SAMP: Architecture, JSAMP and sampjs

Mark Taylor (University of Bristol)

BoF: Interoperability with SAMP
ADASS XXII
Champaign IL
7 November 2012
Outline

- SAMP overview and architecture
- **JSAMP**: Java toolkit for SAMP
  - Diagnostic tools
  - Library for adding SAMP to Java applications
- **sampjs**: JavaScript library for SAMP
  - Library for adding SAMP to web pages
Background and History

SAMP = Simple Applications Messaging Protocol
allows astronomy software tools to exchange control and data

History:

- PLASTIC v1 (Platform for Astronomical InterConnection), Euro-VO protocol 2006
- SAMP v1.11, IVOA Recommendation 2009
- Useful client-side technology for VO work patterns
- . . . but not specific to VO applications

Status:

- Quite widely used in desktop tools
  - Java, Python, Perl, C, C#, Tcl, IDL, . . .
- Visible at ADASS
  - ADASS XIX: 5 subject index entries for “SAMP”
  - ADASS XX: 16 subject index entries for “SAMP” (5th after Java, Python, VO & XML)
  - ADASS XXI: 7 subject index entries for “SAMP”
Principles to maximise interoperability:

- Simple to use and learn for client developers and users
  - Platform independent
  - Lightweight to implement

- Message semantics are typically vague
  - “Here’s a table!” not “Plot entries from this catalogue over the current image”
  - but also extensible
  - Arise from usage, not decreed by committee

Consequences:

- Loosely coupled suites of interoperating tools
  - ... selected by the user
    - ... without conscious effort
  - ... from a pool of tools contributed by developers
    - ... who do not need close collaboration

- It works!
Key Concepts

- Hub-based operation
  - *Hub* is a daemon process, conceptually freestanding (though may run within one client)
  - Clients *register* with Hub to send/receive messages
  - Hub brokers messages and provides directory services
  - All communication is Client ↔ Hub (but messaging conceptually Client ↔ Client)

- Publish/Subscribe
  - Each client *subscribes* to zero or more message types (MTypes)
  - Clients can only receive subscribed messages

- Callable Clients
  - All clients can *send* messages
  - Only *callable* clients can subscribe to/receive messages
  - Callability is optional
3-Layer Architecture

Abstract API
- Data types
- Message structure
- Control functions

Profile
- Transport protocol
- Data encoding
- Hub discovery

MTypes
- Message semantics
- Arguments
- Return values

Standard Profile

Web Profile
- image.load.fits
- table.highlight.row
- coords.sky.pointAt
- ...
Layer 1: Abstract API

Hub API:

- register()
- unregister()
- declareMetadata(map metadata)
- declareSubscriptions(map subscriptions)
- getRegisteredClients()
- getSubscribedClients(string mtype)
- getMetadata(string client-id)
- getSubscriptions(string client-id)
- notify(string recipient-id, map message)
- notifyAll(map message)
- call(string recipient-id, string msg-tag, map message)
- callAll(string msg-tag, map message)
- callAndWait(string recipient-id, map message, string timeout)
- reply(string msg-id, map response)

Callable Client API (optional):

- receiveNotification(string sender-id, map message)
- receiveCall(string sender-id, string msg-id, map message)
- receiveResponse(string responder-id, string msg-tag, map message)
The *Profile* maps the abstract API to bits on the wire (or similar)

- Two profiles currently defined:
  - **Standard Profile:**
    - Suitable for desktop applications
    - Based on XML-RPC
    - Hub discovered using local lockfile `~/.samp` (usually)
    - Callable clients run their own XML-RPC server to receive messages
  - **Web Profile** *(since April 2012 only)*:
    - Suitable for web applications (e.g. JavaScript)
    - Based on XML-RPC
    - Hub discovered at fixed port 21012
    - Special measures for safe sandbox evasion
    - Callable clients use long pull

- Profile interoperability:
  - Each client uses one appropriate profile
  - The hub can accept connections using multiple profiles
  - Clients are treated the same regardless of profile
MTypes (message types) define message semantics

- An MType is:
  - A short hierarchical string (a.b.c)
  - . . . with associated input parameters
  - . . . and associated return type
  - . . . and associated semantics

- Think of it like a function call definition in an API

- Example:

  **table.load.votable**: Loads a table in VOTable format

  Arguments:
  - `url (string)`: URL of table to load
  - `table-id (string)`: Identifier for use with subsequent messages (optional)
  - `name (string)`: Name to label loaded table for user (optional)

  Return values:
  - `None`

- Other examples:
  - `table.load.fits, table.highlight.row, image.load.fits, coord.pointAt.sky, samp.hub.event.register, samp.app.ping`
  - More at [http://wiki.ivoa.net/twiki/bin/view/IVOA/SampMTypes](http://wiki.ivoa.net/twiki/bin/view/IVOA/SampMTypes) (add your own!)
Other Details

Things I haven’t mentioned

- Subscription wildcarding
- Extensible parameter model
- Parameter data model
- Message delivery patterns (call/notify)
- Message targets (broadcast/send)
- Asynchronous processing model
- Error processing
- Client tracking
Hub Availability

Hub Implementations:

- JSAMP (Java)
- SAMPy (Python)
- ... some others, but implementations partial

How do I make sure a hub is running?

