Why events and why DoRA

Presentation to RMIT
May & July 2013

This material is designed to be viewed in presentational mode as an aid to a discussion. Mike Kenyon
Authors K. Chandy & W. Schulte

Figure 1-1: Business pressures affect application architecture decisions.
Domain-oriented Reference Architecture (DoRA) - ambitions

**Ambitions**

- Reduce Integration cost
  - Enable effective & full servicing through choices of channels & devices.
  - Reduce time & cost to deliver new business products and systems.
  - Reduce cost of integration.

- Improve IT change responsiveness
  - IT systems adaptable to business process and organisational change
  - Enable more extensive product & service offerings that cross system support e.g. customer pricing, product bundling.
  - Increase system life expectancy (reduces depreciation costs & increase ROI) e.g. core accounting system longevity.
  - Allow decommissioning of ineffective / costly systems, ie inefficient legacy.
  - Reduce integration and system design / deployment errors redundant / applications.

- Reduce Business risk
  - Enable more accurate & timely sales targeting, customer & product profitability measurement (across activity type & channel etc).
  - Enable notifications/alert in respect to selected events.
  - Up to the minute customer events matched to patterns for risk management, fraud detection, regulatory & compliance, and service improvements.
  - Enable complete view of customer engagement and available in variety of practical forms to suit purpose/channel.

**Value of the new integration approach**

- Enable better business-integration (e.g. M&A)

*All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions*
Quick Scene Setting
All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions.
Integration will be grounded on business fundamentals of:

- BUSINESS Ownership
- BUSINESS Process
- BUSINESS Meaning of Data

Achieved by:

- Changing the way we change a Bank
- Grown from change-out
- Selected delivery of a few simple, corner-stone solutions
All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions.
QUICK OUTLINE Based on process and events, keep it simple

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions.
QUICK OUTLINE: Making integration simpler reduces effort to build/change/control

- Transform the meaning of data exposed during events and deliver to where it's wanted

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions
All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions

**Systems**
- Business Owned
- Autonomous
- Modular
- Grouped into ownership domains (that have a technical consistency for ease of re-configuring) eg to:
  - Plugin/unplug solutions
  - Flex and adjust domains
- Move towards a vision through *existing* change
- Change the way we plan for; design; deliver integration
- Have a few "kick-start" projects
Reference Architecture
Integration is the act of: making something start in one part of the business, as a consequence of something having happened in another part of the business.

**WHAT TO CHANGE**

1. Have one unifying change strategy.
2. Stop doing it wrong: We have built new systems only to integrate old ones – that makes it worse.
3. Plan to deliver the right integration designs.

**DOMAIN PRINCIPLES**

1. Separate business domains to own their systems.
2. Domains enable:
   - Multi-channel processing.
   - STP.
   - Real-time availability of information to those that need it.
4. Systems are independent from one another (no conflicting ties).
5. Universal events, to enable:
   - Re-use of messaging.
   - Plugging in of new systems.
   - Switching systems between domains.
   - Unplugging of old systems.
6. Inter-domain events link the business domains.
7. Customer events matched to know exactly how the Bank participates with customers.
8. Stores of universal events are held for convenient use by presentational systems & analytics.

**BENEFITS**

1. Cheaper to deliver; easier to change; needs less change; lasts longer; disposed of easily. Massively lower TCO.
2. New business products & services faster to market.
3. Adaptable to business reorganisation eg M&A...

---

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions.
Domain composition

Domain, whose business owner:
1) unilaterally requires, explicit integration contracts between the systems in **his/her domain**;
2) bilaterally agrees integration contracts with **other domains**;
3) responsible for the **end-end processing** supported by **his/her domain**.

Independent systems that:
1) have clear **business purpose & business ownership**;
2) are **provided** with information about **events** needed to achieve that purpose, in real-time;
3) keep their own information, under their own control, to achieve their own specific purpose, i.e. **selfish**;
4) **provide** information about **events** they create, when they happen;
5) are **self-contained** and **decoupled**.

Semantic-hub that:
1) is real-time;
2) transforms semantics; routes & forgets;
3) asynchronous;
4) decouples systems using transforms;
5) re-uses messaging via a third and central form (canonical);
6) may perform some minor enrichment.

