Extending the DDS Global Data Space to the Web

- Problem Statement
  - Large Asian bank operating in several countries
  - Expanding its country-specific Financial Trading Services to >10K users, using desktop and mobile devices
  - Requirements:
    - Low cost of operation/distribution
    - Rapid response to changing competitive marketplace
    - End-to-end control
  - Phases:
    I. Market data distribution
    II. Secure mobile trading
Node.js  http://nodejs.org

- Server-side software system designed for writing scalable Internet applications, notably web servers.
- Uses Google's V8 JavaScript engine, the libuv portable event loop library, and a core library written in JavaScript.
- Uses event-driven, asynchronous I/O to minimize overhead and maximize scalability.
- Portable to multiple operating systems.
- Applications are written JavaScript, some also include compiled (C/C++) modules.
- OCI offers a training class in Node.js

OpenDDS  http://opendds.org

- Implementation of the OMG DDS* 1.2 and DDS-RTPS 2.1 specifications
  - *DCPS layer with all optional profiles
- Open source, permissive license, public source repo
- Core libraries written in C++, includes Java API
- Configurable transports:
  - TCP, RTPS, UDP-unicast, UDP-multicast, shared memory
- Configurable discovery: centralized, peer-to-peer (RTPS)
- Modeling tool based on Eclipse with code generation
- OCI offers training classes in both OpenDDS programming with C++ and the OpenDDS Modeling SDK
Solution Architecture

QuickFIX
Feed Handler
OpenDDS

QuickFAST
Feed Handler
OpenDDS

OpenDDS
Node Modules
Web Application

Node.js process

HTTP / WebSocket
HTML5 / Dojo

Demo Application

- OpenDDS process (C++) publishes sample market data
- Web server (Node.js) subscribes to the market data
- Web client receives dynamic updates using WebSockets (Socket.IO module for Node)

```c
module MarketData {

#pragma DCPS_DATA_TYPE "MarketData::SymbolDefinition"
#pragma DCPS_DATA_KEY "MarketData::SymbolDefinition symbol"
struct SymbolDefinition {
    string symbol;
    string description;
    float closing_price;
};

#pragma DCPS_DATA_TYPE "MarketData::Trade"
#pragma DCPS_DATA_KEY "MarketData::Trade symbol"
struct Trade {
    string symbol;
    float price;
    unsigned long quantity;
};

};
```
OpenDDS Module for Node.js

• NPM Module
  – Our module ("npm install opendds") is open source and uses open source libraries and tools, wrapping the C++ OpenDDS code.

• API Design
  – Minimal API: only what’s required to subscribe to DDS data, including optional QoS and Content-Filtering
  – Each data sample is delivered via a callback function which runs on Node’s event loop.

• IDL / JavaScript mini-mapping (see next slide)

Mapping IDL to JavaScript

• The current implementation only needs to translate IDL structs to JavaScript objects, not the other way around
  – Including all data types that structs used with DDS can contain

• IDL Boolean is JavaScript Boolean
• Mapped to JavaScript Number:
  – IDL octet, integral and floating point types
• Mapped to JavaScript String:
  – IDL char, wchar, string, wstring, enum
• Mapped to JavaScript Object: IDL struct, union*
• Mapped to JavaScript Array: IDL sequence, array*

* => not yet implemented
Quality of Service and Content-Filtered Topic

- **DURABILITY QoS Policy**
  - Used by the “Symbols” topic (TRANSIENT_LOCAL kind)
  - Publishing side writes each instance just once, most likely before subscribing side (Node.js) has even started
- **Other QoS Policies are left at default values in this demo**
- **Content-Filtered Topic**
  - Used by the “Trades” topic
  - Only trades involving symbols which the user has requested (using the Web interface) pass the filter
  - Other trades are not sent on the network
    - Due to publisher-side evaluation of the Content-Filtered Topic

Scalability

- **Additional Feed Handlers**
  - New market data sources
  - Each feed handler could be responsible for a subset of symbols (per-channel, or segmented)
  - Redundancy using DDS OWNERSHIP and LIVELINESS QoS
- **Additional Web Servers**
  - Use Node’s “cluster” module to start multiple instances on the same host (in order to make use of multi-core systems)
  - Use web server load balancing to scale to multiple hosts
Future Directions

- OpenDDS Module for Node.js could eventually gain support for:
  - IDL unions and arrays
  - Current features (subscribe) with the full DDS DCPS API:
    - Topic, MultiTopic, ContentFilteredTopic (dynamic parameter changes)
    - Subscriber, DataReader
    - Listeners/Conditions
    - Built-In Topics
  - Ability to publish data samples using a simplified API
    - Example use case: web server load balancing and stats
  - Publication with the full DDS DCPS API:
    - Publisher, DataWriter

For More Information

- OpenDDS: [http://opendds.org](http://opendds.org)
- Node.js: [http://nodejs.org](http://nodejs.org)
- Financial Domain
  - FIX: [http://fixprotocol.org](http://fixprotocol.org)
  - QuickFIX: [http://quickfixengine.org](http://quickfixengine.org)
  - FAST: [http://fixprotocol.org/fast](http://fixprotocol.org/fast)
  - QuickFAST: [https://code.google.com/p/quickfast](https://code.google.com/p/quickfast) (an OCI project)
  - Liquibook: [https://github.com/objectcomputing/liquibook](https://github.com/objectcomputing/liquibook)
- Adam Mitz: mitza@ociweb.com