Launching CDISC CORE v1.0: A New Standard for Automated Study Conformance

Tomás Sabat Stöfsel, Verisian, CEO
Charles Shadle, Head of Data Science, CDISC
17 November 2025

Meet the Speakers

Tomás Sabat Stöfsel

Title: Co-Founder & CEO

Organization: Verisian

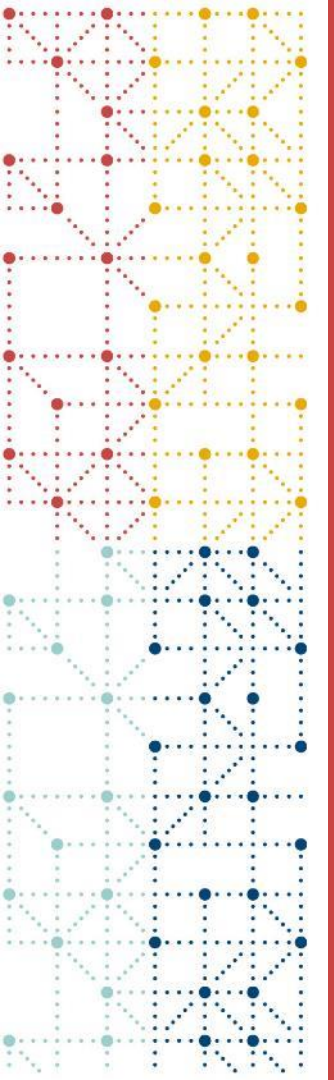
Insert your short bio here as text in sentence form or use notes space below and delete this text.

Charles Shadle

Title: Head of Data Science

Organization: CDISC

Charles Shadle is a data science leader driving innovation in data standards and automation across the regulated biopharma industry. He advances reproducibility, interoperability, and efficiency from protocol design to regulatory submission.



Agenda

1. Overview & Opportunities
2. V1.0 Scope & Status
3. Optimization Plan
4. Conclusion

Overview & Opportunities for Improvement

CDISC Open Rules Engine (CORE)

Purpose

- Open-source
- Reference implementation
- Execute conformance rules

Key Features

- Authoring, testing, and execution
- Standardized, transparent validation
- Open-source access to the global clinical research community
- Adapt to evolving regulatory and research needs

CDISC Open Rules Engine (CORE)

