JAIN-SIP: Architecture, Implementation, Testing

M. Ranganathan
mranga@nist.gov
Advanced Networking Technologies Division
http://w3.antd.nist.gov
National Institute of Standards and Techology
http://www.nist.gov
Gaithersrburg, MD 20899.

Speaker Introduction

- Speaker affiliation:
  - NIST: National Institute of Standards and Technology.
    - ANTD: Advanced Networking Technologies Division.
  - Not a member of JAIN-SIP expert group.
- Our interests are:
  - Applied Networking Research, Standards, Testing
- Architect and main developer of NIST-SIP: public domain JAIN-SIP Java SIP Stack.

Talk Outline

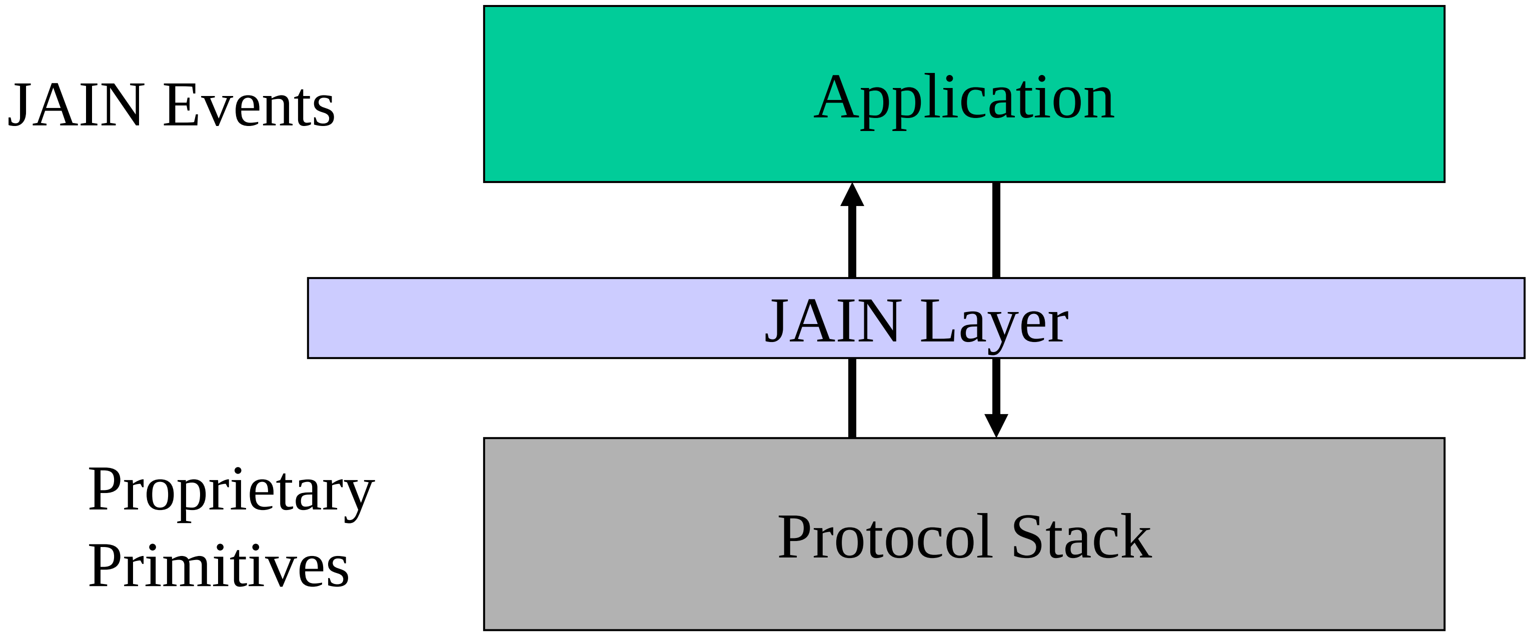
- Brief overview
- The function of JAIN SIP (why use it)
- JAIN architecture preliminaries
- JAIN-SIP abstractions and what they do
- Services provided by the JAIN stack
- Application responsibilities
- A skeleton application

Talk Outline (Contd.)

- Parsing SIP headers
- Wrapping a JAIN implementation around a SIP stack
- Testing the implementation
  - The TCK
  - Other useful test tools
- Loose ends and ideas for future spec

JAIN Architecture

- JAIN Provides an event-layer abstraction for applications.



Overview: JAIN-SIP

- JAVA-Standard interface to a SIP Signaling Stack. Spec Lead: Chris Harris, DynamicSoft charris@dynamicsoft.com
- Wraps the low-level stack and protocol abstractions in a JAVA interface layer
- Allows a JAVA application/servlet or bean to imbed a SIP stack and access low level functions

