Making Mumps Acceptable To The Mainstream

Rob Tweed
M/Gateway Developments Ltd

@rtweed
rtweed@mgateway.com
Enterprise Web Development: Javascript, NoSQL and Big Data

CAN A PHOENIX ARISE FROM THE ASHES OF MUMPS?

January 22, 2013 • by robtweed • in Uncategorized

There’s a major problem that is growing increasingly critical in the Mumps application world: where are the new generation of developers going to come from to support what is a pretty massive legacy of applications? The US Dept of Veterans' Affairs' (VA) VistA Electronic Healthcare Record (EHR) is just one of a large number of healthcare applications that was written in Mumps and is supported and maintained by a dwindling number of developers who understand and/or enjoy using Mumps. The sheer scale of VistA and its growing popularity as an Open Source EHR outside the VA means severe problems ahead if a way of
The challenge

• How to recruit and retain new developers with the expertise to:
  – exploit the new and expanding possibilities of browsers as the modern, ubiquitous UI platform
  – make VistA the healthcare Big Data visualisation platform of choice
  – make Healthcare IT as exciting and cool as the gaming and social media industries?
The Problem: Mumps

• Lots of disinformation and misconceptions about Mumps
• Mumps language considered old-fashioned, arcane and obsolete
  – “barely distinguishable from line noise”: Dan Risacher
• Regarded as something of a joke in the mainstream
The consequence

• Not seen as a sensible career move by most of the new generation of developers
  – Unlikely to want a Mumps development job
  – Unlikely to be retained very long

• Dwindling pool of Mumps skills, despite training efforts

• Significant brake on VistA uptake:
  – Seen as a risk by potential adopters:
    • Old, obsolete technology
    • Shortage of developers
So what’s to be done?
Understanding the bigger picture

- Google Alerts
- Twitter searches & dialogue
- Feedback on articles and comments
- Feedback from other communities

- Understanding the issues from the point of view of those looking at Mumps from the mainstream
VistA: its own worst enemy

• Dan Risacher: Open Source proponent
  – “barely distinguishable from line noise”
  – “just re-write it in MUMPS that doesn't suck so much”
For example

@teemuvesala
#MUMPS worse than #Perl
github.com/OSEHRA/Vista-M...
For example
VistA: Why it’s like it is

• When first written, Mumps had many constraints:
  – 8 character variable name
  – 8 character global names
  – 8 character routine names
  – 8 character labels
  – No variable scoping
  – No functions
  – Everything upper case
VistA: Why it’s like it is

• Hardware limitations at the time:
  – Very expensive
  – Very low-powered hardware
  – Very little memory
  – Developer time relatively less costly

• Code written to:
  – minimise resource usage
  – Maximise performance
  – At the expense of maintainability
Write code that doesn’t suck?

• Modern Mumps coding has none of the original limitations
• Well-written modern Mumps code can be highly readable, understandable and maintainable
But:

- **VistA coding standards: SAC Compliance**
  - Retains most of the coding limitations unnecessarily

- **Original VistA Mumps code depended on leaky, globally-scoped variables**
  - Very difficult to build new, properly-scoped code to work with leaky legacy code
  - Huge task to rewrite properly

- **Most Mumps programmers don’t understand how to write good, modern code that doesn’t suck**
The poisonous conflation

- Mumps is positioned as a language
- It’s also a database
- Mention Mumps to the mainstream and they focus on the language, not the database
- Google search for Mumps language and what do you find almost immediately?
You may not realize it, but the majority of us developers have been living a sheltered professional life. Sure, we’ve got that living disaster of a C++ application and that ridiculous interface between PHP and COBOL written by the boss, but I can assure you, that all pales in comparison to what many, less fortunate programmers have to work with each day. These programmers remain mostly forgotten, toiling away at a dead-end career maintaining ancient information systems whose ridiculously shoddy architecture is surpassed only by the tools used to create it. Bryan H lived in such a world for over two years. Specifically, he worked at a “MUMPS shop.”
Back comes the poison

Andrew Clegg @andrew_clegg
27 Mar
I think this is why no-one uses it ow.ly/jsVcb ... RT @al3xandru:
Mumps: The Proto-Database nosql.mypopescu.com/post/464079714
The reason for the poison?