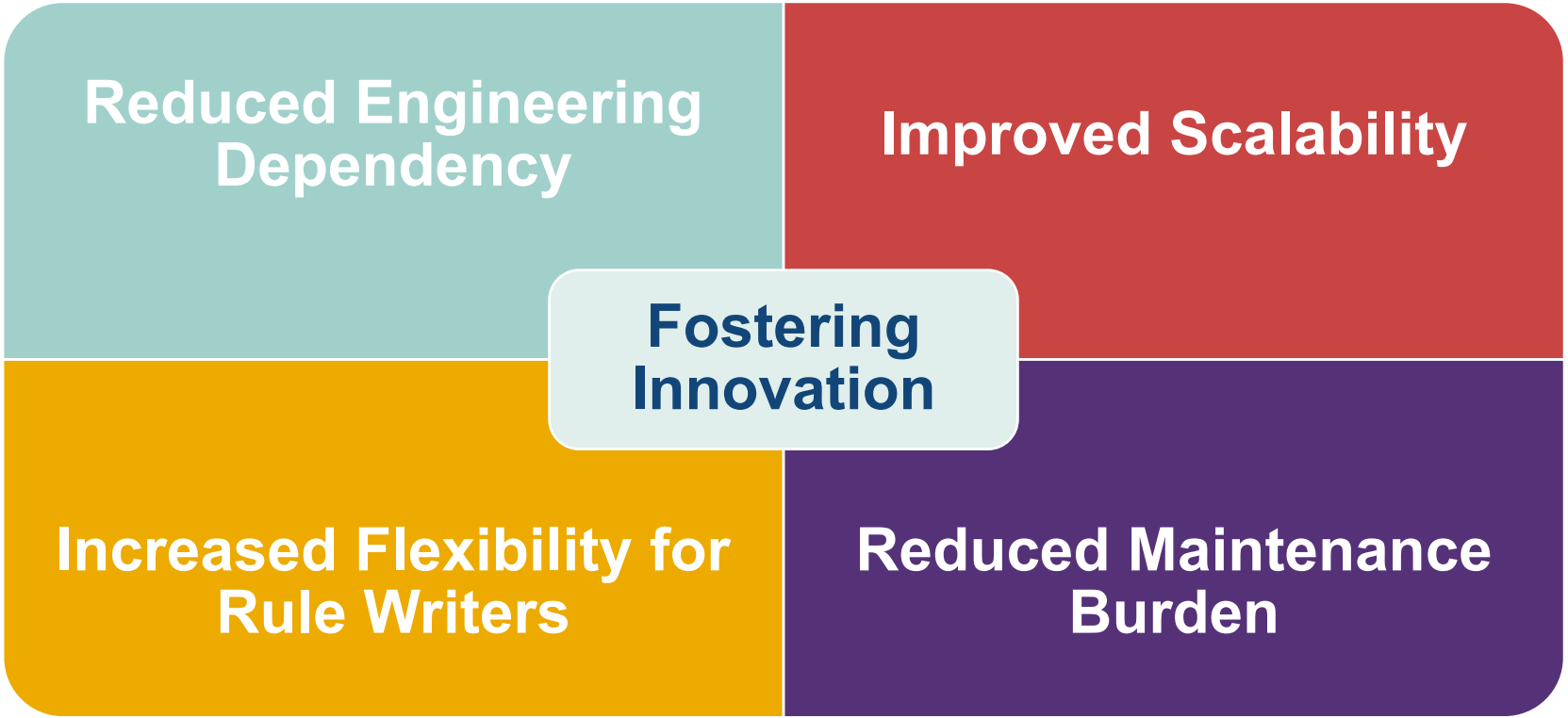
Architecture

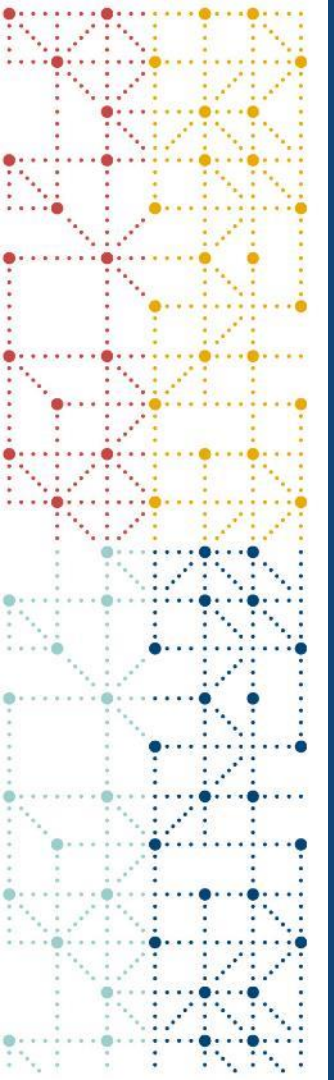
- Written Python, rules authored in YAML
- Rules engine, command-line interface, and testing framework
- Built on a lightweight adaptation of Venmo engine

Future Vision

- Embedding established query languages (e.g. SQL & JSONata)
- Enhance scalability to support new standards
- Easier rule creation, testing, and maintenance

Key Opportunities & Impact





CORE V1.0 Scope & Status

Open Rules v1.0 Scope

Standard	# Rules	# Complete	% Complete
SDTMIG v3.3 and 3.4	505	451	89.3%
SENDIG v3.1.1	312	216	69.2%
FDA Business Rules	292	153	52.4%
TIG v1.0	484	404	83.5%
USDM v3.0 and v4.0	259	78	30.1%
Totals	**1852**	**1302**	**70.3%**

Engine

- Support for the rules
- Initial support for ADaM rules
- Refactoring for performance & flexibility

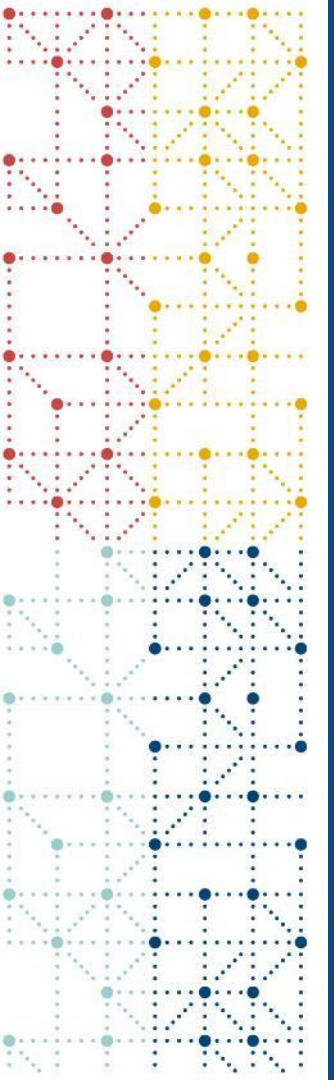
What Follows v1.0

Rules in support of

- All SDTM and SEND versions
- ADaM
- Define-XML
- PMDA Rules
- Biomedical Concepts

Drivers

- Potential AI-Assisted Rule Authoring UI
- Other?



