μService and Scribble
aka
Scribble @ ThoughtWorks

Steve Ross-Talbot
stalbot@thoughtworks.com
(With thanks to Ray Hu)
September 2016
We propose that creating a practice within TW to concentrate on what we will call “Structural Engineering” will provide a focus to concentrate and leverage our experience in this area.

There are five constituent elements:

- Education (Theory, Practices and Research)
- Tools (Architectural Simulation Laboratories)
- Consultancy (offerings for both clients and internal teams)
- Collaboration (with external organisations e.g. Universities)
- TW Logistics (people and cash)
The world of programming

There is an increasing chance of getting it all so terribly wrong

We test a lot

We try to speed up the lifecycle of development so we can make mistakes earlier

We talk a lot (but never enough)
The world of programming

Multiparty session types offer us a way of understanding our digital world better. It offers a way for us to better identify what is good and what is bad as complexity increases. It does this by uncovering structure, the structure of communication based on observable behavior in a distributed plain. Hence structured engineering @ TW!
Multi-party session types & Scribble

Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

Formalisation of W3C WS-CDL [ESOP'07]

Scribble at π² Technology

Multi-party session types

- Global Types and End Point Projection (Abstract Choreography)
  - Potential errors:
    - Communication mismatch: e.g. receiver is sent an unexpected message
    - Protocol violation: executed interaction does not follow the protocol
    - Deadlock: e.g. all endpoints blocked on input

---

Scribble Protocol

- "Scribbling is necessary for architects, either physical or computing, since all great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, somehow it starts to urge you in your mind, in so small a voice but how persistent it is, at that point you start scribbling" - Kohei Honda 2007

- Basic example:

```plaintext
protocol HelloWorld {
    role You, World;
    Hello from You to World;
}
```
μService’s

A Micro service is normally part of a suite of independently deployable services that support a specific business goal using simple, well-defined interfaces to communicate with each other. Typically they are small and have language-agnostic APIs.

“throw-it-away” challenges our conventional IT notion of value and encourages a new form of re-use that is based on thinking and innovation.
μService’s and Serverless Architectures

A μService’s is the leaven bread of serverless computing. Serverless computing is the new Cloud/DevOps that incorporates fast instantiation into a cloud of executable containers each able to execute the bundle of behaviors that a set of μService’s exhibit.

The problem I have found is that very few people understand what a choreography really is, let alone how it can be described and used. Rather a choreography is a loose description or a way of doing things. A style if you will.

Whereas we know differently!!
What we are trying to do

To put meat on the bone of choreography by proving out a tools chain for its use
Behavioral Docking

Containers vs. VMs

Why Docker

Docker Deployment

EasyFSM

Scribble

Business Logic jar

Scribble description
μService’s as FSM’s

A real world example
μService’s as FSM’s

The scribble

```
explicit global protocol PartnershipSupplier ( role loginsvc, requestor, authorisersvc, filtersvc, suppliersvc, contractsvc)
{
    connect requestor to loginsvc;
    login(username, password) from requestor to loginsvc;
    choice at loginsvc {
        loginFailure() from loginsvc to requestor;
        disconnect requestor and loginsvc;
    } or {
        loginSuccess(uuid) from loginsvc to requestor;
        connect requestor to authorisersvc;
        connect authorisersvc to filtersvc;
        do Main(requestor, authorisersvc, filtersvc, suppliersvc, contractsvc);
    }
}
```
µService’s as FSM’s

The scribble

aux global protocol Main ( role requestor, authorisersvc, filtersvc, suppliersvc, contractsvc )
{
choice at requestor {
    // GET SUPPLIER INFO
    getSuppliers(uuid) from requestor to authorisersvc;
    do SuppInfo(requestor, authorisersvc, filtersvc, suppliersvc);
} or {
    // GET CONTRACT INFO
    getContracts(uuid) from requestor to authorisersvc;
    do ContractInfo(requestor, authorisersvc, filtersvc, contractsvc);
}
do Main(requestor, authorisersvc, filtersvc, suppliersvc, contractsvc);
}
μService’s as FSM’s

