FIX Orchestra: The Full Stop at the End of FIX

June 2017

John Greenan, CEO Alignment Systems
FIX Orchestra Working Group
http://twitter.com/AlignmentSys
http://blog.alignment-systems.com

FIX Orchestra Status

- Release Candidate 1 is available
  - Application level orchestration
  - Usable today to describe message structure, scenarios, actors, and states
- Available on GitHub
  - FIX now develops technical standards in a fully open manner
- Release Candidate 2 in progress
  - Support for FIXatdl
  - Expression language
  - ISO 20022 metamodel convergence
  - Session layer orchestration
  - Interface definition

FIX Orchestra History – June 2015

View this email in your browser

Call for Participation - Machine Readable FIX Specification (the FIX Service Profile)

The Repository working group in conjunction with the Specification working group is forming a subgroup to address the work of generating machine readable rules for FIX Specifications, to be known as the **FIX Service Profile**. Machine readable rules of engagement are intended to improve operational efficiency and the value of the FIX Protocol by reducing the time and effort it takes to onboard, certify, and deploy new FIX connections with counterparties. Participants of this group will define the requirements for the FIX Service Profile. We are looking for participants who are knowledgeable in FIX onboarding, certification, and operations to drive these requirements for the FIX Service Profile from this subgroup.

If you are interested in participating in the Specification and Repository Working Group's subgroup for the development of a standard machine readable FIX specification, please contact the Program Office at fix@fixtrading.org.

FIX Orchestra History

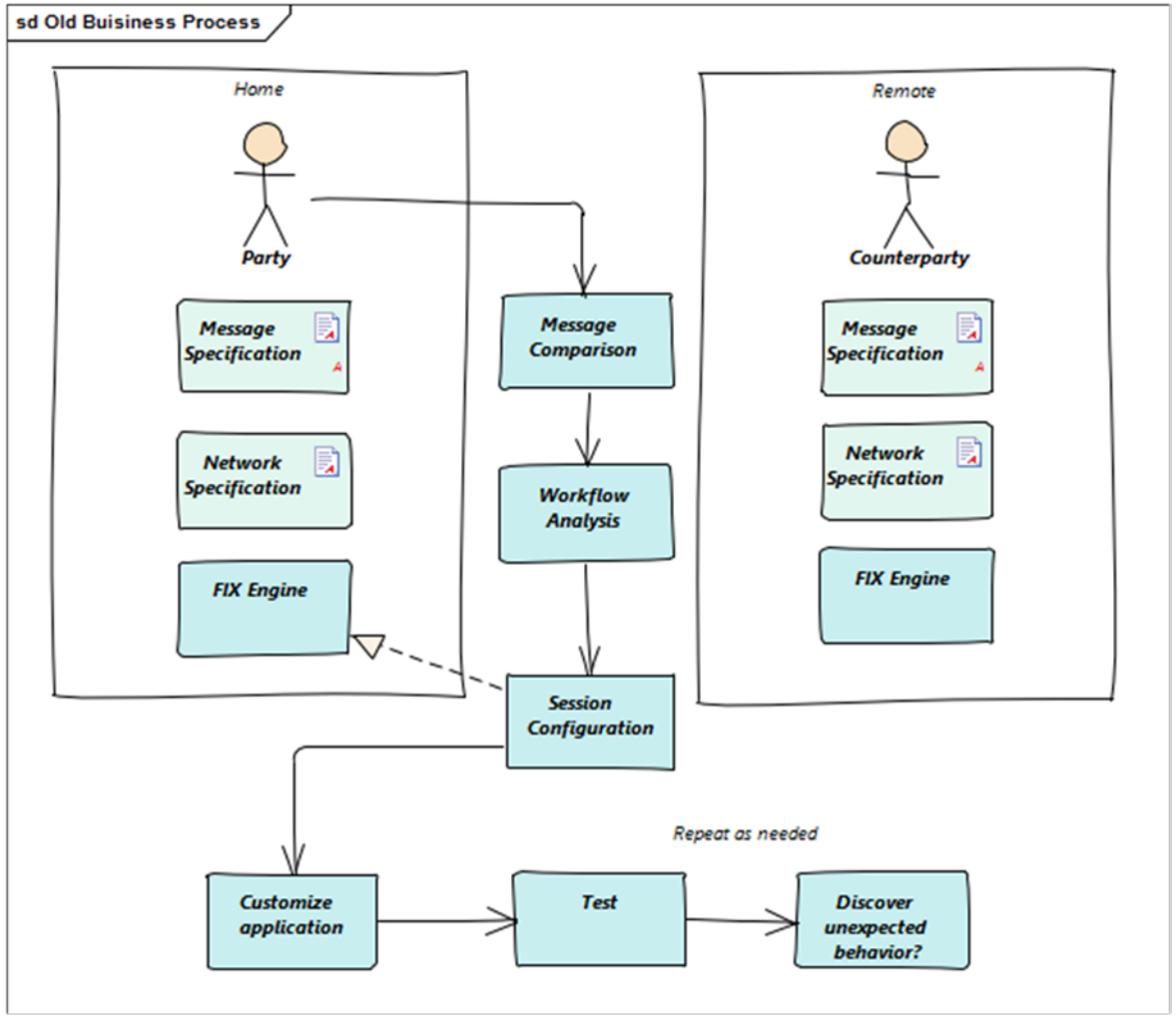
- April 2014 http://blog.alignment-systems.com/2014/04/fidl-fix-interface-definition-language.html

