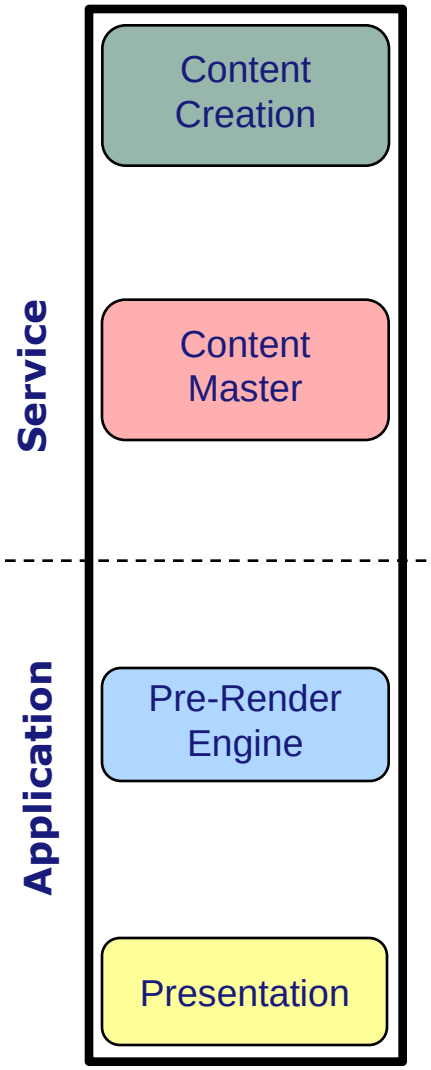
Content Distribution in an Event Oriented World

September 2008

Ian Koenig

Introducing a Pattern for Content Distribution

Content Distribution Pattern



As content traverses the network from creation to presentation, there is a standard pattern defining "best practices".

Content Creation – Focuses on the process of creating content including editing and modifying content. Content may exist in an "unpublished" state.

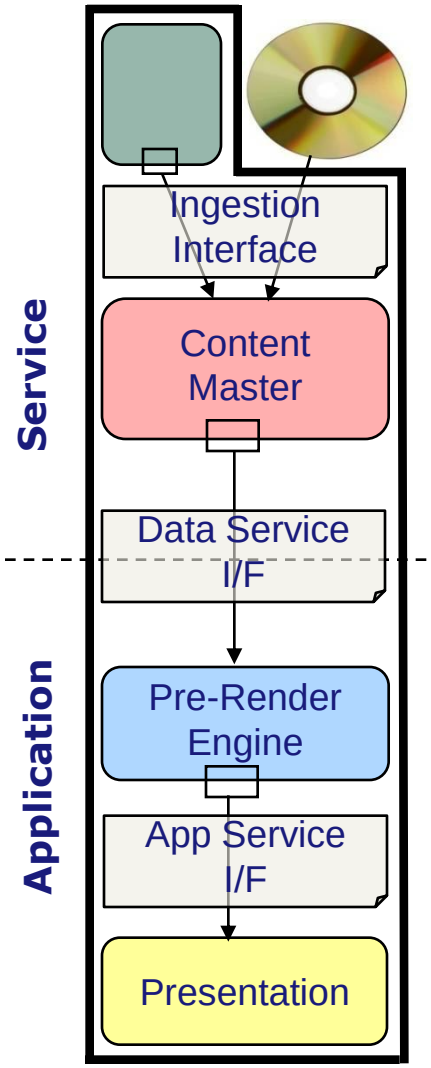
Content Master – Focuses on creating the "single version of the truth". This is part of Master Data Management.

Pre-Render Engine – Focuses on "making predictable queries fast". This often implies transforming the data model received from the master into one easily render-able.

Presentation – Is what you ultimately get.

Extending the Pattern for Content Distribution

Content Distribution Pattern



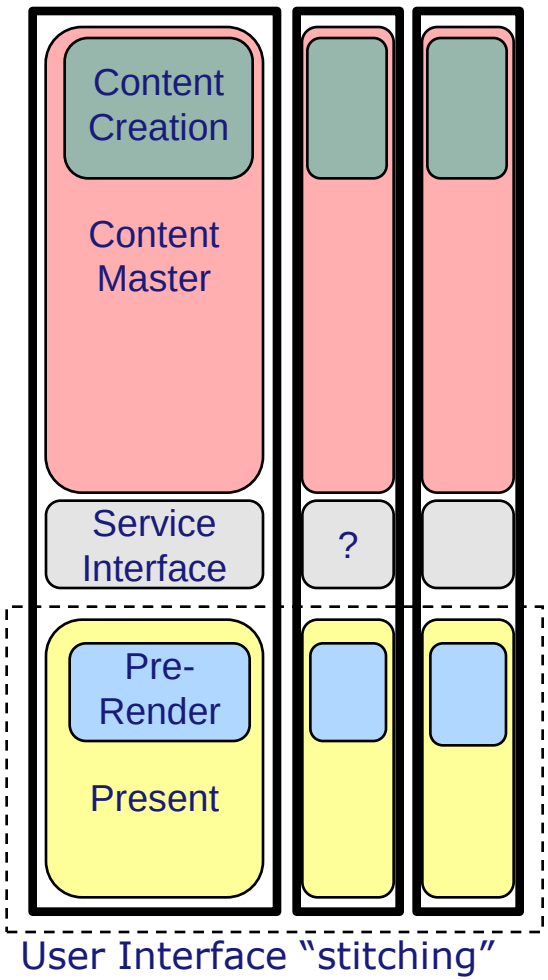
Often, the process of Content mastering involves aggregating content from multiple sources, including external ones

Ingestion Interface – The Boundary Interface (owned by the Master) that consumes content "published" by Creation systems and acquired from 3rd party sources.

