modelsolv

Canonical Modeling for API Interoperability

QCON NEW YORK, JUNE 2014

TED EPSTEIN, FOUNDER AND CEO

MODELSOLV, INC.

Overview

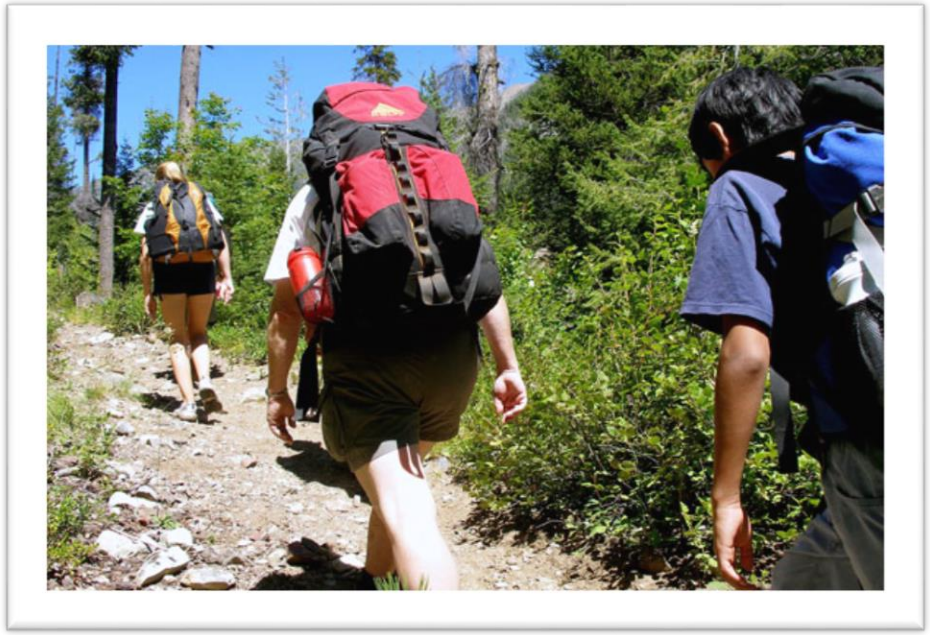
- How APIs help developers, how they don’t
- Canonical models: promises and challenges
- Tackling variability in message representations
- A Better Client Library
- Demo and walkthrough

Developers in the API Jungle

Services Built in isolation

- Many services built to service a single application or org unit.
- No data or API governance, no shared data models.
- No meaningful guidelines. “Just use schema.”
- No easy path to eventual integration or promotion to shared services.

Heavy Burden on Developers



Heavy Burden on Developers

- Find the right service

Heavy Burden on Developers

- Find the right service
- Negotiate data formats

Heavy Burden on Developers

- Find the right service
- Negotiate data formats
- Point-to-point integrations: each one is different

Strained Ecosystem

- Bloated code

Strained Ecosystem

- Bloated code
- Memory footprint

Strained Ecosystem

- Bloated code
- Memory footprint
- High integration costs

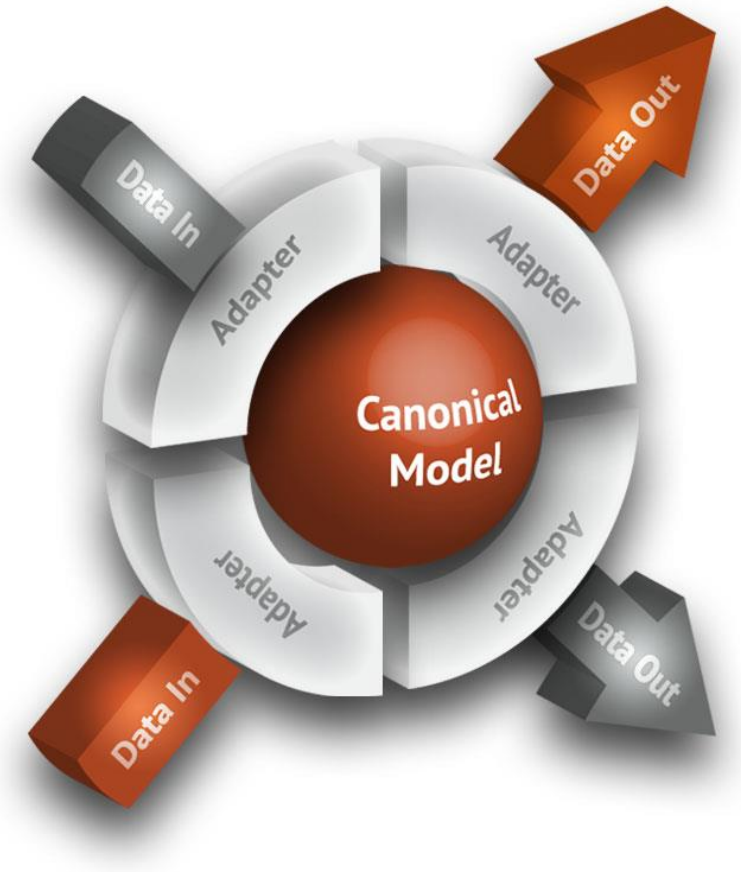
Strained Ecosystem

- Bloated code
- Memory footprint
- High integration costs
- Slow delivery

Canonical Models: Promises and Challenges

A Modest Proposal

- Define a common model for all data communicated between systems.
- Common recommendation in one form or another
  - Patterns of Enterprise Integration (Fowler)
  - SOA Design Patterns (Erl)