"There is some testing software out there (such as Greenline, LaSalleTech and others) but it really exists to automate a manual process without re-engineering. This proposal aims to offer a way forwards for the FIX Trading Community to allow a low value piece of work to be automated and removed from the day-to-day work needed."

- Initial implementation known as FIIDL – FIX Interactive Interface Definition (2013-2014)
- Proprietary, closed-source
- June 2015 New working group, clean-room, fresh start

Why Orchestra?

How to engage a FIX counterparty without Orchestra



What problems are we trying to solve?

- FIX protocol was loosely specified from the start. Plenty of room for interpretation.
- Specifications are usually documented in *human readable documents* that are exchanged between counterparties. Humans must interpret the specs and turn them into executables and configurations.
  - Conditionally required fields are explained in text which must be interpreted and converted to code.
- FIX standards tell the universe of possible values.
  - Which values of OrdType and TimeInForce are accepted by my counterparty?
- Workflow is often not well documented.
  - Under what conditions do I get a Session Level Reject, a Business Message Reject, or an Execution Report with ExecType=Rejected?
  - The same message type may have different contents for different scenarios, e.g. Execution Report for order accepted versus an execution.
- **In short, the information we have is sparse and not directly actionable.**

What is Orchestra?

What is FIX Orchestra and what does it do?

- **FIX Orchestra is a standard for exchanging machine-readable rules of engagement.**
- FIX remains the protocol on the wire. No changes required to your existing FIX engine (unless you want to). *FIX Orchestra is metadata about a specific implementation of FIX.*
- Orchestra is not a product, although FIX Trading Community may kickstart open-source implementations as examples. Vendors and firms are free to develop proprietary implementations, so long as they are conformant to the standard.

What is FIX Orchestra and what does it do?

Orchestra content, all machine readable

- Message structure by each scenario. Implemented as an extension of FIX Repository.
- Accepted values of enumerations by message scenario
- Workflow: when I send this message type under this condition, what can I expect back?
- How external states affect messages, e.g. market phases, order state, price
- Express a condition such as for a conditionally required field using an expression language

What is FIX Orchestra and what does it do?