The scribble

aux global protocol SuppInfo ( role requestor, authorisersvc, filtersvc, suppliersvc )
{
    choice at authorisersvc {  // DENIED
        deny() from authorisersvc to requestor;
        exit() from authorisersvc to filtersvc;
    } or {
        // PREPARE FILTERED SUPPLIER INFO FOR REQUESTOR
        connect authorisersvc to suppliersvc;
        getsuppliers(uuid) from authorisersvc to suppliersvc;
        getsuppliersRtn(supplierdetails) from suppliersvc to authorisersvc;
        do FilterInfo <filterSupplier(usercontext, filters, supplierdetails)> (authorisersvc, filtersvc);
        disconnect authorisersvc and suppliersvc;
        getSuppliersRtn() from authorisersvc to requestor;
    }
}
μService’s as FSM’s

The scribble

aux global protocol ContractInfo ( role requestor, authorisersvc, filtersvc, contractsvc ) {
  choice at authorisersvc {
    deny() from authorisersvc to requestor;
    exit() from authorisersvc to filtersvc;
  }
  or {
    connect authorisersvc to contractsvc;
    getContracts(uuid) from authorisersvc to contractsvc;
    getContractsRtn(contractdetails) from contractsvc to authorisersvc;
    do FilterInfo <filterContract(usercontext, filters, contractdetails)>
      (authorisersvc, filtersvc);
    disconnect authorisersvc and contractsvc;
    contracts() from authorisersvc to requestor;
  }
}

aux global protocol FilterInfo < sig Query > ( role authorisersvc, filtersvc ) {
  Query from authorisersvc to filtersvc;
  filtered() from filtersvc to authorisersvc;
}
μService’s as FSM’s

digraph G {
  compound = true;
  "70" [ label="70: COMPLETED" ];
  "70" -> "72" [ label="requestor -> MakeNewServerSideConnection()" ];
  "72" [ label="72: ";]
  "72" -> "73" [ label="filtersvc -> MakeNewClientSideConnection()" ];
  "73" [ label="73: COMPLETED" ];
  "73" -> "74" [ label="requestor Receive getsuppliers(uuid)" ];
  "74" [ label="74: COMPLETED" ];
  "74" -> "75" [ label="requestor Send deny()" ];
  "75" [ label="75: ";]
  "75" -> "73" [ label="filtersvc Send end()" ];
  "74" -> "76" [ label="suppliersvc -> MakeNewClientSideConnection()" ];
  "76" [ label="76: ";]
  "76" -> "77" [ label="suppliersvc Send getsuppliers()" ];
  "77" [ label="77: ";]
  "77" -> "78" [ label="suppliersvc Receive suppliers()" ];
  "78" [ label="78: COMPLETED" ];
  "78" -> "79" [ label="filtersvc Send filterSupplier(usercontext, filters, supplierdetails)" ];
}
μService’s as EasyFSM’s

EasyFSM Configuration file

FINITE STATE MACHINE
In dot notation

```xml
<STATE id = "STATE_START">
  <MESSAGE id = "roleName="requestor"" action = "connectionRequestFrom" nextState = "STATE_STARTED" />
</STATE>

<STATE id = "STATE_STARTED">
  <MESSAGE id = "roleName="filtersvc"" action = "connectionRequestTo" nextState = "STATE_CONNECTION_REQUEST_TO_filtersvc_OBTAINED" />
</STATE>

<STATE id = "STATE_CONNECTION_REQUEST_TO_filtersvc_OBTAINED">
  <MESSAGE id = "roleName="requestor" messageType="getSuppliers(uuid)" action = "receiveMessage" nextState = "STATE_receiveMessage_REVEIVED_FROM_requestor_PROVIDING_getSuppliers(uuid)" />
  <MESSAGE id = "roleName="requestor" messageType="getContracts(uuid)" action = "receiveMessage" nextState = "STATE_receiveMessage_REVEIVED_FROM_requestor_PROVIDING_getContracts(uuid)" />
</STATE>

<STATE id = "STATE_receiveMessage_REVEIVED_FROM_requestor_PROVIDING_getSuppliers(uuid)">
  <MESSAGE id = "roleName="requestor" messageType="deny()" action = "sendMessage" nextState = "STATE_sendMessage_SENT_TO_requestor_USING_deny()" />
  <MESSAGE id = "roleName="suppliersvc"" action = "connectionRequestTo" nextState = "STATE_CONNECTION_REQUEST_TO_suppliersvc_OBTAINED" />
</STATE>
```

