Large Scale Integration

John Davies

Agenda

- Problem? What problem?
  - Surely integration is commodity now?
- Some of the shit we have to deal with
  - FIX, FpML, SWIFT
- Just create a big canonical model, that'll solve everything
  - Err - no!
- Metadata management and Java-binding

Integration - Old hat?

- In 2000 we thought seen the end of integration so we started a BPM company

- By 2002 we'd given up on BPM and were selling SWIFT integration

- By 2006 we had most of the large investment banks as customers

- In 2007 we'd sold the company

More and more Integration

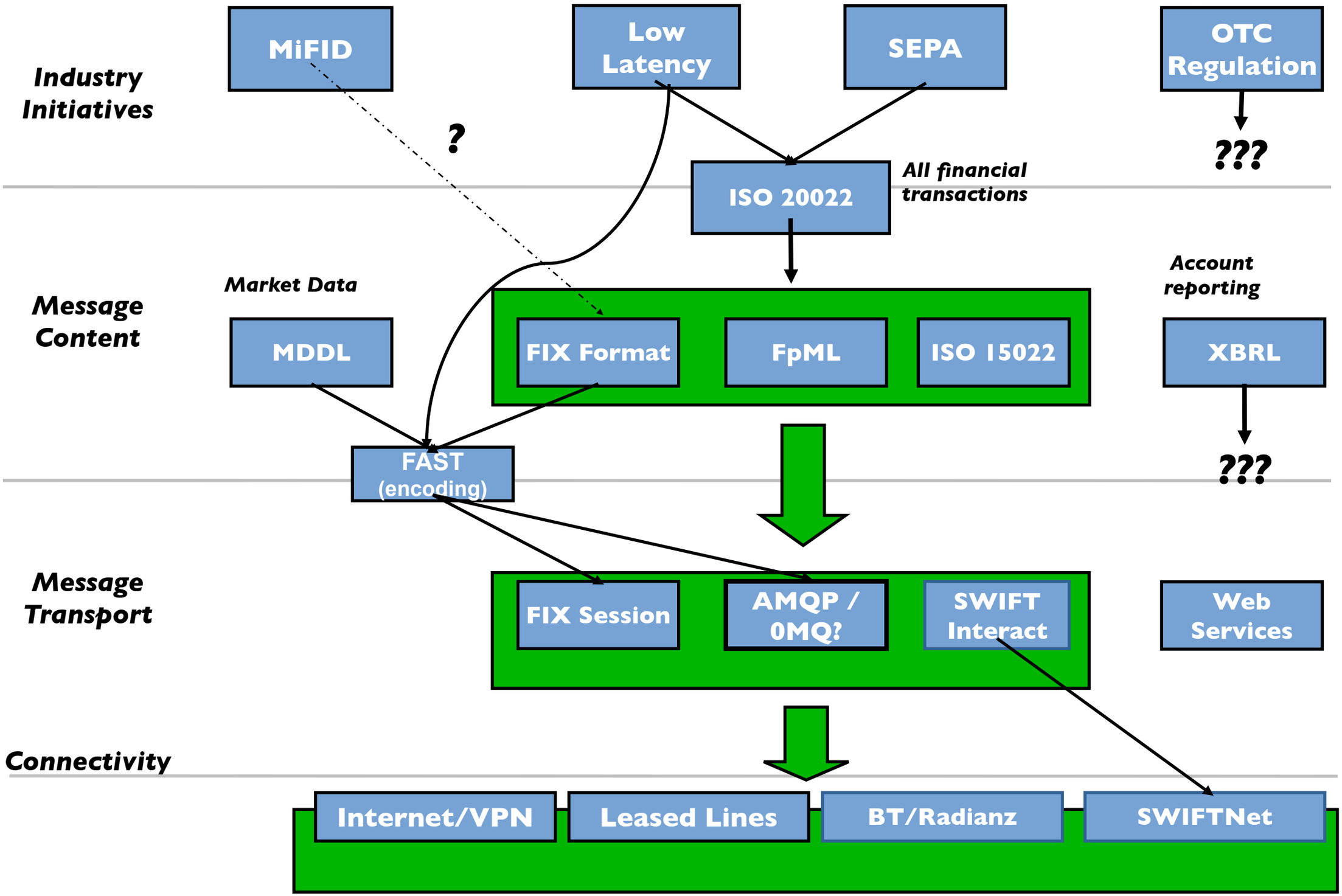
- When you think about it, as we become more and more distributed and a increasingly global market, guess what?

- We need more and more integration

- SOA, ETL, ESB, Spring Integration, Mule, JMS, MQ Series, Tibco RV, ReST, WS, RMI, Remoting etc. etc. etc.
  - Integration is everywhere

Incept[5]

The Financial Services Landscape



Integration - High Volume

- Front Office
  - Very high volume (100-100,000 / sec), usually simple messages
  - Latency is critical (< 10ms)
  - FIX, FAST, ASN.1, IIOP are most common payloads and protocols
  - Light-weight XML only (if any)

- Credit card processing
  - ISO-8583, Binary, NVP, Batch
  - 10,000 / sec or 180m / day

- Tax processing
  - Individual census / population records

FIX

- Common protocol in the Front Office is FIX
  - FIX comes in several flavours - 4.0, 4.1, 4.2, 4.3, 4.4, 5.0, (all the above as FIXML), FAST and FIXatdl

- FIX is both a message standard and a Protocol
  - As of FIX 5.0 the session protocol is split out meaning the transport is independent

