Robust Streaming Data Pipelines through Model Driven Development

Vamshi Jandhyala
London, UK

Abstract

Model-driven engineering (MDE) is a software development methodology that focuses on creating and exploiting domain models, which are conceptual models of all the topics related to a specific problem.In this paper, we show how MDE can be adapted for the purpose of creating high quality streaming data pipelines that adhere to Data Governanace principles.

High Level Process Overview

MDE process for data pipelines consists of the following sub processes:

1. Model Authoring and validation.
2. Model registration and versioning.
3. Data In Motion library Generation.
4. Channel and contract creation.
5. Data validation, publishing and consumption.

Model Authoring

The first step in understanding data is to be able to model it accurately and unambiguously. For this purpose we need an expressive programming language agnostic meta model that can be used to formally specify models for data and meta data. The logical data modelling is usually done using UML or SysML but one can also use a lightweight Domain Specific Language e.g. Pegasus Data Language for this purpose. The meta model/DSL should also enable application developers/data stewards to annotate data models, create definitions, tag attributes as PII/Confidential etc.Along with the logical data model, one should also be able to specify a set of data quality rules or constraints e.g. using Natural Rules Language(NRL). The UML diagram below is a simple, quite generic model of a settlement transaction. Sample NRL rules for the model are shown below.

```
Model "settlement_transaction.uml2"

Context: Transaction
Validation Rule "t1"
If the type is equal to 'PURCHASE' then exactly two parties are present

Context: Settlement
Validation Rule "s2"
discount is greater than or equal to 0 and discount is less than or equal to 1
```

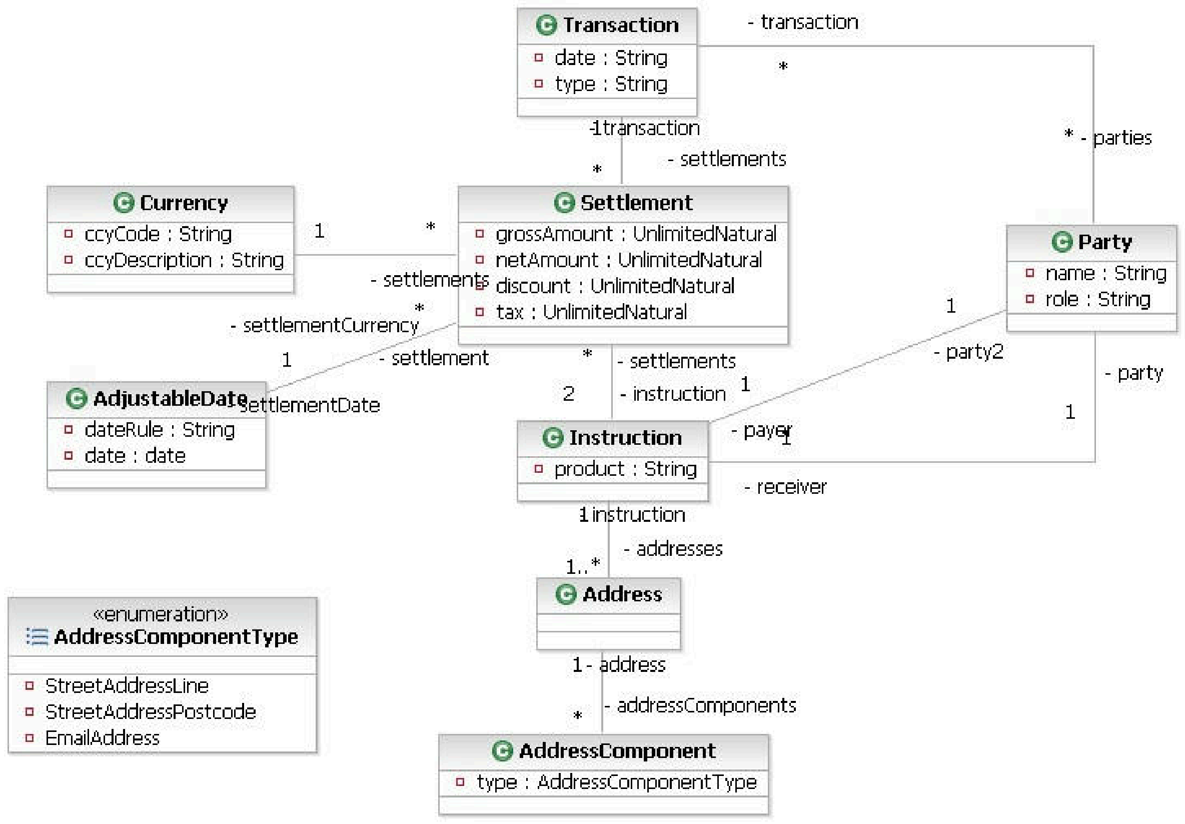

Figure 1: UML Settlement Transaction Model

Model registration and versioning

Once a model is validated, it needs to be registered in a central Data Catalog. This enables transparency, auditability and factilitates reuse. Every update to the model should be versioned in the Data Catalog.The Data Catalog can be realized through a DataHub like application. The Data Catalog should enable the following:

1. Search and Discovery of Data models.
2. Add/retrieve data definitions of entities, attributes and relationships.
3. Tag data models using a controlled vocabulary.
4. Visualize models, their relationships, lineage etc.