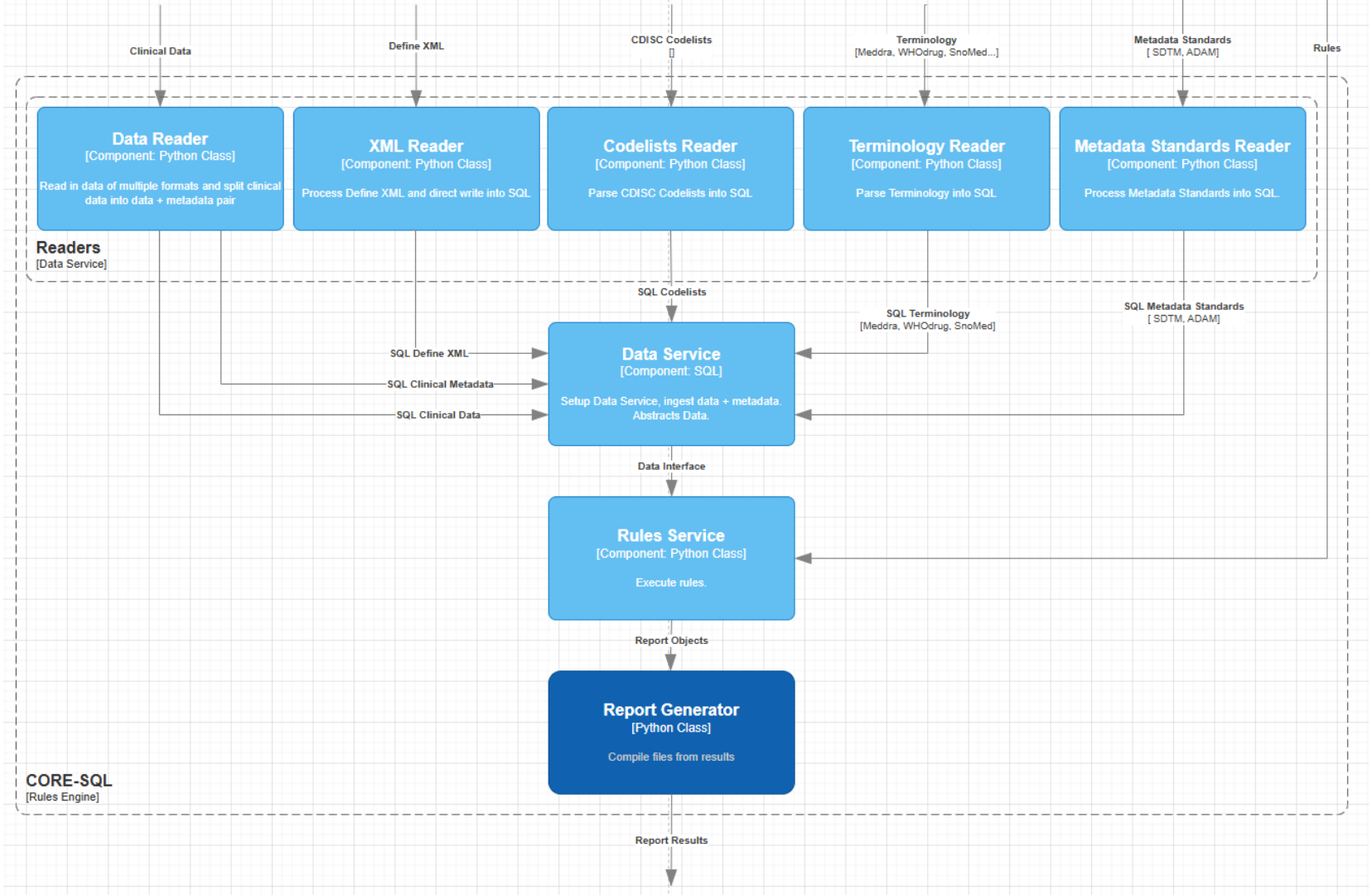
CORE Optimization Plan

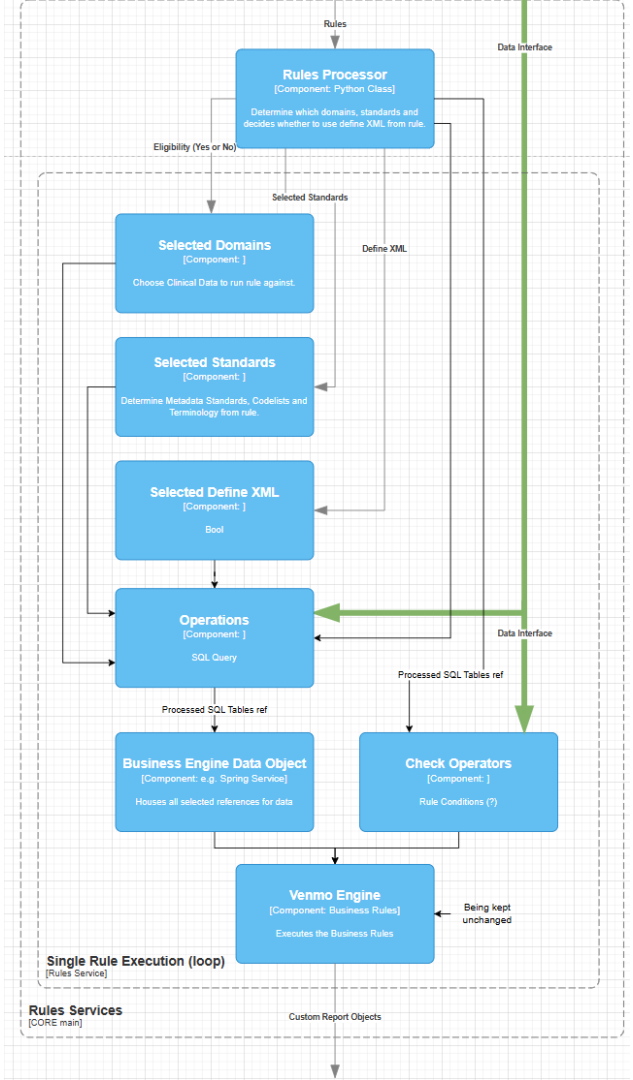
A Scalable Path to Smarter, More Maintainable Conformance Rules