- Having a FIX engine doesn't mean you can understand the messages
  - Conversely being able to understand the messages doesn't mean you can communicate through FIX

FIX 4.4 - Post-Trade Confirmation

- This is a FIX 4.4 Post-Trade Conformation
  - There's no time for the "<" and ">"

```
8=FIX.4.4 9=1 35=AK 49=STRING 56=STRING 90=1 91=D 34=1 50=STRING
142=STRING 57=STRING 143=STRING 144=STRING 145=STRING
52=20020101-00:00:00.000 122=20020101-00:00:00.000 212=1 213=D
347=ISO-2022-JP 369=1 627=1 628=STRING 629=20020101-00:00:00.000
630=1 664=STRING 772=STRING 859=STRING 666=0 773=2 797=N 650=Y
665=4 453=1 448=STRING 447=B 452=1 802=1 523=STRING 803=1
60=20061122-00:00:00.000 75=20061122 55=STRING 65=STRING
48=STRING 22=1 454=1 455=STRING 456=1 460=1 461=STRING 167=FAC
762=STRING 200=200201 541=20020101 224=20020101 225=20020101
239=RP 226=1 227=1.0 228=1.0 255=STRING 543=STRING 470=AF 471=GB
472=STRING 240=20020101 202=1.0 947=USD 206=0 231=1.0 223=1.0
207=XLON 106=STRING 348=1 349=D 107=STRING 350=1 351=D
691=STRING 667=200611 875=99 876=STRING 864=1 865=99
866=20061117 867=4.3 868=STRING 873=20061117 874=20061117 80=400
54=2 862=1 528=A 529=12 863=200 79=STRING 6=1.5 381=123.45
118=115.78 93=6 89=STRING 10=000
```

FIX isn't complex

- FIX is "very" simple, it's basically tag/value pairs

```
8=FIX.4.1 9=154 35=6 49=BRKR 56=INVMGR 34=236
52=19980604-07:58:48 23=115685 28=N 55=SPMI.MI
54=22 7=200000 44=10100.000000 25=H 10=159
```

- So, simple, the tag represents the field...
  - 44 refers to Price
  - 52 is sending Date/Time
  - 55 refers to the symbol