Different Uses for Canonical Data Models:

API Design

- SOA Governance: Enforce consistency with the canonical model ("canonical schema")
- Tools: Compose message formats from canonical data models

SDK Development

- Abstract physical format
- Optimize for performance and/or developer productivity

Integration

- Transform messages to/from canonical format.
- Boundary translation to/from industry standards

Analysis

- Analyzing data landscape, service landscape
- Compliance, internal audit, MDM, large-scale integration efforts.

Why is this so hard?

Intrinsic Challenges

- Getting everyone to the table, getting to agreement
- Versioning, Change impact analysis, and lifecycle management

Economic Alignment Challenges

- Service developers need to get it done
- Code-first frameworks promise low-cost to service developers

Modeling Mismatch: Viewpoint

- What's the purpose of the model?
- What level of abstraction? What level of detail?
- How does it related to concrete representations?

Modeling Mismatch: Language and Method

- ER, UML, OWL, etc.
- The attractive nuisance of XML Schema

Result: confusion about what the canonical model is supposed to be.

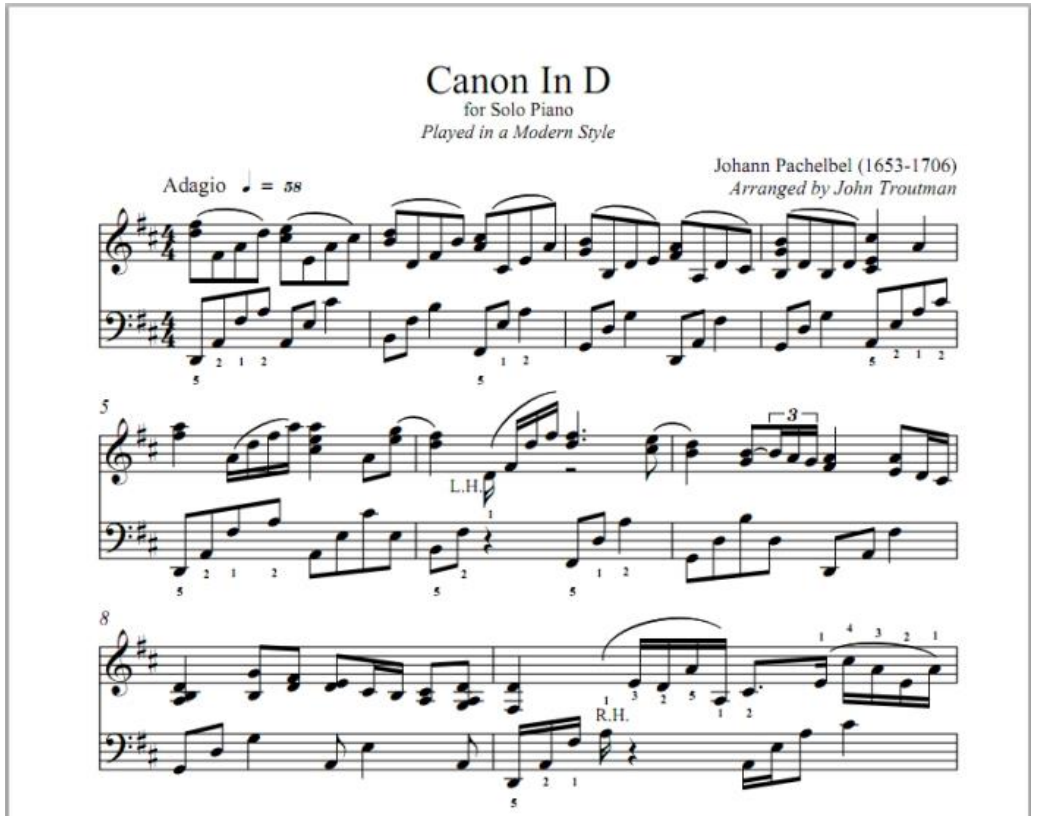
The Pervasive Problem: Variability

- One size fits none
- Message representations vary by
  - Level of detail
  - Perspective
  - Topology, Granularity
  - Contextual constraints
  - Metadata

Realization: Decoupling Models and Messages

Another Look at the Variability Problem

- There's a common theme *(canon)* underneath these variations.
- Can we describe the theme and variations separately?
- Can we model the variations as adaptations, augmentations of the theme?
- There's a name for that: Realization.



Realization: Property Subset

API requests or responses may only need a subset of properties defined in the canonical model.

Realization model may specify a list of included properties.