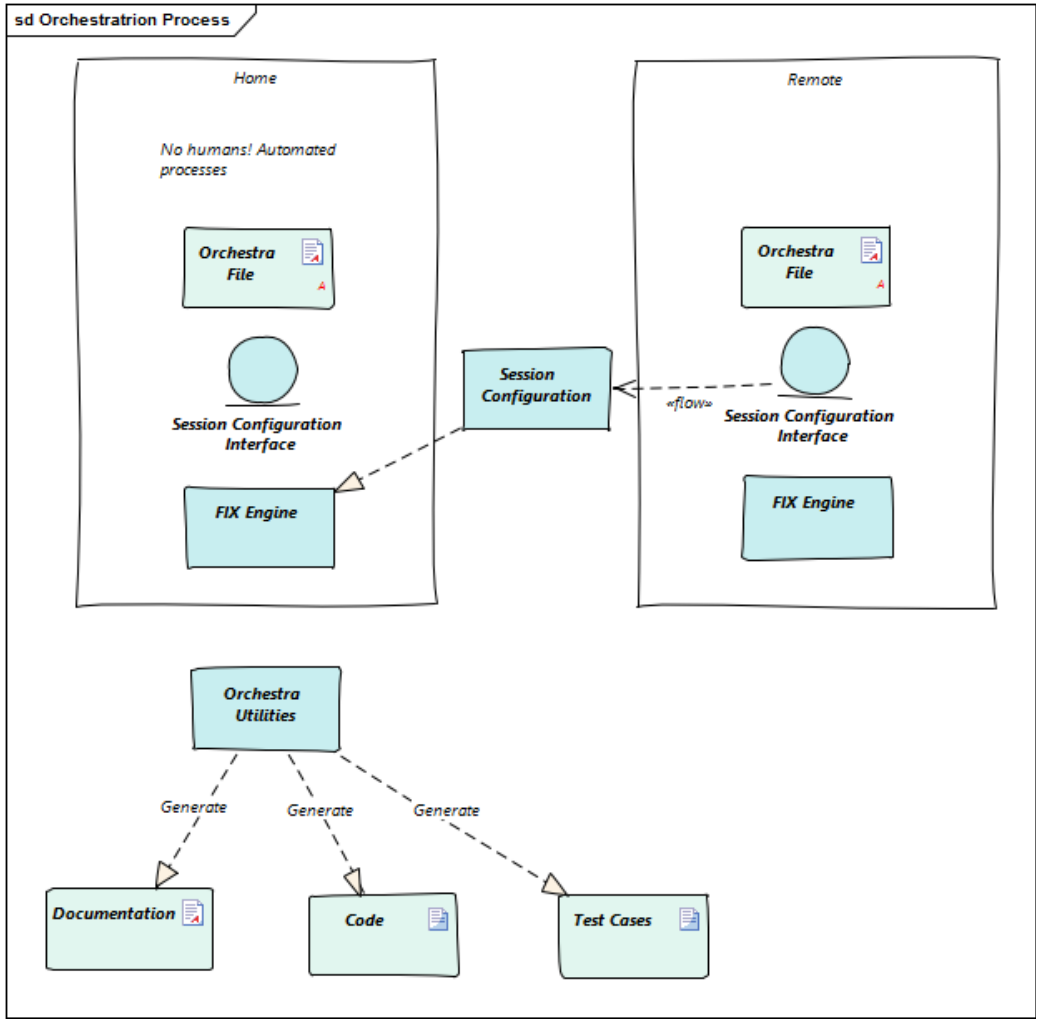
- Content is a composed of multiple feature categories.
  - Application layer structure and behavior – independent of encoding such as tag=value, FIXML, SBE
  - Session layer behavior
  - Operational: session configuration—identifiers and transport settings
- A firm does not need to implement every feature of FIX Orchestra to gain some benefit.
  - *Just want to share message definitions and conditional fields?* That's doable
  - *Want to extend to describe message responses, scenarios, and basic states?* That's doable
  - *Want to fully model in detail the FIX service?* That's doable

What is FIX Orchestra and what does it do?

Orchestra process of engagement

- Counterparties exchange their Orchestra files, either statically or exposed through network interfaces for discovery.
- Counterparties compare their own file with that of their partner.
  - Discover differences and restrictions
- Automatically generate:
  - FIX engine configuration
  - Application configuration and code
  - Test cases and sample messages
  - Documentation for those pesky humans

How to engage a FIX counterparty with Orchestra



Orchestra is an interface definition

- Orchestra defines an interface to service offerings or service endpoints
- You don’t need to modify the internals of your applications
- Almost all existing FIX infrastructure has the provisioning information of FIX connections and data dictionaries stored in multiple places and multiple formats.
- Orchestra can be used to define the service in one place within version control
- Then simple scripts can be created to read Orchestra files and update configuration files for various services
- The benefits of Orchestra can be available with minimal investment

FIX Orchestra supports innovation

Possible uses and tools

- Generate and run conformance tests
- Capture best practices as an Orchestra file instead of text
- Regulate internal flows within a large organization as well as between counterparties
- Orchestra is a contract for behavior – use it to generate an emulator for testing
- Capture an Orchestra file from FIX logs
- Analyze FIX logs for conformance to specified behavior
- Let's go further…
  - Generate Execution Management, Order Management, Smart Order Routing, Order Matching behavior based upon exchange of state machine descriptions contained within FIX Orchestra files