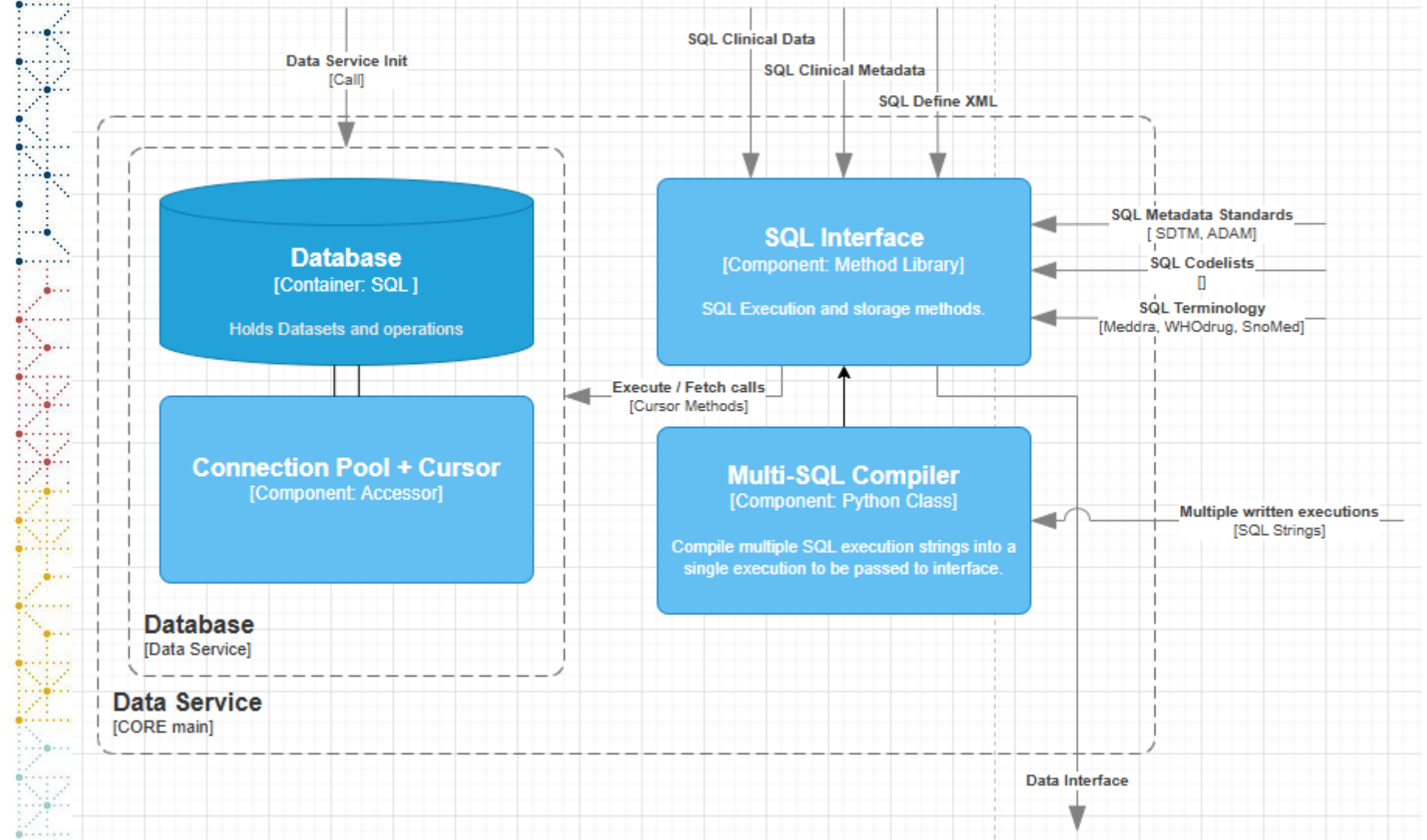
Overview of CORE Enhancements