Data Service Interface – The Boundary Interface between Content Masters and Application Pre-renderer Engines including the publish-side contract of the master and the consume-side contract of the pre-renderers

Application Service Interface – The Boundary Interface between Application Pre-renderer Engines and presentation engines (or other applications). Often, this is a web service

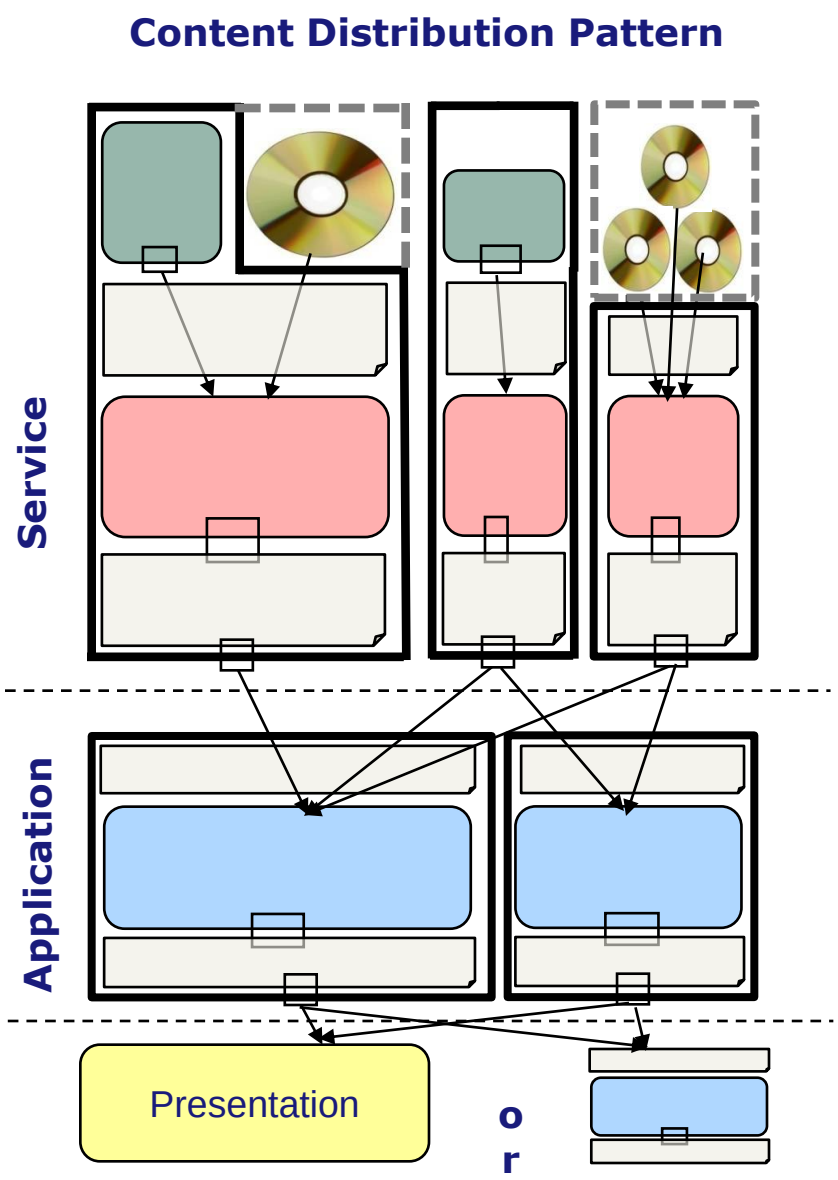
Why do we need this pattern?



Because:

- Most companies accrete content silos organically, either by acquisition or by the individual practices of individual businesses. Even if they did not, the concept of a Service Oriented architecture implies that as you carve the overall functional architecture into independent services, you need an information architecture to put the content based services back together in a consistent fashion
- A content silo is a content set that is tightly coupled between the content master, who collects and stores the content and the application processes that present the content for analysis / display.
- Many silos provide an API directly from the master DB, forcing the presentation layer to do all the pre-rendering "business logic", which is not properly isolated. Its also messy to aggregate content from multiple sources.
- Sometimes it is the U/I that "stitches" it all back together creating the "Frankenstein effect"

Content Silos are inflexible for multiple reasons



What we want is to decouple the tiers into independent services so that we can build more sophisticated applications and re-use content (both created and derived content) to the greatest extent possible

An example from the real world

Content Distribution Pattern

… If you squint just a little tiny bit …You might have actually seen this pattern before….

Content Sources

Ingestion Interface

Content Master

… And its just as valid for systems dealing with Stock quotes handling 100,000's updates per second

Data Service Interface

… But for quote systems (like anything) the interfaces, protocols and databases used must be fit for purpose. The pattern remains the same.