• Written by guys who have had to maintain the leaky code in Mumps applications such as Epic and VistA

• They believe (and perpetuate the belief) that it’s how you have to code in the Mumps language
  – nobody told them why that code was the way it was, and that it didn’t need to be that way
  – now all modern developers believe it too!
Then we have the Misconceptions

Alan C. Viars @aviars 19 Feb
@rtweed @jeffbrandt I get the comparison and understand it widely used in legacy systems, but I wouldn’t start a new project on UNIVAC.
Expand   Reply   Retweet   Favorite   More

Alan C. Viars @aviars 19 Feb
@rtweed @jeffbrandt MongoDB is not new and shiny at this point. Very proven. Interfaces in all langs.
Expand

Alan C. Viars @aviars 19 Feb
@rtweed @jeffbrandt MUMPS problems: Lack of interfaces for most langs. No support for XML/JSON. Built before TCP/IP. Small community.
View conversation
More misconceptions

jeffbrandt @jeffbrandt
18 Feb
RT @rtweed: @aviars Why not use cobol? it works but.... limiting
How many mumps programmer? How many schools teach
MUMPS, Madison
Expand

jeffbrandt @jeffbrandt
18 Feb
RT @rtweed: @aviars I'd be interested to hear your reasoning
behind that conclusion 1960 technology, Silo data, no interfaces...
Expand

jeffbrandt @jeffbrandt
18 Feb
RT @aviars: Mumps: <- Anyone building something new, MUMPS
over MongoDB needs head... | you can drive your car w/ your feet
but Why :)
Expand

jeffbrandt @jeffbrandt
18 Feb
RT @aviars: RT @Mumps Univ NoSQL DB <- Anyone building
something new on MUMPS over MongoDB needs their head
examined. |Agree 100%
Expand
Reason

• Epic
  – I don’t know if criticisms of Epic’s interoperability are founded or not, but if they are, it’s a commercial decision by Epic, not a technical limitation of Mumps
  – However, this is naively being translated into a poisonous “truth” that Mumps has significant technical limitations

• Now being turned into a broader “truth”:
  – Mumps’ dominance in the sector is the root of all ills in Healthcare IT
    • ie it’s old, out-of-dated technology that is fundamentally flawed and needs replacing
Is MUMPS Infecting EHRs?

CareCloud’s Ahmed Mori questions the effectiveness of the MUMPS EHR programming language in healthcare.

The word ‘mumps’ has two meanings in the healthcare industry. Most popularly, mumps is a virus characterized by a painful swelling of the salivary glands, and is rather contagious.

Its other usage is an acronym for Massachusetts General Hospital Utility Multi-Programming System, a programming language that is almost
Step 1: remove the conflation

- Focus on Mumps, the database
  - It’s the database, not the language, that is the unique and powerful part of the technology
  - Assumed to be as antiquated and obsolete as the language
  - By the Mumps community focusing on the language, the database is being dismissed by the mainstream
NoSQL

• Prior to 2009, Mumps wasn’t a proper database!
  – Relational ruled
• NoSQL Movement explodes onto the scene
  – Relational fails at “Internet Scale”
  – New Models needed:
    • Key/Value (eg Redis)
    • Columnar (eg Cassandra)
    • Document (eg MongoDB, CouchDB)
    • Graph (eg Neo4j)
A Universal NoSQL Engine, Using a Tried and Tested Technology

Rob Tweed (rtweed@mgateway.com web: http://www.mgateway.com)
George James (GeorgeJ@georgejames.com web: http://www.georgejames.com)

Introduction

You wouldn't expect a programming language from the 1960s to have anything new to teach us, especially one that diverged from the mainstream around the time that Dartmouth BASIC became popular. Even more especially a programming language called MUMPS.

However, surprisingly there is one aspect of this archaic language that is still ahead of it's time. MUMPS has a pearl in its oyster called Global Persistent Variables. These are an abstraction of the B-tree structures that are normally used by MUMPS to store large volumes of data. Global Persistent Variables (usually simply referred to as "Globals") are an expressive and highly efficient way of modelling all of the common use cases that are targeted these days by NoSQL databases.
Addressing the Problem

• Need to clearly separate the language from the database
• Database has everything going for it:
  – A highly functional NoSQL database at a time when NoSQL is hugely fashionable
  – Highly scalable, high-performance database
  – Multi-facettted, not a NoSQL “one-trick pony”
• But being diminished and dismissed because of real and imagined limitations of its language
Solution: use another language?