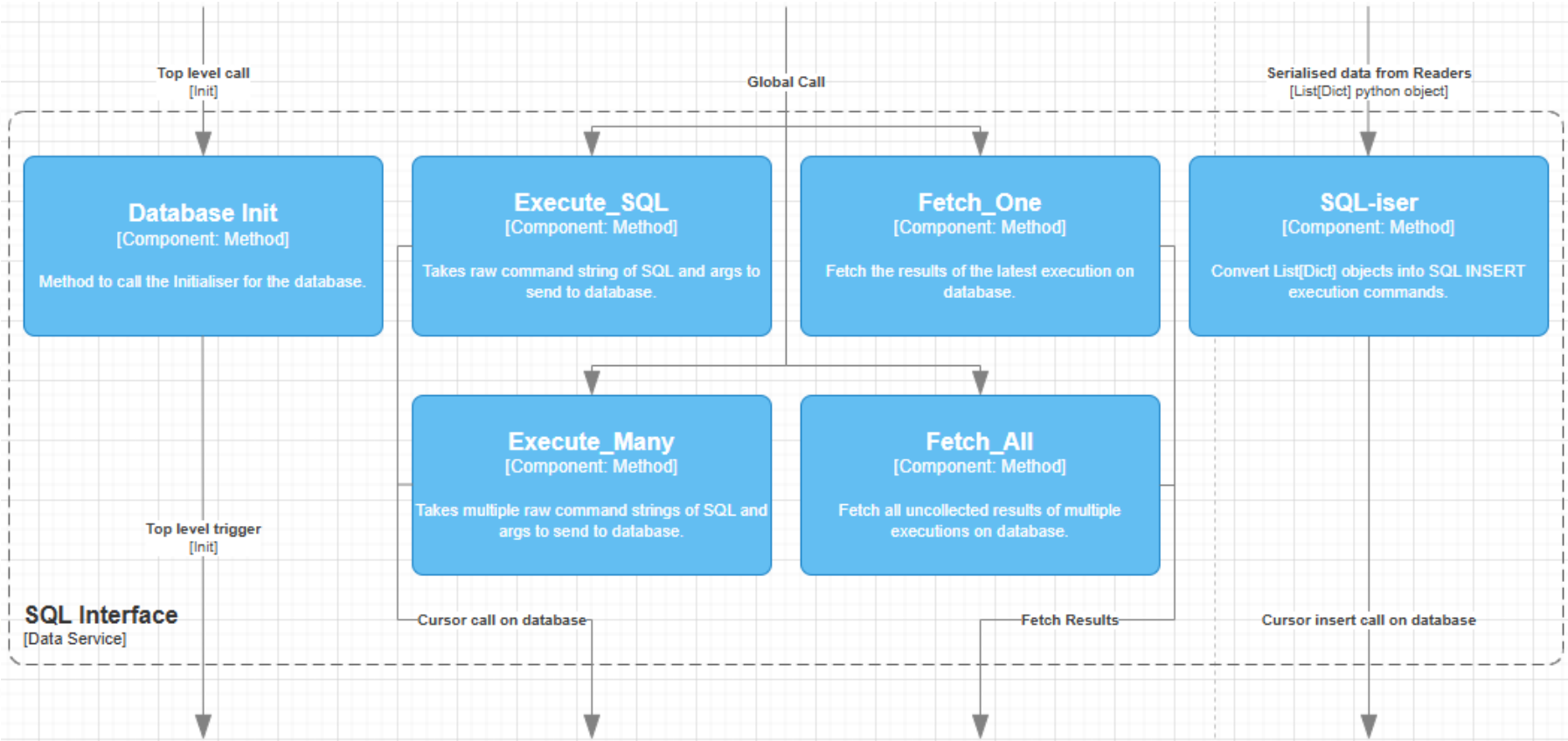
Refactoring for Long-Term Sustainability