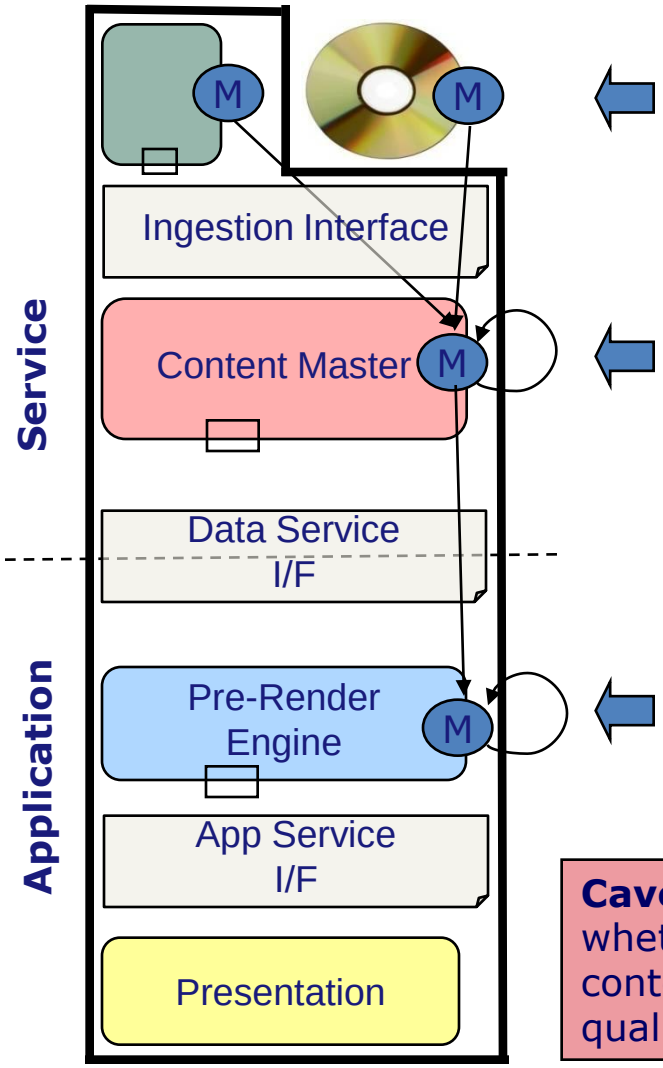
Application Pre-Render Engine

Application Service Interface

Presentation

Adding metadata to the model

Content Distribution Pattern



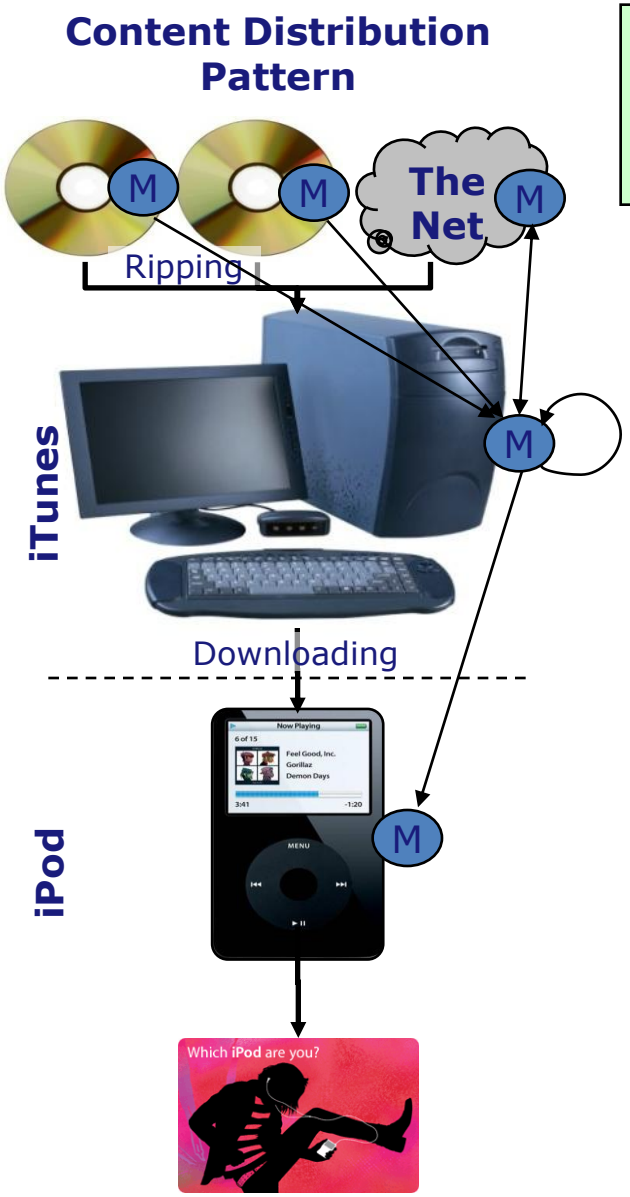
In addition to the data that flows through the system, associated metadata flows too.

Metadata - exists everywhere data exists. In general it is carried like an associated dataset through the system, but may have a distinct schedule by which it is refreshed.

In general it is a good idea to separate metadata that changes only when the data model changes (i.e. a new version is published) from metadata that can be changed dynamically (i.e. without changing versions). We call this distinction design-time metadata vs. run-time metadata.

Caveat: One person's data is another person's metadata and whether you treat an item as data or metadata can often be contextual and change over time. So it is important to always qualify the use of the term metadata in practice.

Metadata in iTunes

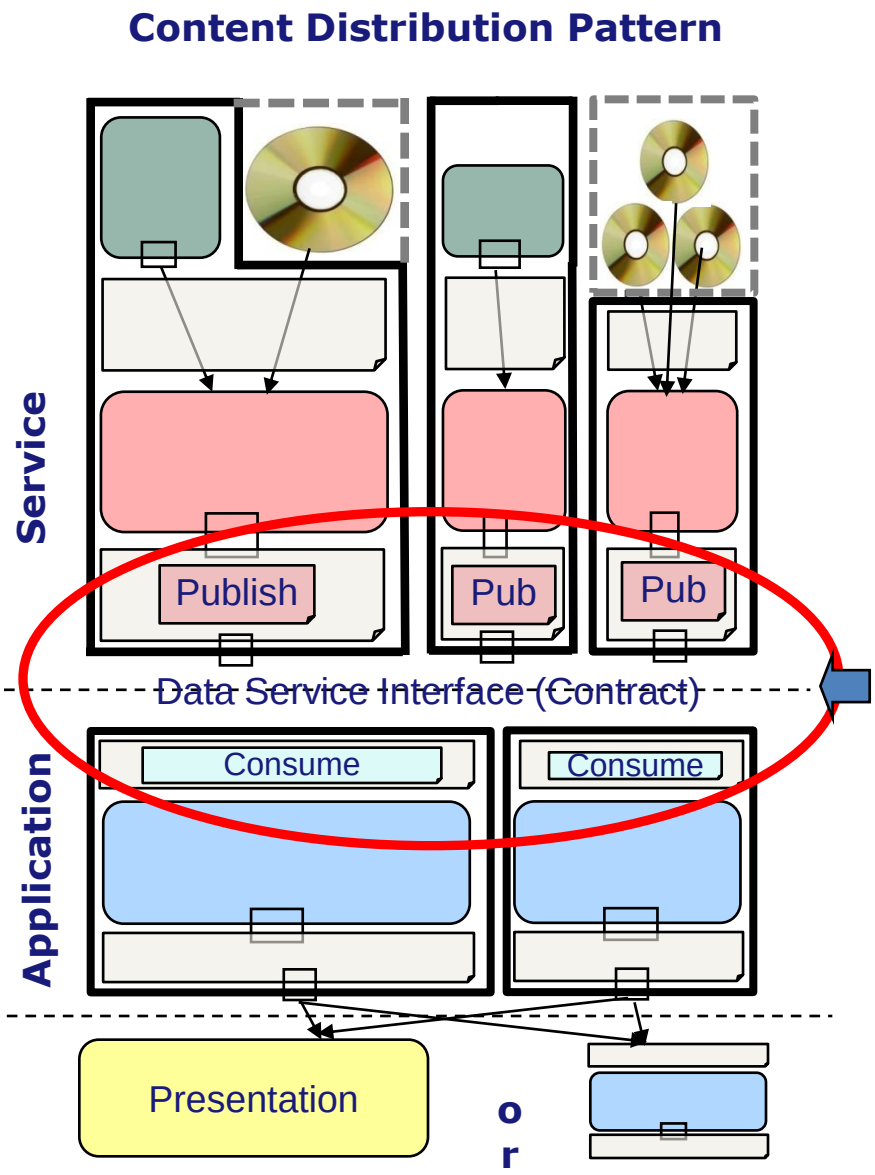


In the iTunes model, the "Data" is the digital representation of the song. The metadata is the descriptive information, such as: Artist, Genre, Album, Track, Song Title, and your personal content like: rating.

- Metadata is initialized from the source and may be edited / corrected in the master (iTunes on your PC). It also may be updated from a more dependable source or the content may be extended from a more complete source (e.g. loading album art from the iTunes store on The Net).