- Both Cache and GT.M allow use of other languages, eg:
  - Java
  - .Net
  - Python
  - Ruby
  - PHP

- None have been ideal bed-fellows with Mumps
  - All have been grafted on
    - with a Mumps coding mind-set, so not natural to that language’s developers; or
    - by people who don’t really understand Mumps
  - The things that make the Mumps language special are missing in most of these languages; and/or
  - The languages can’t fully exploit the power of Mumps Globals
Key features of Mumps language

• Interpreted
• Loosely typed
• Able to be run in its own shell
• Exceptionally high performance and scalability
• Intimately integrated with the database
  – Difference between in-memory and on disk is denoted by ^
• Database is dynamic, schema-free
Mismatches

• **Java & .Net:**
  – Compiled
  – Pre-defined class-based objects

• **Python & Ruby:**
  – Good potential but don’t scale well

• **PHP:**
  – Rapidly going out of favour
  – Poor language
  – Constantly changing APIs at each release
The Ideal Solution

• A language that has similar good parts to Mumps
• Object oriented, but dynamic objects
• High performance and scalability
• Highly popular language that is likely to stay that way
• Similar intimate integration with the Mumps database
JavaScript: ticks all the boxes

• A language that has similar good parts to Mumps
• Object oriented, but dynamic objects
• High performance and scalability
• Highly popular language
  – Games, 3-d, social media industries
  – It’s not going to disappear
  – It’s the language new developers are learning
One Box Unticked

- Similar intimate integration with the Mumps database as the Mumps language
  _ ^

- …but read on
Javascript instead of Mumps

• How to get a new generation of developers for VistA (and other Mumps-based applications)?
  – Training them in Mumps language?
    • can’t recruit and retain for reasons explained so far
  – Access Mumps database from JavaScript, but
    • in a way that they “get” Mumps globals
    • In a way that combines the power of Javascript and the power of Globals, that is natural to a Javascript developer
Node.js

- Server-side implementation of Javascript
- Started as an open-source project by Ryan Dhal
- Uses Google’s V8 Javascript engine
- Designed for network programming
  - Event-driven
  - Normally asynchronous I/O
- Very high performance and scalability
- Runs on Linux, Windows & Mac OS X
Node.js: LinkedIn

After evaluation, some of the advantages of Node.js were:

- Much better performance and lower memory overhead than other tested options, running up to 20x faster in some scenarios
- Programmers could leverage their JavaScript skills.
- Frontend and backend mobile teams could be combined into a single unit.
- Servers were cut to 3 from 30. Enough headroom remains to handle 10x current levels of resource utilization.

Node.js

• Many open-source modules, in particular:
  – built-in Web server
  – Socket.io: web-sockets
InterSystems Node.js Interface

• Initially introduced for their free Globals database
  – Used their very low-level interface into the core global database engine
  – Written by Chris Munt (M/Gateway Developments)
  – Very high performance

• Ported to Caché (2012.2)
  – Official standard interface
  – function() API added to invoke Mumps functions from Javascript
Equivalent for GT.M?

• David Wicksell: NodeM
  – https://github.com/dlwicksell/nodem
NodeM

- API-compatible reverse-engineer of InterSystems’ Node.js interface
  - Behaves identically
Node.js Interface

Node.js

C++ Interface

ISC Interface

NodeM

GT.M / Cache

Callin Interface

• In-process
• Synchronous
Example of how it’s used

```javascript
var globals = require('mumps');
var db = new globals.Gtm();
db.open();

var node = {global: 'zewd', subscripts: ['config', 'RootUrl']};
var x = db.get(node).data;

Same as: set x=^zewd("config","RootUrl")
```
OK, but too low-level

- Doesn’t make natural sense to a JavaScript developer
- Need to understand Mumps Global storage to use it in this way, using the core APIs of the interface
- ie, lacking the intimacy of ^ in the Mumps language
- ... but this has now been addressed
https://github.com/robtweed/ewdGateway2

• Node.js gateway for EWD:
  – Alternative to using Apache or IIS + Cache/GT.M gateway (CSP, WebLink, m_apache)
  – Works identically on both GT.M and Cache
    • uses InterSystems interface (Cache) or NodeM (GT.M)
  – Apache 2 licensed
  – Web-sockets made simple and easy
  – EWD applications written entirely in Javascript

• Dynamic OO abstraction of Globals
  – Layered on top of the low-level interface APIs
  – Mumps Globals made meaningful to JavaScript developers
ewdGateway2 Architecture