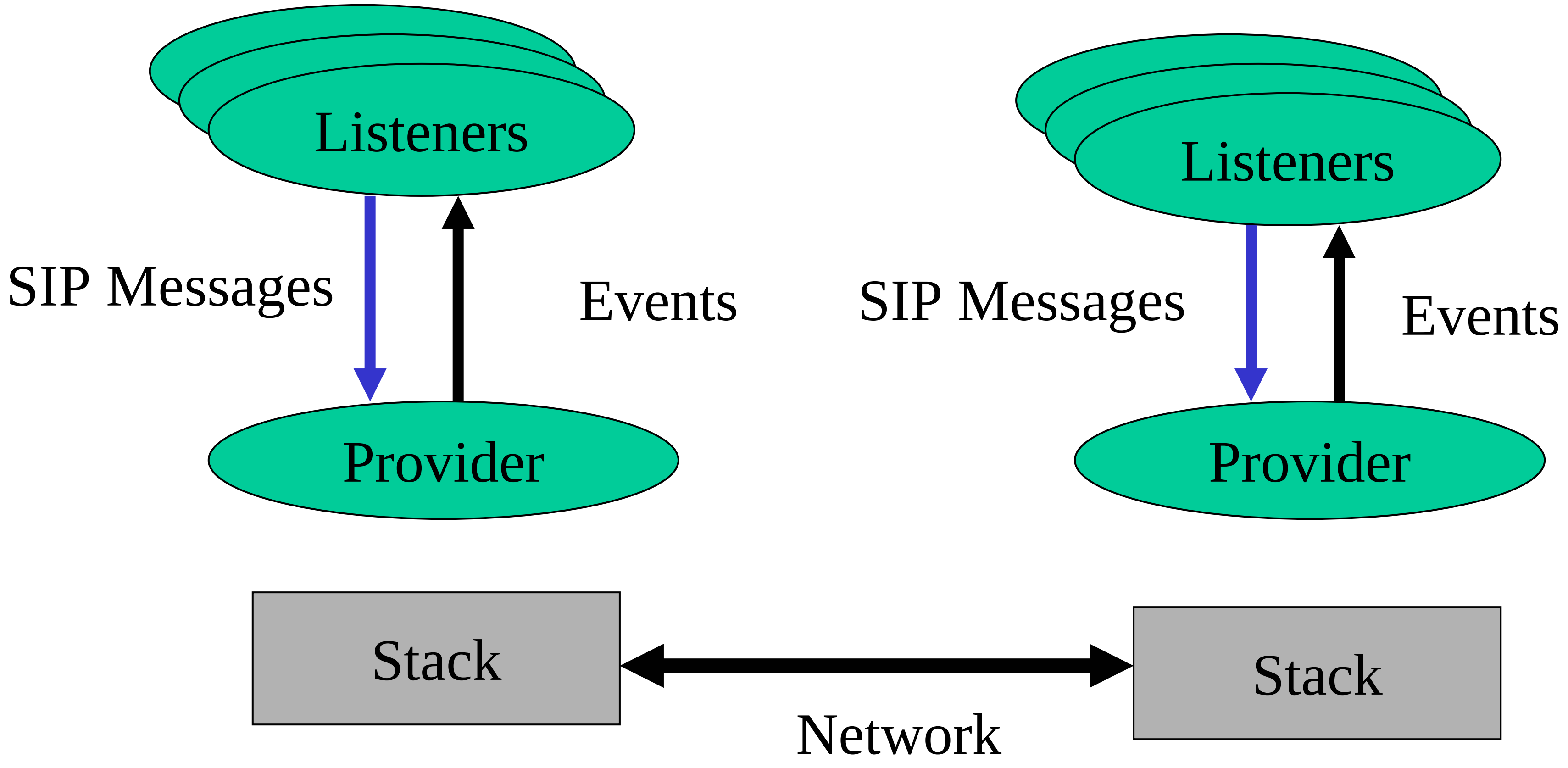
Overview: JAIN SIP (Contd.)

- Simplifies the construction of SIP components:
  - User Agents, Proxy Servers, Presence Servers.
- JAIN SIP can be utilized in a User Agent or Proxy
- Holy Grail: Application portability between JAIN SIP stacks via definition of interfaces and run-time behavior. Ensure interoperability via the TCK.

Application Portability

- How does JAIN-SIP aim to achieve application portability between JAIN-SIP compliant stacks?
  - Standardize the interface to the stack.
  - Standardize the events and event semantics.
  - Standardize transactional semantics.

JAIN-SIP Architecture



Responsibilities the Application

- Application MUST go through the provider for all interactions with the stack (no direct access to the wire protocols).
- Application registers an implementation of the SipListener interface with the stack.

Responsibilities the Application (Cont.)

- Application sees all signaling traffic and is responsible for sending responses via the SipProvider.
- Application is responsible for retransmission processing on timeout for stateless implementation.

Responsibilities of the Stack

- Provide methods to format and send SIP messages
- Parse incoming sip messages and allow application to access / modify fields through a standardized JAVA interfaces
- Invoke appropriate application handlers when protocol significant events occur
- Provide transaction support
- Manage transactions on behalf of a user application

JAIN-SIP Packages

- jain.protocol.ip.sip
  - Stack, provider and other packages.
- jain.protocol.ip.sip.header:
  - Header factories, interfaces for each supported header.
- jain.protocol.ip.sip.message
  - Message factories : Create messages for sending out.
- jain.protocol.ip.sip.address
  - Address factories: Parse and create URL and address objects.

JAIN-SIP Abstractions

- jain.protocol.ip.sip.SipFactory:
  - Creates the main Stack object.
- jain.protocol.ip.sip.SipStack
  - Event generator: Fields incoming messages and generates events.
  - Transaction handler: Manages transactions and generates transaction timeout events. Transaction objects are not directly accessible by the application.

JAIN-SIP Abstractions (Contd.)

- jain.protocol.ip.sip.ListeningPoint:
  - Corresponds to a Stack Address (UDP/TCP) – IP address and port from which the stack can receive and send messages.
  - The stack is configured with one or more listening points.
- jain.protocol.ip.sip.Provider
  - Provides helper facilities for the application program (sendRequest, sendResponse, sendAck…)

The SipListener Interface

JAIN-SIP Application programs must implement the jain.protocol.ip.sip.SipListener interface

```
public interface SipListener extends java.util.EventListener
{
public void processResponse(SipEvent responseReceivedEvent);
public void processRequest(SipEvent requestReceivedEvent);
public void processTimeOut(SipEvent transactionTimeoutEvent);
}
```

JAIN-SIP Application Skeleton

Create a SipFactory object instance

```
sipFactory = SipFactory.getInstance();
sipFactory.setPathName("gov.nist");
```

Create a SIP Stack instance

```
 try {
            sipStack = sipFactory.createSipStack();
 } catch(SipPeerUnavailableException e) {
            System.exit(-1);
} catch(SipException e) {
            System.exit(-1);
}
```

JAIN-SIP Application Skeleton (Contd.)

Create factories to format headers and send messages

```
HeaderFactory headerFactory =
sipFactory.createHeaderFactory();
AddressFactory addressFactory =
   sipFactory.createAddressFactory();
MessageFactory messageFactory =
   sipFactory.createMessageFactory();

… format and send off  invite message using sendMessage
```

JAIN-SIP Application Skeleton (Contd.)

