
Connectivity Management
```

```
User Interaction
```

```
Terminal Capabilities
```

```
Framework
```

```
Call Control
```

```
Account Management
```

```
Connectivity Management
```

```
Data Session Control
```

```
Mobile
```

```
Enterprise
```

```
Domain
```

```
Application Server
```

```
3rd Party Domain
```

```
Network Domain
```

```
Call Control
```

```
Account Management
```

```
Charging
```

```
Terminal
```

```
Capabilities
```

```
Framework
```

```
Different part of the Parlay 3.1 API specification
```

```
Connectivity Management
```

```
User Interaction
```

```
Terminal Capabilities
```

```
Firewall
```

```
PSTN
```

```
Parlay Gateway
```

```
Mobile Network
```

```
Enterprise
```

```
Domain
```

```
Application Server
```

```
3rd Party
```

```
Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```

```
Network Domain
```
From SUT Specification to ATS (II): Behavior Functions

- Behavior functions are generated:
  - Call_InitiateAuthentication()
  - Call_SelectEncryptionMethod()
  - Call_Authenticate()
  - Call_AuthenticationSucceeded()
  - GetCall_Authenticate()
  - GetCall_AuthenticatedSucceeded()
  - Call_RequestAccess()
  - Call_ObtainInterface()

---

Case Study: Generated TTCN-3 Code Snippet

```ttcn-3
function call_viaPort_IpInitial__initiateAuthentication(
  inout IpInitial port_p,
  in org__csapi__fw.TpAuthDomain IpInitial__clientDomain_p,
  in charstring IpInitial__authType_p,
  in template org__csapi__fw.TpAuthDomain rtn_templ_p)
runs on IpInitial_Tester
return org__csapi__fw.TpAuthDomain rtnValue := …;

  port_p.call (IpInitial__initiateAuthentication: {
    IpInitial__clientDomain_p, IpInitial__authType_p
  }, T_CLIENT) { …
    port_p.getreply (IpInitial__initiateAuthentication, org__csapi.TpCommonExceptions: ?) {
      setverdict (inconc);
    } …
    port_p.getreply { setverdict (fail); }
    port_p.catch (fail); 
  } …
  port_p.catch (timeout) { setverdict (fail); }

  return rtnValue;
```

The highlighted part of the code snippet represents the core test behaviour. Everything else is implicit, but still required

---

Could be hidden away from the user based on predefined rule/pattern
OSA Parlay Case Study: Results and Findings

- TTCN-3 Library for OSA-Parlay successfully generated
  - Very helpful in speeding up development process
- Tests successfully executed against real-life implementations
- Brute force not efficient
  - Too much potentially unused code generated
- Human intervention is required for selecting relevant SUT interfaces and messages for more efficient code generation
- Approach applicable for Asynchronous (Message-based) communication as well, however
  - Refinements of behaviour model required
  - More convenient Approach for patterns definition and code generation rules required (currently hard-coded in XML)

Conclusions and Outlook

- Pattern-oriented Test Development has been used successfully in testing operation-based systems with TTCN-3
- Approach is also applicable for message-based systems (e.g. protocol stacks).
- A mean for expressing test patterns is required to decouple from the complexity of the systems to be tested
- Test Patterns can effectively fill the gap between system specification and test specification => facilitate test automation
Outlook: Which notation for TTCN-3 Patterns?

- Issue of notation is not essential but worth discussing
- SoA
  - UML & affiliates (U2TP)
  - SDL
  - XML
  - Others?
- Future
  - TTCN-3?