D Adoption Case Study

Andy Smith
andyrsmith@gmail.com
Outline

• Quick Adoption History
• Business overview
• Software requirement
• Where D addresses these
• Event Sourcing Description
• Architecture
D Adoption History
Business Overview

- Group within a Fund management firm
  - Accountable at Group, Firm, & Regulatory Authority
- Technology function to support business
  - Market Data Recording
  - Trading Frameworks
    - Interact directly with brokers
  - Introduce new data sources
  - Simulation / Analysis tools
- Competitive / Time pressure environment
Requirements ...

- Correctness
- Testability
- Reliability
- Modifiable
- Productive
- Performant
What makes D a good citizen

- Fast development iterations (DMD)
- Built-in unit-tests
- C-like Syntax
- Posix Availability
- Good Standard Library
- Easy to modify
- (So far) no nasty language surprises
Phobos Goodness

• Time savers
  – CommandLine option parsing
  – JSON Parsing
  – DateTime module
  – Atomics
  – Bitop
  – CSV. Inescapable in finance!
Event Sourcing
Event Sourcing

• Represent Everything as stream of events
  – Ordered
  – Persisted

• Examples of Events
  – Orders
  – Executions
  – User actions
    • Button Press
    • Numeric Field change
  – Heartbeats
System is a pure State Function

\[ (S_{n+1}, O_{n+1}) = f(S_n, I_n) \]

- Inputs (State, Input Event) 2-tuple
- Outputs (State, Output Event) 2-tuple
System is a 'fold-left' over events

\[ S_1 = f(S_0, E_0) \]
\[ S_2 = f(S_1, E_1) \]
\[ S_3 = f(S_2, E_2) \]

\[ S_3 = f(f(S_1, E_1), E_2) \]
\[ S_3 = f(f(f(S_0, E_0), E_1), E_2) \]

i.e. a pure function of initial + input events

\[ S_n = f_2(S_0, E_0, E_1, E_2 \ldots E_{n-1}, E_n) \]
Purity?
If we are 'pure' we get ...

- **Determinism**
  - Same result every time. Repeatable behaviour
  - Regression testing
  - Post-Trade analysis
  - Auditable

- **Resilience**
  - Copy events off to another box for standby system

- **Parallelizable**
But ...
But ... lied a bit :-(

- Pure functional version performs badly
  - Allocate new state for every event
  - Even with persistent structures not good enough
- Imperative code with state mutation much faster
  - That's what we **ACTUALLY** have
- However ...
  - Same input still produces same outputs
  - Mutation still okay
  - c.f. Clojure 'transients'
  - Lose ability to cache intermediate state objects
Architecture
Layered Separation of Concerns

Concurrency
Persistence
Event Dispatch

Business Logic
Inner layer – Business Logic

- Simple vanilla callback code
- Handles
  - Order Logic
  - Stats calculations
  - Profit/Loss calculations
- Single threaded
  - Cache friendly
- Gets time from the outer layer
Outer Layer (the D parts!)

• Handles
  – Concurrency
  – Persistence
  – Event Delivery

• Implemented in terms of
  – Stream consumers
  – Event Loop (Live or Simulation)
    • Decides (and persists) event firing order
Live Event Sources

- Market
- From Broker
- Carbon Units
- MD
- Exec
- Loop
- Order
- Cons
- To Broker

<table>
<thead>
<tr>
<th>MD Index</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>12321</td>
<td>1</td>
</tr>
<tr>
<td>12432</td>
<td>1</td>
</tr>
<tr>
<td>12543</td>
<td>2</td>
</tr>
</tbody>
</table>
Simulation

- Single Threaded
- Streams are 'delay lines' not concurrency
- Only one 'fundamental' source, the Market Data

Diagram:
- MD → SimLoop
- SimLoop → Exec
- SimLoop → Order
- Exec → Simulator
- Order → Simulator
Where is D used?
Where is D used?
Why there?

- Require C-linkage for optimal API usage
- Alternatives?
  - JNI (homegrown)
  - JNI (vendor)
  - JNA (maybe)
  - C / C++
- D with C-linkage + SHM kills two birds with one stone
- Intention was to rewrite in C/C++ (probably C++11)
  - But stuck with D
Stream == ???
Stream Candidates
(Contiguous vs Circular Array)

• Credit
  - Martin Thompson
  - Peter Lawrey

• Contiguous
  - Simple, mmap required memory segment
  - Not so simple in Java-land mmap takes integer :
  - Numpy friendly
  - Page Faults
  - Bounded, can run out!!

• Circular Array
  - Less simple
  - Cache friendly
  - Need journal of retired entries
Stream Implementation
with circular array

Head
(readers)

Tail
(writers)
**MMFile Layout**

<table>
<thead>
<tr>
<th>ReserveTail</th>
<th>CommitTail</th>
<th>Head1</th>
<th>Head2</th>
<th>HeadN</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>N * T.sizeof</td>
</tr>
</tbody>
</table>
Atomic 'incrementAndGet'
(courtesy of mnovak)