Handle incoming messages (delivered as events):

```
public void        processRequest(SipEvent requestReceivedEvent) {
   Request request = (Request)requestReceivedEvent.getMessage();
   long serverTransactionId = requestReceivedEvent.getTransactionId();
   try {
       if (request.getMethod().equals(Request.INVITE))
            processInvite(request,serverTransactionId);
        else if (request.getMethod().equals(Request.ACK))
            …..
   } catch (SipParseException ex) {
            ex.printStackTrace();
   }
}
```

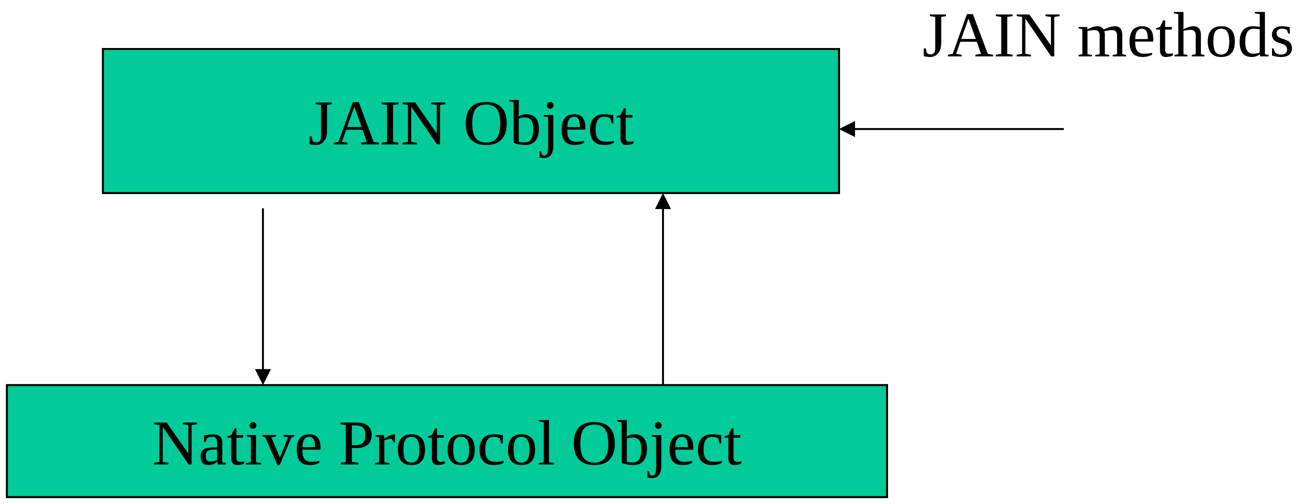
JAIN-SIP Application Skeleton (Contd.)

Handle Timeout Events

```
public void processTimeOut(jain.protocol.ip.sip.SipEvent
        transactionTimeOutEvent) {
   try {
     if(transactionTimeOutEvent.isClientTransaction()) {
         get the request for this transaction
             sipProvider.sendRequest(request);
     }
   } catch (Exception ex) {
        ex.printStackTrace();
   }
}
```

JAIN Implementation Wrappers

- JAIN API are defined as interfaces.
- Interface implementation is JAVA
- Protocol object can be JAVA/Native



Building a SIP Stack

- Parsing incoming messages.
- Rewriting/responding to messages:
  - Generating outgoing messages from incoming messages.
- Routing outgoing messages:
  - Sending them off to the right next hop.
- Transaction handling:
  - Matching requests to responses.

Parsing SIP Headers

- The SIP grammar is specified in ABNF form.
- Grammar is compositional (includes grammar of other RFCs).
  - Introduces some syntactic ambiguities.
- SIP headers can be parsed using a “hand crafted” parser or by inputting the grammar to a parser generator.

Parsing SIP Headers (Cont.)

- Parser generators are good for clearly identifying and dealing with syntactic ambiguities.
- Parser generators can generate slow and huge parsers.
  - Lots of code means slow class loading.
  - May not be good for production SIP stacks.

Parsing SIP Headers (Cont.)

- RFC grammar makes it difficult to directly use parser generator tools directly.
  - Must be able to directly handle ABNF.
  - Must be able to cleanly deal with ambiguities.
- A few tools like antlr are suitable:
  - Allows closure operations on non-terminals.
  - Syntactic and semantic predicates allow systematic dealing with syntactic ambiguities.
  - Grammar composition features.

Eager and Lazy Parsing

- JAIN-SIP allows for either eager or lazy parsing.
- Eager parsing – the entire message is parsed as soon as it is received.
- Lazy parsing – the headers are parsed as needed by the application. Can provide performance advantages.
  - When the message is received, text is stored but not parsed.
  - Portions of the message parsed when needed.

Rewriting/Responding to Messages

- Follow RFC 2543 Rules in generating Responses to requests.
- These rules are implicit in the JAIN-SIP Spec/implementation.

Parsing: Error Reporting

- Errors are reported up to the application using *SipParseException*.
  - Thrown on both set and get methods.
  - Allows for lazy or eager parsing.
- *SipParseException* is supposed to capture the portion of the header that lead to the error.

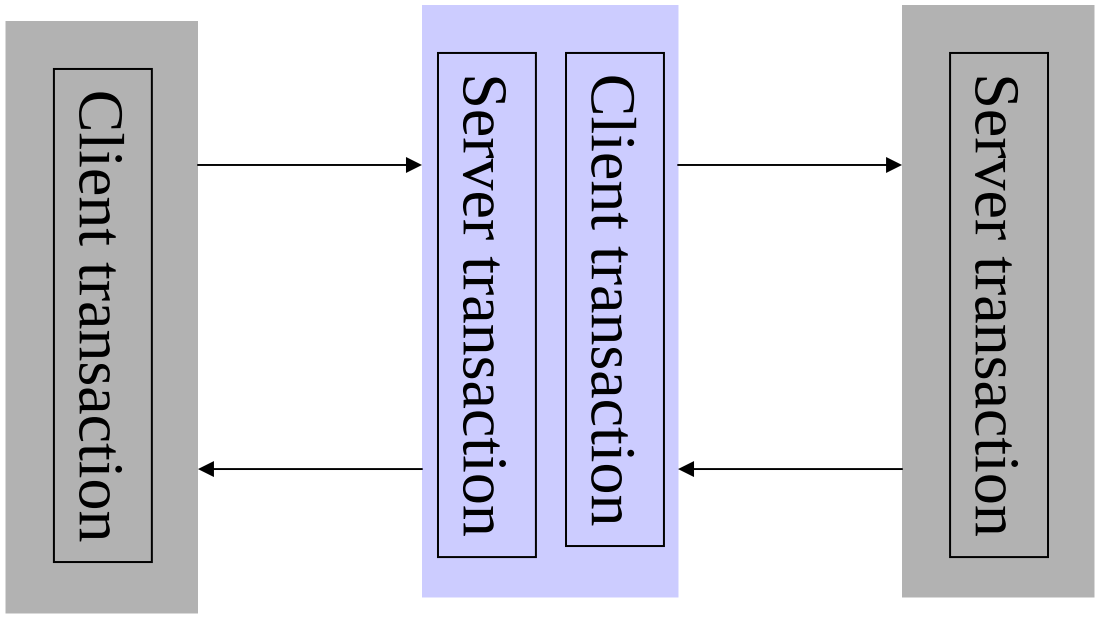
Lazy Parsing

```
public interface HeaderIterator
{
    public boolean hasNext();
    public Header next()
                    throws HeaderParseException,
                        NoSuchElementException;
}
```