- Runs within some applications
  - Tools which include hub capability often run one on startup, if not already running (Aladin, TOPCAT, Iris, ...)
- Start one externally as an application (download and run JSAMP/SAMPy)
- Start one using WebStart (JSAMP) — e.g. http://astrojs.github.com/sampjs/hub/webhub.jnlp

Rule of thumb: If you’re using a hub-capable toolkit, try starting a hub on startup if none is already running. Otherwise, don’t worry — probably someone else (another application or the user) will start one. You do not need to implement a hub to be a SAMP citizen.
JSAMP Java Toolkit/Library

- Contains:
  - Hub implementation
  - Client library
  - Diagnostic tools

- Availability:
  - Java 1.4+ (may move to 1.5)
  - One jar file (750 Kb), no external dependencies
  - Open source, unrestricted licence (Academic Free/BSD)

http://software.astrogrid.org/doc/jsamp/
JSAMP hub status GUI shows:

- Which clients are registered
- Metadata for each client
- MType subscriptions for each client (what messages they receive)
- All messages sent/received with content and responses

Availability:

- Optionally displayed by hub
- Optionally view from client using JSAMP lib (only messages to/from that client shown)
- Standalone command-line `hubmonitor` tool (only messages to/from hubmonitor shown)
JSAMP GUI

JSAMP hub status GUI shows:

- Which clients are registered
- Metadata for each client
- MType subscriptions for each client (what messages they receive)
- All messages sent/received with content and responses

Availability:

- Optionally displayed by hub
- Optionally view from client using JSAMP lib (only messages to/from that client shown)
- Standalone command-line hubmonitor tool (only messages to/from hubmonitor shown)
Log SAMP communications at HTTP, XML or RPC level

- Standard profile (desktop applications):
  - Use java system property `-Djsamp.xmlrpc.impl=xml-log|rpc-log`
  - Set it on the hub to see all communications
  - Set it on any JSAMP client to see only communications to/from that client
    (no application code changes required to switch logging on/off)

- Web profile:
  - Hub flag `-web:log http|xml|rpc`
  - Can log full HTTP communications including headers etc

Logging can be verbose, but it’s extremely useful to see exactly what
HTTP/XML/RPC is being exchanged to diagnose problems.
Other JSAMP Capabilities

Command-line tools:

- **messagesender**: command-line send tool
- **snooper**: subscribes to some/all MTypes and logs messages
- **hubtester**: hub test suite
- **calcstorm**: hub stress tester

Non-standard profile support

- Run multiple desktop hubs at once
- Set fixed hub XML-RPC endpoint URL
- Implement custom/experimental profiles
- Tweak/relax web profile authorization policy

Multi-host support

- Hub flag `-std:httplock`: use HTTP URL not local filename as lockfile
- **bridge**: join two hubs (maybe on different hosts) together
JSAMP Client Library

- Basic client use:
  - **HubConnection** object provides all hub methods
    - HubConnection c = profile.register(); do SAMP stuff; c.unregister();
    - Suitable for short-lived or send-only clients
  - **HubConnector** creates HubConnections as required
    - Watches for hubs starting and stopping
    - Manages registration, metadata and subscriptions across hub reconnections
    - Keeps track of other clients (live id→Client map)
    - Suitable for long-lived, GUI-based, send/receive clients

- GUI features:
  - Registered client icon panel
  - Subscribed client send menus
  - Hub view with client and message status display
  - Hub start/reg/unreg methods and Actions
  - MType-specific send menus
JSAMP Example 1: Send table to all subscribed clients (HubConnection)

```java
public static void main(String[] args) throws SampException {
    // Prepare message to send.
    Map params = new HashMap();
    params.put("url", args[0]);
    params.put("name", "Command-line");
    Message msg = new Message("table.load.votable", params);

    // Register with hub.
    HubConnection conn = DefaultClientProfile.getProfile().register();

    // Send message (send-and-forget to all).
    conn.notifyAll(msg);

    // Unregister.
    conn.unregister();
}
```
public static void main(String[] args) throws SampException {

    // Prepare message to send.
    Map params = new HashMap();
    params.put("url", args[0]);
    params.put("name", "Command-line");
    Message msg = new Message("table.load.votable", params);

    // Register with hub.
    HubConnection conn = DefaultClientProfile.getProfile().register();

    // Declare application metadata.
    Map meta = new HashMap();
    meta.put(Metadata.NAME_KEY, "Sender");
    meta.put("author.name", "Mark");
    conn.declareMetadata(meta);