**Business to business gateway and domain to domain gateway** that:
1) secures domain boundary;
2) converts syntactical forms;
3) is treated the same as any other system, by the Semantic-hub;

**Warehousing** that:
1) consumes same event types in same (canonical) form, wherever its source;
2) stores events from the semantic-hub in a common foundation layer;
3) foundation layer a consistent base for MI & analytics.

**Information service** that:
1) consumes same event types in same (canonical) form, wherever its source;
2) stores knowledge for effective exposure via drill-down, RESTful interfaces with conversational gateways;
3) one-way events (consume only).

**Conversational gateway and Channel agnostic router** that:
1) secures domain boundary;
2) converts syntactical form;
3) is real-time;
4) routes messages from presentational systems to systems inside the domain boundary.

**Presentational System** that:
1) ReSTfully requests information
2) receives & presents information
3) captures & emits new events

Normal challenges to DoRA concept:
1) how to **match** customers ?
2) how to **KYC** before on-boarding ?
3) how to **BTR**; collect transactions across all accounts and view them in different categories & sequences ?
4) why is workflow control only necessary to people based processing ?
5) can information services be more up-to-date than traditional sources ?

**DoRA has answers to all these challenges!**

**Bigger things solved:**
Product bundling
Customer centric billing
Customer centric limit management
Customer centric servicing & sales

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions
Ownership 1: a business owner owns “run-the-bank” domain and all system owners of that domain, in respect of their systems, report to their domain owner.

Ownership 2: a business owner owns each “run-the-bank” system including the semantic-hub and are the consumers of change-the-bank activities.

Our channels focus on presentational processing based on RESTful information suppliers from information services and core systems. Core systems are the target for events captured in channels.

Our information services and core systems support each of the presentational systems in the same way. Enabling multi-channel business processes.

Ownership 3: project owners own “change-the-bank” developments and are responsible for their own integration. NB1 the run-the-bank system owners may provide the skill-base to develop the change but will not have to fund it. NB2 No project has to develop the interfaces for another project.

Ownership 4: domain owners bilaterally agree inter-domain integration contracts.

Each Domain will have an architect responsible for modelling the domains current deployment & vision for deployment.

Each Project will be required to describe its change(s) in terms of the modelled architecture of the domains it will change.

The SDLC may change.

Integration planning will be based on business process support and be the corner stone to business & IT alignment.

Integration planning will be completed before projects design phases are initiated.

Systems decoupled by events.
Messaging styles compared

Mike Kenyon
Why send events – part 1 (axioms)!

BEST: to already have all the information needed to do one’s job

BEST: to share your information at the point of creation (when that information is to hand)

WORST: time, is to be asked for information when one is busy doing something else
Why send events – part 1 (build effort)!

BEST: No extra interface ends to be built

BEST: 2 interface ends to be built

WORST: 4 interface ends to be built
More if we apply:
Synchronous acknowledgements;
Responses to negative acknowledgements;
Rollbacks/compensation;
etc
Why send events – part 1 (orchestration, aggregation, composition pattern)!

This has poor reliability & high latency

This gets worse for nested requests. Reliability falls at a compound rate; latency is additive.
Why send events – part 1 (pure event pattern)!

Much better to provide the information in advance of it being needed, i.e. when it is first captured.
When one’s systems requiring data either cannot or chooses not to hold data “in case” then still use a request but do not nest! And request asynchronously!!
Why send events – part 1 (pure event pattern but with change system scope)!

Some improvement to the previous pattern’s latency can be had by adjusting the sequence that systems are messaged!
Why send events – part 1 (design, overview)!

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions.
Why send events – part 2 (performance)!

When events occur in operations, i.e., before they are needed, their messaging can be assumed to have: “zero” latency and 100% reliability.

1st pattern to finish

3rd pattern to finish

2nd pattern to finish

4th pattern to finish

Could be faster where it is possible to operate the calls to C, D & E in parallel but that is more difficult & costly to deliver. AND this pattern already has the worse performance & is the most costly to deliver.
Why send events – part 2 (reliability)!

SOA reduces reliability. When this drops too low one can redesign the integration. Alternatively, make one or more systems 99.9%. Adding a 9 doubles the cost of system delivery & change!!!

Why send events – part 2 (reliability)!

Reliability is not diminished with event oriented integration

Success = the compounded reliability of all systems in the service.