FIX Orchestra working group

Standard development tasks

- Gather requirements
  - Ongoing
- Propose and discuss possible solutions with actual code or mockups
  - Ongoing
- Write a specification for the standard
  - Release Candidate 1 - available
  - Release Candidate 2 – in process
- Develop examples and common utilities
- Promote the standard

FIX Orchestra roll-out and adoption

FIX Trading Community tasks

- Standardization
  - Working group uses GitHub to collaborate on schema and samples
  - Working group proposes standard to Global Technical Committee
  - Big bang or roll out in phases? e.g. message structure/FIX Repository 2016 edition, scenarios and state machines, condition DSL, session layer
- Develop Orchestra files representing existing FIX standards
  - Enhancement of FIX Repository
  - Best practices by asset class, region, etc. – delegate to various working groups
  - Publish in GitHub
- Develop open-source utilities
  - File comparison and reporting tools
  - Validation against schema

FIX Orchestra roll-out and adoption

Firm and vendor tasks

- Develop open-source or proprietary utilities
  - Compose Orchestra file from other sources
  - Orchestra file editors
  - Create or adapt configuration, test and code generators
  - Web interface for session configuration – web services or semantic web technologies
  - Certification tests
- Exchange files with counterparties
  - Pilot program for early adopters
- Give feedback to working group

FIX Repository → FIX Orchestra

- Orchestra and Repository 2016 Edition share a common XML schema
- The distinction is usage
  - Repository declares message structures
  - Orchestra adds workflow and conditional behavior.
- Users may select the set of features they wish to use.
- For this reason we are going to move to the term ***FIX Orchestra*** to refer to all representations of financial messaging protocols

Deep Dive into FIX Orchestra

Deep dive in FIX Orchestra

Contents

- Common elements
  - Abbreviations
  - Datatypes with mapping to XML Schema types and General Purpose Datatypes (ISO 11404)
  - Categories for documentation
  - Sections for documentation
  - Code Sets – sharable sets of valid values with underlying datatype
  - Fields—sharable in many messages
- For each version of FIX
  - Components, including common blocks and repeating groups
  - Messages refer to components and fields
- Provenance
  - Artifact described by Dublin Core Terms—who, what, when
  - Each message element can convey history—when added, changed, deprecated

Deep dive in FIX Orchestra

- *New:* a DSL to specify when conditionally required fields are required or forbidden (Boolean expression, may reference other fields.)

```
<fixr:fieldRef id="44" name="Price" presence="conditional">
        <fixr:required>
                <fixr:when>OrdType in [Limit, StopLimit ]</fixr:when>
        </fixr:required>
        <fixr:forbidden>
                <fixr:when>OrdType = Market</fixr:when>
        </fixr:forbidded>
</fixr:fieldRef>
```

Presence values are optional, required, conditional, forbidden, ignored, constant (need not be transmitted on wire).

Deep dive in Orchestra

- *New:* a code set is now a first-class object and may be shared among several fields.
- A code set has an underlying FIX datatype; may be char, int, string.

```
<fixr:codeSet name="TimeInForceCodeSet" type="char" default="Day">
        <fixr:code value="0" name="Day“/>
        <fixr:code value="1" name="GTC“/>
        <fixr:code value="2" name="AtTheOpening“/>
        <fixr:code value="3" name="IOC“/>
        <fixr:code value="4" name="FillOrKill“/>
</fixr:codeSet>
```

The code set is the “type” of this field.

```
<fixr:field id="59" name="TimeInForce" type="TimeInForceCodeSet“/>
```

Deep dive in FIX Repository 2016 Edition

- Datatypes section was enhanced to map FIX datatypes to General Purpose Datatypes (ISO 11404) as well as XML Schema types.
- Datatypes are about value space and should be independent of encoding.
- Both FIX and XML sometimes confused value and lexical spaces since they were originally character-based encodings. But now FIX has binary encodings, so we have to get this right.
- Example: a FIX datatype is Price. Its value space is exact numbers. Therefore, it should not be considered a subclass of float, as it was in the past.
  - General Purpose Datatypes has this covered with Scaled number type with factor and radix=10 parts.
  - XML Schema standard confuses value and lexical spaces of numbers. It is says that integer is derived from decimal!?

