Semantic Models That Run:

From CSV
to Context-Rich
Linked Data

A live, standards-based transformation pipeline.

Powered by a real-time domain model in Jargon.

No hardcoded rules.

No hidden scripts.

Just declarative mapping, semantic precision, and end-to-end transparency.

A public demonstration - submitted to the OMG Semantic Augmentation Challenge 2025 and shared openly with the world so we can reclaim the executable benefits that modelling promised - but never fully delivered.

> # *Semantic Models That Run: From CSV to Context-Rich Linked Data*
>
> ### *What if meaning could be added to raw data — and made executable, not just descriptive?*

CSV isn't just flat - it's fragile. No datatypes. No relationships. No identifiers. No context. Every value is just a string - and every integration relies on humans guessing what columns mean. That's the semantic gap this challenge asked us to close.

The challenge asked for a mapping format - a way to describe how columns relate to semantics. This submission includes just that - mapping that is: readable, declarative, and aligned to real-world vocabularies like FIBO, GeoSPARQL, and schema.org. BUT the mapping is just one artefact. **What you'll see here is a fully working system** - one that transforms CSV into structured JSON, validates it with JSON Schema, expands it with JSON-LD, and outputs RDF - all with no custom code.

This demo shows how semantic models can lift those flat structures into context-rich, machine-readable JSON-LD, using declarative mappings and live validation, all driven by a single source of truth.

From this:

```
SERVTYPE_DESC,ADDRESS,CITY,COUNTY,STALP,ZIP
"FULL SERVICE - BRICK AND MORTAR","18001 Saint Rose Rd", "Breese", "Clinton", "IL", "62230"
```

To this:

```

{
    "https://banks.data.fdic.gov/ontology/hasServiceType": {
         "@type": "http://www.w3.org/2001/XMLSchema#string",
         "@value": "FULL SERVICE - BRICK AND MORTAR",
    },
    "fdic:address": [
      {
        "https://schema.org/addressLocality": [
          {
            "@type": "http://www.w3.org/2001/XMLSchema#string",
            "@value": "Breese"
          }
        ],
        "fdic:county": [
          {
            "@type": "http://www.w3.org/2001/XMLSchema#string",
            "@value": "Clinton"
          }
        ],
        "https://schema.org/addressRegion": [
          {
            "@type": "http://www.w3.org/2001/XMLSchema#string",
            "@value": "IL"
          }
        ],
        "https://schema.org/streetAddress": [
          {
            "@type": "http://www.w3.org/2001/XMLSchema#string",
            "@value": "18001 Saint Rose Rd"
```

```
          }
        ],
        "https://schema.org/postalCode": [
          {
            "@type": "http://www.w3.org/2001/XMLSchema#string",
            "@value": "62230"
          }
        ]
      }
}
```

This is a fully runnable transformation pipeline, powered by real-time domain models authored in Jargon.

At the centre of this solution is a live Jargon model. It isn't a static diagram. It's the working source that drives every transformation, validation, and semantic expansion in this demo. Every step is traceable, standards-aligned, and running in real time.

> ***Reusable and Extensible***
> *This solution is not tied to FDIC data. The mapping format and model-driven approach apply to any structured dataset - enabling scalable, repeatable semantic transformations across domains. Whether banking, insurance, supply chains or climate data; the same workflow applies.*

The process begins with a simple CSV file. A declarative mapping - aligned to the domain model - transforms that input into structured JSON. The result is then validated using JSON Schema, and expanded with JSON-LD to include globally meaningful identifiers and types - all derived from the same model.

> The process is driven by The Model that Drives Everything. The only other input is the competition's CSV file. The transformation pipeline proceeds through these stages:

- Use the Mapping to transform CSV to structured JSON
- Validate the transformation with JSON Schema
- Expand the JSON with JSON-LD and RDF vocabularies
- Validate semantic alignment of the expanded JSON

The model also includes vocabularies such as GEOSPARQL, schema.org, and selected elements of FIBO. These were transformed from published RDF sources into reusable domain models inside Jargon. Some were imported directly. Others, like FIBO, use cherry-picked IRIs.

Once imported, these behave like native domains: composable, semantic, and runnable - without OWL reasoning or traditional ontology tooling.

