# MOSDEX

## A New Standard for Data Exchange with Optimization Solvers

Dr. Jeremy A. Bloom

jeremyblmca@gmail.com

July 8, 2020

## Synopsis

- Rationale for a new data exchange standard
- Overview of MOSDEX
- MOSDEX Syntax
- MOSDEX Example – Transshipment in Instance Form
- MOSDEX Example – Transshipment in Query Form
- Optimization Architecture and MOSDEX
- The Big Data Future
- Additional MOSDEXCapabilities
- MOSDEX Resources

## Rationale

- MPS format is the de facto standard for data exchange and model specification for many optimization solvers
- It has several advantages:
  - Sparsity
  - Text-based
  - Non-proprietary
- However, it also has many deficiencies:
  - Lack of an output standard
  - Lack of model-data separation
  - Difficulty in scaling
  - Lack of indexing
  - Column orientation
  - Extensions beyond linear models

# The MOSDEX Standard

## Mathematical Optimization Solver Data Exchange

**Overview**

- Efficient for machines, readable by humans
- Represent the data in relational form
- Use the JSON standard
- Augment the data representation with mathematical modeling objects

**Design Principles**

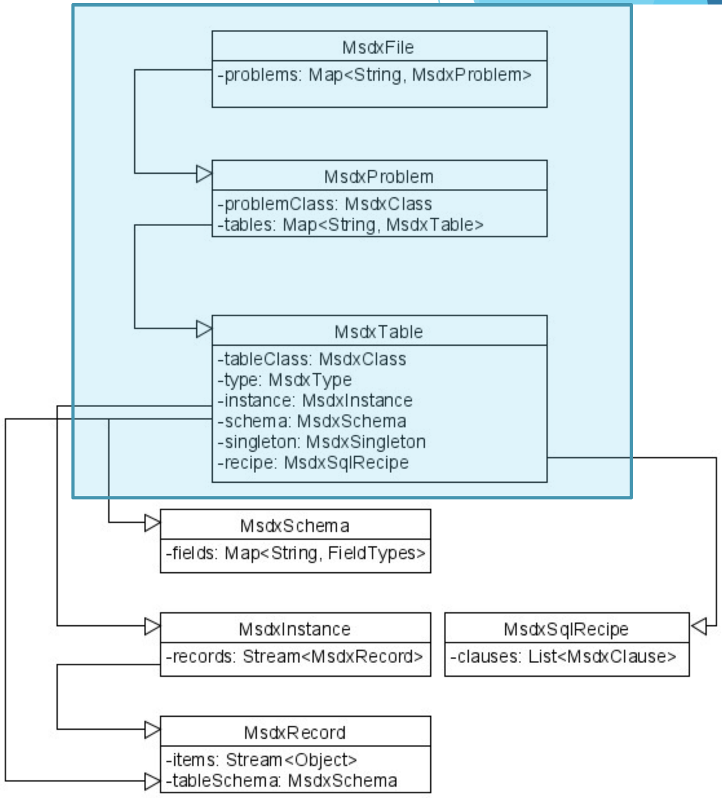
- Independence from and support for
  - multiple optimization solvers and APIs,
  - multiple algebraic modeling languages, and
  - multiple programming languages.
- Minimize custom coding required of the target solvers and modeling languages
- Rely on the public, published APIs of the target solvers and languages
- Avoid requiring users to customize the parser and/or code generator or emulator

## Why JSON?

- A standard format for data exchange http://json.org/
- Simple syntax
  - Primitive: String, Integer, Double, IEEEDouble
  - Object: { “name” : value, …} unordered key: value pairs
  - Array: [ value, …] ordered items of mixed types
  - Nesting: Array can contain Objects, Object can contain Arrays
- Support in many languages
  - Statically typed (e.g. C++, Java) require classes that map to JSON objects and arrays
  - Dynamically typed (e.g. JavaScript, Python) can create objects on the fly from JSON
- JSON Schema http://json-schema.org/
  - A meta-standard for describing JSON files