- Metadata is downloaded with the data and kept "associated", otherwise all you would have is a stream of bits.

Further extending the pattern



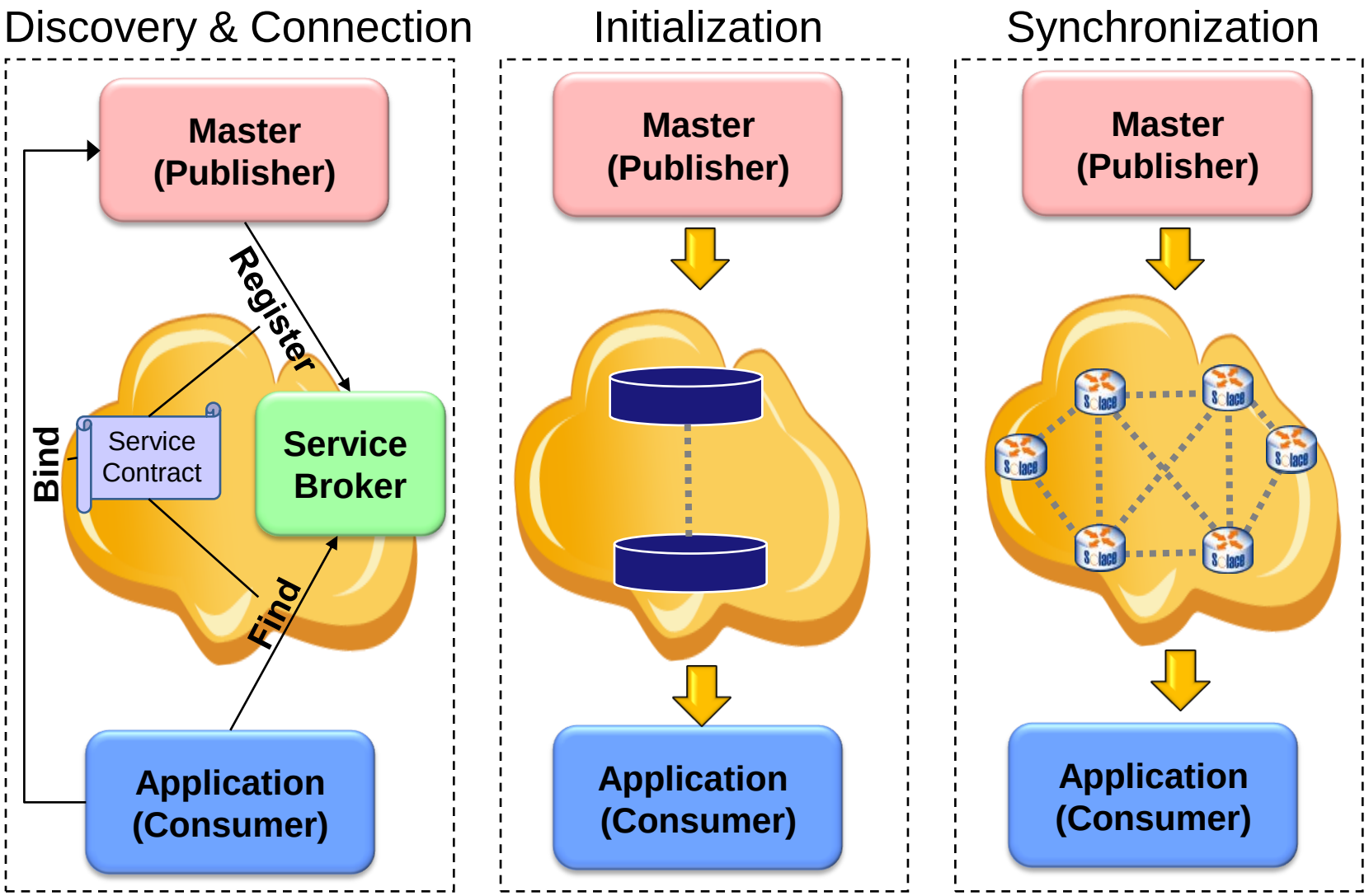
Often, the application pre-render engine aggregates content from multiple masters.

When this occurs, the application pre-render engine is creating unique new value and is therefore in essence a new "service".

Data Service Interface – What we've called the Data Service Interface so far is really two parts. The publish side interface that is owned by the master and the consume side interface that is owned by the application pre-render engine.

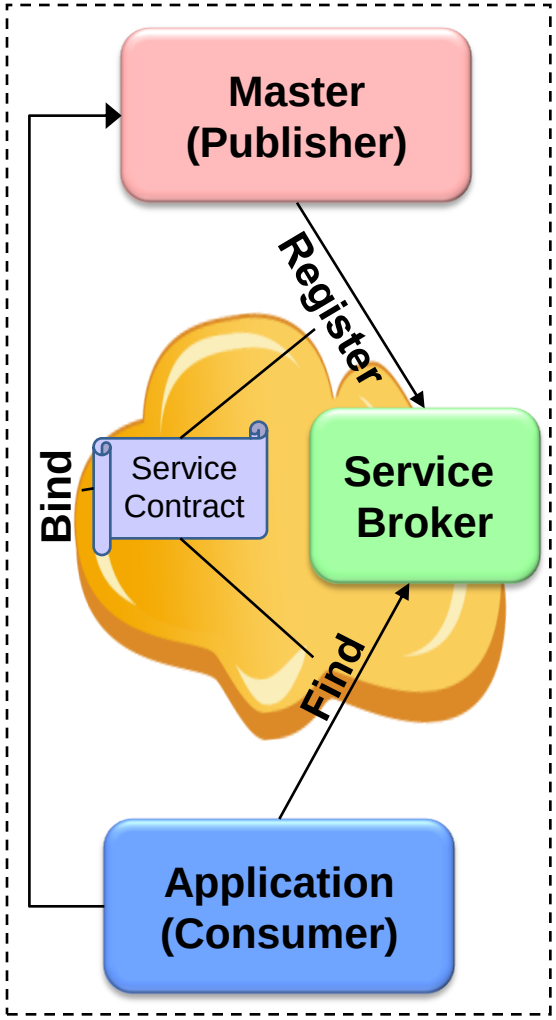
It is the contractual relationship between these two sides of the interface that is key to proper service orientation and loose coupling.