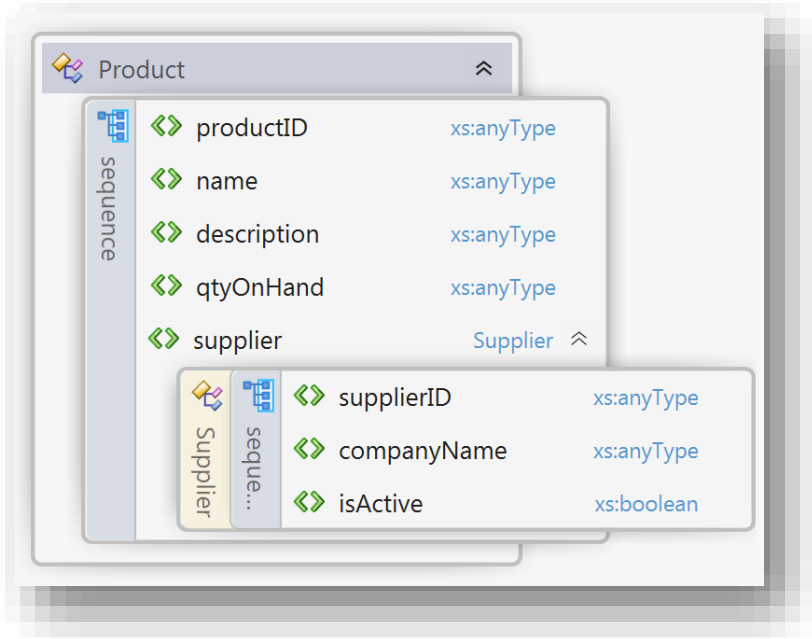
Name	Type	Documentation
filingID	string	A unique, system-assigned identifier for the tax filing.
taxpayer	Person	Reference to the person who owns this filing.
~~jurisdiction~~	~~string~~	~~Country, province, state or local tax authority where this is being filed.~~
year	gYear	Tax year
~~period~~	~~int~~	~~Period within the year, if any~~
~~currency~~	~~string~~	~~Currency code~~
~~grossIncome~~	~~decimal~~	~~Total income reported on tax filing.~~
~~taxLiability~~	~~decimal~~	~~Net tax liability~~

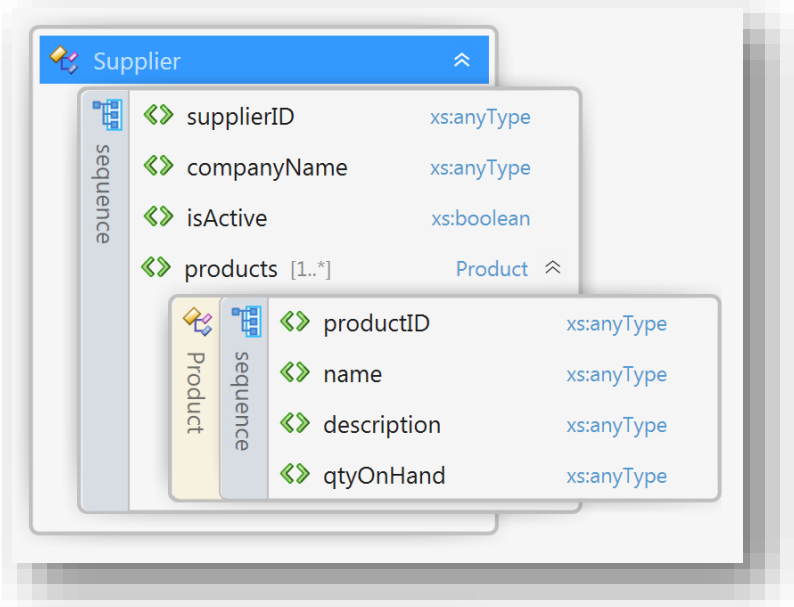
Realization: Perspective

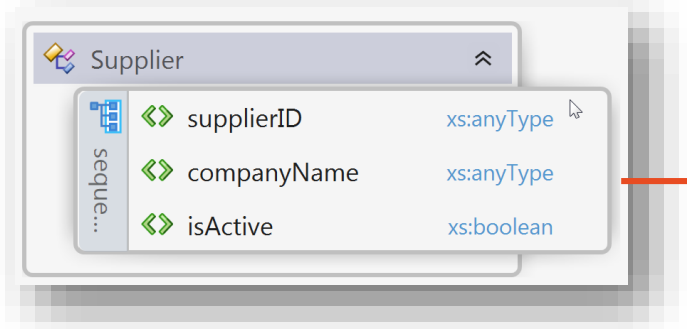
Message and resource structures project different views from the same logical data model

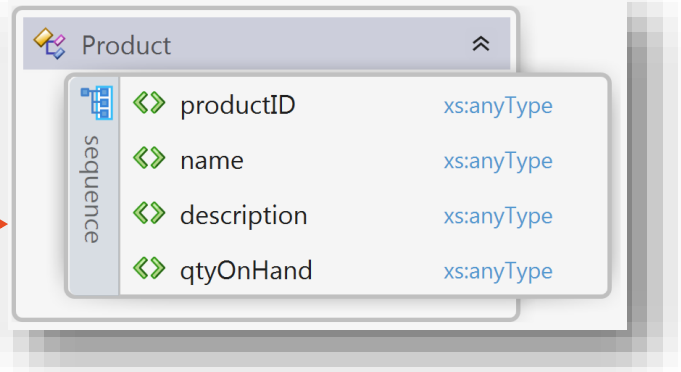
Canonical model should support bi-directional references.