Deep dive in FIX Orchestra

Contents

- A superset of FIX Repository
- Actors with state variables and state machines
- Adds responses to a message, aside from structure
  - Message response (workflow)
  - State changes
  - State machine transitions
  - Each response qualified by “when” condition in DSL

Deep dive in FIX Orchestra

Actor example (snippet)

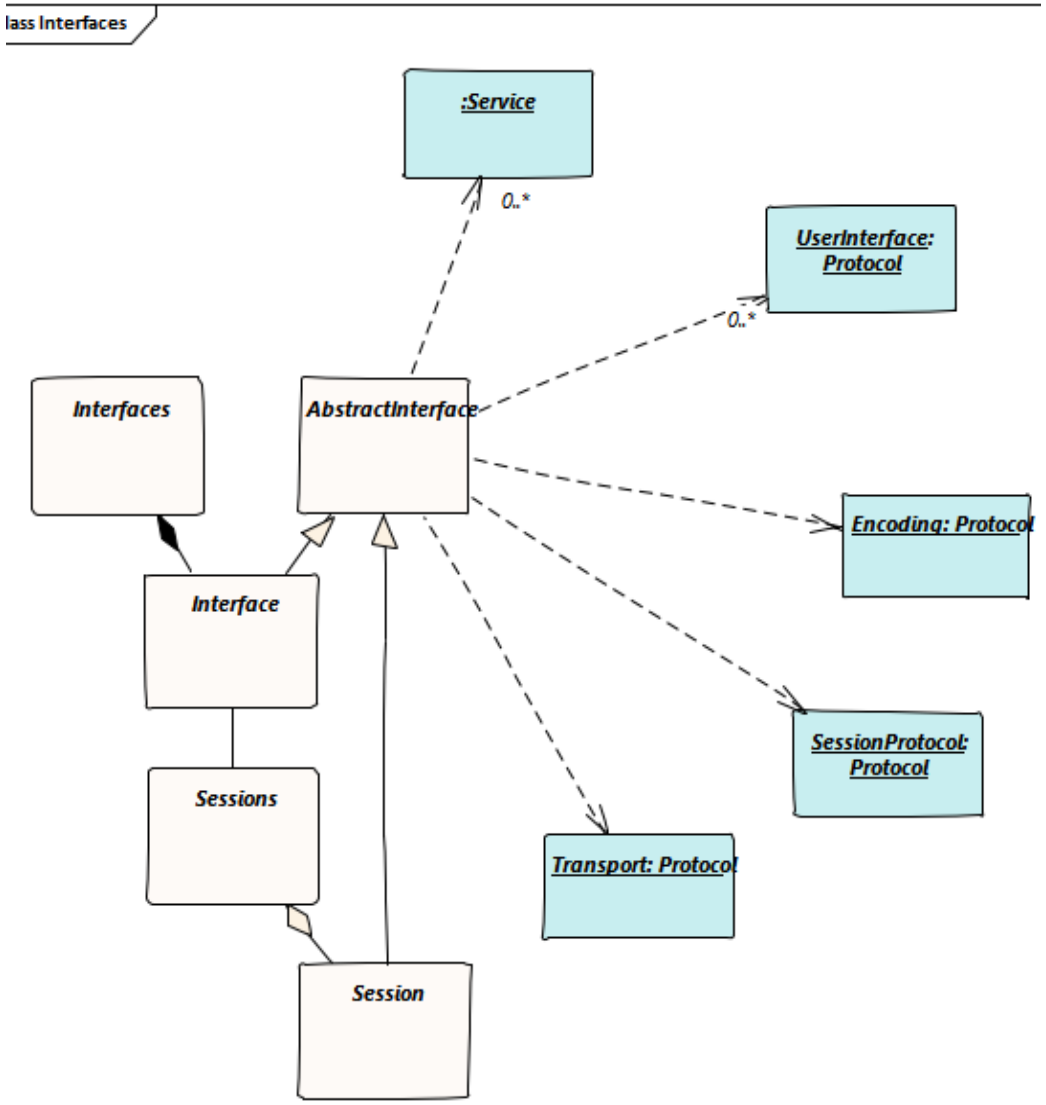
```
<fixr:actor name="Market">
        <!-- fields used as variables, not part of a message -->
        <fixr:field id="336" name="TradingSessionID" type="String"/>
        <fixr:field id="75" name="TradeDate" type="LocalMktDate"/>
        <!– a state machine -->
        <fixr:states name="Phase">
                <fixr:initial name="Closed">
                        <fixr:transition name="Reopening" target="Preopen"/>
                </fixr:initial>
                <fixr:state name="Halted">
                        <fixr:transition name="Resumed" target="Preopen"/>
                </fixr:state>
                <fixr:state name="Open">
                        <fixr:transition name="Closing" target="Preclose"/>
                </fixr:state>
```