---

### Probability chart - events

<table>
<thead>
<tr>
<th>Systems F</th>
<th>Outcome probability</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>0.99</td>
<td>success</td>
</tr>
<tr>
<td>failure</td>
<td>0.01</td>
<td>failure</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1.0000000000</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Probability chart - services

<table>
<thead>
<tr>
<th>Systems A</th>
<th>Systems B</th>
<th>Systems C</th>
<th>Systems D</th>
<th>Systems E</th>
<th>Outcome probability</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>0.99</td>
<td>success</td>
<td>0.99</td>
<td>success</td>
<td>0.99</td>
<td>success</td>
</tr>
<tr>
<td>failure</td>
<td>0.01</td>
<td>failure</td>
<td>0.01</td>
<td>failure</td>
<td>0.01</td>
<td>failure</td>
</tr>
</tbody>
</table>

Add formulas and calculations as needed.
Why send events – part 3 (capacity/throughput)!

Let’s first conjure up the events that create the data needed in subsequent processing.

Watch the clock!
For the same timeframe throughput differs for different patterns.
Assume 10,000 transactions. How many run-time messages needed to complete processing...

NB events will have been sent in advance (when they occurred) and therefore not counted. Where the ratio of events used by a system is low and cannot be increased by selecting only those events with specific characteristics, then it is acceptable to request information.

Where the ratio of events used by a system is high then it is best to send events in real-time (and in advance of them being needed).

Sometimes it is possible to even adjust the processing sequence to enrich triggers prior to processing them.

It is rarely effective to use this kind of pattern.
Move towards brokered integration, no need to leap there

Use point to point in ways that decouple systems & enable reuse
Tolerate batch and service architectures until replacement beneficial!

Until legacy can emit events, tolerate batch and service oriented architectures

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions
Scenario discussions

Mike Kenyon
Accept new account application for existing customer!

Identify customer
Choose product
Select profile & use it to pre-fill application
Apply
Validate application
Expose application to rest of bank

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions
Complete application for existing customer!

- Expose application to rest of bank (from earlier process)
- Check for fraud
- Check for sanctions
- Check for credit worthiness
- Check for AML
- Decide if account can be opened
- Open account
- Issue tokens/cards/other facilities
- Enquire on account details/progress

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions
Re-matching for customer updates!

Expose a customer update to Party Matching (PM).

Eg could be end of process in previous slide but any core accounting/customer based system update to customer is exposed to the semantic-hub in the same way.

PM updates its view of a system’s customer.

PM does another best-guess & a re-match.

PM creates & exposes “updated customer event(s)*” BUT only if the best-guess or matching has changed!

Customer/account systems all are able to receive updates (if required) and update their own data (as required).

Systems have just received the “best” update available for customer data and cannot improve it. Nonetheless they may choose to hold a different view (eg a constraint to field sizes) which they then share with PM to assure effective future matching/de-matching.

A system’s own chosen updated view of customer should not alter the original matching or best-guess results and thus no recursive looping will exist.

Systems might undertake other, consequential, tasks eg categorising or rating customers. Such consequences are likely to be different events and will not be of interest to PM.

Recursive messaging is possible in any style of integration but is avoided with judicious application of business logic.
Capture & process a payment instruction!

- Examine recent transactions
- Select payment template & beneficiary list
- Select beneficiary
- Complete details
- Validate details
- Validate account use
- Check for sufficient funds
- Post accounting entries
- Execute payment
- Reverse failed payment postings
- Monitor payment progress

Channel abstraction layer:
1. Conversational Gateway
2. Screen mash-up

Semantic Hub:
- Channel banking system
- Channel agnostic payment validation system
- Account-use validation system
- Sufficient funds & limits check system
- Customer's transaction history
- Customer's beneficiaries log
- Customer's contact event log
- Communication/evidence image store

Business to Business Gateway:
- Core accounting system (batch postings)
- Core accounting system (real-time postings)
- Account posting engine
- Warehousing

All models shown are examples chosen to emphasise architecture, not intended to represent either existing or planned solutions.
The customer does NOT have to wait for confirmation!

*Customer gets on with other stuff; *they see their request is pending; *they see their request is noted; *they see their request is valid for use; *they see their request is approved by Bank; All displayed whilst they are doing other stuff!

*Customer waits for a response; *they see their request is pending; *eventually it is approved by Bank; They can now do something else.