Node.js

Web Server (HTTP/HTTPS)

Queued Request Dispatcher

Queue

Node.js

Cache or GT.M

Node.js

Cache or GT.M

Node.js

Cache or GT.M

Worker Child Processes

Fully asynchronous
Dynamic OO Global Abstraction

• Underlying concept: any object and its properties can be easily modelled as a corresponding Mumps Global node, eg:

  – patient.address.town
  – patient.address.zip
  – patient.name

  – ^patient(id,"address","town")=“New York”
  – ^patient(id,"address","zip")=123456
  – ^patient(id,"name")=“John Smith”
Dynamic OO Global Abstraction

- Property === Subscript
- But globals are dynamically-defined
- No pre-determined subscripts
- No schema
- Javascript’s objects are dynamic
Dynamic OO Global Abstraction

- **GlobalNode** Javascript Object
  - Represents a physical global node
  - May or may not physically exist on disk
  - Has a set of properties and methods, eg:
    - `_value` (read/write)
    - `_delete()`
    - `_increment()`
    - `_forEach()`
    - `_getProperty()` or `$(())`
Dynamic OO Global Abstraction

- GlobalNode(globalName, subscriptArray)
Dynamic OO Global Abstraction

- GlobalNode(globalName, subscriptArray)

```javascript
var patient = new GlobalNode('patient', [1234567]);

Sets a pointer to ^patient(1234567)
```
Dynamic OO Global Abstraction

```javascript
var patient = new GlobalNode('patient',[1234567]);

var name = patient.$('name')._value;
```

Equivalent to:
```
set name=^patient(1234567,"name")
```
Dynamic OO Global Abstraction

var name = patient.$('name')._value;

• $() returns a GlobalNode object for the specified property (subscript)
• Also instantiates the named property in the parent GlobalNode
Dynamic OO Global Abstraction

var name = patient.$('name')._value;

• Now name is instantiated as a property of patient, so $(\cdot)$ is no longer needed:

patient.name._value = "David Smith";

– writes the value directly to the Global, but appears to be a standard JavaScript object

– JavaScript equivalent of ^
Dynamic OO Global Abstraction

var address = patient.$(‘address’);

– address is a GlobalNode object
– Pointer to ^patient(123456,”address”)

• var town = address.$(‘town’)._value;
Dynamic OO Global Abstraction

Could have also done this instead:

```javascript
var town = patient.$('address').$('town')._value;
```

- Chain as much as you require
- Each `($) instantiates that property in parent, so now we can do:

```javascript
var zip = patient.address.zip._value;
```
Javascript + Mumps Globals

• Similar level of intimacy as Mumps
• Globals are just manipulated as if they were normal Javascript objects

• JSON to/from Globals:
  – Mumps as a document database
    • Alternative to MongoDB
Document Storage

• GlobalNode objects have two methods:
  – _setDocument()
    • Maps a JSON document into an equivalent global
  – getDocument()
    • Maps a GlobalNode’s sub-tree into a JSON object
var gridData = [
  0: {col1: 1, col2: 1, name: ‘rec1’},
  1: {col1: 4, col2: 4, name: ‘rec4’}
];

session.$('newGridData')._setDocument(gridData);
JavaScript Document Storage

```javascript
var gridData = [
    0: {col1: 1, col2: 1, name: 'rec1'},
    1: {col1: 4, col2: 4, name: 'rec4'},
];

session.$('newGridData')._setDocument(gridData);
```

%^zewdSession("session",4020,"newGridData",0,"col1")=1
%^zewdSession("session",4020,"newGridData",0,"col2")=1
%^zewdSession("session",4020,"newGridData",0,"name")="rec1"
%^zewdSession("session",4020,"newGridData",1,"col1")=4
%^zewdSession("session",4020,"newGridData",1,"col2")=4
%^zewdSession("session",4020,"newGridData",1,"name")="rec4"
```
JavaScript Document Retrieval

```javascript
var gridData = session.$('newGridData')._getDocument;

console.log(gridData);

[
  0: {col1: 1, col2: 1, name: 'rec1'},
  1: {col1: 4, col2: 4, name: 'rec4'}
]
```

Note: JSON not generated in Mumps code - mapping via JavaScript
Example

• ONC / Mitre JSON data from Cypress Server:
  – Storage as JSON document in GT.M
  – Manipulation from within Javascript as objects

• Cypress uses MongoDB
• A Mumps database is a much more powerful document database
Mumps as a Native JSON Database