The Data Interface Problem Domain



Data Interface: Discovery & Connection

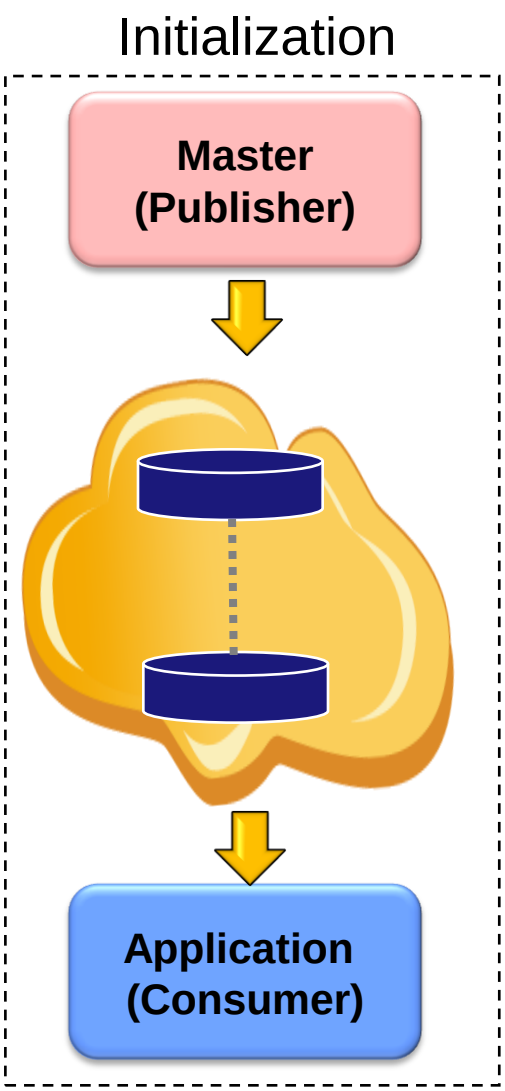
Discovery & Connection



In a typical environment, it is desirable for publishers (masters) and consumers (applications) to be de-coupled. This allows independent versioning and cycle-times. It also allows new applications and new publishers to be added to the "network" without breaking anything.

- Publishers should not a priori "know" consumers
- Publishers therefore need a "broking facility" so that consumers can find them.
- Loose coupling and versioning are of paramount importance
- SLAs and rebuild / update schedules are part of the interface contract.
- UDDI (Universal Description, Discovery and Integration) is an appropriate model to follow (http://en.wikipedia.org/wiki/UDDI)

Data Interface: Initialization



The master must provide a way for the application to build from an empty state. Often this is through an extract file that is FTP'ed across the network and then loaded into the target database.

A slight advance over this model is to create a full extract on a schedule and then a Delta file from that point to the present.

Initialization

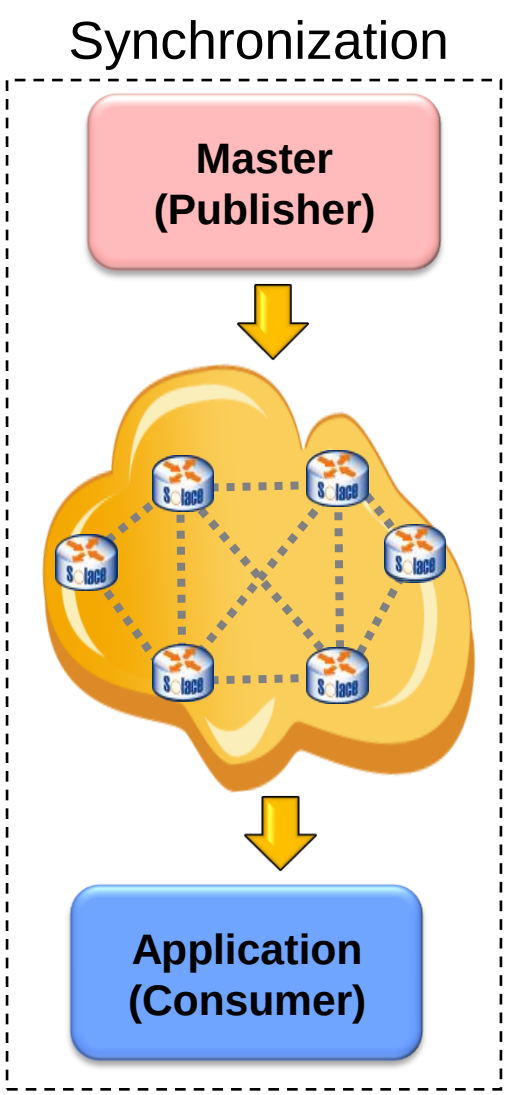
Full Image

An application pre-render database / cache upon initialization or after recovery needs a mechanism to fully rebuild its state from the beginning of time.

Cumulative Delta

A cumulative delta includes all changes from the previous full rebuild, so to initialize, an application generally processes the most recent Full rebuild AND the most recent Cumulative Delta

Data Interface: Synchronization



Once an application has initialized (reached a synchronization point) it begins to go stale, it needs to stay updated as changes are made to the master.

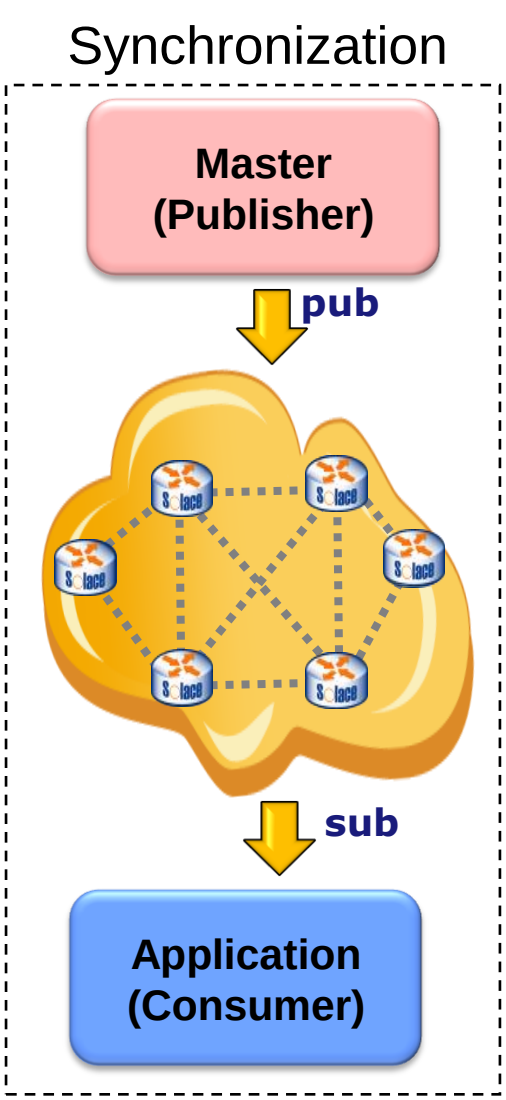
- If the master provides no synchronization interface, then the application is forced to periodically start over (i.e. re-initialize)
- Alternatively, masters can package all changes from a previous point into an "Incremental Delta" file and distribute that.