Transactions

- SIP transaction consists of a single request and any responses to that request, which include zero or more provisional responses and one or more final responses (from RFC 2543).
- The SIP protocol provides enough state in the SIP message to extract a transaction identifier.

Client and Server Transactions

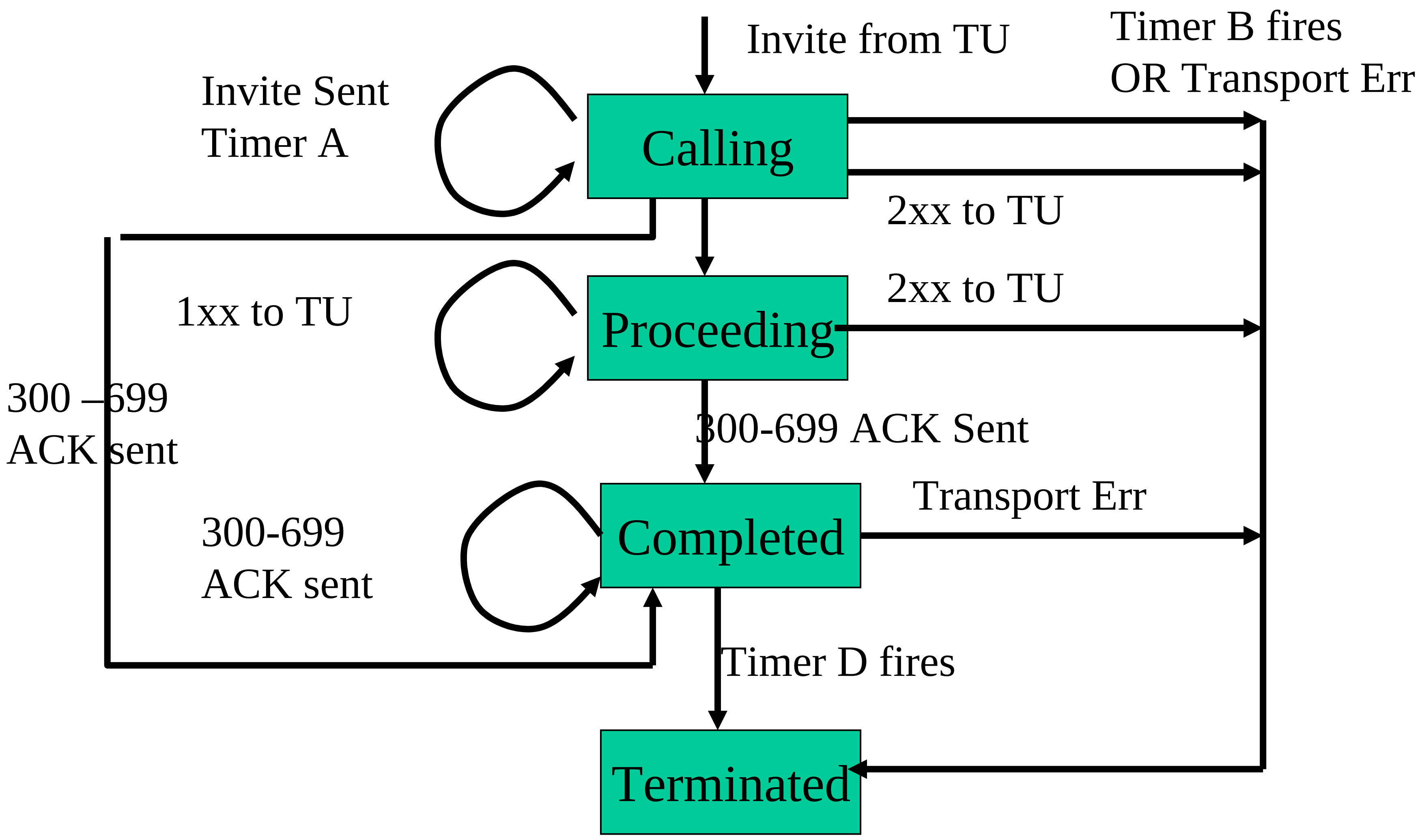


UAC Statefull proxy UAS

Transactions in JAIN-SIP

- JAIN-SIP identifies transactions by a long integer.
- Transactions are associated with all sipProvier.sendXXX methods.
- SipListener.processRequest(SipEvent)
  SipListener.processResponse(SipEvent)
  SipListener.processTimeout(SipEvent)
- The transaction identifier is retrieved from the SipEvent for the Timeout Event.

INVITE Transaction (RFC 2543)



Transaction Timeouts

- Stateless implementation:
  - The stack generates periodic timeout notifications for outstanding transactions.
  - The application keeps track of the retransmission state machine.
  - Timeouts are delivered to the application in an exponentially decaying fashion.

Transaction Timeouts (Contd.)

- Statefull implementation:
  - The stack handles retransmissions.
  - Timeout when the transaction does not complete as expected (longest arc in the previous diagram).

