The Coalesce Data Abstraction Layer:
A Micro-services Architecture
for Unified Semantics, Pedigree and Polymorphic Views

Brought to you by:

InCadence
STRATEGIC SOLUTIONS

Visit our Coalesce Open Source Website at:
https://www.github.com/incadence/coalesce

For further information about the Coalesce System contact:
Michael C. Daconta, VP of Advanced Technology
(mdaconta@incadencecorp.com)
and
Dave Boyd, VP of Data Solutions
(dboyd@incadencecorp.com)

Table of Contents

Introduction

The Coalesce System is a data abstraction layer that unifies diverse storage engines into a cohesive whole to improve the performance, pedigree, richness and semantics of your data storage and retrieval. The Coalesce system is in use today in several large-scale production systems delivering significant benefits over conventional data storage engines. On two of those systems, Coalesce was chosen to meet a plethora of difficult requirements that could not be satisfied with a single type of database. Specifically, when you need to support relational queries, geospatial queries, free-text queries, pl-3 security, cross-domain transfer, metadata versioning and robust pedigree – coalesce is the only system that can deliver those features in an easy-to-implement, unified approach. A unified approach with a single data abstraction layer is important because data that is isolated in multiple, uncoordinated storage systems tends to get out of sync causing parts to become outdated, erroneous, or non-authoritative. At the same time, there are some data features that are currently combined within single storage engines that are better separated into different engines – examples of this are version-control, linkages and free-text – while all of these can be accomplished within a single, relational database – it will not do them well; instead, linkages are best done with a graph database; version control is best done in an XML or a document store and free-text is best stored in a text search engine. So, colloquially, we are proving the old adage that "one size does not fit all" by separating what is best to be separated and unifying what needs to be unified. This is accomplished by wrapping multiple individual storage engines into a single data abstraction layer. In essence we "coalesce" multiple storage engines into a single solution as depicted in Figure 1.

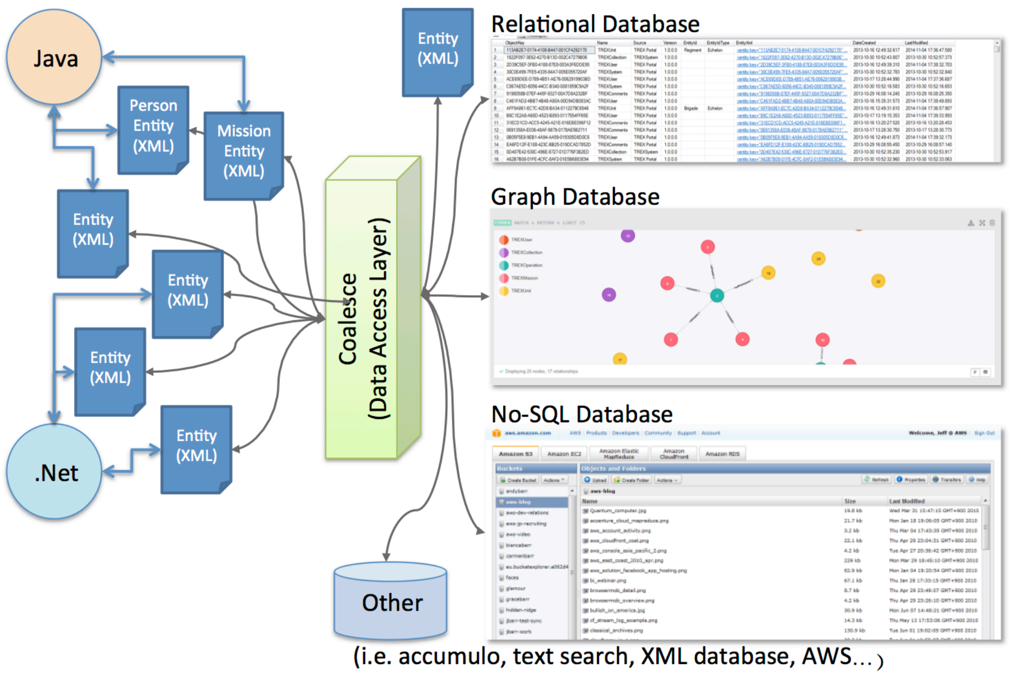

Figure 1 A high-level view of the Coalesce Architecture

In figure 1, we see your current systems/applications as the Circles representing either a system in either Java or a Microsoft .Net programming language. In the middle of the diagram, represented by the Green box, is the Coalesce System. On the right side of the diagram are one or more data storage engines. Coalesce serves as a unified data access layer to multiple storage engines. Now, let's delve into some specific problems plaguing traditional data storage systems throughout Government and industry.