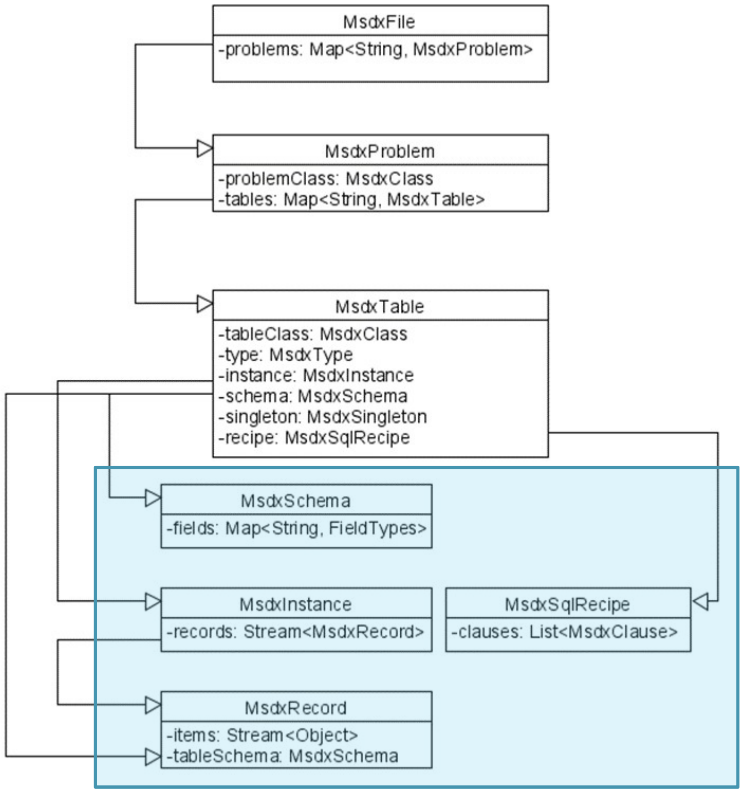
## MOSDEX Syntax

- FILE contains one or more named PROBLEMS
  - A FILE is intended to be self-contained but may link to other FILES
- PROBLEM contains one or more named TABLES plus auxiliary keyword objects
  - A PROBLEM may contain a full optimization problem, data only, or a module of a decomposition
- TABLE is the fundamental data structure of MOSDEX
  - Think of a table in a relational database with a fixed number of columns and an indefinite number of records (rows)
  - A TABLE's class is DATA or a modeling object – VARIABLE, CONSTRAINT, OBJECTIVE, or TERM
  - A TABLE's type depends on its class – e.g. VARIABLE types are CONTINUOUS, INTEGER or BINARY
  - A modeling object TABLE represents a family of related objects differentiated by a key.
    A key (i.e. subscript) can have one or more dimensions
  - A TABLE can have either **Instance** form or **Query** form



# MOSDEX Syntax (continued)

- Instance-form TABLE contains data
- Query-form TABLE is specified by an SQL query
- Singleton TABLE is a special instance that contains a single record
- Every TABLE has a SCHEMA
  - Specifies the name and datatype of each field (column) in the TABLE
  - Instance TABLE has explicit SCHEMA
  - Query TABLE has SCHEMA implied by the query
  - Singleton TABLE has SCHEMA implied by the data
- Data TABLE can have any reasonable schema
  Modeling object TABLE has a schema dictated by the solver
- PROBLEM can contain a mixture of instance, query, and singleton TABLES



## Why SQL?

- There is a deep and intimate relationship between the syntax of optimization models and the structure of relational data
- SQL permits compact specification of complex data transformations and optimization structures
- SQL is widely known and used by developers
- Relatively easy to learn
- Many platforms support it
- ANSI standard (with product-specific variances)
- Portable
- Efficient data manipulations through query optimization

Bloom, Optimization Modeling and Relational Data. https://github.com/JeremyBloom/Optimization---Sample-Notebooks/blob/master/Optimization%2BModeling%2Band%2BRelational%2BData%2Bpub.ipynb

## Modeling Objects – Why do we need them?

- MOSDEX is a data standard, not a modeling language
  - MOSDEX lacks syntactic features (e.g. a sum operator, set operators) that make modeling languages friendly for users, although many of these are available through SQL
- MOSDEX needs compatibility with MPS
  - MPS format combines data exchange with definitions of the modeling objects - rows (constraints and objectives) and columns (variables and coefficients)
  - However, MPS was defined before solvers had modeling APIs and before optimization domain-specific languages had been invented
  - Today, when many solvers have modeling APIs and with a number of optimization domain-specific languages, modeling objects are less important
- Modeling objects are still needed for solvers without modeling APIs (e.g. Clp)
- Standardized modeling objects are potentially useful for model export (serialization) and import (deserialization)
- However, even without modeling objects, a standard for data exchange can still realize most of the value