- Basic but it's still better than XML when latency comes into play

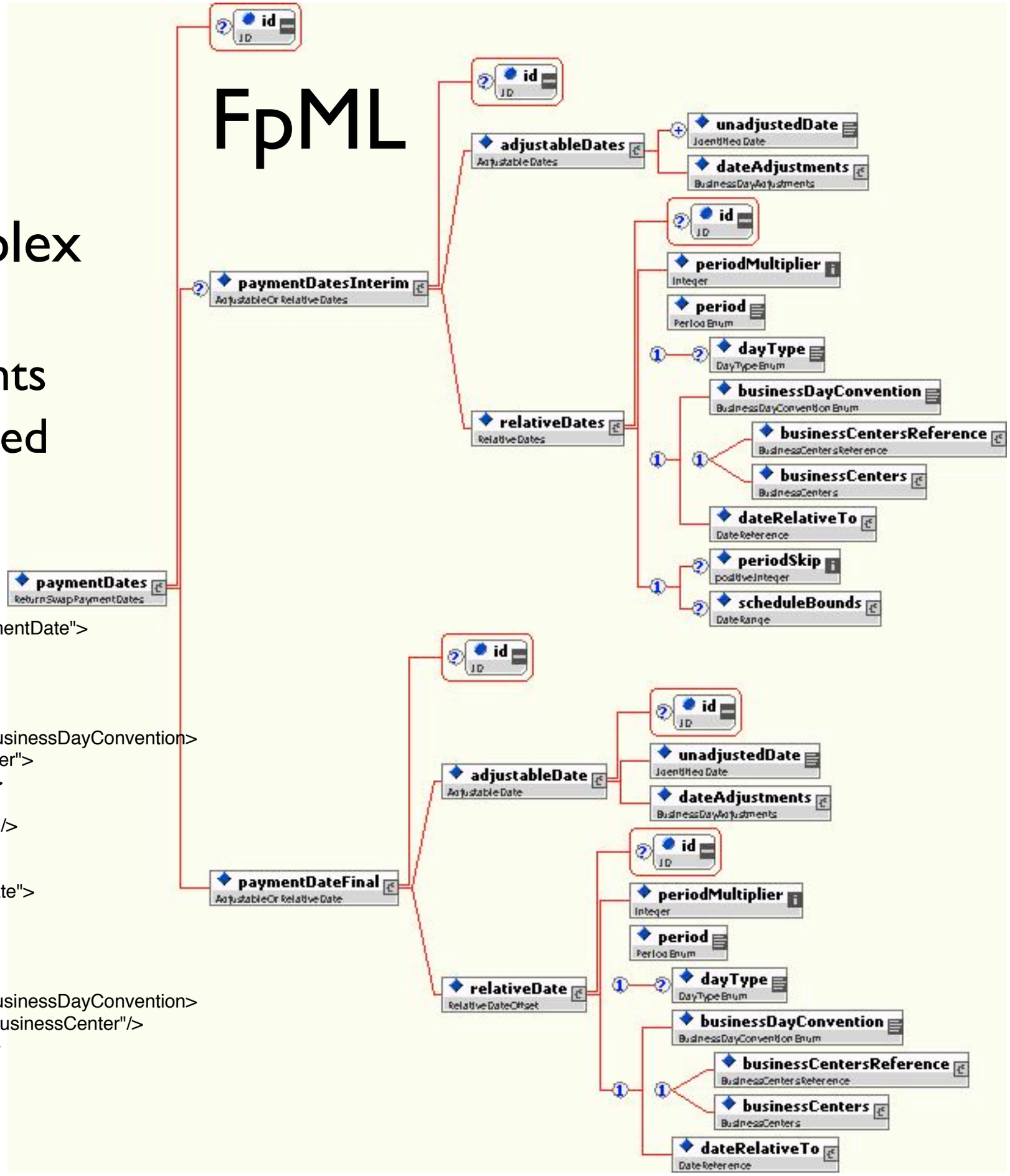
Integration - Complex

- Middle Office
  - Volumes are medium to high (1-1000 / sec), very complex messages
  - Calculations are complex and grid/HPC is usually required
  - Derivative contracts on ISDA's FpML
  - Corporate Actions (also FpML)
  - SEPA on ISO-20022, Murex, SwapsWire, CSVs are also common
  - XML widely used but usually over MQ & JMS

- Tax processing
  - Wealth records, inheritance, PAYE etc.

FpML

- FpML - Complex
  - 15 levels
  - >3000 elements
  - But well defined




```
<paymentDates id="EquityPaymentDate">
  <paymentDatesInterim id="InterimEquityPaymentDate">
  <relativeDates>
    <periodMultiplier>3</periodMultiplier>
    <period>D</period>
    <dayType>CurrencyBusiness</dayType>
    <businessDayConvention>FOLLOWING</businessDayConvention>
    <businessCenters id="PrimaryBusinessCenter">
      <businessCenter>USNY</businessCenter>
    </businessCenters>
    <dateRelativeTo href="InterimValuationDate"/>
  </relativeDates>
 </paymentDatesInterim>
 <paymentDateFinal id="FinalEquityPaymentDate">
  <relativeDate>
    <periodMultiplier>3</periodMultiplier>
    <period>D</period>
    <dayType>CurrencyBusiness</dayType>
    <businessDayConvention>FOLLOWING</businessDayConvention>
    <businessCentersReference href="PrimaryBusinessCenter"/>
    <dateRelativeTo href="FinalValuationDate"/>
  </relativeDate>
 </paymentDateFinal>
</paymentDates>
```

Integration - High Value

- Back Office
  - Low volume (10-1000 / hour)
  - Very high value messages, strict compliance and validation
  - Proprietary networks, mostly SWIFT

SWIFT - A seriously reliable network

- SWIFT is 3 things, a secure network, a standards body and a connectivity provider
  - It is used by over 8,000 banks (>80,000 branches), in over 200 countries handling over 15 million messages a day (>2 billion/year)
  - Mostly payments and securities, Europe is >65% of the volume
  - SWIFT is over 30 years old, has a systems availability of 99.986% (<1½ minutes/week) , they've NEVER lost a message

- The figures are impressive but the messages are a real bastard!
  - Around 330 types of message
  - >400 complex types, >1000 complex validation rules

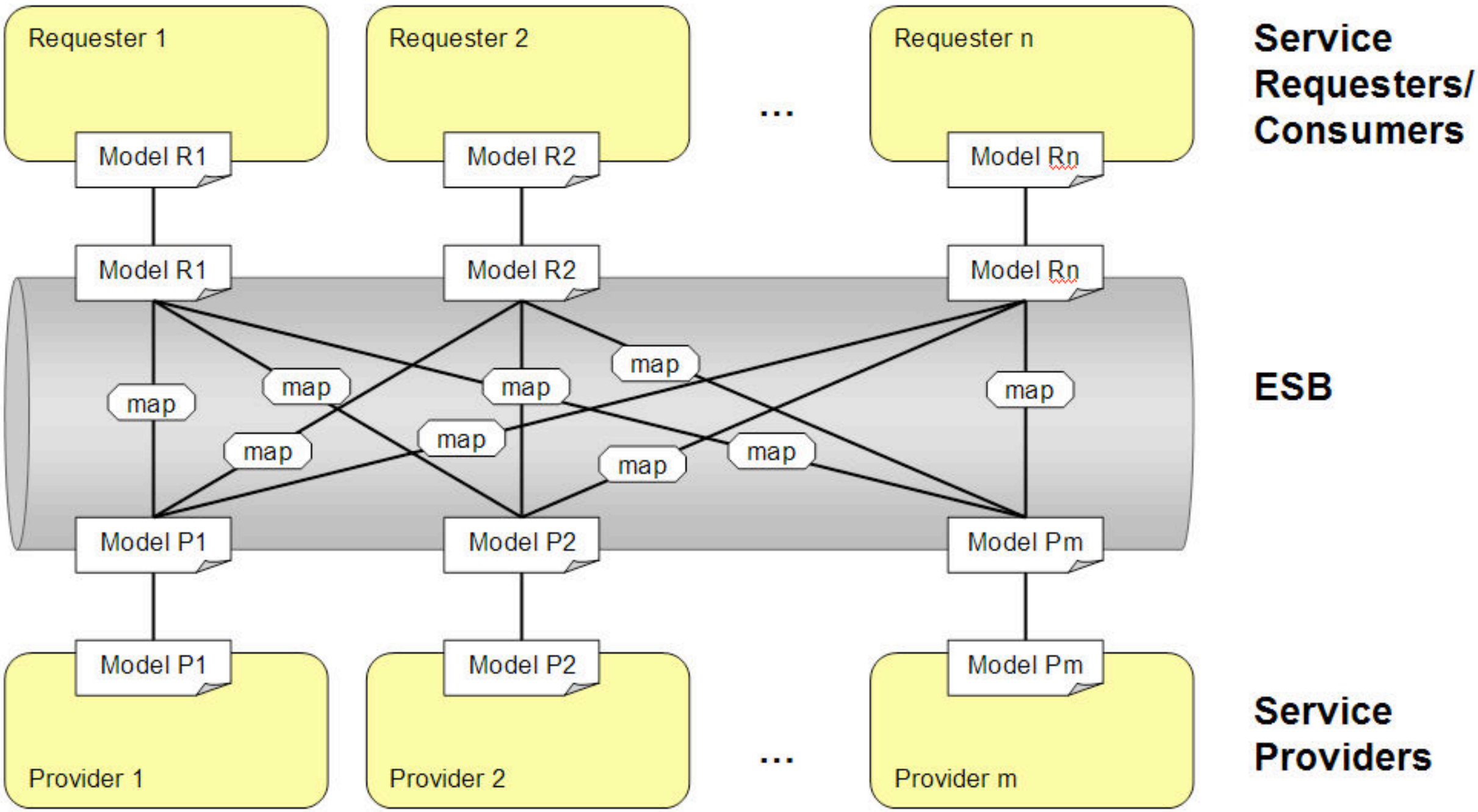
SWIFT - MT564 Corporate Action Notification