Realization model should allow embedded or linked representations.





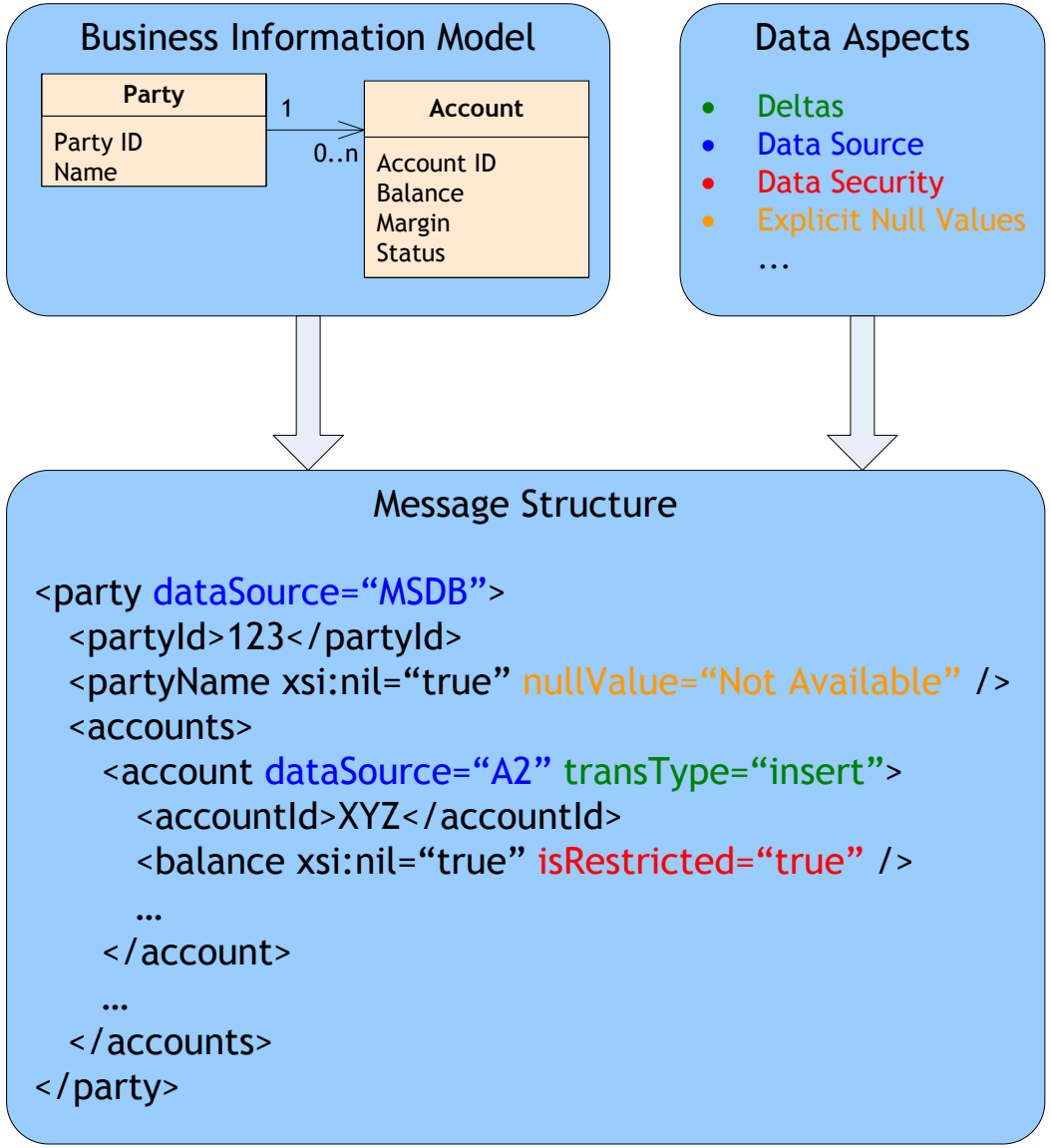




Realization: Metadata

APIs may need to augment essential data with descriptive metadata.