The Legacy Data Architecture Problem

Having served as the Metadata Program Manager for the Department of Homeland Security (DHS) and as team lead for the Federal Enterprise Architecture (FEA) Data Reference Model (DRM), it became evident, through numerous examples, that large organizations (private and public) suffer from three systemic problems: Stove-piped Systems, Brittle Schemas and Single-Scoped Storage. Let's examine each in detail:

Stove-piped Systems

A stove-piped system is an enterprise system that follows a tightly-coupled, vertical integration pattern. By vertical integration, we mean that the User Interface (UI) only communicates with a single middle-ware or business logic server which, in turn, only communicates with a single data storage solution as depicted in Figure 2.

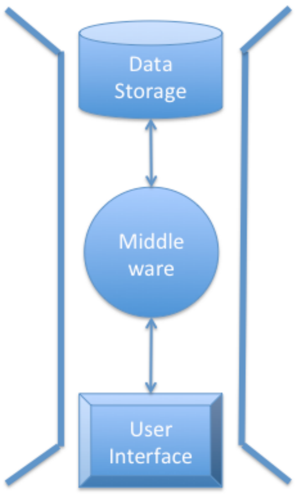

Figure 2 Stove-piped System

It is important to understand that this is the default case for almost every IT system because it is the simplest and fastest way to satisfy the goals of that project. The problem with developing a system with blinders on is that it is oblivious to all the other systems in the organization. Those blinders are what cause disparities between systems across the organization. What is needed is for programmers to not program towards a specific back-end, but to program towards a single data abstraction layer like Coalesce.

Brittle Schemas

A schema defines the structure of your data models and metadata models. Specifically, a schema defines the entity names, field names, relationship names and corresponding data types. For example, if you were designing an "Organization" Entity it would have fields like "name", and "creationDate" with data types of String and Date respectively. Defining your entities, fields and relationships should never be done in isolation because of the tendency towards stove-piped systems as discussed previously. Besides that semantic stove-piping which may not surface as a problem until years down the road, a more immediate and on-going problem arises almost before the ink is dry on the very first data model – change happens. System requirements change (or are misinterpreted), functional needs change (or new ones are requested), or mistakes were made in the original models. Unfortunately, almost all traditional data storage engines and especially relational database systems suffer from brittle schemas that are not flexible under change and thus break the other parts of the system. This is the classic "brittle schema" problem. When you change a schema by adding, modifying or deleting entities, fields, relationships or data types your system often fails in the following ways:

- Not allowing the change at all.
- Forcing application code to be rewritten.
- Forcing existing data to be deleted or transformed.

Simply stated, brittle schemas increase cost and introduce delays in the schedule. Anything a data storage solution can do to ameliorate this problem is a significant improvement over the status quo.

Single-Scoped Storage

Similar to the root cause of stove-piped systems (i.e. narrow thinking), most traditional storage engines are designed and optimized for a single, specific type of data. Relational database are optimized for structured, tabular data. Graph databases are for graphs (linked data). Document stores are optimized for text documents. Geospatial engines store geospatial data. Search engines store free form text. This singular scope results in one of two outcomes – engines that are optimized for one type of data yet shoe-horn in other types in an inefficient manner; or solutions that leverage multiple engines and handle synchronization between the engines at the application level.

The Coalesce Solution

The Coalesce Unified Data Abstraction Layer solves the enterprise problems specified above. Let's now examine how Coalesce solves those problems. We do this by defining a set of fine-grained services that are divided into jobs, tasks, plug-ins and providers.

Micro-Services Architecture

In Coalesce, we have applied the concepts of micro-services to data storage and retrieval services. A service provider is at the top level and it exposes an endpoint for each service provided. The service endpoint is either a SOAP or REST service. Each service uses Jobs and Tasks in a worker framework where a Job is further sub-composed into one or more subtasks that can run independently. For example, the job may be "Entity Persistence" with Create, Read, Update and Delete tasks. These services are deployed in a Karaf container and further leverage OSGI modularity features and the OSGI bundle lifecycle. All Coalesce services use a contract (or "interface" in the Java programming language) which allows the service implementation to be dynamically loaded at Runtime to support different implementations. The most obvious example is the Coalesce "Persister" interface with methods like "saveEntity()", "createEntity()", etc. Any number of Persisters, Cachers, Compressors, Synchronizers or Searchers can be created, configured and dynamically loaded into a running Karaf container. As stated before, a Karaf container is a lightweight Java application server that supports the OSGI modularity framework. Karaf exposes its REST or SOAP endpoints via a built-in Jetty web server. A coalesce abstraction layer is typically composed of one or more Karaf containers that front one or more data storage engines.