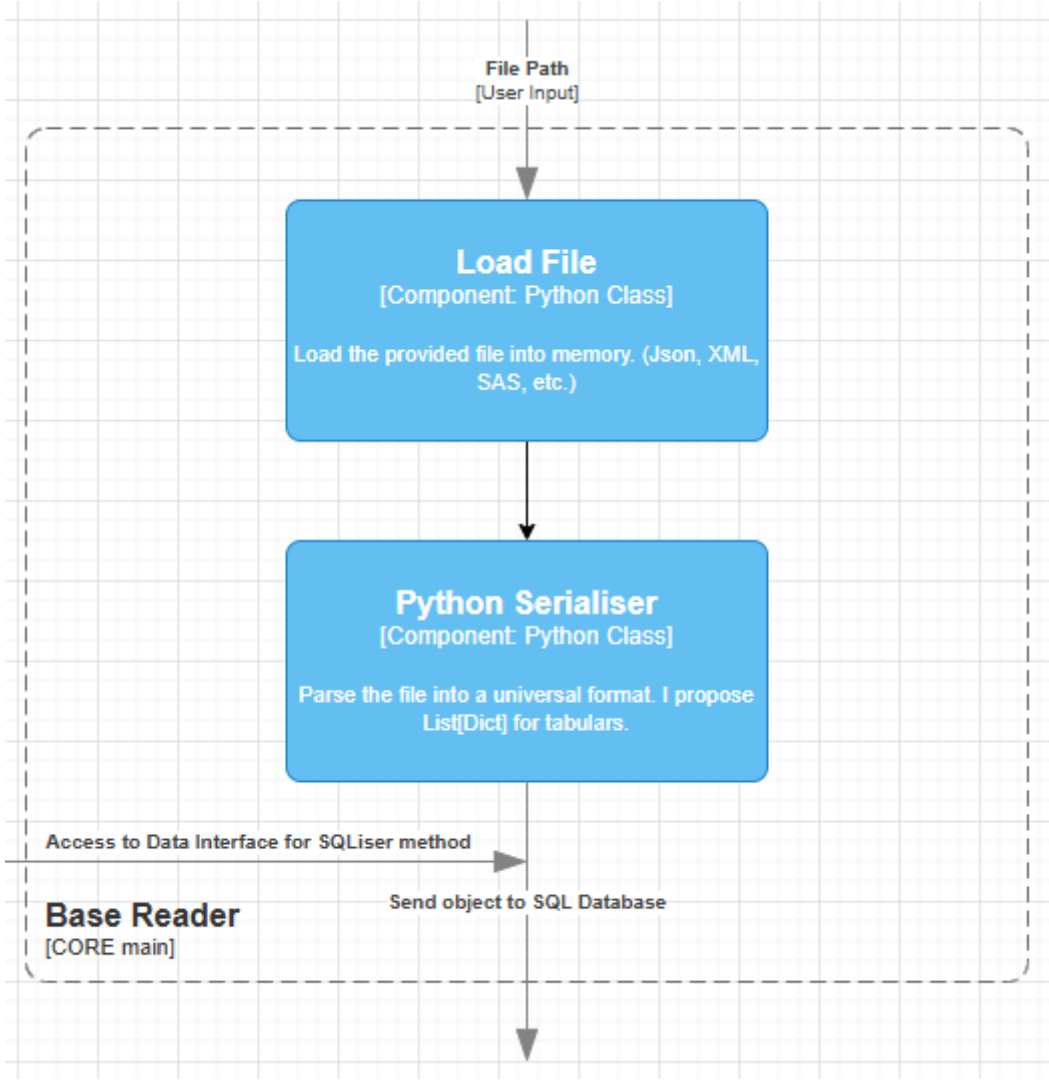
- Aiming for a scalable, modular architecture
- Improved rule execution performance and resource management


