Consuming and Creating VistA Web Services using EWD

Rob Tweed
M/Gateway Developments Ltd

http://www.mgateway.com
Twitter: @rtweed
Consuming
Web Services with EWD
EWD

• Agile web application development framework for Cache & GT.M
Web Application Frameworks

• Look after:
  – Session management
  – Security management
  – State management
  • Creating the illusion of a stateful environment on top of the stateless web architecture
Mainstream web-enablement of VistA
VistA Web Enablement

• The mainstream forget or don’t realise:
  – Mumps (Cache or GT.M) isn’t just a database
  – Has its own integrated scripting language
  – Mumps is capable of doing everything required for:
    • Parsing HTTP requests
    • Creating HTTP responses (HTML, XML, JSON ..)
    • Session, state and security management
  – Tailor-made for the web
EWD runtime Architecture

Browser

Browser

Browser

Web Server (Apache)

m_apache

xinetd – based persistent connections

MGWSI

GT.M

EWD Runtime

EWD Page routines

Your scripts

Your data

http/https
EWD runtime Architecture: WebLink

- WebLink
- Browser
- Browser
- Browser
- Web Server (Apache/IIS)
- Persistent connections
- WebLink interface
- Caché
- EWD Runtime
- EWD Weblink routines
- Your scripts
- Your data

http/https
EWD runtime Architecture: CSP

Browser

Web Server (Apache/IIS)

CSP

Persistent connections

Caché

http/https

Your scripts

Your data

EWD CSP pages
EWD runtime Architecture: Node.js

- Browser
- Browser
- Browser

Node.js process

ewdGateway Module

GT.M or Caché

EWD Runtime

stdin/stdout

Child_process connections

http/https

Your scripts

Your data

EWD routines

Your scripts

EWD routines
Consuming Web Services

• How can EWD consume external web services?
  – Require HTTP/HTTPS request to get XML response
  – Parse and process the response
EWD

- Its core is a native implementation of the XML DOM APIs
  - Uses Globals as a graph database
- EWD includes:
  - XML DOM Parser
  - XPath implementation
  - HTTP Client
EWD & Web Services

- EWD can be used to build desktop & mobile web applications that consume Web Services
  - MDWS
EWD Consuming Web Services

Caché / GT.M

MGWSI

EWD Runtime

EWD Page routines

OnBefore Render Method

EWD HTTP Client

HTTP/HTTPS Request

Web Service
EWD Consuming Web Services

Diagram:
- Caché / GT.M
- MGWSI
- EWD Runtime
- EWD Page routines
- OnBefore Render Method
- EWD HTTP Client
- EWD XML Parser
- XML DOM
- SOAP Response
- Web Service
EWD Consuming Web Services

- Caché / GT.M
- MGWSI
- EWD Runtime
- EWD Page routines
- OnBefore Render Method
- XML DOM
- EWD XPath Queries
- EWD Session
- Web Service
EWD Consuming MDWS Services

Caché / GT.M

VistA

EWD Runtime

EWD Page routines

OnBefore Render Method

EWD HTTP Client

HTTP/HTTPS Request

MDWS Interface
Demo/Proof of Concept

• Desktop Web UI using ExtJS
• 4 MDWS services
  – connect
  – login
  – getClinics
  – getPatientsByClinic
• Basic scheduler/calendar interface
• Running in the VHA Innovations Sandbox
Downside of MDWS

- Current implementation uses RPC Brokers via .Net layer
- Ultimately this architecture will not scale
With current VistA usage

- CPRS
- Broker connections
- Terminal
- Statefull connections
- GT.M or Cache
- Persistent Processes
MDWS adds to the problem

User

CPRS

Terminal

GT.M or Cache

Statefull connections

Broker connections

MDWS
.Net interface

Persistent Processes

Broker connection

One per MDWS user
Web Services should be stateless

- EWD and its supporting architecture can make this possible
Handling stateless requests

1. Request arrives from a user
Handling stateless requests

2. MGWSI establishes Socket and Session
Handling stateless requests

3. User request is processed on GT.M server
Handling stateless requests

4. Meanwhile a second user makes a request.
Handling stateless requests

Web Server

User 1
User 2
User
User

Apache/IIS
m_apache/WebLink/etc

Processing...
Processing...