Unified Semantics

More important than the architectural flexibility afforded by Coalesce Microservices is the unification of your data and metadata semantics across all your data storage engines. Semantics refers to the meaning of your data and Coalesce provides you with a consistent, yet flexible framework that both structures and organizes your application data. The Coalesce Semantic Framework consists of a metamodel, templates, Coalesce Entity and Relationships. Let's examine each in detail and the benefits they provide.

The framework begins with a metamodel which both underpins and frames all the higher-level data, metadata and information concepts it supports. A metamodel is a model of models; or, in terms of data, a model for creating concrete models. Specifically, the Coalesce metamodel defines all the core data structures you need in a hierarchical structure to include entities, sections, record sets (equivalent to a relational table), records, fields, list fields, dynamic fields and data types. These generic data structures can define data of any type and provide additional features like unique identifiers for each item, time stamps on each item, field-level classification, versioning, changing history and source information. This metamodel is stored in a human-readable text document in XML format (Note: there are alternate formats like JSON available). A metamodel allows us to think about data manipulation in a generic way and is the key tool for alleviating the brittle schema problem. Furthermore, using the generic concept of a Coalesce entity in our service calls frees us up from creating brittle "data-type" specific services. Less brittle data and services significantly reduce maintenance costs for IT systems. It also can reduce initial development costs in the later stage when change becomes more costly.

As we previously stated, a metamodel enables the creation and support of domain-specific data models. For example, if the domain of your IT system is law enforcement, then you want to create data models for concepts like Defendant, Officer, Arrest, Court, etc. These domain-specific concepts are modeled as coalesce entities via Coalesce Templates. An "entity template" specifies your domain-specific names for things, how your entity is organized into sections, field names (also called attribute names) field data types and field-level validation. Furthermore, coalesce templates can define which fields are "searchable". Searchable fields can be stored separately by a persister in order to optimize the performance of queries against those fields. For example, in a relational database persister, search fields would be stored in a separate search table in order to provide efficient searches. This notion of distinguishing the needs of search from the needs of storage and modeling is a key Coalesce design objective as depicted in Figure 3.

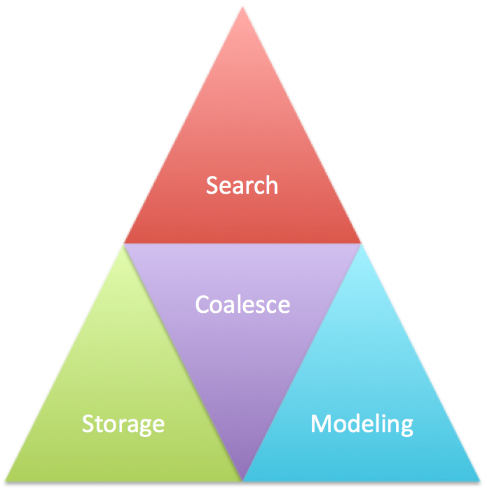

Figure 3 Distinct Dimensions of Data Storage & Retrieval

Separating Search, Storage and Modeling sets Coalesce apart from other "one-size fits all" solutions.

To manage templates, the Coalesce System provides a template registry, visual template editor, and template services. This could provide the core of a robust metadata management system to consolidate an organization's data models across the enterprise.

The Coalesce Entity is the central concept of the Coalesce Data Abstraction Layer. As discussed above, Coalesce entities are defined by a template and organized within the Metamodel. The Coalesce Entity is also supported by the Coalesce micro-services we discussed above. A Coalesce Entity is stored in multiple ways including a default, XML document representation. There may also be one or more "search" representations. Finally the ability to generically pass around and manipulate textual representations of entities simplifies your IT architecture and enables robust versioning, pedigree and cross-domain transfer. These features especially benefit from the concept that an entity can contain a running tally of its state from cradle-to-grave (of course, that "running state" can be easily and efficiently pruned from the document tree if necessary).