Code generation

The next step in the process is generation of Data In Motion(DiM) libraries which are essentially class libraries in various programming languages (Python, Java, Typescript etc.) based on a data model. Data provider applications can then embed the DiM libraries in their application to generate records that conform to the logical data models. The DiM library can also provide utility functions that enable producers/consumers to serialize/deserialize data into Avro, Parquet, XML or JSON as required. In addtion, they should also provide the ability to validate the records against the data quality rules associated with the logical data models. The typical workflow is as follows:

1. User creates or edits a model and uploads it to the Data Catalog.
2. The Data Catalog detects changes, validates and registers a new version of the model.
3. User creates and runs a build pipeline against a versioned model to generate DiM libraries which are stored in an artifact repository.

Channel and contract creation

Channel is a logical abstraction that links a Data Provider, a Data Contract, a set of consumers each having their own Consumer View(CV), the details of the physical transport mechanism (e.g. Kafka broker and topic). A Data Contract is a logical data model that the Providing application is guaranteed to adhere to. A Consumer View is a “subset” of the data contract on a given channel that the consumer is interested in. The Data Catalog should enable Data Providers to create channels and consuming applications to subscribe to existing channels in a self-service fashion. The Data Provider application generates DiM libraries based on the Data Contract and each consuming application on a channel generates a DiM library based on the Consumer View.

Data validation and publishing

A typical workflow for a Streaming Data Provider is as follows:

1. Define a Logical Data Model.
2. Generate a DiM libary based on the logical model.
3. Set up the physical transport mechanism e.g. Kafka cluster.
4. Create a channel in the Data Catalog - The Data Provider creates a channel in the Data Catalog and registers the logical data model created in step 1. as a Data Contract on that channel.
5. Create a provider application that embeds the DiM libraries and a client specific to the physical transport/programming language (e.g. Python Kafka client).
6. Validate a record created through the DiM against the data quality rules and publish exceptions on the exception channel corresponding to the data channel.
7. Publish validated records using the client on the data channel.

A typical workflow for a Streaming Data Consumer is as follows:

1. Search for data models and identify channels that transport data as per the required data model.
2. Register as a consumer against an existing chanel.
3. Define a Consumer View based on the Data Contract of the channel.
4. Generate a DiM libary based on the Consumer View.
5. Create a consuming application that embeds the DiM libraries and a client specific to the physical transport/programming language (e.g. Python Kafka client).
6. Validate a record created through the DiM against the data quality rules and publish exceptions on the exception channel corresponding to the data channel.
7. Consume validated records.

High Level Architecture

A number of components need to work seamlessly to enable organizations realize the benefits of Model Driven Engineering. This section describes the capabilities provided by the various components and how they need to work together. A high level architecture diagram is given below.

Model Authoring Platform

Model authoring platform provides a set of tools that enable Information Architects as well as Application Developers create logical data models based on a standard meta model that can be stored and versioned in the Data Catalog. For complex models, based on UML, applications like MagicDraw could be used. For models of simple to medium complexity, one can use a lightweight DSL that is compatible with the standard meta model. One advantage of a lightweight DSL is that it can be **validated** through code editor extensions (e.g. VSCode, IntelliJ etc.) which enables tight feedback loops and ensures that model authoring is integrated with the existing SDLC processes.