Statefull Vs. Stateless

- JAIN-SIP does not specify whether the implementation should be statefull or stateless.
  - Controversy in mailing list discussions.

Statefull Vs. Stateless (Contd.)

- Statefull implementation:
  - Makes it easier to implement user agents/b2bua/statefull proxy.
  - Not suited for stateless proxy servers.
- Stateless implementation:
  - Thinner implementation.
  - Allows implementation of stateless proxy servers but puts more burden on implementation of user agents.

Dealing with Extensibility

- SIP extensibility: can add new request methods and new headers.
- JAIN-SIP provides support for all the headers in RFC2543.
- Headers can be created/accessed by name.
- Messages can be created with any method name (implementation can reject it and throw a SipParseException if needed).

The TCK

- A test suite that is supposed to ensure interoperability between JAIN implementations.
- Ideally – if two implementations pass the TCK then they are plug replace-able.

JAIN-SIP TCK

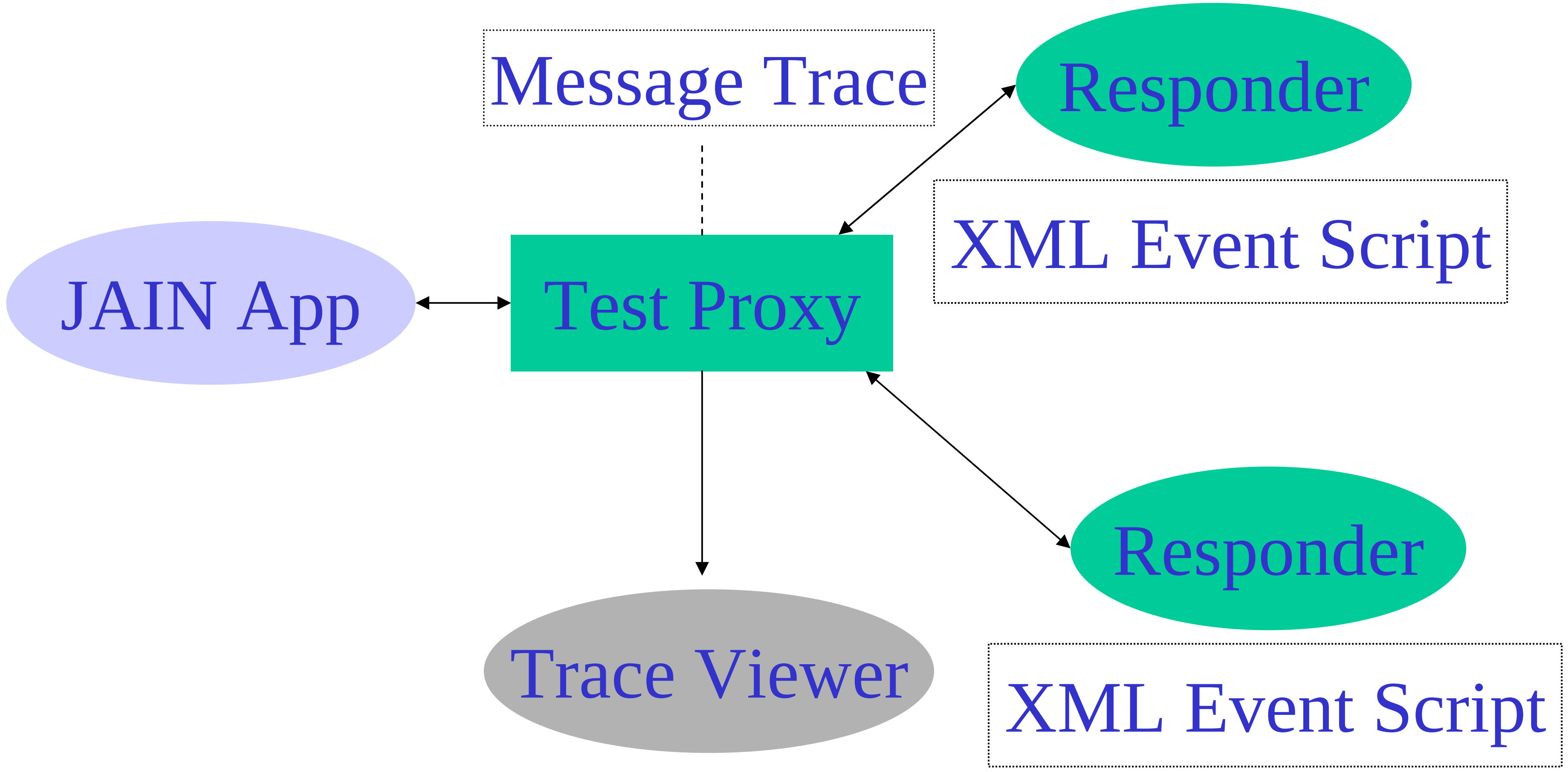
- The TCK is a unit test. Can test all the set/get methods, stack creation, message send/receive and some transaction processing.
- Does not test timeouts and detailed transactional semantics.
- Lots of good protocol test tools available to test your application/implementation.
  - NIST responder tool: allows you to test against a call flow.