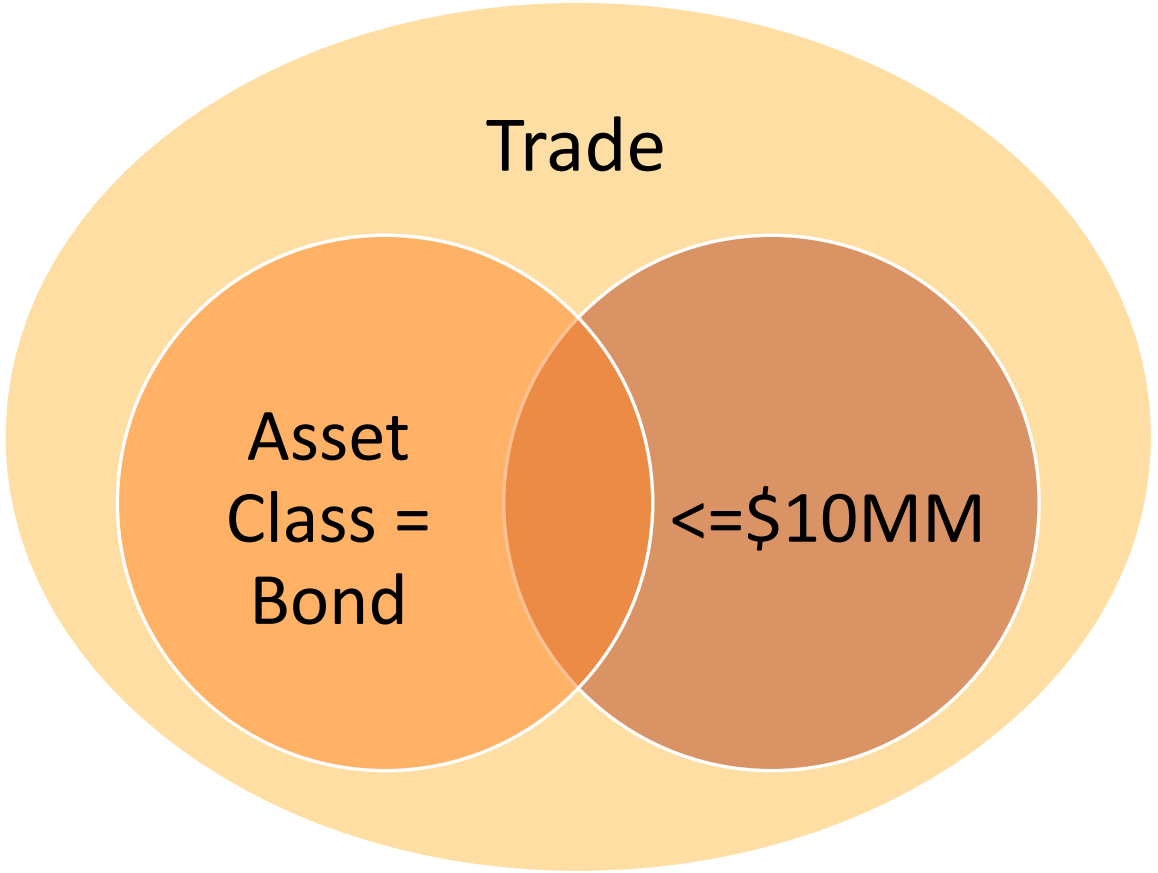
Data aspects are cross-cutting concerns that may be woven together with canonical data as part of the interface realization.



Realization: Contextual Constraints

Services may have specific constraints that are not intrinsic to the data definitions.

Realization model may specify constraints on requests or responses. Constraints may take different forms: range, subtype, logical expression, etc.