Cache/GT.M Server

5. 2nd session
Established by MGWSI
Handling stateless requests

6. User 2’s request is completed.
Handling stateless requests

7. User 2’s processing stops. Connection is maintained...
Handling stateless requests

9. Session 2 is re-used for incoming Request from User 3
Handling stateless requests

9. Session 1 has finished Handling user 1's request and is now free
Handling stateless requests

9. 2\textsuperscript{nd} request from User 2 is handled by process 1
Handling stateless requests

- Each back-end GT.M/Caché process supports multiple browsers/users
  - Perhaps 10:1 to 20:1 ratio of concurrent users to physical processes
  - Highly scalable architecture
  - Extremely high performance
    - Unused back-end processes aren't shut down
    - Process startup overhead only for the 1\textsuperscript{st} request
    - gateway connects an incoming request to an existing waiting back-end process
EWD-based Web Applications

– High performance & massive scalability
  • Its stateless architecture significantly reduces the resources needed to support large numbers of users
    – Far fewer processes
    – Less memory
    – Less CPU
    – Far better utilisation of resources
Gateway scaling

**Browsers**

http over TCP/IP

Apache/IIS

Telnet over TCP/IP

Multiple servers

GT.M/Caché Server

GT.M/Caché Server

GT.M/Caché Server

Key:

- Green: Gateway Components
Gateway scaling

Key:
- Gateway Components

Gateway Components

Apache/IIS

GT.M/Caché Server

Apache/IIS

GT.M/Caché Server

GT.M/Caché Server
Creating
Web Services with EWD
EWD-based MDWS?

- EWD’s back-end can process HTTP requests or XML/SOAP request payloads
- EWD normally serves up pages or fragments:
  - HTML
  - Javascript
  - JSON
- Can serve up XML instead
  - EWD’s XML DOM APIs can be used to create the XML dynamically
EWD-based *openMDWS*

- *openMDWS* Proof of Concept
- 4 services
  - connect
  - login
  - getClinics
  - getPatientsByClinic
- Back-end uses the same VistA Mumps code used by the RPC Brokers used by MDWS versions
  - Now completely stateless using EWD
- Running on a GT.M-based WorldVistA server
openMDWS

Core Re-usable Functions

No reliance on leaking VistA variables

No leakage out of VistA variables

Full variable scoping

VistA Legacy Code
openMDWS

Core Re-usable Functions

set ok=\$\$login^myRou(.inputs,.results)

Full variable scoping

VistA
Legacy Code

inputs("accessCode")
inputs("verifyCode")
openMDWS

Core Re-usable Functions

set ok=\$login^myRou(.inputs,.results)

Full variable scoping

VistA
Legacy Code

results("DUZ")
results("DT")
Results("greeting")
openMDWS

HTTP/XML Web Service wrapper functions

set ok=\$\$login^\$myService(sessid,localCall)

Map HTTP request name/value pairs to input array

Map output array to XML-mapped local array

openMDWS Core Function
openMDWS

HTTP/XML Web Service wrapper functions

set ok=\$\$login^\$myService(sessid,localCall)

Map HTTP request name/value pairs to input array

Map output array to XML-mapped local array

openMDWS generates XML from array and dispatches As HTTP response
XML-Array Mapping

s outerTag="UserTO"
d outerTag%^zewdMDWS(.array,outerTag,sessid)
s array(outerTag,"name")=$g(results("username"))_$_c(1)_"text"
s array(outerTag,"SSN")=$g(results("SSN"))_$_c(1)_"text"
s array(outerTag,"DUZ")=$g(results("DUZ"))_$_c(1)_"text"
s array(outerTag,"siteId")=siteId_$_c(1)_"text"
s array(outerTag,"greeting")=$g(results("greeting"))_$_c(1)_"text"
d createOutput%^zewdMDWS(localCall,.array,sessid)

<UserTo>
    <name>Rob Tweed</name>
    <SSN>1234567</SSN>
    <DUZ>23</DUZ>
    <siteId>99</siteId>
    <greeting>Welcome!</greeting>
</UserTo>
EWD Consuming remote openMDWS Service

- Caché / GT.M
- EWD Runtime
- EWD Page routines
- OnBefore Render Method

- EWD HTTP Client

- HTTP/HTTPS Request
- XML response
- openMDWS Interface

- VistA
EWD Consuming local openMDWS Service

- Direct function call
- XML-mapped array returned
- No HTTP Processing overhead
- No XML Parsing overhead
openMDWS

• No additional technologies required
  – Uses native stateless architecture
  – Coding is identical on Cache & GT.M

• Open Source

• openMDWS services can be written by anyone

• XML/HTTP, JSON, RDF, etc

• One-off exercise of wrapping legacy code
  – Future-proofed all-purpose re-use
EWD and MDWS: Conclusions

• Fast and easy to create web/mobile applications that use MDWS web services

• Provides a significantly better technical architecture for implementing MDWS web services
  – Very high performance
  – Highly scalable
  – Significantly reduced resource requirements