........
μService’s as EasyFSM’s

```xml
<STATE id="STATE_CONNECTION_REQUEST_TO_filtersvc_OBTAINED">
    <MESSAGE id="roleName="requestor" messageType="getSuppliers(uuid)"">
        action = "com.thoughtworks.org.receiveMessage"
        nextState = "STATE_receiveMessage_RECEIVED_FROM_requestor_PROVIDING_getSuppliers(uuid)" />
    <MESSAGE id="roleName="requestor" messageType="getContracts(uuid)"">
        action = "com.thoughtworks.org.receiveMessage"
        nextState = "STATE_receiveMessage_RECEIVED_FROM_requestor_PROVIDING_getContracts(uuid)" />
</STATE>

<STATE id="STATE_START">
    <MESSAGE id="roleName="requestor" action = "connectionRequestFrom" nextState = "STATE_STARTED" />
</STATE>

<STATE id="STATE_STARTED">
    <MESSAGE id="roleName="filtersvc" action = "connectionRequestTo"
        nextState = "STATE_CONNECTION_REQUEST_TO_filtersvc_OBTAINED" />
</STATE>

<STATE id="STATE_CONNECTION_REQUEST_TO_filtersvc_OBTAINED">
    <MESSAGE id="roleName="requestor" messageType="getSuppliers(uuid)"">
        action = "receiveMessage"
        nextState = "STATE_receiveMessage_RECEIVED_FROM_requestor_PROVIDING_getSuppliers(uuid)" />
    <MESSAGE id="roleName="requestor" messageType="getContracts(uuid)"">
        action = "receiveMessage"
        nextState = "STATE_receiveMessage_RECEIVED_FROM_requestor_PROVIDING_getContracts(uuid)" />
</STATE>

<STATE id="STATE_receiveMessage_RECEIVED_FROM_requestor_PROVIDING_getSuppliers(uuid)">
    <MESSAGE id="roleName="requestor" messageType="deny()"
        action = "sendMessage"
        nextState = "STATE_sendMessage_SENT_TO_requestor_USING_deny()" />
    <MESSAGE id="roleName="suppliersvc" action = "connectionRequestTo"
        nextState = "STATE_CONNECTION_REQUEST_TO_suppliersvc_OBTAINED" />
</STATE>
```

……..
μService’s as EasyFSM’s

Recipient or Provider (depends on direction)

Current state

Java like syntax for business logic binding

DONE

<STATE id = "STATE_CONNECTION_REQUEST_TO_filtersvc_OBTAINED">
  <MESSAGE id = "roleName=requestor" messageType="getSuppliers(uuid)"">
    action = "com.thoughtworks.org.receiveMessage"
    nextState = "STATE_receiveMessage_RECEIVED_FROM_requestor_PROVIDING_getSuppliers(uuid)"
  </MESSAGE>
  <MESSAGE id = "roleName=requestor" messageType="getContracts(uuid)"">
    action = "com.thoughtworks.org.receiveMessage"
    nextState = "STATE_receiveMessage_RECEIVED_FROM_requestor_PROVIDING_getContracts(uuid)"
  </MESSAGE>
</STATE>

Represents a choice

Next state

parameters
μService’s and business logic

Some sort of java jar file incorporating all of the business function method calls and their parameter types as names.

To be implemented by the programmer is the business logic itself within the method calls and the parameter types as member variables to be stored as needed.
Scribble and μServices runtime

Lazy instantiation of scribble-defined μServices
Scribble and μServices runtime

Lazy instantiation of scribble-defined μServices
Scribble and μServices runtime

Velocity of agile delivery is increased through:
- Capture of tacit knowledge as complexity increases
  - Behavioral correctness of multi-parties (less rework)
- Standing up of stubbed out behaviors
- Reduction in effort and cost to deploy behaviors
  - Behavioral on-the-fly instantiation

This is the next generation of serverless architectures.

But that is not enough

To prove out the tools chain

To apply it to existing systems
But that is not enough

We have to be able to **understand what we have**. The legacy issue. And we need to understand it just enough to **make sensible decisions**.

So we need be able to modeling what we find, and to **analyse** it and have it give us advice on what to do (i.e. where to start first, the scope of what we need to change)