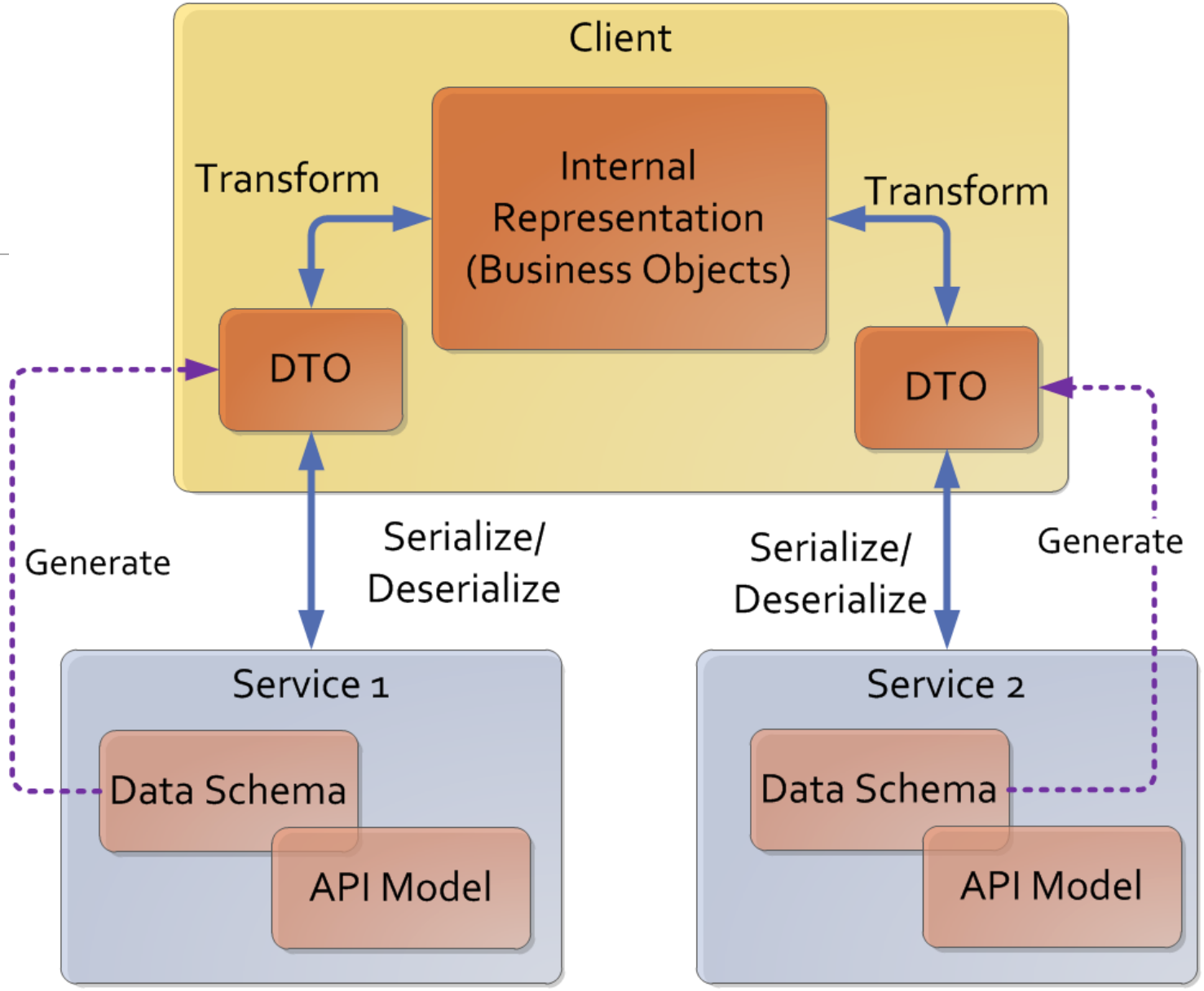


Canonical Modeling Reloaded

- New understanding of the model vs. message:
- Canonical data models describe business information at the conceptual level
  - Semantically rich
  - Technology independent
- Realization models afford variability, with clear limits
  - Bend the canonical model, don't break it
  - Realized representations must be recognizable as instances of the canonical model.
- Some new terms:
  - *Interface Data Model*: A realization model used to define data exchanged through an API.
  - *Resource Data Model*: An interface data model for a RESTful (resource-oriented) API.

A Better Client Library

Today's Client Library



How many ways from Sunday does this suck?

1. Canonical data model gets lost in the translation.
2. Awkward structures introduced by message format.
3. Annotations are specific to a single API, message format.
4. Not usable as business objects
5. Extra code to populate these DTOs, move data between them and internal representations.
6. High memory footprint

Just say no to DTO.