## MOSDEX Examples
### Transshipment Network

$$minimze\ TotalCost = \sum_{(i,j)\ in\ LINKS} cost[i,j] * Ship[i,j]$$

$Subject\ to:$

$for\ all\ \{k\ in\ CITIES\}$

$$Balance[k]: \sum_{(k,j)\ in\ LINKS} Ship[k,j] - \sum_{(i,k)\ in\ LINKS} Ship[i,k] = supply[k] - demand[k]$$

$$0 \le Ship[i,j] \le capacity[i,j]\ for\ all\ (i,j)\ in\ LINKS$$

from https://ampl.com/BOOK/EXAMPLES/EXAMPLES2/net1.mod

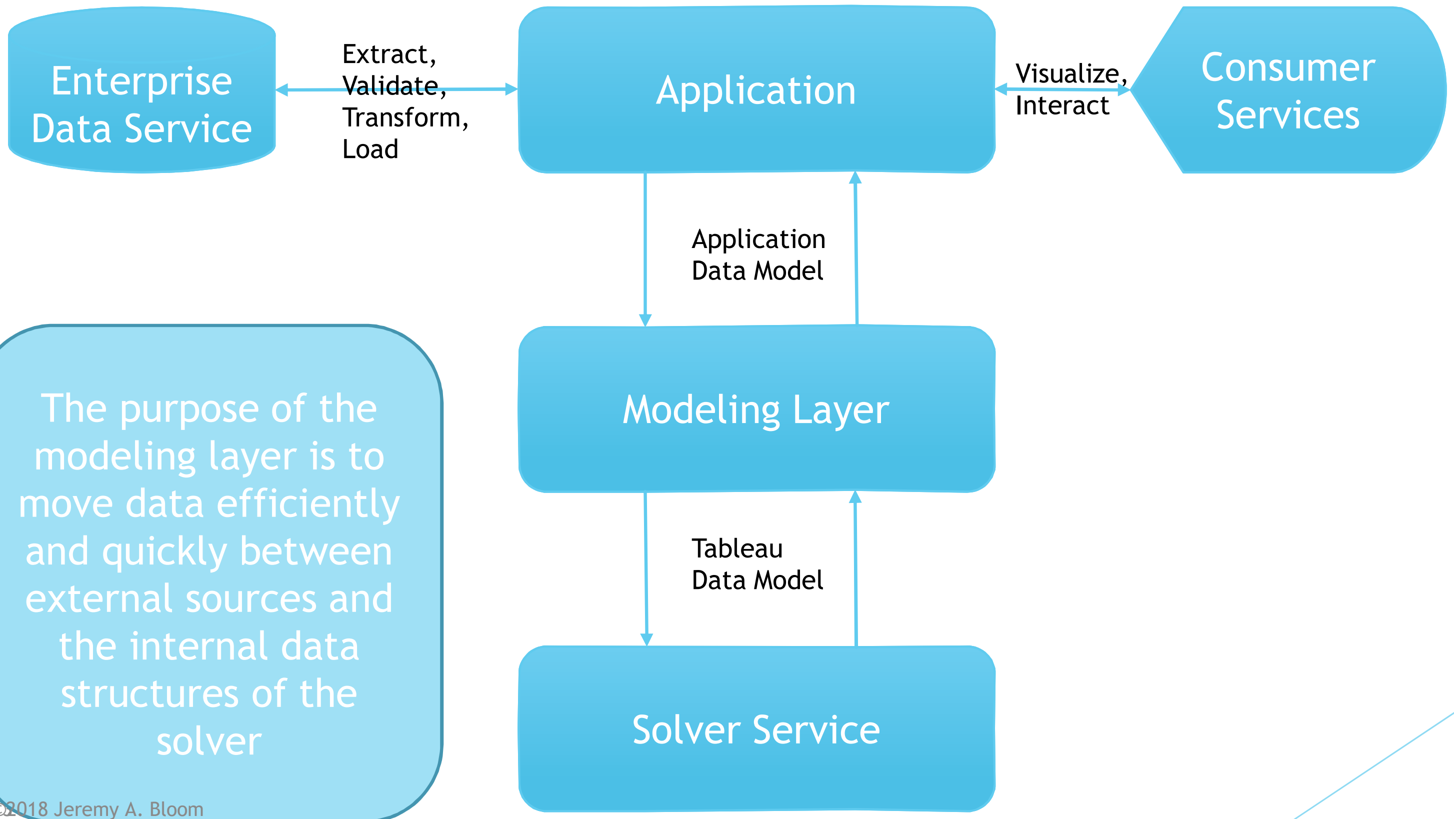
## MOSDEX Examples
## Transshipment Network in Instance Form

