The AMI 2.0 metadata ecosystem: new design principles and features

What is AMI?

- AMI (ATLAS Metadata Interface) is a generic ecosystem for metadata:
  - Primitives for metadata extraction and processing
  - Heterogenous and distributed database connectivity
  - High level tools for selecting data by metadata criteria

- The ecosystem has development kits for:
  - Developing JAVA business objects (server-side)
  - Developing metadata-oriented Web applications (client-side)

- AMI is designed for:
  - Scalability, evolutivity and maintainability
History of AMI

- AMI is mature ecosystem of more than 18 years of existence
- Originally developed for the ATLAS experiment at CERN
  - 2000 → Metadata bookkeeping for the Liquid Argon (LAr) calorimeter
  - 2001-2016 → ATLAS software release management (TagCollector)
  - 2006-ongoing → AMI provides the ATLAS official dataset discovery tool
  - 2015-ongoing → parameter definition for dataset processing (AMI-Tags)
  - ... Used by the ATLAS production system
- Development of AMI 2.X started in 2015 for:
  - Simplifying deployment
  - Avoiding subsystem entanglements
  - Improving maintainability and performance
- Since 2017, AMI 2.X is used by:
  - ATLAS (migration ongoing)
  - ROSETTA (Institut de Planétologie et d'Astrophysique de Grenoble)
Overview of the AMI ecosystem

- AMI JAVA Core: core library server-side
- AMI HTTP Services: AMI Commands (proprietary) or REST API
- AMI Task Server: aggregating and processing metadata
- AMI Web Framework: developing metadata-oriented Web applications
- Lightweight clients: accessing AMI from anywhere
AMI JAVA Core
AMI JAVA Core features

- **AMI JAVA Core** is the central part of the AMI ecosystem

**Main features:**

- Authentication and authorizations
- Command engine (~100 generic commands, ~500 ATLAS-specific commands)
  - Metadata queries (trivial [SQL, MQL] or more complex, read or write), experiment-specific commands, service administration, ...
- High level primitives for manipulating data
  - DB rowsets, JSON documents, XML documents, remote access, ...
- Metadata Query Language (MQL) and Structured Query Language (SQL)

---

**Command layer**

- **authentication and roles**
- **command engine**
- **data formatter**

**Metadata layer**

- **high level primitives for manipulating data**
- **distributed transactional engine**
- **connection pool**
- **reflexion and MQL**
- **JDBC drivers**

---

**n-tiers architecture**
Command layer

Authentication sub-system
(password / X.509 certificate / SSO)

Authorization sub-system

Command sub-system (JAVA classes)

Formatter (XSLT)

clients (HTTP services)

AMI conf

Metadata layer

SQL
NoSQL

Other (files, brokers, ...)

Example of commands:

GetSessionInfo
SearchQuery -catalog="..." -sql="..."
GetDatasetInfo -logicalDatasetName="..."

(for ATLAS, getting detailed dataset info)
Metadata layer

Connection pool (HikariCP)

Transaction pool

MySQL  Oracle  NoSQL
Metadata layer

Connection pool (HikariCP)

MySQL  Oracle  NoSQL

MySql
Oracle
NoSQL

Relation extraction for SQL only
(foreign keys, indices, ...)

Transaction pool

Transaction #1  Transaction #2  ...

MQL → SQL
automatic generation of SQL Joins

Metadata Query Language

Reflexion sub-system

free  used  ...

07/04/18
Metadata layer

Command sub-system

Command #1
(transaction #1)

Command #2
(transaction #2)

Command #3
(transaction #2)

...

Data primitives

Very high level rowset

MQL → SQL
automatic generation of SQL Joins

Transaction pool

Transaction #1

Transaction #2

...

Reflexion sub-system

Relation extraction for SQL only
(foreign keys, indices, ...)