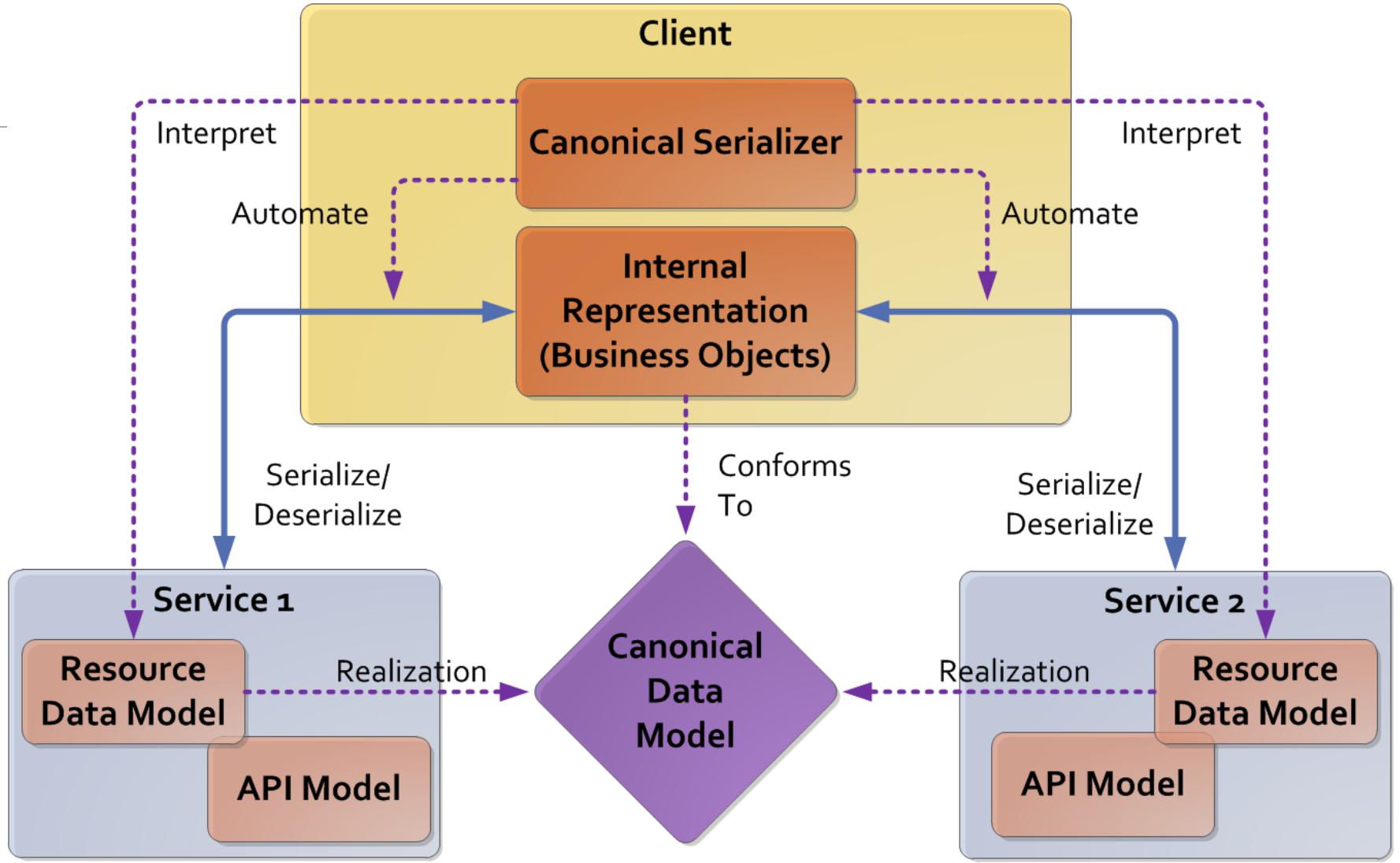
Canonical Serializer

No DTOs, no need to code transformations

Single object graph serialized to multiple RDMs, message formats.

Canonical classes/interfaces can be used as business objects.

Flexibility: pluggable formats for canonical model, RDM, object graph and media type.



Canonical Serializer: Implementation

What do we call a serializer that
shoots representations out of a *canon*?

Kaboom Serializer!

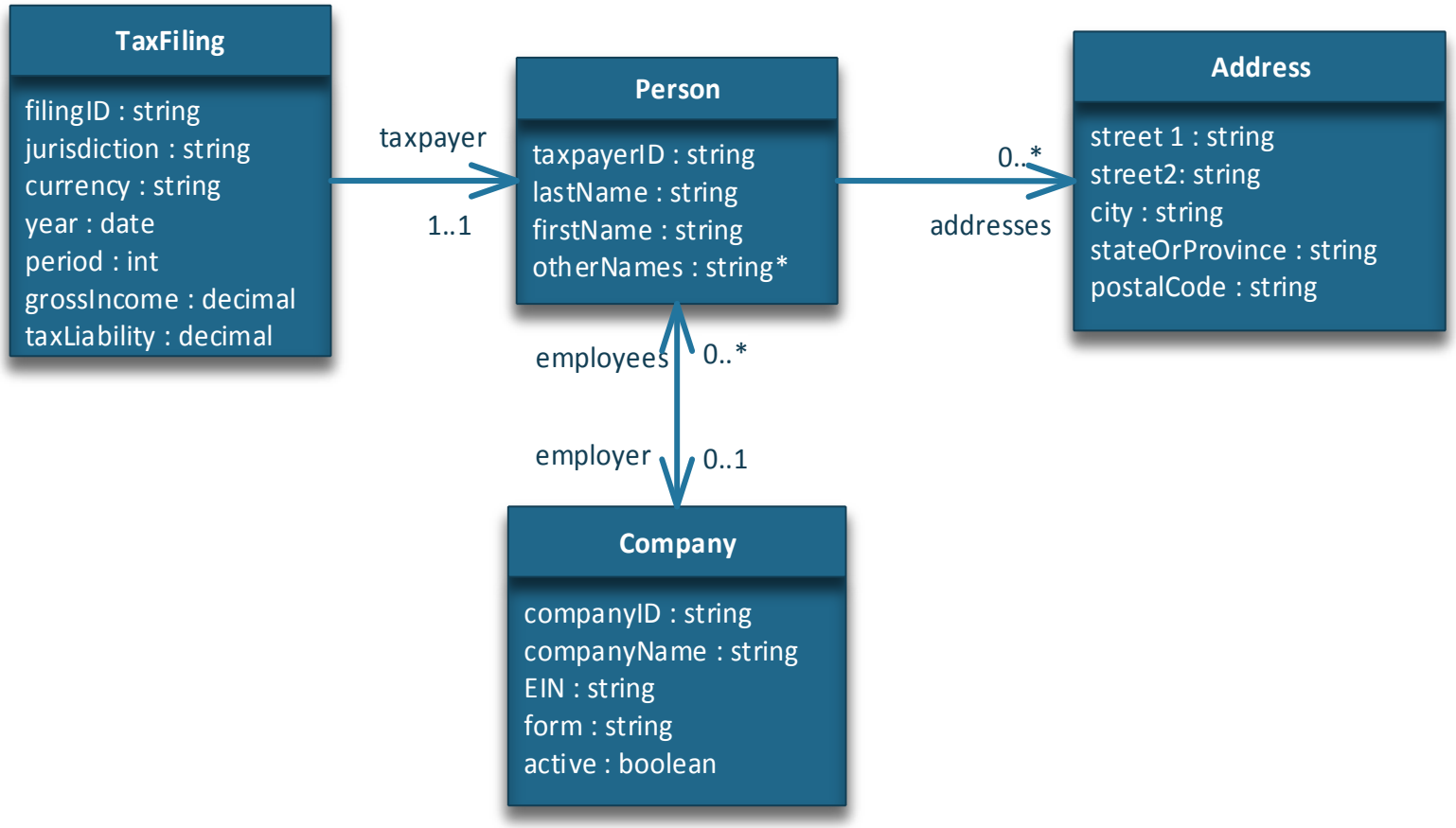
https://github.com/modelsolv/Kaboom

(Just a demo now, but feedback & contributions welcome.)