Protocol Test Tool

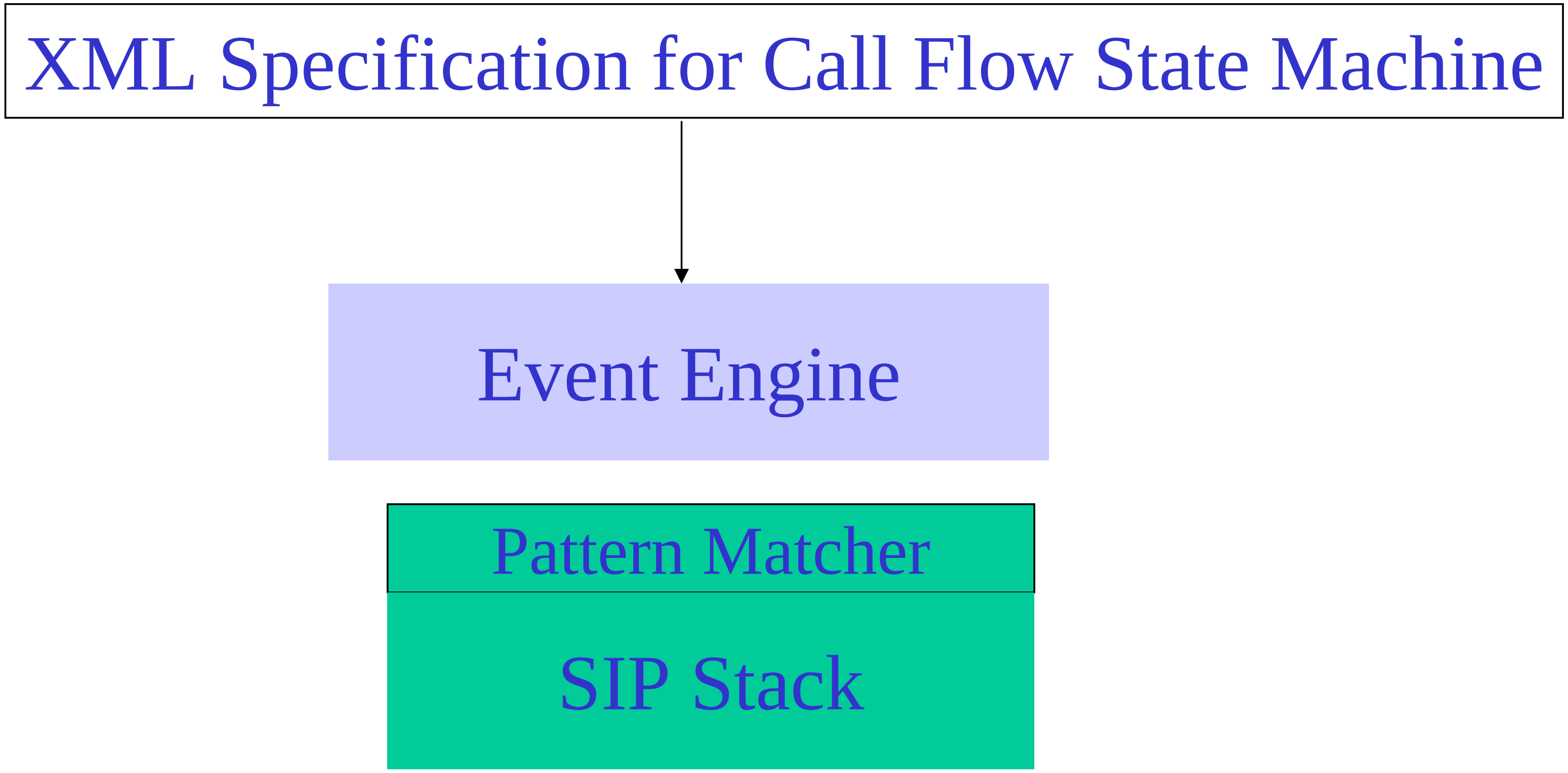
- **Flexible framework for..**
  - Call Flow Generation
  - Test Scripts, Error Scenarios
- **Scripting Technology**
  - XML+NIST-SIP+JAVA/JYTHON
  - XML State machine templates
  - Event triggers, responses specified in XML+JAVA or XML + JYTHON
  - NIST-SIP event engine drives script and implements responses.

```
<!-- Expect an ACK and send a BYE 40 seconds after the ACK.
 Note that this includes some imbedded code that triggers when
 the ACK arrives. When the transaction complets the onCompletion
 attribute specifies the code fragment that runs
-->
<EXPECT
        nodeId              = "node6"
        enablingEvent       = "INVITEReceivedOKSent"
        triggerMessage      = "ACK"
        generatedEvent      = "ACKReceived"
        executeOnTrigger    = "onACKReceived"
        executeOnCompletion = "onTransactionComplete"
>
<GENERATE
        delay       = "100"
        retransmit = "true"
>
<SIP_REQUEST>
<REQUEST_LINE
        method = "BYE"
/>
</SIP_REQUEST>
```