- Show demo

## MOSDEX Examples
## Transshipment Network in Query Form

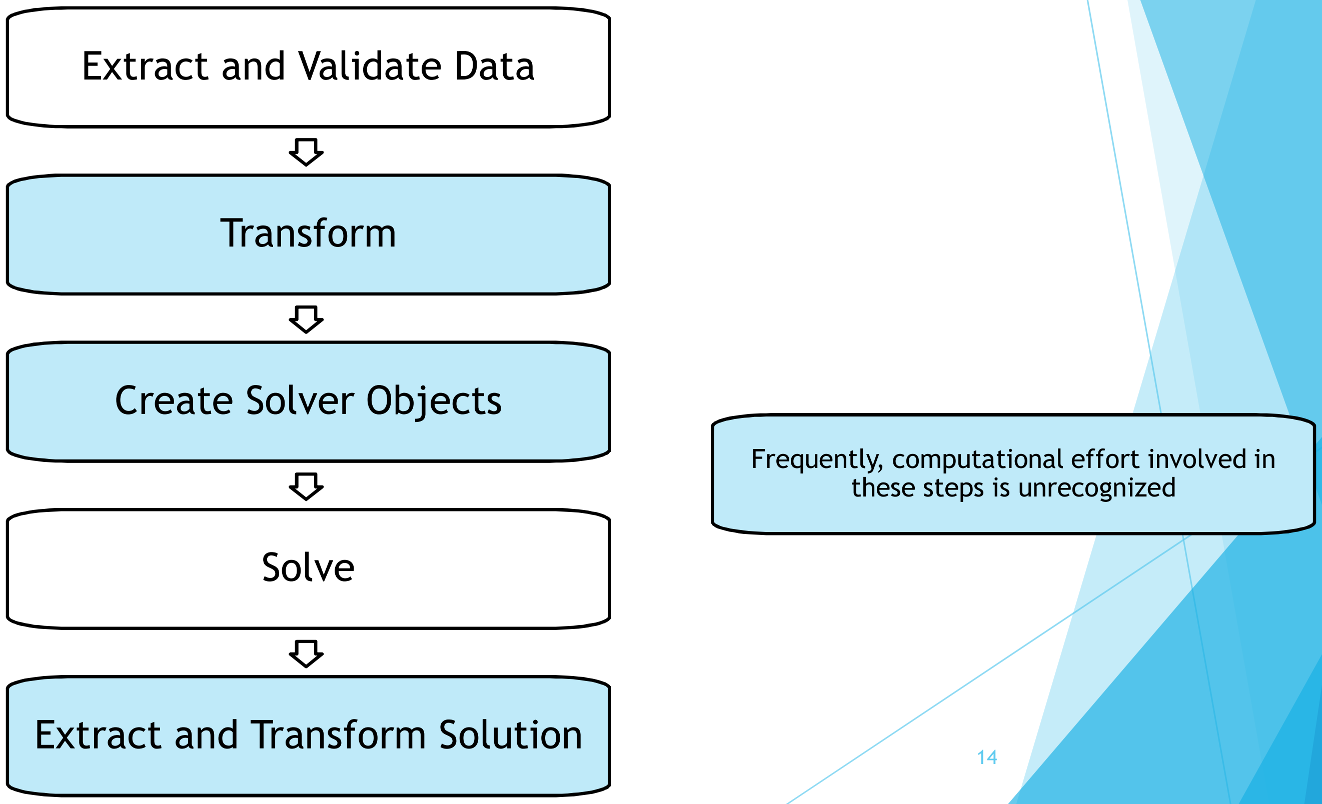
- Show demo

## Business Solution Architecture



The purpose of the modeling layer is to move data efficiently and quickly between external sources and the internal data structures of the solver

## Dataflow for an Optimization Application


Extract and Validate Data
Transform
Create Solver Objects
Solve
Extract and Transform Solution
Frequently, computational effort involved in these steps is unrecognized