But other forces are at work …

- The "more up-to-date" the application is (i.e. the more its state represents the state of the master) the more value.
- Large files tend to clog up the network, so operationally we tend to move these during off-hours. Unfortunately that can mean small time windows and if many masters are moving many large files at overlapping times, switches can overload and file transfers will fail.

Which just moves us in the direction of moving smaller bits of data more often, which is really no surprise since market data systems have been doing that since they were first invented.

Data Interface: Event based Synchronization



The highest value and most efficient mechanism for synchronizing application (consumers) from content masters (publishers) is for the master to publish events signifying that a change has occurred.

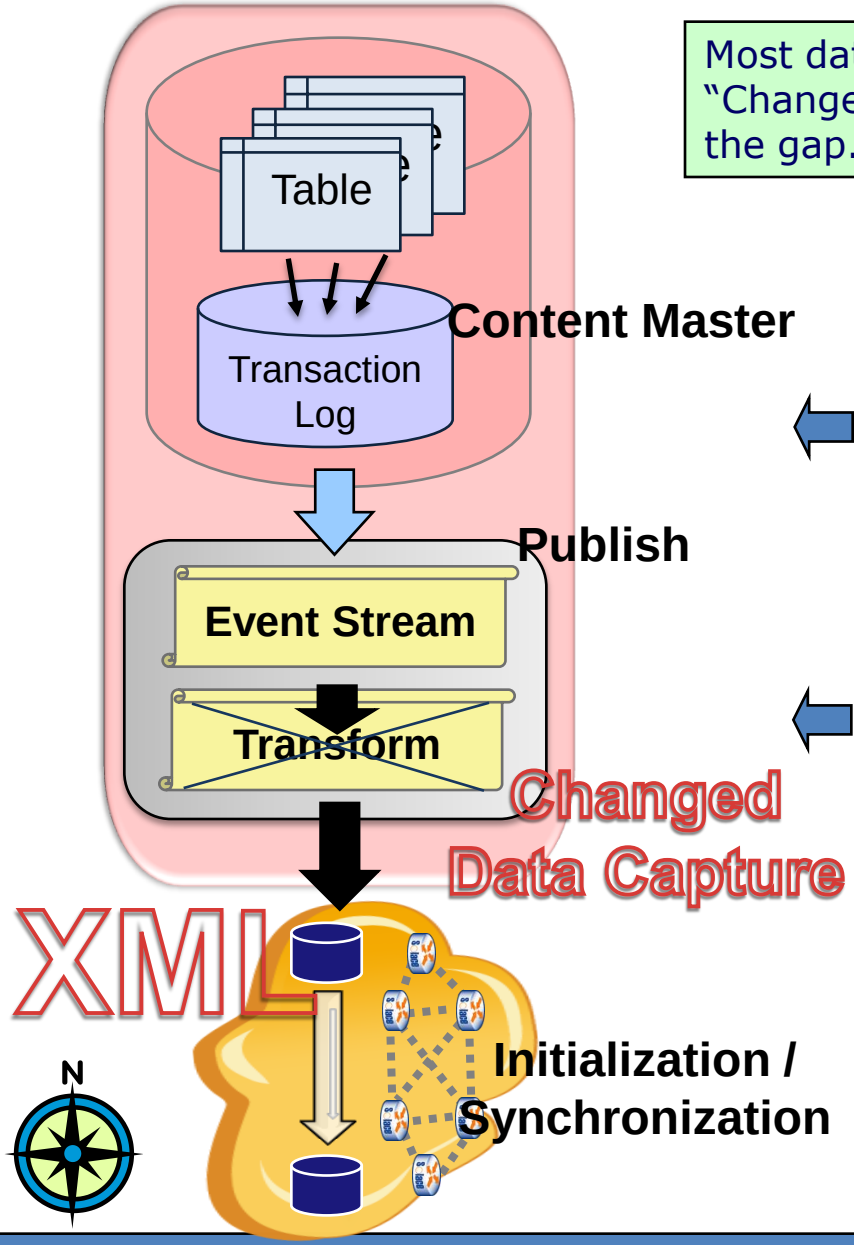
- When the master creates an event message representing a change made to its database, it is expected that message represent an entire transaction (i.e. is fully process-able).
- The ordered set of event messages from a single master constitutes an event stream
- The set of event messages across all content masters constitutes and event cloud.

The internal physical data model of the master should be transformed in a canonical data model for distribution. This is a data model where there are no fragile keys. In other words, all internal keys used to link tables internal to the master have been removed and all keys used across masters are represented by proper public identifiers or properly mastered permanent GUIDs to external Entities (another discussion)