Connection pool (HikariCP)

free

used

MySQL

Oracle

NoSQL
More about MQL

- MQL is a kind of SQL without `FROM` clause nor join
- It makes it possible to build queries without (precisely) knowing relations
- Joins are automatically generated from the reflexion sub-system info
- MQL turns DB-oriented point of view to metadata-oriented point of view
- When there are cycles in relations, there is a dedicated syntax to apply path constraints
- Example:

  ```sql
  SELECT BLOCK.ID
  WHERE BLOCK_PARAM.NAME = 'foo'
  AND DATASET.NAME{BLOCK.DATASETFK} = 'bar'
  ```

  MQL to SQL
AMI Task Server
AMI Task Server features

- The AMI Task Server is used for:
  - Extracting metadata from primary sources (pull mode)
  - (Re)Processing metadata
  - Storing metadata in AMI
- It can run any kind of tasks (shell, python, java, ...)
- When needed, it can benefit from the AMI Java Core library
- Main features:
  - Kind of super CRON
  - Web interfaces and monitoring
  - The AMI Task Server is distributed
  - Mutual exclusion mechanism between tasks
  - Priority lottery scheduler for avoiding starvation (not real time)
  - One shot tasks

In ATLAS:
- Metadata extraction from Tier0 (real data)
- Metadata extraction from ProdSys (simulated data, reprocessing)
- ActiveMQ messages from RUCIO (data placement)
- ~50 other tasks
AMI Web Framework
AMI Web Framework (AWF)

- A Web framework for designing metadata-oriented applications
- AWF can be used without the AMI Java HTTP Service
  - Server-side, libraries AMIMini{PHP, Python, JAVA} can easily bridge AWF to existing services
- AWF is based on standard technologies:
  - JS6 (transpiled to JS5), CSS3, HTML5
  - JQuery, Twitter Bootstrap 4
  - AMI-Twig (a JavaScript homemade version of the Twig template engine)
Features and patterns

- Authentication
- URL router, short URLs
- Object paradigm (emulated)
- Sub-applications and reusable graphic controls
- Centralized resource live cycle management (CSS, JS, JSON, xml,Twig files; AMI sub-applications; AMI controls)
- Wizards for generating sub-application and control skeletons
- **MVC** pattern:
  - **Model** → AMI commands
  - **View** → TWIG templates
  - **Controller** → classes ami.SubApp, ami.Control (JavaScript)
Default controls and applications

- Controls can be embedded in external Web pages such as wikis
  - See details in poster session

- Applications are generally built by assembling controls

- Main available controls:
  - Dialog boxes
  - Controls for searching (Google-like Search, Criteria Search, …)
  - Controls for displaying (Schema Viewer, Tab, Table, Element Info, …)
  - Controls for annotating entities (WhiteBoard, …)

- Main available applications:
  - Basic CMS
  - AMI command interpreter
  - Admin Dashboard and Monitoring
  - Schema Viewer, Table Viewer, Simple Search, Criteria Search, Search Modeler, …
Searching ATLAS datasets by criteria

Displaying search results in AMI

A control embedded in a wiki and connected to the central AMI service

This control executes the GetDatasetInfo command
Conclusion
Conclusion

- AMI is mature metadata ecosystem of more than 18 years of existence
- Originally developed for the ATLAS experiment:
  - i) Official dataset discovery tool (millions of datasets, billions of files). ii) Used by the ATLAS production system (parameter definition for dataset processing [= AMI-Tags]). …
- AMI version 2 released this year, web site soon available
- AMI Java Core
  - High level server-side JAVA library for processing metadata
    - i) High level primitives for manipulating metadata. ii) Metadata Query Language. iii) datasource connectivity.
- AMI HTTP Services + lightweight clients
  - AMI commend service (proprietary), REST interface
- AMI Task Server
  - Distributed system for extracting, processing and storing metadata
- AMI Web Framework
  - For developing metadata-oriented Web applications and graphic controls
Questions?