## How MOSDEX Supports the Dataflow

- MOSDEX standardizes the dataflow
  - Query-form tables document extraction, validation, and transformation of data in a platform independent manner
- MOSDEX parser translates JSON into the MOSDEX Object Model in the underlying programming language, e.g. C++, Java, Python, etc.
- MODEX Object Model transforms into classes of the solver's API, standardizing the solver interface
- The solver's API classes transform back into the MOSDEX Object Model which then provides access to the solution by the consuming applications; MOSDEX documents the execution of these transformations

## The Future: Big Data and Streams

- Analytics in general is moving strongly and rapidly to applications on big data sets
- Optimization applications already support large data sets (O(1M) variables and O(100K) constraints and solvers on the horizon could increase instance sizes by another order of magnitude or more
  - Examples: control of millions of energy storage batteries in hybrid vehicles, stochastic electricity unit commitment, individualized marketing offers to millions of customers
- Data handling tools such as Hadoop and Apache Spark can now process enormous data sets, using distributed, parallel processing
- Data exchange format for optimization needs to be able to adapt to big data as well

## Big Data Operations

- **Parallel Processing:** Multi-processor architectures are becoming more common that permit operating on data sets in parallel
- **Streaming**: Data flows from a source (e.g. file, Twitter feed, etc.) to a destination (e.g. an optimization solver) without an intermediate resting place
  - Creating intermediate objects can overwhelm the stack space of the processor
  - Streams can support parallel processing on multiple processors
  - Operators work directly on a stream rather than on the individual items in the stream (unlike an iterator)
  - Many languages now support some form of stream processing
- **Transformation**: transform each row of a data set to a new row in another data set (e.g. a Map operation)
- **Terminal Action**: produce a result, e.g. a scalar, that is not a data set (e.g. a Reduce action)
- Big data handlers (e.g. Hadoop, Spark) are optimized to perform these operations efficiently.
  - Transformations are evaluated in *lazy* fashion; that is they are performed only when a action is initialed and they may be reformulated for efficient execution
- Thus, to the extent possible, data handling operations should be performed inside a big data handler

## Additional MOSDEX Capabilities

- Linear, Mixed Integer, and Quadratic Optimization Models
- Modular Structures - Decomposition, Stochastic Programming, etc. (experimental)
- Nonlinear Optimization Models (experimental)
- MOSDEX is designed for extension

## Next Steps for MOSDEX

1. ✓Agree on the basic structure of MOSDEX.
2. ✓ Draft rigorous syntax specifications for the new standard.
3. ✓ Write examples of the new standard using widely understood optimization problems (of which there are many published examples) with a view towards testing and extending the new standard where necessary. Examples should include (but not be limited to) network models, time-staged models involving lagged variables (e.g. production/inventory problems), and stochastic programs.
4. Code samples demonstrating how the new standard utilizes different solvers' APIs and different modeling languages.
5. Code parsers for the new standard for reading and writing files in the various target languages. In this step, adapt existing JSON parsers in the target languages to accept the syntax of the new standard.
6. ✓ Publish the documentation and code developed in the previous steps on the COIN-OR Github site and solicit comments from users.

## Summary

- MOSDEX is intended as a data and model exchange format that supports multiple solver APIs in multiple programming languages and multiple modeling languages
- The MOSDEX includes the following aspects
- It represents the data in relational form
- It uses the JSON standard
- It augments the data representation with mathematical modeling objects
- There is a defined development path for MOSDEX

## MOSDEX Resources

https://github.com/coin-modeling-dev/MOSDEX-Examples/tree/master/MOSDEX-1.2

- MOSDEX Standard
  - MOSDEX Syntax v1-2.docx
- MOSDEX Schema
  - MOSDEXSchemaV1-2.json

## MOSDEX Examples

https://github.com/coin-modeling-dev/MOSDEX-Examples/tree/master/MOSDEX-1.2

- Volsay_1-2.json – a simple 2-variable, 3-constraint linear program illustrating instance form tables
- net1a_1-2.json – a network flow linear program illustrating query form tables
- net1b_1-2.json – the same network flow linear program using instance form tables
- sailco_1-2.json – a production planning linear program illustrating lagged inventory decision variables
- warehousing_1-2.json – a facility location mixed-integer linear program illustrating a large-scale, structured problem in query form
- multicommodity_1-2.json – a multi-commodity network flow linear program in extensive form
- cuttingStock_1-2.json – a cutting-stock mixed-integer linear program illustrating use of modular structure for decomposition in a column generation algorithm
- trafficNetworkQP_1-2.json – a quadratic programming problem
- trafficNetworkNLP_1-2.json – the same problem formulated as a nonlinear program with expression graph