Scenario

- TaxBlaster: new tax preparation app.
- Integrates with e-filing service
- Integrates with client billing service
- Common data model, different views

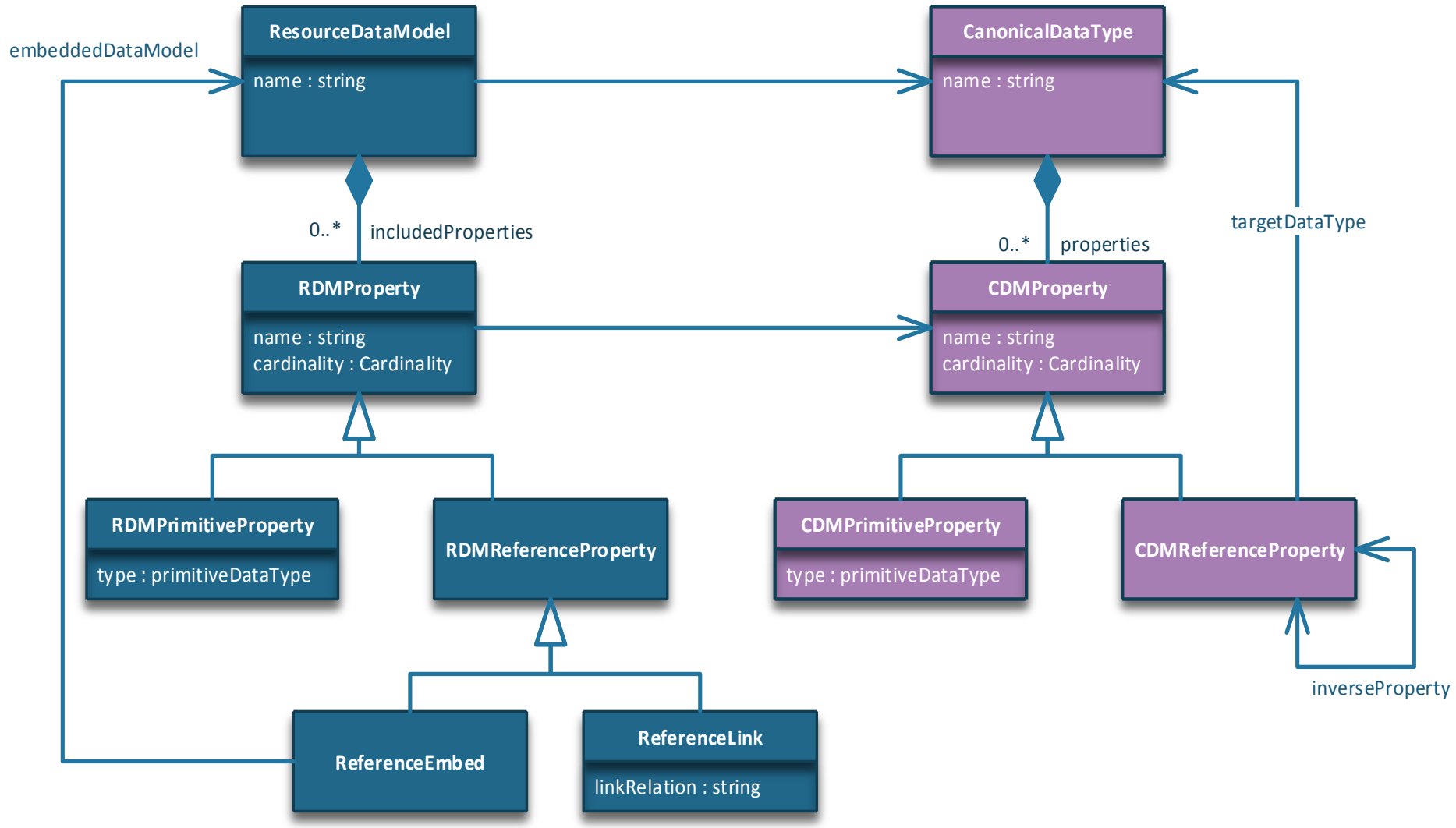
TaxBlaster: Canonical Data Model



Internal Metamodel

Pluggable implementations:

- CDM
- RDM
- Canonical Object Reader/Writer
- Serializer



Recipe for a model-oriented API client

- A canonical modeling language
- Data available at runtime that conforms to the canonical model
- An API description facility that
  - Realizes the canonical model as an Interface Data Model
  - Described as formalized variations
- A runtime serializer/deserializer that
  - Interprets the API model
  - Serializes and deserializes between the the canonical object graph and the the realized message format.

Conclusion

Canonical models are a way to capture organizational agreement on data definitions

We need the right degree of coupling between the canonical model and API representations.

We do this by identifying the kinds of variations that we need to support, and formalizing these in a realization mapping.

Tooling can support realization modeling and apply it in client libraries, SDKs, middleware, etc.

Potential benefits: better interoperability, lower integration cost, higher developer productivity.

Questions

THANK YOU!