> *This submission isn't a prototype or wireframe. It's a running, standards-based implementation.*
> *Every step is live. Every artifact is generated directly from the model.*
>
> *Traditional modeling tools focus on completeness or theoretical rigor. But they often stop short of producing outputs that real systems can use. This results in models which may look precise, but are still reliant on brittle scripts or manual interpretation.*
>
> *This demo takes a different approach. The Jargon model powers everything you see.*
> *JSON Schema, OpenAPI, JSON-LD, RDF, and validation logic all come from a single, traceable source of truth.*
> *No translation layer. No handoffs. No drift.*
>
> *This is modeling that runs. Not simulation. Not documentation. A living contract between data, systems, and semantics - in real time.*

This demonstration uses W3C and industry standards from start to finish:

- **JSON Schema** for validation
- **JSON-LD** for semantic context
- **RDF** for graph reasoning

Jargon makes these standards usable without custom code, scripts, or hand-built vocabularies.

The Model That Drives Everything

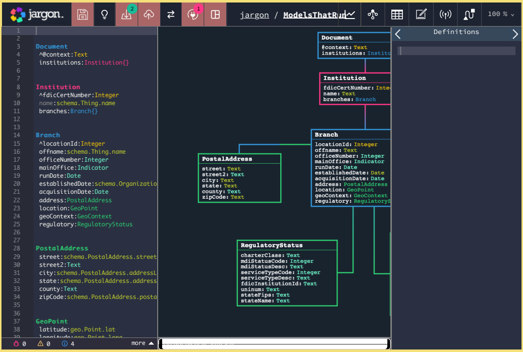
This isn't a diagram. It's the source of truth — and the engine behind everything you're about to see.

And Jargon is the engine behind the model:

- It defines the structure and relationships in the data
- It links each property to its source column
- It generates the mapping, schema, context, and OpenAPI spec automatically
- It serves the model live, so you can trace every semantic rule back to its source

Every transformation in this demo starts here and stays aligned throughout. No manual edits. No duplication. No drift.

> ***This isn't a screenshot. It's a live Domain Model served by Jargon.***
> *What you see below is the actual model running behind this demo - the same source that powers every mapping, validation, and semantic rule on the page.*
> *Not a mockup. Not a snapshot. A real model, live and in use.*



```

Document
 ^@context:Text
 institutions:Institution{}


Institution
 ^fdicCertNumber:Integer
 name:schema.Thing.name
 branches:Branch{}


Branch
 ^locationId:Integer
 offname:schema.Thing.name
 officeNumber:Integer
 mainOffice:Indicator
 runDate:Date
 establishedDate:schema.Organizatio
 acquisitionDate:Date
 address:PostalAddress
 location:GeoPoint
 geoContext:GeoContext
 regulatory:RegulatoryStatus


PostalAddress
 street:schema.PostalAddress.street
 street2:Text
 city:schema.PostalAddress.addressL
 state:schema.PostalAddress.address
 county:Text
 zipCode:schema.PostalAddress.posta


GeoPoint
 latitude:geo.Point.lat
```

🚀 Open the Full Domain Model in Jargon

CSV Input

Just rows and columns for now — but it's the foundation we'll transform into something semantically rich.

This is the raw data used in the challenge: a flat CSV file with basic policyholder and provider details. **Other than the domain model, this is the only input required.**