- Plenty of time for the "<" and ">" but it's 30 years old and 80,000 banks already use it

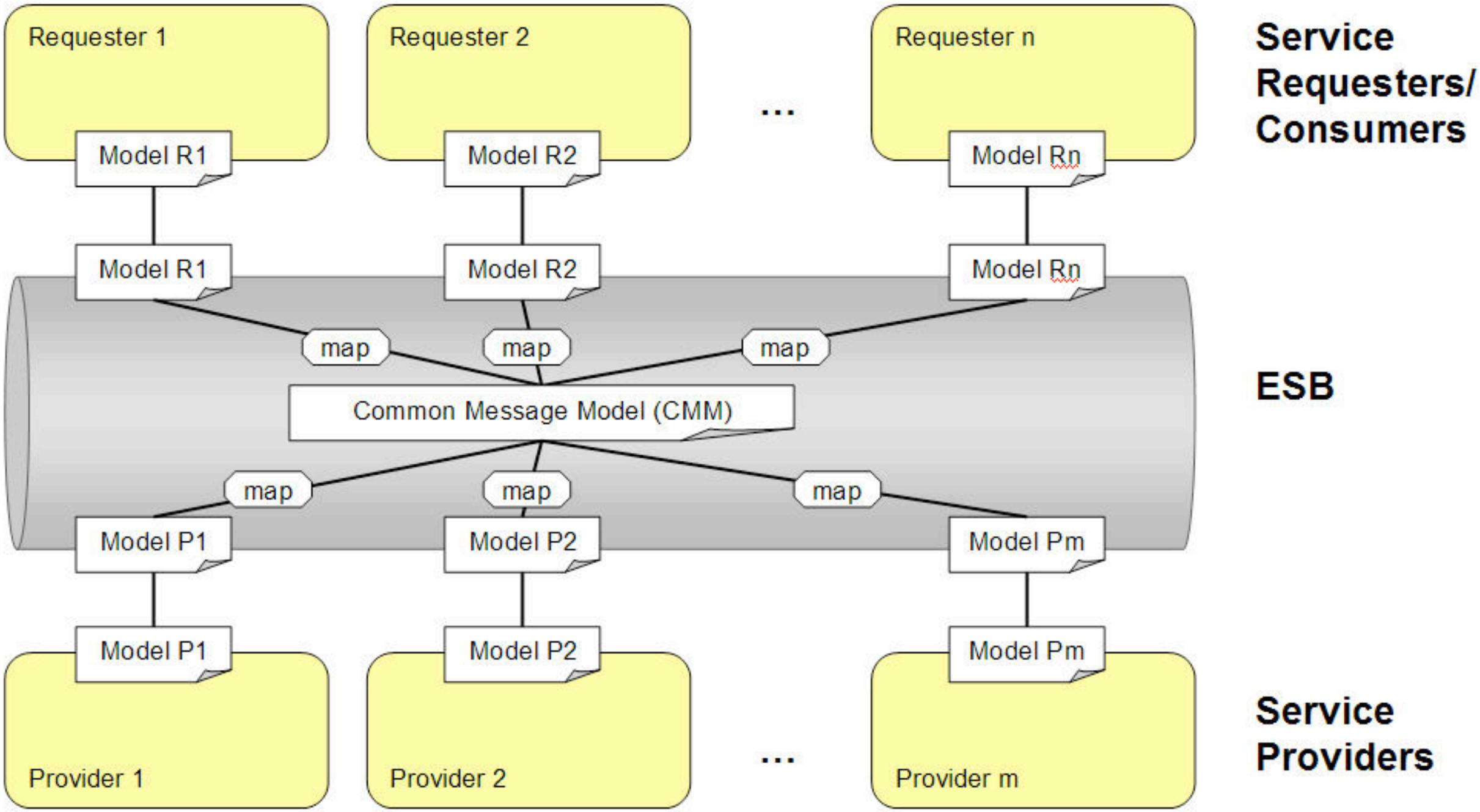
{1:F01INTRUS33AXXX9999999999}
{2:O5640947040127FRNYUS33AXXX4218183425040127094 7N}{3:
{108:MT564}}{4:
:16R:GENL
:20C::SEME//2003041800000042
:20C::CORP//12345
:23G:NEWM/CODU
:22F::CAEV//XMET
:22F::CAMV//VOLU
:98A::PREP//20010901
:25D::PROC//PREC