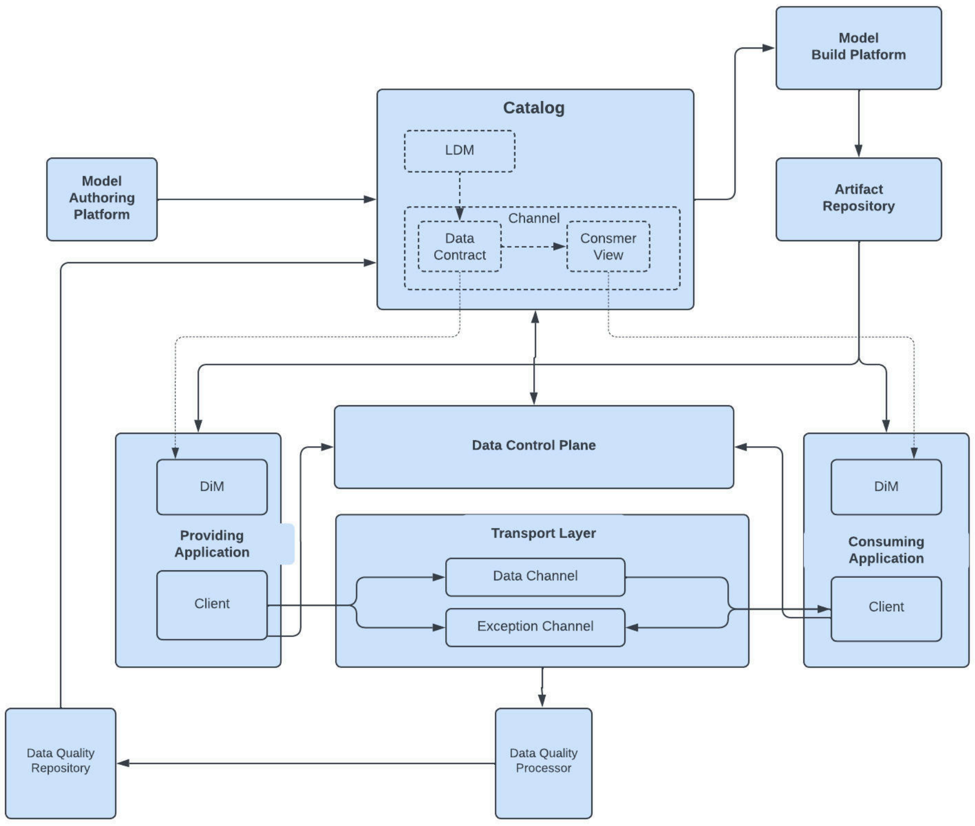

Figure 2: High Level Architecture

Data Catalog

The Data Catalog is one of the most important components of the architecture. The Catalog is the central repository for all logical data models, channels, their versions and the associated metadata(which uses controlled vocabulary). The Catalog should faciliate search and discovery of models and channels. It should also provide data quality scores against data channels which incentivizes providers to improve data quality and enables consumers trust the data they are consuming. The catalog should also expose the data through a property-graph model that can be easily queried for identifying dependencies, visualizing lineage, tracking downstream consumers etc.

Model Build Platform and Artifact Repository

The Model Build Platform enables producers and consumers to generate a versioned DiM library in one or more programming languages against any versioned model in the Catalog and store it in the artifact repository.

Streaming Data Client

Streaming Client SDKs which provide common abstractions that work over multiple underlying transport layers e.g. Kafka can be provided that can help the consumers migrate from one transport to another. The main role of client SDK is to authenticate publishers and consumers on a channel by connecting to the Data Control Plane for authentication and help serialize records to the underlying transport layer.

Streaming Transport Layer

The Streaming Transport layer could be a Apache Kafka based platform or any other messaging platforms that enables one pubisher exchange data asynchronously with multiple consumers using a pub/sub model.

Data Control Plane

The Data Control Plane helps authenticate providers and consumers before they publish to a channel. The control plane also tracks channel usage as well as telemetry.

Data Quality Processor

The Data Quality Processor consumes and processes exception messages from the Exception channels and then persists the results in normalized fashion in the Data Quality Repository.

Data Quality Repository

The Data Quality Repository stores exception records and keeps track of data quality metrics for all channels that can be exposed through the Catalog.

Benefits of MDE

This section summarizes a number of benefits to using MDE.

Data Model Discovery and Reuse

Well defined data models centrally stored in a Data Catalog enable a shared understanding of data across the organization. Business domain data once formally modelled can be reused across multiple applications. By making the models and channels discoverable,the need to create new data pipelines also reduces as new consumers can start consuming data from existing channels.

Controlled model evolution

The data provider on a channel has an obligation to produce records which comply with the channel contract. As the channel contract is formally specified, any model changes on the provider end can be automatically checked for compliance against the data contract. The Consumer Views also shield the consumers from any changes to attributes in the data contract outside of what is being consumed. Automatic compatibility checking between data contracts and consumer views enable providers to make changes to the data contract confidently without impacting any of the consumers of a given channel.

Automated evergreen fine grained data lineage

Typically on a channel, multiple consumers can consume data provided by a data provider. Every consumer can potentially consume a different subset of attributes in the data contract using their own Consumer View. As all the Consumer Views against a channel are registered in the Catalog, we have fine grained attribute level lineage between Providers and Consumers. All provider/consumer applications of a given channel are authenticated with the Control Plane at run time before publishing/consumption of records. This enables identification of channels which have not been used recently.

Embedded Data Quality

Data Quality is not an afterthought by tightly integrated with the data modelling/publishing/consumption process. Both consumer and publishers can validate records published/consumed on a data channel against the data quality rules associated with the data model and publish exception records on the corresponding exception channel. The records in the exception channel can be analysed and stored for the purposes of data quality scoring and remediation. The data quality scores for channels can also be made available through the Catalog to enable consumers get a better understand the quality of data being published on the channels they are interested in.

Automatic Code Generation

Automatically generating standards compliant serializing and deserializing code for multiple languages from the model saves the providers and consumers a lot of boilerplate code. In most cases providers and consumers do not have to worry about data serialization but they have the flexibility to choose specific serialization formats (Avro, JSON, XML etc.) for their use case as they see fit.