- We need 'LOCK XADD' ASM instruction on X86_64 for wait free operation in a MPMC queue
- AKA 'UNSAFE.incrementAndGet' in Java 7+ land
- Unavailable in Phobos (as of writing), but not a problem...

```cpp
version (X86_64)
{
  T atomicOp(string s : "+=", T)(ref shared T val, T mod) pure nothrow @nogc
  if (traits(isIntegral, T))
  {
    T oval = void;
    static if (T.sizeof == 8)
    {
      asm pure nothrow @nogc
      {
        mov RAX, mod;
        mov RDX, val;
        lock;
        xadd[RDX], RAX;
        mov oval, RAX;
      }
      return oval + mod;
    }
  }
}
```
struct ManyToManyWriter( T, int Consumers, int Capacity ) if (isPow2(Capacity)) {
    mixin ManyToManyCommon!(T, Consumers, Capacity);

    long cacheTail;
    long cacheHead;

    bool reserved = false;
    long reservedPos = long.max;

    T* reserve() {
        enforce(!reserved);
        reservedPos = atomicOp!"+="(header.reserveTail.value, 1) - 1;
        while ( reservedPos - cacheHead == Capacity ) {
            cacheHead = getHead();
        };
        reserved = true;
        return &data[indexOf(reservedPos)];
    };

    void commit() {
        enforce(reserved);
        while ( !cas( &header.commitTail.value, reservedPos, reservedPos + 1 ) ) {};
        cacheTail = reservedPos + 1;
        reserved = false;
        reservedPos = long.max;
    };
}
Multiple Heads

```c
mixin template MultipleHeads() {
  long getHead() {
    long getMinHead( uint X )( long prev ) {
      static if ( X == 0 ) {
        return prev;
      } else {
        return getMinHead!(X-1)( min( prev, nthHead!X ) );
      };
    },
    long nthHead(uint X)() if (X>=1 && X<=Consumers) { 
      return atomicLoad!(MemoryOrder.acq)( header.heads[X-1].value );
    }
    return getMinHead!(Consumers-1)( nthHead!Consumers );
  }
};
A D solution to false sharing

```cpp
template Padded(T) {
    const postAmbleLength = 128 - 4 * long.sizeof - T.sizeof;
    struct Padded {
        private long[4] preamble;
        T value;
        private byte[postAmbleLength] postAmble;

        alias value this;
    };
}
```

- Java alternatives not very attractive
Market Data Consumption

API → callbacks → D → populate struct → MD
D Market Data Message

```
align(1) struct BidAskChange {
    int   messageType;
    int   securityId;
    long  time_stamp;
    long  bidQty;
    double bidPrice;
    long  askQty;
    double askPrice;
};

#pragma(msg, "Size is ", BidAskChange.sizeof);
static assert (BidAskChange.sizeof == 48);
```
import sun.misc.Unsafe;

public class BidAskChange {
    long address;

    Unsafe unsafe;

    private static final int MESSAGE_ID_OFFSET = 0;
    private static final int SECURITY_ID_OFFSET = MESSAGE_ID_OFFSET + INT_SIZE;
    private static final int TIMESTAMP_OFFSET = SECURITY_ID_OFFSET + INT_SIZE;
    private static final int BID_VOLUME_OFFSET = TIMESTAMP_OFFSET + LONG_SIZE;
    private static final int BID_PRICE_OFFSET = BID_VOLUME_OFFSET + LONG_SIZE;

    public int getSecurityId() {
        return unsafe.getInt( null, address + SECURITY_ID_OFFSET);
    }

    public long getTimeStamp() {
        return unsafe.getLong( null, address + TIMESTAMP_OFFSET);
    }

    public double getBidPrice() {
        return unsafe.getDouble( null, address + BID_PRICE_OFFSET);
    }
}

Reading Structs in Java
void dumpType(T, string member)() {
    auto val = T.init;
    auto sizeof_ = __traits(getMember, val, member).sizeof;
    auto alignOf = __traits(getMember, val, member).alignof;
    auto offsetOf = __traits(getMember, val, member).offsetof;
    auto typeof_ = typeof(__traits(getMember, val, member)).typeof;
    printf("%20s %4s align=%s sizeof=%20s offset=%s",
        member, sizeof_, alignOf, sizeof_, offsetOf);
};

void dumpInfo(T)() {
    foreach(member; __traits(derivedMembers, T)) {
        dumpType!(T, member);
    }
};

void main() {
    dumpInfo!BidAskChange;
};
Output...

Size is 48LU
Size is 48LU
Size is 48LU

messageType 4 align=4 stringof= int offset=0
securityId 4 align=4 stringof= int offset=4
timeStamp 8 align=8 stringof= long offset=8
  bidQty 8 align=8 stringof= long offset=16
  bidPrice 8 align=8 stringof= double offset=24
  askQty 8 align=8 stringof= long offset=32
  askPrice 8 align=8 stringof= double offset=40

• Enough info to generate the java reader code at compile time
Electronic Trading
Trading

genenerate
parse
d_gateway
call
cb
API
Trading API

- Relatively straightforward
- Process performs two tasks
  - Convert outbound structs to strings (main thread)
  - Convert inbound strings to structs (cb thread)
- One thread dedicated to each task
  - No contention/locking
FIX Protocol

• 'Human Readable ?'

  8=FIX.4.29=17835=849=PHLX56=PERS52=20071123-05:30:00.00011=ATOMNOCCC999090020=3150=E39=E55=MSFT167=CS54=138=1540=244=1558=PHLX EQUITY TESTING59=047=C32=031=0151=1514=06=010=128

• Warty Protocol
  – Conflates OSI session + application layers in ugly ways
  – Compare with
    • MIDI
    • Military Protocols
    • Native exchange

• Parsing / Generation done with old school C-style string processing
Conclusion

- D is very useful addition to toolbox
- Adoption was worth it
- Project completed faster than could have with C / C++
- Has a definite niche in finance
Q?