Example Envelope for an Event Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentEnvelope majorVersion = "1" minorVersion = "0.7"  xmlns=""
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:item="http://contentSet.schemas.myCompany.com/2008-07-20/ >
    <Header> Some Header Info </Header>
    <Body " majorVersion = "1" minorVersion = "2.0">
        <ContentItem action="Add">
            <NewsML xmlns ="http://iptc.org/std/NewsML/2003-10-10/">
                Your NewsML
            </NewsML>
        </ContentItem>
        <ContentItem action="Modify">
            <NewsML xmlns ="http://iptc.org/std/NewsML/2003-10-10/">
                Your NewsML
            </NewsML>
        </ContentItem>
    </Body>
</ContentEnvelope>
```

Changed Data Capture as an Enabling technology



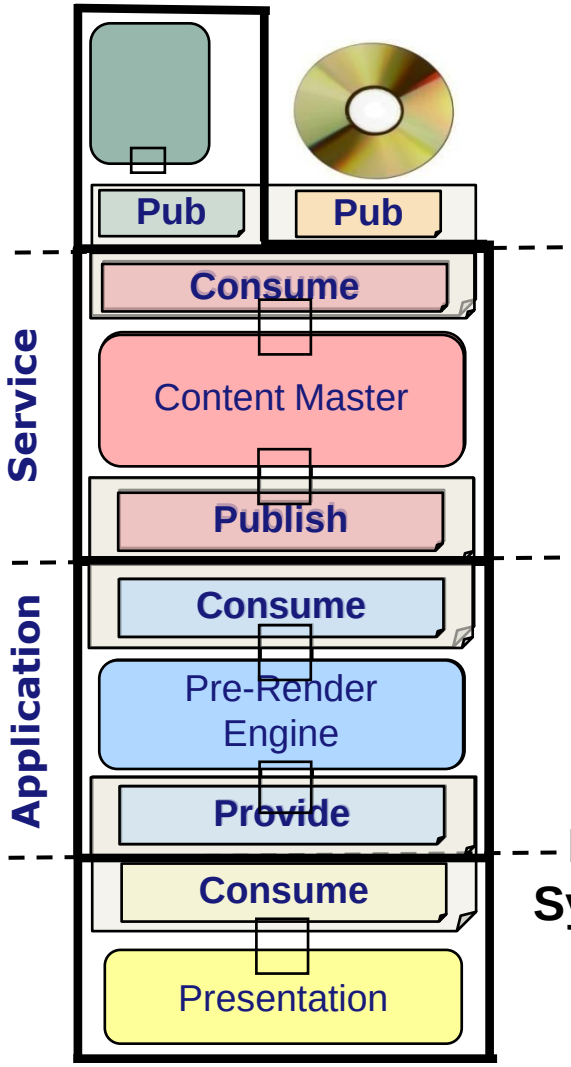
Most databases were not built to propagate changes, but "Changed Data Capture" technology can be utilized to fill the gap.

Log Mining – is a technique that watches the transaction log that modern databases use to capture all changes as they are made.

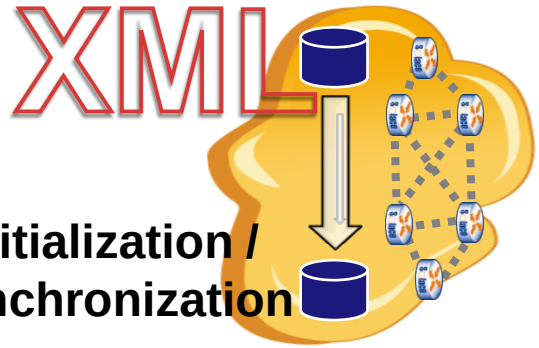
Transformation– The final step is transforming the transactional changes made to the databases to XML messages that capture the "business event" process-able downstream.

Event driven Content distribution Services

Content Distribution Pattern



With a keen eye towards encapsulation and loose coupling, and assuming that the application pre-rendering tier adds "enough" value, we can view each tier as an independent Service in an SOA, one which has an event-oriented interface, one which has an inquiry-response interface

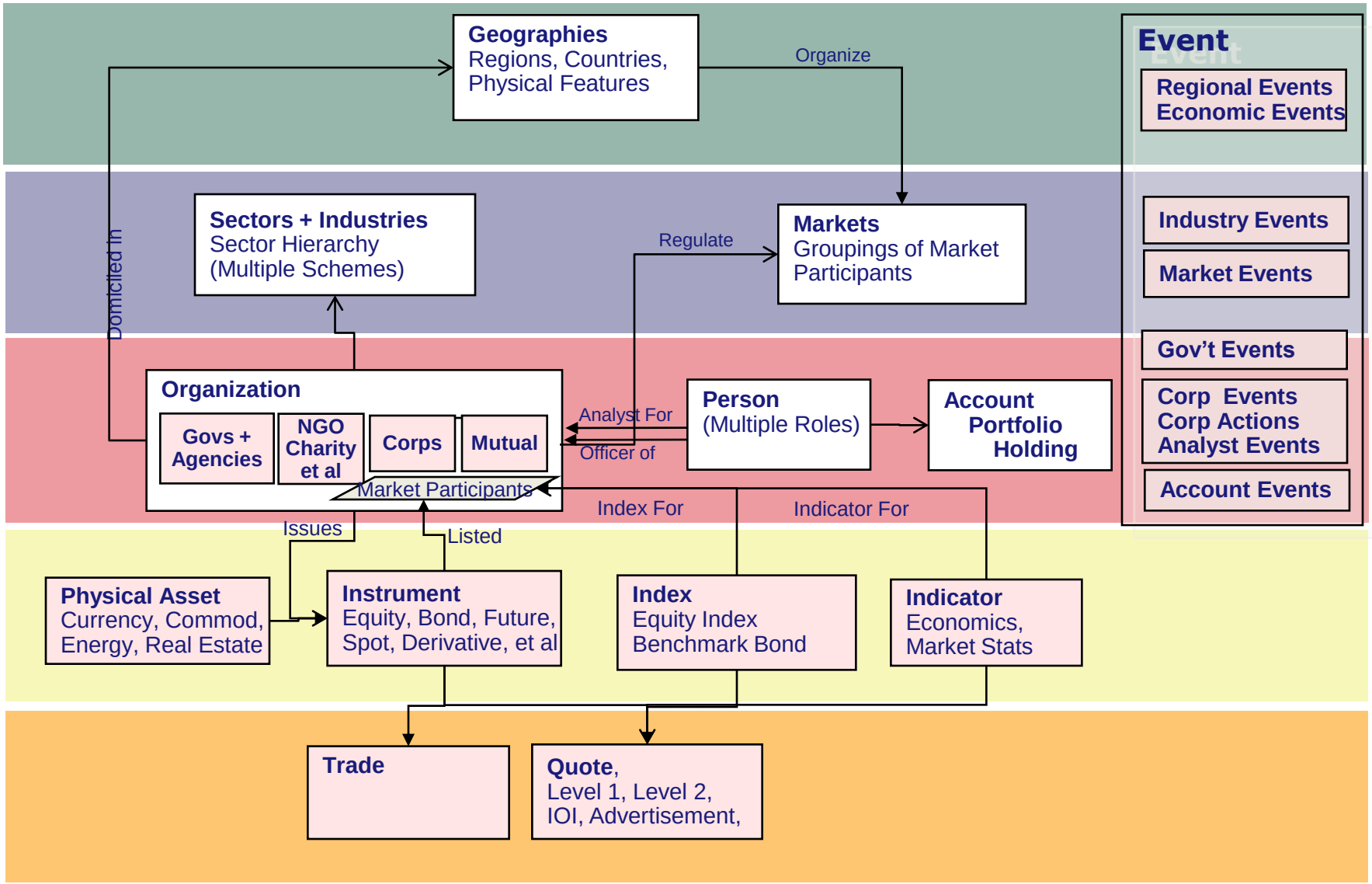


The End

Everybody’s infrastructure needs improvement

APPENDIX

A Financial Entity Model – to link Content sets together

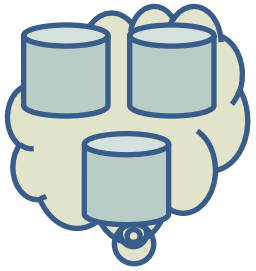


Content Masters perform Master Data Management

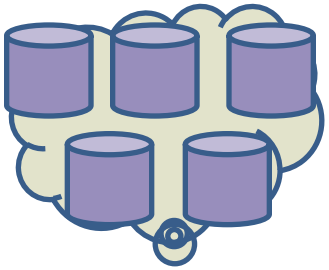
- There are two types of Content Masters: Entity Authorities and Data-set authorities.
- Entities are those Content sets (e.g. Companies, People, Geographies, Instruments, et al) that link the other Data-sets together (e.g. in the Financial world: Company Fundamentals, M&A, Company Ownership, et al)
- Entities are joined via Relationships. An RDF-style triplet Resource Description Framework approach is used to do this (http://en.wikipedia.org/wiki/Resource_Description_Framework)
- It is considered Best Common Practice to assign all Entities Permanent GUIDs to uniquely represent them and to use these permanent GUIDs as early in the API call tree as possible in deference to mnemonic aliases.

For example "C" is not the symbol for "Citigroup Inc." It is the symbol for the NYSE quote for Citigroup Common Stock, but only since Chrysler Corp was acquired by Daimler and gave up the symbol "C", which it had for its common stock. So not only does the symbol not actually represent the company consistently in time, many companies issue ADRs and list on multiple exchanges making the use of quote symbols as company identifiers highly problematic

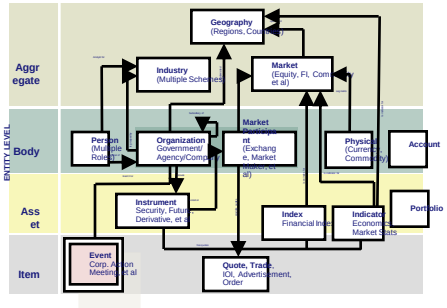
Entity Authorities



Data-set Authorities



Entity Model



Content is Pre-Rendered to make "predictable queries fast"

- Content Services distribute Content across their Data Interface in their Canonical data Model.
- The Services Canonical data model is the data Model optimized for distribution and specific to no one consumer.
- When an application consumes content from a service, its first order of business is to transform that content into something quickly renderable / presentable through its Service Interface.
- It is not allowed for the Pre-render step to create new content (otherwise the new content is not really mastered), except under rigorously controlled circumstances

For example, it is acceptable for the pre-render step to calculate data "on-the-fly", as long as all of the values needed for the calculation are mastered and the formula upon which the calculation is based is mastered.

Content Distribution – Ten Governance Policies (1 of 2)

1. **Human Interfaces:** Presentation servers access content from the pre-Render layer through a well defined Service Interface. They never access Content Masters and Data interfaces directly.
2. **Application Pre-render Databases:** access data from Content Masters through a well defined Data Interface (API). Pre-Render databases / caches are Copies. They do not create "new" data except they may perform 'on the fly' calculations as long as the result is not persisted (ex. Currency conversion)