Finally, a key component of the Coalesce metamodel is the ability to create named links, or relationships, between entities. This native support for graph concepts elevates Coalesce from a data storage abstraction to supporting robust knowledge-base concepts. For example, Coalesce entities can easily be modeled with robust relations like genealogy, friendship, part-whole, and other custom relationship types. This raises relationships to the level of first-class objects in the Coalesce abstraction Layer and even enables connections to entities stored in previously stove-piped systems.

Pedigree and Cross-Domain Features

Linkages are critical to enabling Coalesce to support the storage and retrieval of your data's pedigree and provenance. Though the definitions for pedigree and provenance are somewhat vague and overlapping, we define them as follows:

Provenance captures the source or origin of the data.

Pedigree captures the event, change and ownership history of the data.

We have leveraged Coalesce to implement robust pedigree and provenance capture for a Navy Big Data project. While there are several pedigree and provenance models to choose from; we used a model that leverages pedigree event entities and Coalesce linkages to tie a pedigree event to every activity executed on the data (including its creation and origin) from cradle to grave. Robust pedigree and provenance gives you confidence in your data and is vital to the creation (and validation) of "Authoritative Data."

Since Coalesce Entities are stored in their base representation as XML documents and possess field-level security, they can easily be pruned for cross-domain transfer. Additionally, Coalesce entity templates can be converted into distinct XML schemas for cross-domain validation. Coalesce has been proven to effectively work with Cross-domain guards to enforce transfer rules and safely pass Coalesce entities across Security Classification domains.

Polymorphic Data Views

Polymorphism is an important feature of object-oriented development. The word is literally translated as "many forms". For software development, it allows multiple classes to all implement the same single interface. Thus, if I have a "speak" method and three Animal classes that implement it – Dog, Cat and Cow – I would call "speak" on each one and they would respond accordingly with "Bark", "Meow" and "Moo." Thus, one interface, many forms of implementation. Coalesce applies this principle to your data by providing a single abstraction layer over many data storage engine implementations. This empowers your developers to satisfy multiple data storage requirements within one clean and consistent interface. Thus using a single, non-proprietary Coalesce interface you can retrieve data in document form, relational form, geospatial form, textual form, graph form and any other data type that implements that interface. Coalesce interfaces are separated into hierarchical "capability" levels representing various degrees of conformance. Given that Coalesce includes persistence to Big Data Stores "Out of the Box," Coalesce is a robust approach to achieve the concept of a "Data Lake." A data lake is a fairly recent concept proposed by a few Hadoop vendors which basically translates to "store all of your data in Hadoop, and specifically, the Hadoop File System (HDFS)." While the simplicity of this concept aligns with enterprise goals and the elasticity of HDFS enables unlimited volumes of storage, the concept proves impractical in practice without a mechanism for bridging existing data models (via a metamodel) and polymorphic views of the data while maintaining change control, pedigree and provenance. Thus, Coalesce is your easiest and most robust approach for actually achieving the vision of a data lake.

How Coalesce Solves the Data Architecture Problem

At the start of this paper we examined three systemic problems that plague most enterprises: stove-piped systems, brittle-schemas, and single scoped storage. To that we should add a new requirement for data architectures: Big Data Integration. After many years of working on partial solutions to these problems, Incadence Strategic Solutions is proud to offer the Coalesce Unified Data Abstraction Layer as a robust, comprehensive and open-source solution to these problems. See https://www.github.com/incadence/coalesce to download the software and see a demonstration of its capabilities. Let's recap how coalesce solves each of these problems:

- Coalesce wraps and bridges your disparate stove-piped data storage engines into a unified abstraction layer.
- Coalesce alleviates the brittle-schema problem via a metamodel framework, template-based models, versioning support, document-based entities and dynamic field support.
- Coalesce shifts your Systems reach from single-scoped storage to polymorphic data views.
- Coalesce provides Big Data integration out of the box with a built-in persister for Accumulo (a Hadoop database).

Conclusion

The Coalesce Unified Data Abstraction Layer is the result of over a decade of trial and error in developing robust and resilient enterprise software solutions. InCadence has many engineers actively enhancing and supporting the product every day. The product is proven and currently deployed in production systems for multiple Government customers. As an open-source product, you can explore the demonstrations and try the Coalesce features in your organization today. We are confident that the Coalesce System will immediately provide valuable capabilities that solve your systemic, real-world data problems.