Incept[5]

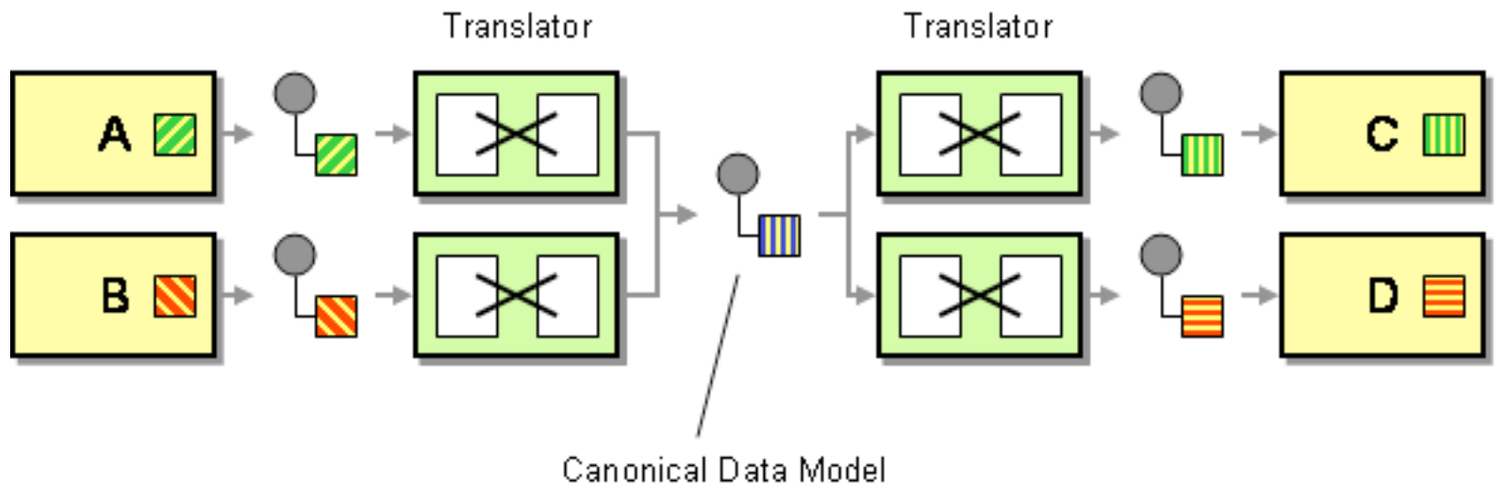
Classic Integration



Canonical Model



The pattern



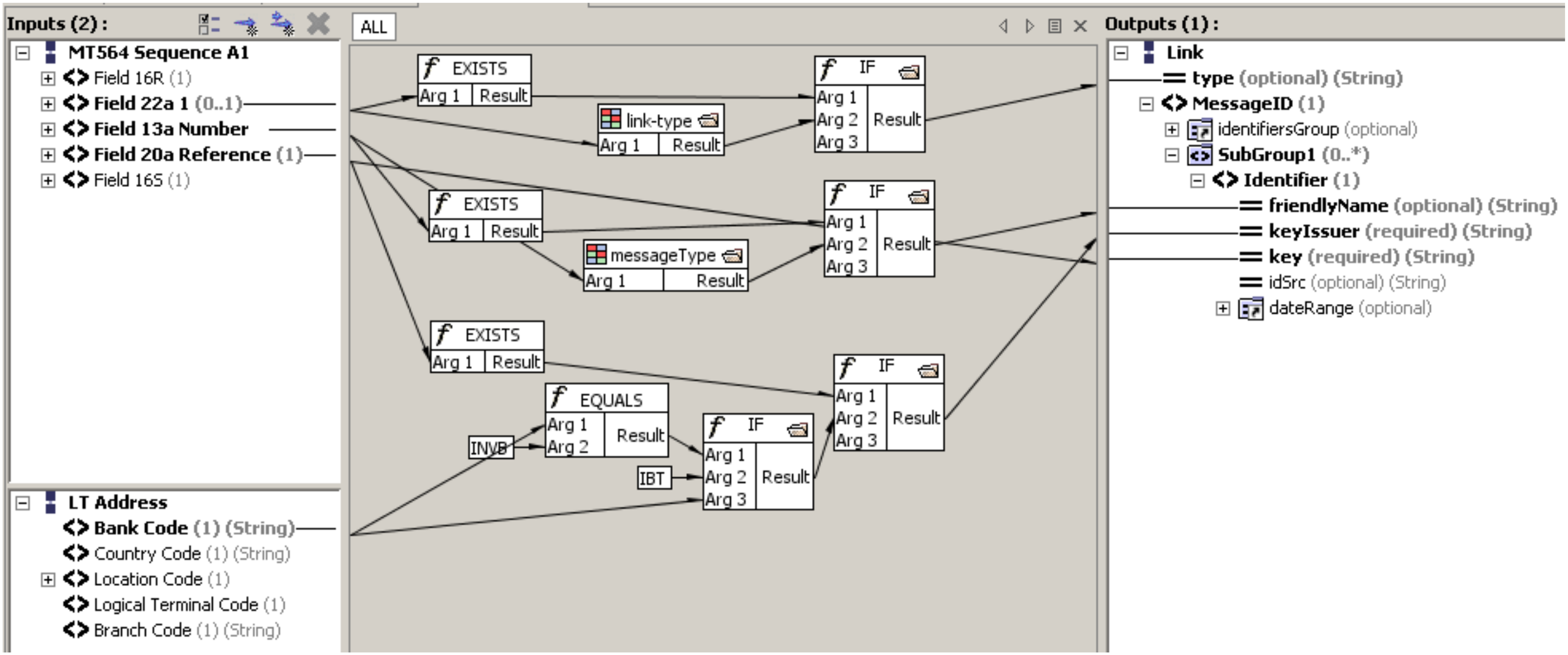
- As viewed in "Gregorgrams" (from Gregor Hohpe)
  - Also as you'd see it in Spring Integration
- The frequent need for bi-directional mapping seems to often get left out
  - The latency and CPU-cost of a parse, two transformations and formatting (output) is huge

Why transform everything?