NIST-SIP Protocol Test Tool

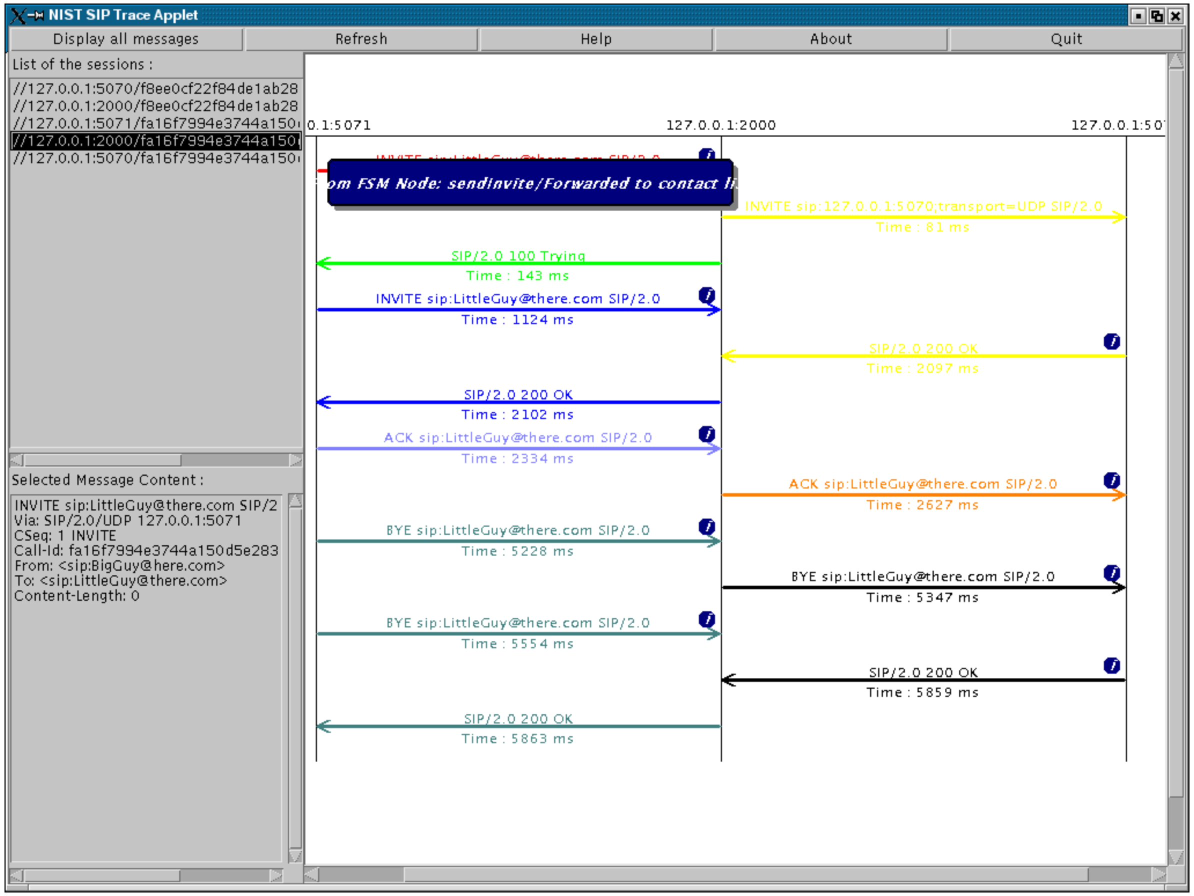


Responder


XML Specification for Call Flow State Machine
Event Engine
Pattern Matcher
SIP Stack

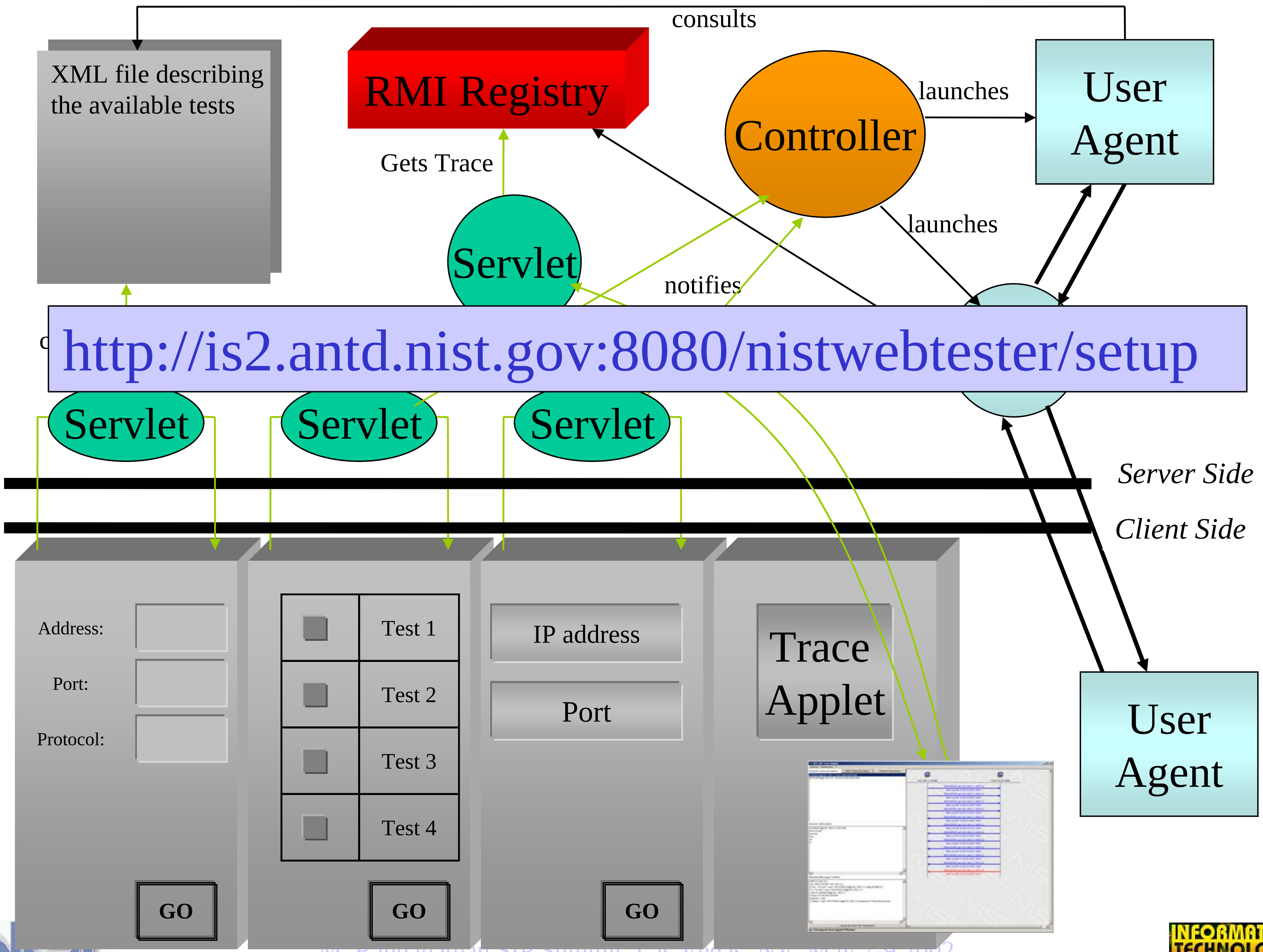
Visualizing Multiparty Traces

- Java Applet collects and visualizes distributed call flow trace files.
- Augmented with XML script state information.
- Enables debugging call flows & test scripts.



```
//127.0.0.1:5070/f8ee0cf22f84de1ab28
//127.0.0.1:2000/f8ee0cf22f84de1ab28
//127.0.0.1:5071/fa16f7994e3744a150
//127.0.0.1:2000/fa16f7994e3744a150
//127.0.0.1:5070/fa16f7994e3744a150
```

```
INVITE sip:LittleGuy@there.com SIP/2
Via: SIP/2.0/UDP 127.0.0.1:5071
CSeq: 1 INVITE
Call-Id: fa16f7994e3744a150d5e283
From: <sip:BigGuy@here.com>
To: <sip:LittleGuy@there.com>
Content-Length: 0
```



http://is2.antd.nist.gov:8080/nistwebtester/setup

Selected Design Issues

- Issues from discussion list (gathered by Phelim O'Doherty, sun Microsystems).
- Introduce management capabilities for ListeningPoints
- Including proxies capabilities explicitly in the API :
  - Make the transaction model configurable?