3. **Content Masters**: contain the "single version of the truth" for all data, created, acquired or derived.
4. **Scalability:** Content Masters tend to scale proportional to the amount of content. Application pre-render databases / caches tend to scale proportional to both amount of content and usage. Content Masters should be architected to scale vertically and Application pre-render databases should be architected to scale horizontally
5. **Physicalization:** Content Masters and Application pre-render databases / caches belong in a core network in the Data center (i.e. not a DMZ). Presentation / Human Interface databases / caches belong in the DMZ. In a three network zoning model (i.e. where there is an application zone between the DMZ and the Core network zone, the application pre-render database / cache belongs in the application zone (duh!)
6. **D/R:** Content Masters should have synchronized copies in at least two Data Centers, preferably in the same geo-region. Masters should be backed up and restored from tape in a disaster. Application pre-render databases should be rebuilt from peers and synchronized to the master. They may not need to be backed up to tape at all and hopefully never need to be restored from tape. The same goes for presentation caches. Don't confuse D/R and HA.
7. **Data Elements (Facts):** (i.e. columns) The Content Master is responsible for creating a unique Data Element Identifier for every piece of data it owns.

Content Distribution – Ten Governance Policies (2 of 2)

8. **Data Items**: (i.e. rows of data) Only the Content Master is permitted to create an item of that data class. The full list of Content Items (i.e. Entities and Datasets) is defined by the Information Architecture. The Content Master is responsible for allocating a permanent GUID for every data item it creates
    a. Once a permanent GUID is allocated it may be sunset (if the data item it is associated with has its "Effective To" date set). It may never be reused or take on a new meaning.
    b. Only the Permanent GUIDs of Entities should be used to link content between datasets

9. **Copying Data:** When content is copied between databases, the following contract applies:
    a. It is the responsibility of the copy to ensure it stays synchronized with the source. If the source does not support a "push" style interface, then it is desirable that the copy is periodically "dumped" and rebuilt.
    b. The copy is responsible for preserving the "name" of the Data Elements and the FactIds (for Traceability). Copies may not rename data
    c. When a master copies data from another master (e.g. for the purpose of deriving new data), the copy may not be on-passed from that DB; only the derived data can.
    d. The exception to the on-pass rule is for "symbology and classification data"

10. **Value Add Content**: is derived and is distinct from vs. "As Collected" content
    a. When new content is created (derived), this "value added" content must be mastered. This could either be the value data itself or the business logic which drives the calculations in the Application pre-render database.
    b. It is best practice to master value-add data with the data class that it is most closely aligned (rather than create a new class and a new master)
    c. The process of creating value added data must not "lock out" the process for adding new data or modifying data.

The Data Interface

- In the Content Distribution pattern, the Data Interface is defined to provide the loosely-coupled interface contract across the boundary between the content master and the application pre-render database. It effectively governs the Service / Application boundary and the Producer / Consumer boundary (i.e. its pretty important)
- A bus – style implementation of the data interface providing pub/sub capability is considered best common practice. The subscription protocol could either be topic based (like JMS) or content based (like provided by emerging content routers (e.g. http://www.solacesystems.com/)
- The Data Interface is:
  - One way
  - Encoded in XML in the Canonical Data Model of each Content Master
  - Keyed by non-Fragile Unique Permanent Entity GUIDs
  - Loosely Coupled: Sources should not 'know' targets. Sources publish. Targets subscribe. The Data Interface Bus mediates. Interface contracts are enforced.