Deep dive in FIX Orchestra

Responses example – conditional DSL may reference message elements or actor states

```
<fixr:responses>
        <fixr:response name="DKTrade">
                <fixr:messageRef name="DKTrade" msgType="8“
                 context="DontKnowTrade"/>
                <!-- validate value in incoming message -->
                <fixr:when>^OrdType IN [Market, Limit, Stop]</fixr:when>
        </fixr:response>
        <fixr:response>
                <fixr:messageRef name="BusinessMessageReject" msgType="j"/>
                !-- test current state of a state machine -->
                <fixr:when>$ApplicationState == DOWN</fixr:when>
        </fixr:response>
</fixr:responses>
```

Orchestra Interface Metamodel



Interface Definition

We start with the interface definition

```
<fixi:interfaces xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:fixi="http://fixprotocol.io/2016/fixinterfaces"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://fixprotocol.io/2016/fixinterfaces
../../main/resources/xsd/FixInterfaces.xsd">
   <fixi:metadata>
   <dcterms:subject>Service offerings and sessions example file</dcterms:subject>
   <dcterms:description>Mock-up presentation for concepts</dcterms:description>
   <dcterms:date>2017-04-21</dcterms:date>
   </fixi:metadata>
   <fixi:interface name="Private">

                    <!-- … details live in here -->

   </fixi:interface>
</fixi:interfaces>
```

Interface Definition – define our service

Here we can define our stack

```
<fixi:interfaces >
<fixi:metadata/>
<fixi:interface name="Private">
    <!-- one or more service offerings, with local orchestration file or internet address -->
    <fixi:service name="orderEntry”
    orchestration="https://mydomain.com/orchestra/orderEntry.xml"/>
    <!-- the protcol stack -->
    <fixi:userInterface name="ATDL" orchestration="https://mydomain.com/orchestra/algo.xml"/>
    <fixi:encoding name="TagValue"/>
    <fixi:sessionProtocol name="FIXT.1.1" reliability="recoverable"
    orchestration="https://mydomain.com/orchestra/session.xml">
        <fixi:annotation>
            <fixi:documentation langId="en-us">FIX session protocol</fixi:documentation>
        </fixi:annotation>
    </fixi:sessionProtocol>
    <fixi:transport name="TCP"/>
    <fixi:sessions/>
</fixi:interface>
```