```javascript
var patient = new ewd.GlobalNode('patient', [123456]);
var document = {
    "name": "John Doe",
    "city": "New York",
    "treatments": {
        "surgery": ["appendectomy", "biopsy"],
        "radiation": ["gamma", "x-rays"],
        "physiotherapy": ["knee", "shoulder"]
    }
};
patient._setDocument(document);
var physioArray = patient.treatments.physiotherapy._getDocument();
physioArray.push('ankle');
physioArray.push('wrist');
physioArray.push('neck');
patient.treatments.physiotherapy._setDocument(physioArray);
```
Mumps as a Native JSON Database

^patient(123456,"city")="New York"
^patient(123456,"name")="John Doe"
^patient(123456,"treatments","physiotherapy",0)="knee"
^patient(123456,"treatments","physiotherapy",1)="shoulder"
^patient(123456,"treatments","physiotherapy",2)="ankle"
^patient(123456,"treatments","physiotherapy",3)="wrist"
^patient(123456,"treatments","physiotherapy",4)="neck"
^patient(123456,"treatments","radiation",0)="gamma"
^patient(123456,"treatments","radiation",1)="x-rays"
^patient(123456,"treatments","surgeries",0)="apendicectomy"
^patient(123456,"treatments","surgeries",1)="biopsy"
Meaningful Mumps

• The Mumps database
• Presented in terms that are meaningful to a JavaScript developer

• No need for the Mumps language to be mentioned
  – Yet Mumps programming against the same Globals can continue if required
Interested in finding out more?

Node.js integrates with M: a tutorial, part one
Posted 2 Apr 2013 by Luis Ibanez

Average: ★★★★★ (2 votes)

Luis Ibanez is a Technical Lead at Kitware Inc., and is Director of Software and IP at the Open Source EHR (OSEHRA). At Kitware he is closely involved in the development of open source software for medical imaging applications, in particular, working in the Insight Toolkit (ITK). Luis is a strong supporter of Access, and one of the editors of the Insight Journal, a journal that enforces the verification of reproducibility in collaboration with other instructors,
What about the legacy code?

- Wrapper and abstract it
- Hide it
- Make it accessible to all other technologies
  - openMDWS
  - Legacy Mumps functions exposed as JavaScript functions

- Move away from the “how” to the “what”
What about the legacy code?

• Use development technologies that avoid the need to write much application code at all
  – less code = easier, less-costly maintenance
  – Web Applications
    • Most of the coding is for the UI
    • Most UI complexity can be handled by frameworks
    • Back-end code is straightforward:
      – Getting data from the database
      – Validating and saving data
    • Running pre-built APIs & RPCs (eg openMDWS)
Hybrid environment

• Legacy Mumps code, buried in the core, abstracted as services or JavaScript functions
  – Natural fit with VSA

• Small number of Mumps developers needed for ongoing support
  – we can continue to recruit Mumps developers in sufficient small numbers for this

• New back-end and application code in Javascript
  – The new generation of developers, and lots of them
  – Exploit the exciting growing capabilities of the browser
    • make Healthcare IT as exciting as Games/Social Media industries
    • a lot more worthwhile!
Making it palatable?

• Never attempt to defend the language: you will *always* fail and end up looking foolish!
  – *A Case of the Mumps* is their poisonous ammunition

• Just admit that the language is obsolete
  – agree with them and they have no ammunition left

• Focus attention on Mumps as a multi-faceted, high-performance NoSQL database

• Demonstrate its unique power and capabilities to a new generation of developers

• Now you can make the critics look foolish:
  – “You’re supposedly a database expert, yet you’re criticising Mumps as a database on the basis of an optional *language*?”
Mumps: Native JSON Database

• JavaScript as the primary language
• Persistent JavaScript Objects
• Native JSON Storage
• Significantly better performance than MongoDB
• Much more flexibility than MongoDB
  – MongoDB “unit of storage” is a complete JSON document
  – Mumps allows sub-document access
Mumps Database Renaissance

• Big Data Visualisation in the browser
• Big Data-capable multi-facetted NoSQL Database
• The NoSQL database of choice for JavaScript developers
  – Document/ Native JSON database
  – Key/Value store
  – Columnar database
  – Graph database
Mumps, the database:

Acceptable to the mainstream

Blog: http://robtweed.wordpress.com
Twitter: @rtweed