Selected Design Issues (Contd.)

- sendBye should take a Request or a Response as argument as it does not rely on a previous transaction.
- sendAck should not generate a new transaction.

Loose Ends: Dialogs and Tags

- Dialog: A peer-to-peer relationship between communicating SIP entities.
- Dialog is identified by to tag, from tag and call id.
- JAIN SIP does not deal directly with dialog objects.
- Need to deal with legacy issues (JAIN-SIP is based on RFC 2543 (no bis)). Semantic changes since the first spec release

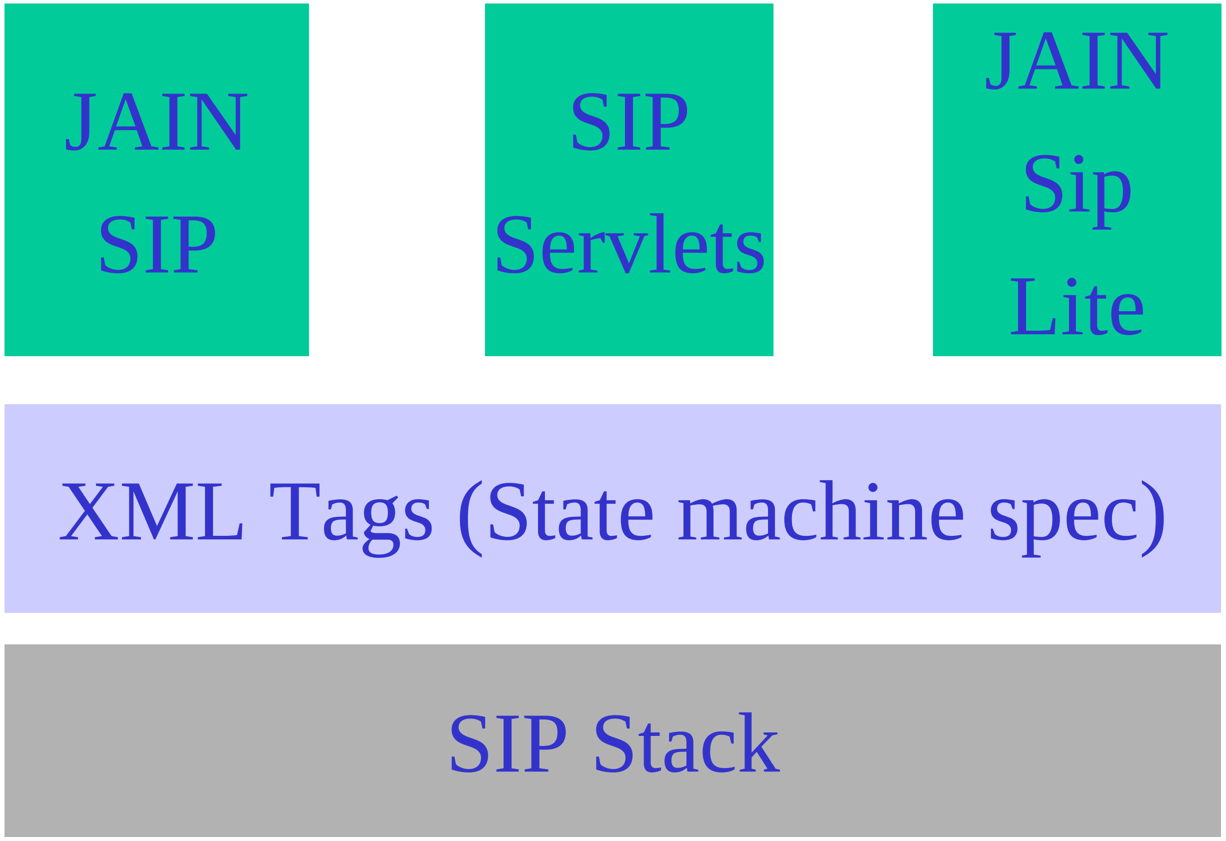
Loose Ends: Statefulness

- Statefull versus stateless implementation?
  - Application portability problem (a portable application has to be written assuming the least common denominator -- needless complexity).
  - What is the correct spec interpretation?
  - Better way to define the spec?

Loose Ends: Stack Configuration

- JAIN does not address stack configuration.
- Configuration parameters include such things as:
  - IP address/port on which the listener will be configured.
  - What transport types the listener will support.
  - Proxy address.
  - Transaction model (?).

Consistent Spec Definition
(Some Ideas -- comments solicited)



Related Specifications

- JSR-164 JAIN-SIMPLE: a set of extensions made to the SIP protocol to support presence and instant messaging.
- JSR-125 JAIN-SIP-Lite: is aimed User Agent application development. Hides the gory details of the SIP protocol and provides a high level abstraction for developers of User Agent (UAC/UAS) software.
- JSR-141 SDP API: Builds a wrapper around a SDP interface

Related Specifications

- JSR 21 JAIN Call Control API: Protocol Independent Call Control. Note that JAIN-SIP-LITE is UA only and protocol aware.
- JSR-116 SIP Servlets: HTTP Servlet like interface for SIP. Primarily for use on proxy servers.

Useful URLs

JAIN SIP Specification:
http://jcp.org/jsr/detail/032.jsp

JAIN-SIP Disucssion List:
mailto:JAIN-SIP-INTEREST@java.sun.com

NIST IP-Telephony Project page (NIST-SIP):
http://is2.antd.nist.gov/proj/iptel
Web-based protocol tester (in development):
http://is2.antd.nist.gov:8080/webtester/setup.html

Acknowledgement

- Chris Harris (DynamicSoft) and the JAIN-SIP expert group.
- JAIN-SIP interest mailing list.
- SIP-IMPLEMENTORS mailing list.
- Numerous contributors to the NIST-SIP project from the user community.
- NIST guest researchers: Olivier Deruelle, Christophe Chazeau, Marc Bednarek.