- If you can understand this from a SWIFT message...
  - 8=FIX.4.1 9=154 35=6 49=BRKR 56=INVMGR 34=236 52=**20030418**-07:58:48 23=115685 28=N
- And you need this...
  - java.util.Date
  - Then just parse it
- If however someone/something needs this...
  - :20C::SEME//**20030418**00000042
  - Then why use an intermediate format?
- As long as you understand that the FIX field is the same as the SWIFT field then there is no transformation
  - Just re-formatting of the values in a new message

Complex stuff is complex

- It will always be complex
- If the input is vastly different from the output then you are going to need "classic" transformation...



The rule

- Keep data, as far as possible, in its original format
  - But as a bound Java Object - An Integration Object
- The Integration Object can read and write itself (parse and format) with no loss of information
  - Parsing includes syntactic and semantic validation
  - JUnit tests in the CI validate these features
- Validated Integration Objects conform to the Metadata model of our systems
- The Integration Objects are the "canonical" messages
  - But only the elements are common, not the message formats

FIX 4.4 - Post-Trade Confirmation

- We model the FIX message...

```
8=FIX.4.4 9=1 35=AK 49=STRING 56=STRING 90=1 91=D 34=1
50=STRING 142=STRING 57=STRING 143=STRING 144=STRING
145=STRING 52=20020101-00:00:00.000
```

Confirmation

Component	Type	Cardinality
Confirmation	Confirmation	
StandardMessageHeader	StandardMessageHeader	1
BeginString	Field 008 - BeginString	1
BodyLength	Field 009 - BodyLength	1
MsgType	Field 035 - MsgType	1
StandardMessageHeaderFields	StandardMessageHeaderFields	1
SenderCompID	Field 049 - SenderCompID	1
TargetCompID	Field 056 - TargetCompID	1
OnBehalfOfCompID	Field 115 - OnBehalfOfCompID	0..1
DeliverToCompID	Field 128 - DeliverToCompID	0..1
SecureDataLen	Field 090 - SecureDataLen	0..1
SecureData	Field 091 - SecureData	0..1
MsgSeqNum	Field 034 - MsgSeqNum	1
SenderSubID	Field 050 - SenderSubID	0..1
SenderLocationID	Field 142 - SenderLocationID	0..1
TargetSubID	Field 057 - TargetSubID	0..1
TargetLocationID	Field 143 - TargetLocationID	0..1
OnBehalfOfSubID	Field 116 - OnBehalfOfSubID	0..1
OnBehalfOfLocationID	Field 144 - OnBehalfOfLocationID	0..1
DeliverToSubID	Field 129 - DeliverToSubID	0..1

Incept[5]

SWIFT - MT564 Corporate Action Notification

{1:F01INTRUS33AXXX9999999999}{2:O564094704012 7FRNYUS33AXXX4218183425040127 0947N}{3:{108:MT564}}{4:

:16R:GENL

:20C::SEME//2003041800000042

:20C::CORP//12345

:23G:NEWM/CODU

:22F::CAEV//XMET

:22F::CAMV//VOLU

:98A::PREP//20010901

:25D::PROC//PREC

:16R:LINK

:22F::LINK//INFO

:13A::LINK//992

:20C::RELA//ABC

:16S:LINK

:16S:GENL

:16R:USECU

:35B:/ISIN/IDENTIFIER12

:16R:FIA

:12C::CLAS//ESVUFR

:11A::DENO//AUD

MT564 Message

Component	Type	Cardinality
MT564 Message	MT564 Message	
Block1	*Basic Header Block 1*	1
Block2		1
Iblock2	Iblock2 (local)	1
Oblock2	Oblock2 (local)	1
Input Output Identifier	Output Identifier	1
Message Type	Message Type (local)	1
Input Time	Input Time	1
Message Input Reference	*Message Input Reference*	1
Output Date	Output Date	1
Output Time	Output Time	1
Message Priority	Message Priority	0..1
Block3	User Header Block 3	0..1
Block4	MT564 Text Block	1
SeqA	MT564 Sequence A General	1
Field 16R	Field 16a Type (local)	1
Field 20a Reference	Field 20a Type (local)	1..*
Field 23G Function of the	Field 23G Type (local)	1
Field 22a 1	Field 22a Type (local)	1..*
Field 98a 2	Field 98a Type (local)	0..1
Field 25D Status Code	Field 25D Type (local)	1

What we get for free

- We need to be able to parse any message type
  - Binary (ISO-8583), CSV, Proprietary (SWIFT, FIX etc.), well structured but complex (FpML)

- If we could treat the Integration Object **as if** it were XML we could execute XPath on anything parsable