Clinical Data
Define XML
CDISC Codelists
[]
Terminology
[Meddra, WHOdrug, SnoMed...]
Metadata Standards
[SDTM, ADAM]
Rules
Data Reader
[Component: Python Class]
Read in data of multiple formats and split clinical data into data + metadata pair
XML Reader
[Component: Python Class]
Process Define XML and direct write into SQL
Codelists Reader
[Component: Python Class]
Parse CDISC Codelists into SQL
Terminology Reader
[Component: Python Class]
Parse Terminology into SQL
Metadata Standards Reader
[Component: Python Class]
Process Metadata Standards into SQL
Readers
[Data Service]
SQL Codelists
SQL Terminology
[Meddra, WHOdrug, SnoMed]
SQL Metadata Standards
[SDTM, ADAM]
SQL Define XML
SQL Clinical Metadata
SQL Clinical Data
Data Service
[Component: SQL]
Setup Data Service, ingest data + metadata. Abstracts Data.
Data Interface
Rules Service
[Component: Python Class]
Execute rules.
Report Objects
Report Generator
[Python Class]
Compile files from results
CORE-SQL
[Rules Engine]
Report Results


Rules
Data Interface
Rules Processor
[Component: Python Class]
Determine which domains, standards and decides whether to use define XML from rule.
Eligibility (Yes or No)
Selected Standards
Define XML
Selected Domains
[Component:]
Choose Clinical Data to run rule against.
Selected Standards
[Component:]
Determine Metadata Standards, Codelists and Terminology from rule.
Selected Define XML
[Component:]
Bool
Operations
[Component:]
SQL Query
Data Interface
Processed SQL Tables ref
Processed SQL Tables ref
Business Engine Data Object
[Component: e.g. Spring Service]
Houses all selected references for data
Check Operators
[Component:]
Rule Conditions (?)
Venmo Engine
[Component: Business Rules]
Executes the Business Rules
Being kept unchanged
Single Rule Execution (loop)
[Rules Service]
Rules Services
[CORE main]
Custom Report Objects


Data Service Init
[Call]
SQL Clinical Data
SQL Clinical Metadata
SQL Define XML
Database
[Container: SQL]
Holds Datasets and operations
SQL Interface
[Component: Method Library]
SQL Execution and storage methods.
SQL Metadata Standards
[SDTM, ADAM]
SQL Codelists
[]
SQL Terminology
[Meddra, WHOdrug, SnoMed]
Execute / Fetch calls
[Cursor Methods]
Connection Pool + Cursor
[Component: Accessor]
Multi-SQL Compiler
[Component: Python Class]
Compile multiple SQL execution strings into a single execution to be passed to interface.
Multiple written executions
[SQL Strings]
Database
[Data Service]
Data Service
[CORE main]
Data Interface


Top level call
[Init]
Global Call
Serialised data from Readers
[List[Dict] python object]
Database Init
[Component: Method]
Method to call the Initialiser for the database.
Execute_SQL
[Component: Method]
Takes raw command string of SQL and args to send to database.
Fetch_One
[Component: Method]
Fetch the results of the latest execution on database.
SQL-iser
[Component: Method]
Convert List[Dict] objects into SQL INSERT execution commands.
Execute_Many
[Component: Method]
Takes multiple raw command strings of SQL and args to send to database.
Fetch_All
[Component: Method]
Fetch all uncollected results of multiple executions on database.
Top level trigger
[Init]
SQL Interface
[Data Service]
Cursor call on database
Fetch Results
Cursor insert call on database


File Path
[User Input]
Load File
[Component: Python Class]
Load the provided file into memory. (Json, XML, SAS, etc.)
Python Serialiser
[Component: Python Class]
Parse the file into a universal format. I propose List[Dict] for tabulars.
Access to Data Interface for SQLiser method
Send object to SQL Database
Base Reader
[CORE main]