    // Locate the first client that can load VOTables.
    Map tableClients = conn.getSubscribedClients("table.load.votable");
    String id1 = tableClients.keySet().iterator().next().toString();
    String name1 = (String) conn.getMetadata(id1).get(Metadata.NAME_KEY);

    // Send message (call and wait for response to a single client).
    System.out.println("Send to: " + name1);
    Response reply = conn.callAndWait(id1, msg, 5);
    System.out.println(reply.isOK() ? "... OK" : "... failed: " + reply.getErrInfo().getUsertxt());

    // Unregister.
    conn.unregister();
}
public static void main(String[] args) {
    final HubConnector connector = new HubConnector(DefaultClientProfile.getProfile());

    // Post a button which will broadcast a Ping message.
    postButton(new AbstractAction("Ping") {
        public void actionPerformed(ActionEvent evt) {
            try {
                connector.getConnection().notifyAll(new Message("bof.ping"));
            } catch (SampException e) { e.printStackTrace(); }
        }
    });

    // Respond to a Ping with a Pong; respond to a Pong by doing nothing.
    connector.addMessageHandler(new AbstractMessageHandler("bof.ping") {
        public Map processCall(HubConnection conn, String senderId, Message msg) throws SampException {
            conn.notify(senderId, new Message("bof.pong"));
            return null;
        }
    });
    connector.addMessageHandler(new AbstractMessageHandler("bof.pong") {
        public Map processCall(HubConnection conn, String senderId, Message msg) {
            return null;
        }
    });

    // Prepare connector with subscriptions and metadata, and set it running.
    Metadata meta = new Metadata();
    meta.setName("PingPong");
    meta.setDescriptionText("Sends and receives ping messages");
    connector.declareMetadata(meta);
    connector.declareSubscriptions(connector.computeSubscriptions());
    connector.setAutoconnect(5);
}
JavaScript SAMP library

- 1000 lines of JavaScript
- No dependencies, but optionally comes with Flash machinery for old browsers
- Initially written as proof of concept, not intended for release
- But got used; seems to work
- Listed at http://astrojs.org/
- Source and docs on GitHub http://github.com/astrojs/sampjs/
- Documentation includes live examples (SAMP-enabled web pages)
- Contributions encouraged
**sampjs Example 1: Send a table**

```html
<html>
<head><title>Send Table</title></head>
<body>
<script src="samp.js"></script>
<script>
// Broadcasts a table given a hub connection.
var send = function(connection) {
    var msg = new samp.Message("table.load.votable",
        {"url": "file:///mbt/data/table/messier.xml"});
    connection.notifyAll([msg]);
};

// Adjusts page content depending on whether the hub exists or not.
var configureSampEnabled = function(isHubRunning) {
    document.getElementById("sendButt").hidden = !isHubRunning;
};

// Arrange for document to be adjusted for presence of hub every 2 sec.
var connector = new samp.Connector("Sender");
onload = function() {
    connector.onHubAvailability(configureSampEnabled, 2000);
};
onunload = function() {
    connector.unregister();
};
</script>
<p><b>I have a table.</b></p>
<button id="sendButt" type="button" onclick="connector.runWithConnection(send)">Send It!</button>
</body>
</html>
```
sampjs Example 2: Steer (e.g.) Aladin from a web page

```html
<html>
<body>
  <script src="samp.js"></script>
  <script>
    // Set up hub registration/unregistration.
    var meta = { "samp.name": "SkyNavigator" };    
    var connector = new samp.Connector("SkyNavigator", meta);
    onload = function() { document.getElementById("regPanel").appendChild(connector.createRegButtons()); };    
    onunload = function() { connector.unregister(); };    

    // Action to send message when sliders change value.
    var posChange = function() {
      var ra = document.getElementById("RA").value;
      var dec = document.getElementById("Dec").value;
      document.getElementById("pos").innerHTML = ra + ", " + dec;
      var message = new samp.Message("coord.pointAt.sky", {"ra": ra, "dec": dec});
      connector.connection.notifyAll([message]);    
    }
  </script>

  // Page content.
  <div id="regPanel"></div>
  <div>RA: <input id="RA" type="range" onchange="posChange()" min="0" max="360" step="0.25" /></div>
  <div>Dec: <input id="Dec" type="range" onchange="posChange()" min="-90" max="90" step="0.25" /></div>
  <div>Pos: <span id="pos"></span></div>
</body>
</html>
```
Resources

SAMPI info Page: http://www.ivoa.net/samp/

SAMPI Standard: http://www.ivoa.net/Documents/latest/SAMP.html

Mailing list: apps-samp@ivoa.net

JSAMP: http://software.astrogrid.org/doc/jsamp/

sampjs: http://github.com/astrojs/sampjs/samp.js