- It just needs an XPath navigator (Jaxen, Saxonica)
  - We could build XPath routing rules or extract data using XPath regardless of the input format (XML, CVS, SWIFT etc.)
  - We could also enrich the Schema validation features to include cross-field references and also validate **any** data source we can parse

Complex validation

- ISO-8601 DateTime in XML Schema is well defined
  - Well almost, there are still inconsistencies about time-zones and time offsets
  - Problem is to restrict one field based on the content or existence of another field(s)
    - If //@AlternateEmail then at least two emails must be defined
    - //TradeDate must be before or the same as the //SettlementDate

sentry-50k.csv

	A	B	C	D	E	F	G	H	I
1	13825693	0000000013	1.0242E+10	10242	256164000	17-Mar-97	1-Jan-71	19-Mar-07	USD
2	33	33_MUREX_L	5969000012	5969	3.1503E+11	15-Nov-95	1-Jan-71	8-Dec-05	JPY
3	818172	818172_MUR	1.0242E+10	10242	256164000	17-Mar-97	1-Jan-71	15-Mar-07	JPY
4	837142	837142_MUR	1.0242E+10	10242	256164000	2-May-97	1-Jan-71	3-May-07	JPY
5	837145	837145_MUR	1.0242E+10	10242	256164000	2-May-97	1-Jan-71	3-May-07	USD
6	893022	893022_MUR	1.0242E+10	10242	256164000	29-Aug-97	1-Jan-71	30-Aug-07	USD

- This problem isn't unique to XML, it is true for almost any type of data

XPath on any message

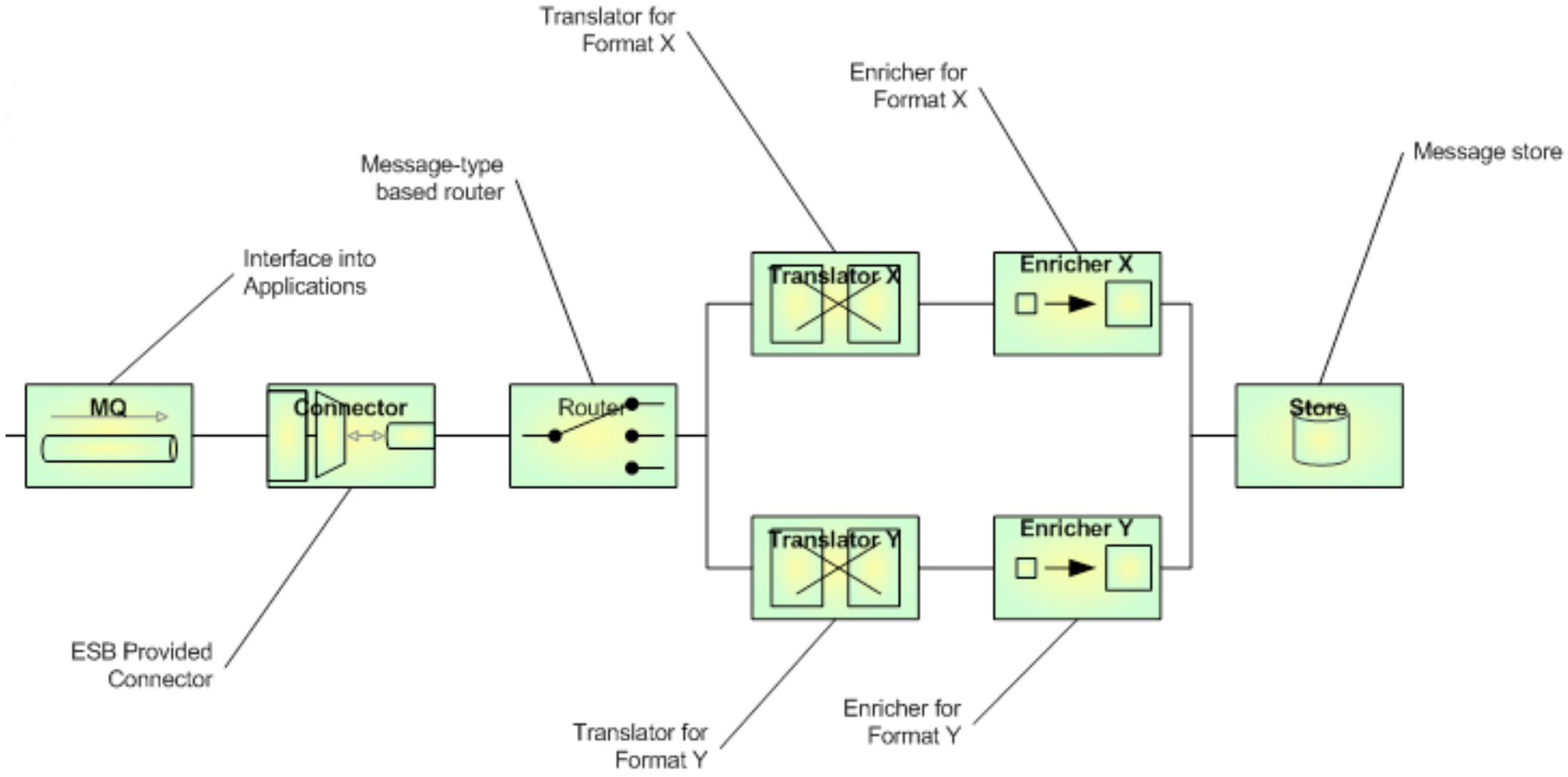
- Take the earlier SWIFT message...