Overview of CORE Enhancements

PostgreSQL Integration

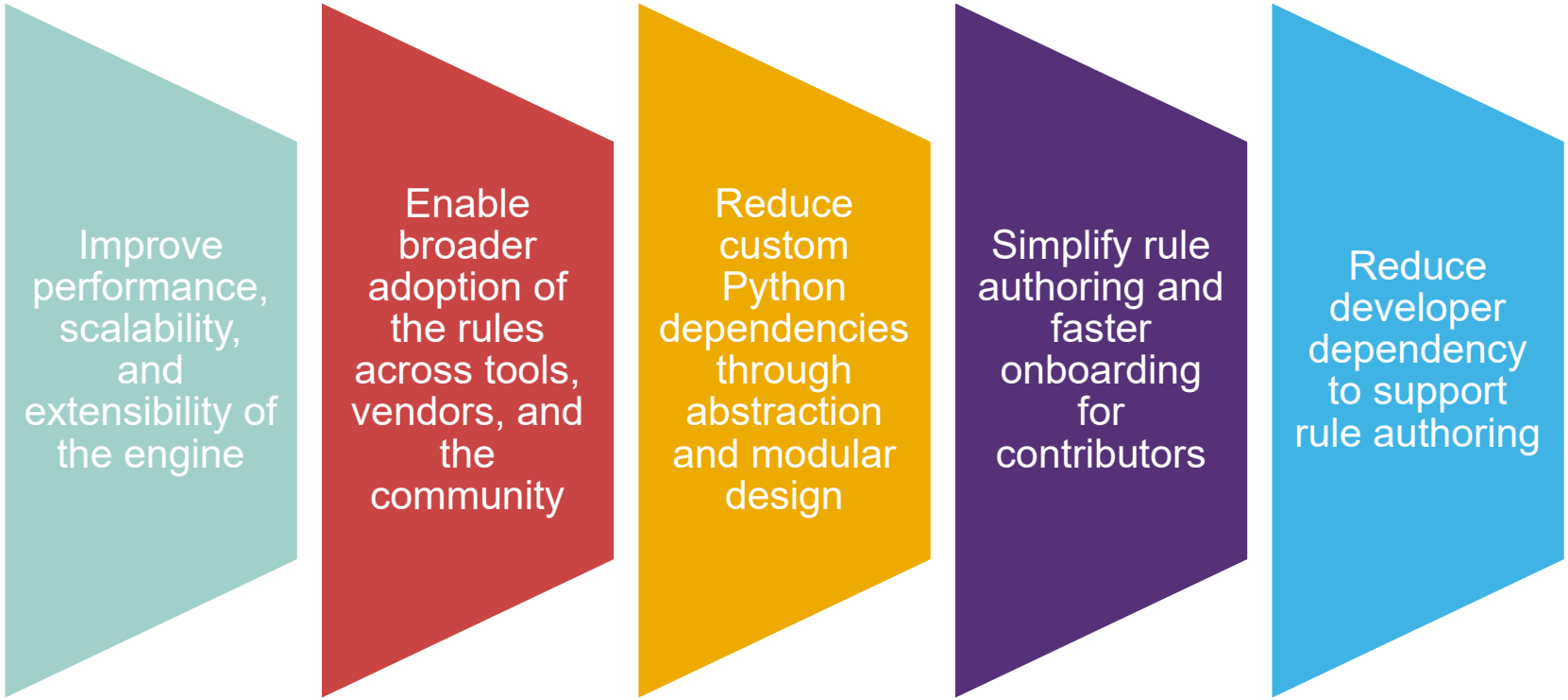
- Adds lightweight, embedded data persistence layer
- Supports portability of rules across systems
- Improves scalability and performance for rule execution and caching
- Enables standalone use without requiring complex DB infrastructure
- Supports faster testing and rule validation at scale

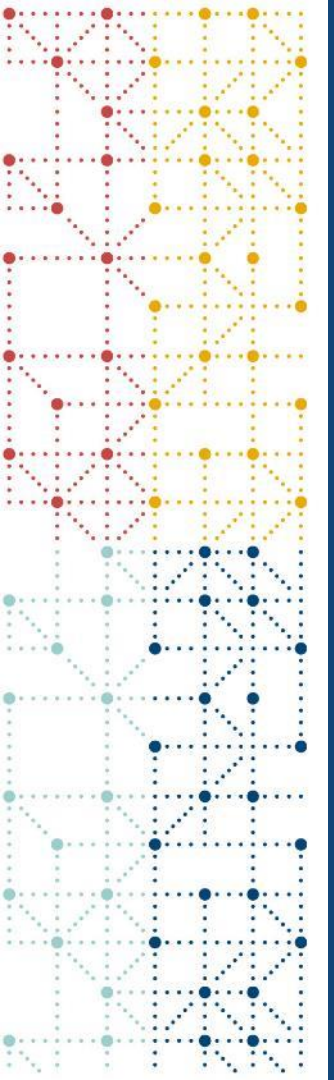
Overview of CORE Enhancements

JSONata Integration

- Enables advanced query and transformation capabilities within rules
- Supports powerful data manipulation without complex backend changes
- Greater flexibility for rule writers
- Improves the expression logic available in YAML rules
- Simplifies complex rule conditions and transformations

Strategic Benefits





Conclusion

Conclusion

- LinkedIn group: CDISC CORE Collaboration
- Volunteer

LinkedIn Group

Volunteer Form

Thank You!

tomas@verisian.com

ndedonder@cdisc.org