Interface Definition

Now let us define a session

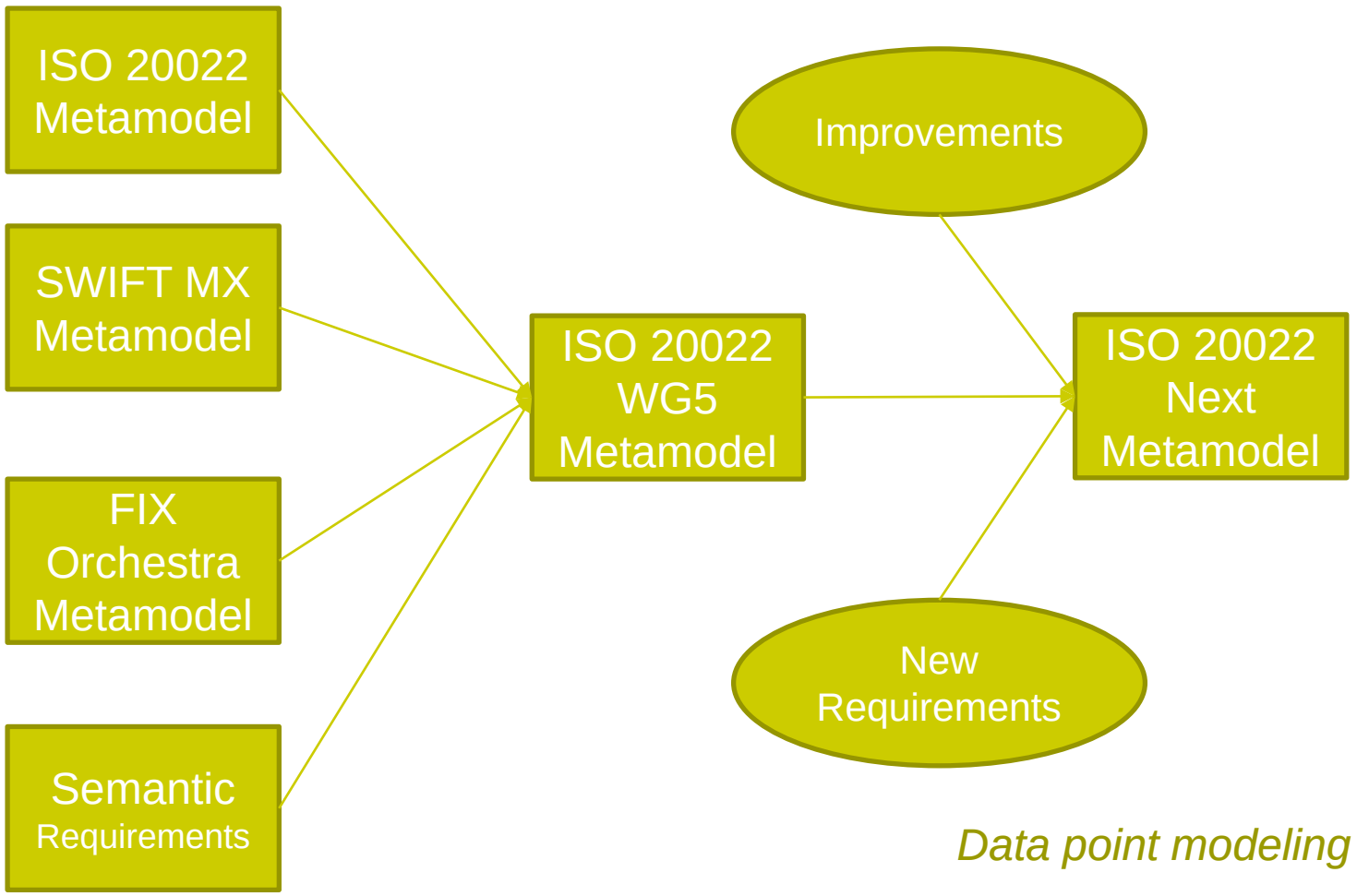
```
<fixi:interfaces >
<fixi:metadata/>
<fixi:interface name="Private">

<!– Service and Protocol Definitions removed -->

    <fixi:sessions>
        <fixi:session name="XYZ-ABC">
            <!-- inherits services and protocols from interface -->
            <!-- alternate addresses are supported -->
            <fixi:transport address="10.96.1.2:567" use="primary"/>
            <fixi:transport address="10.96.2.2:567" use="secondary"/>
            <!-- there can be any number of identifiers -->
            <fixi:identifier name="SenderCompID">
                <fixi:value>XYZ</fixi:value>
            </fixi:identifier>
            <fixi:identifier name="TargetCompID">
                <fixi:value>ABC</fixi:value>
            </fixi:identifier>
            <!-- tells when session becomes effective so it can be configured in advance -->
            <fixi:startTime>2017-05-17T09:30:00Z</fixi:startTime>
        </fixi:session>
    </fixi:sessions>
</fixi:interface>
</fixi:interfaces>
```

Interface Definition

Now for something completely different a new service

URI to local file or web resource

```
<fixi:interface name="OrderRouting">
   <fixi:service name="orderRouting"
   orchestration="https://mydomain.com/orchestra/orderRouting.xml"/>
   <fixi:encoding name="GPB" messageSchema="file://something.proto">
   <fixi:annotation>
   <fixi:documentation>Message schema attribute demonstrates
   extensibility</fixi:documentation>
   </fixi:annotation>
   </fixi:encoding>
   <fixi:protocol name="TLS" version="1.2" layer="transport">
   <fixi:annotation>
      <fixi:documentation>Additional protocols may be
      added</fixi:documentation>
   </fixi:annotation>
   </fixi:protocol>
   <fixi:sessions>
</fixi:interface>
```



Data point modeling (DPM)? FpML?

Similar concepts

- Need to specify how a message is actually used
  - Business rules
  - Restrictions
  - Extensions