{1:F01INTRUS33AXXX9999999999}{2:O5640947040127FRNYUS33AXXX4218183425040127094 7N}{3:{108:MT564}}{4:
:16R:GENL
:20C::SEME//2003041800000042
:20C::CORP//12345
:23G:NEWM/CODU
:22F::CAEV//XMET
:22F::CAMV//VOLU
:98A::PREP//**20010901**
:25D::PROC//PREC
:16R:LINK

- As examples of XPath
  - To read the whole line with the date in it... `/Block4/SeqA/Field98a2`
  - To read just the date... `/Block4/SeqA/Field98a2/A/DateYYYYMMDD`
  - To read all the dates in Block 4... `/Block4//DateYYYYMMDD`
  - Count the number of dates in Block 4... `count(/Block4//DateYYYYMMDD)`

XPath routing

- This is the logical architecture of a large European clearing house



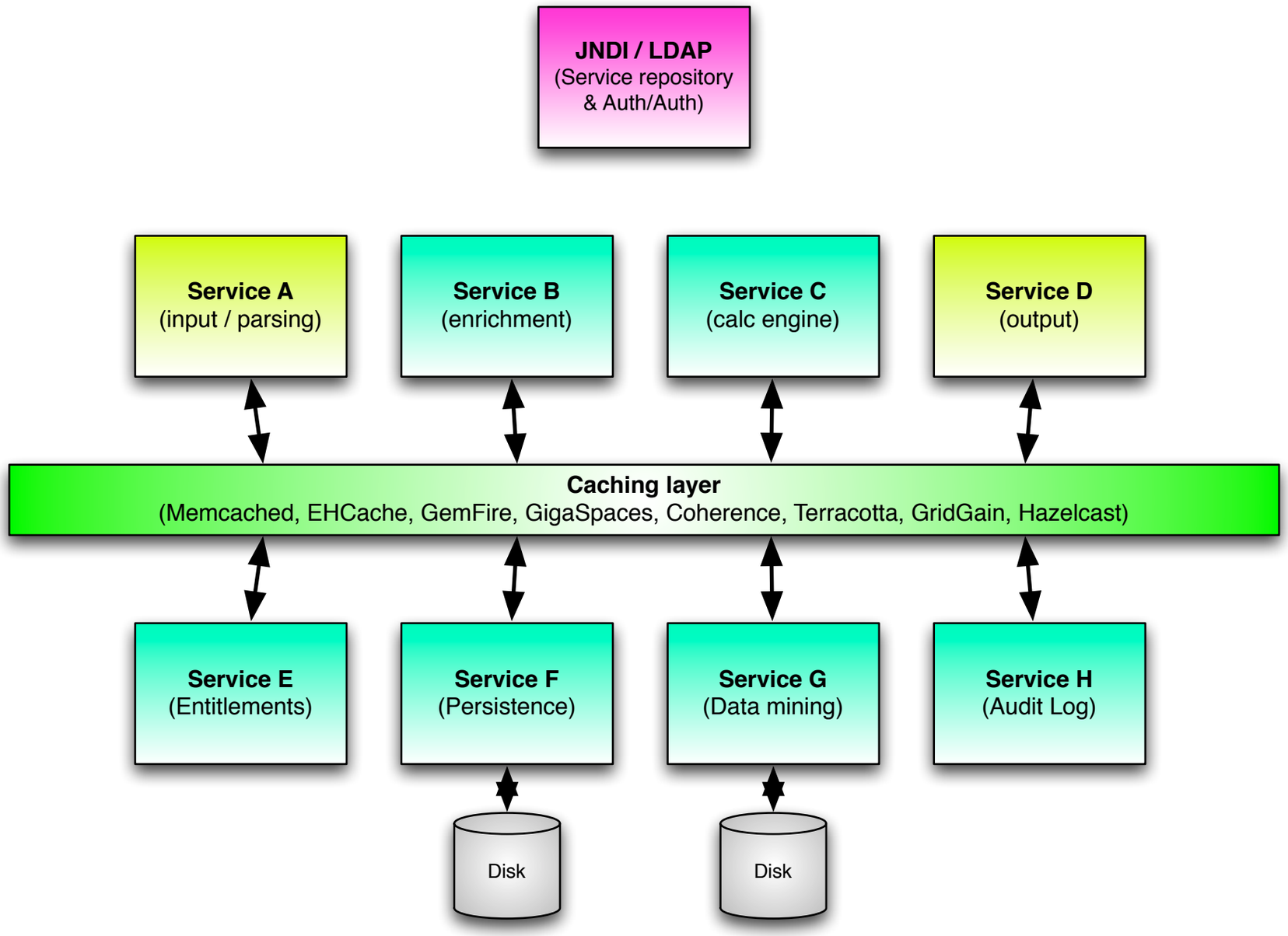
- Routing can be performed by applying XPath queries on the incoming Integration Objects

Persistence

- How do you store something as complex as FpML or SEPA's ISO-20022 messages in a relational database?
  - FpML is typical, it has over 1000 elements and umpteen levels of depth
  - Normalising FpML would result in the mother of all databases and SQL queries up to a page long
- How do you manage multiple versions?
- Answer
  - Don't use a relational mapping
  - Simply store it as XML (in a CLOB) and extract the indices you need with XPath
  - Many databases (Oracle, Sybase, DB2 V9 etc.) offer XML data types but they usually don't implement all the schema features and slow the insert times down to a crawl
  - The true power comes from caching in Memcached, EHCache, GemFire, GigaSpaces, Coherence, Terractotta etc.

Logical Architecture

- It looks like and ESB but we're using a cache, the messages are Integration Objects



It's question time...