> ***This is a live, working system.***
> *You can modify the CSV below. Your changes will be used as inputs to the mapping process.*
> *There's no hidden logic or manual setup. The structure, relationships, and validation are all driven by the model.*
>
> ***Note:***
>
> *This demo runs on a 100-row sample of the official FDIC dataset to ensure instant responsiveness in the browser. The full 80,000-row dataset has been processed using the exact same model and mapping — with no code changes. The resulting expanded JSON-LD output (over 450MB) is included in the submission ZIP as evidence of successful full-scale execution.*

```
X,Y,OBJECTID,ACQDATE,ADDRESS,ADDRESS2,BKCLASS,CBSA,CBSA_DIV,CBSA_DIV_FLG,CBSA_DIV_NO,CBSA_METRO,CBSA_METRO_FLG,CBSA_METRO_NAME,CBSA_MIC
RO_FLG,CBSA_NO,CERT,CITY,COUNTY,CSA,CSA_FLG,CSA_NO,ESTYMD,FI_UNINUM,ID,LATITUDE,LONGITUDE,MAINOFF,MDI_STATUS_CODE,MDI_STATUS_DESC,NAME,
OFFNAME,OFFNUM,RUNDATE,SERVTYPE,SERVTYPE_DESC,STALP,STCNTY,STNAME,UNINUM,ZIP
-89.554910003,38.6842860180001,1,42539,18001 Saint Rose Rd,,SM,"St. Louis, MO-IL",,0,,41180,1,"St. Louis, MO-
IL",0,41180,14761,Breese,Clinton,"St. Louis-St. Charles-Farmington, MO-
IL",1,476,30964,9231,223055,38.6842860182208,-89.5549100030003,0,,NULL,1NB Bank,ST. ROSE FACILITY BRANCH,3,45401,11,FULL SERVICE -
BRICK AND MORTAR,IL,17027,Illinois,223055,62230
-89.372215024,38.6181820130001,2,,1350 12th St,,SM,"St. Louis, MO-IL",,0,,41180,1,"St. Louis, MO-IL",0,41180,14761,Carlyle,Clinton,"St.
Louis-St. Charles-Farmington, MO-IL",1,476,25373,9231,232078,38.6181820130991,-89.3722150239032,0,,NULL,1NB Bank,1350 12TH STREET
FACILITY,1,45401,11,FULL SERVICE - BRICK AND MORTAR,IL,17027,Illinois,232078,62231
-89.612338026,38.6117310070001,3,,500 W Harrison St,,SM,"St. Louis, MO-IL",,0,,41180,1,"St. Louis, MO-
```

Mapping Definition

The bridge between model and data — generated, not hand-written, and entirely traceable.

This mapping shows how each CSV column connects to elements in the domain model. It wasn't written by hand. Jargon generated it automatically based on the model. Each field you see is derived from a class or property in that model.

> ***Model-driven and web-native:***
> *To include a property in the mapping, the author just adds a `[csv.columnName]` annotation in the model.*
>
> ***Jargon does the rest:***
> *Generating a plain JSON mapping file - inspired by JSONPath, but tailored for CSV-to-JSON transformation.*
>
> *Want to see how it works? [View the mapping format.]*

GENERATED MAPPING (FROM JARGON)

```
{
  "type[]": {
    "constant": "Document"
  },
  "institutions[{@CERT}].fdicCertNumber": {
    "column": "CERT",
    "type": "number"
  },
  "institutions[{@CERT}].type[]": {
    "constant": "Institution"
  },
  "institutions[{@CERT}].name": {
    "column": "NAME",
    "type": "string"
```

Transform CSV to JSON

Here's where the model proves itself. One click turns tabular data into structured content - ready for meaning.

This is where all of the pieces come together. Where the magic happens.

You've seen the model and its mapping. Now you can run the transformation.

The button below takes the CSV input and applies the mapping generated by Jargon. It's not handcrafted logic. It's model-driven structure running in real time.

This transformation is live and reactive. Modify the CSV above and rerun this step to see updated results immediately.

Transform

GENERATED JSON (LIVE)

```
Click "Transform" to generate JSON from the CSV input and mapping above.
```

Validate with JSON Schema

We check the shape of the data — and ensure it matches the model's expectations exactly.

The specific structure of the transformed JSON isn't what matters — it's that the structure we use is known, descriptive, and enforceable. Without that, we can't reliably apply meaning or process the data. JSON Schema gives us that guarantee.

The schema shown here wasn't written by hand. Jargon generated it automatically from The Model that Drives Everything — the single source of truth for this entire pipeline. You can verify this by opening the dropdown in the embedded model above - the schema matches exactly.

Jargon doesn't just help you design models. It makes sure those designs are enforced, from transformation to validation and beyond.

> ***A quick note on the source data:*** *the provided CSV contained a number of structural inconsistencies — the same logical columns represented in different ways (for example, 1 vs TRUE, date serials vs date strings), and even a repeated header row midway through the file.*
>
> *I don't know the provenance of the file, and it may represent a heroic effort to gather data from diverse contributors. But this is precisely why structural validation matters. A standards-based JSON Schema can catch these inconsistencies early — helping teams avoid costly surprises downstream.*

JSON SCHEMA (FROM JARGON)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "type": {
      "type": "array",
      "readOnly": true,
      "default": [
        "Document"
      ],
      "items": {
        "type": "string"
      },
```

Validate

```
Click "Validate" to check the generated JSON aligns with the JSON Schema.
```

Expand with JSON-LD

Now every field gains meaning. The data isn't just structured — it's semantically explicit.

Even richly structured JSON isn't enough on its own. Machines still need help understanding *what* the data means, not just *how* it's shaped.

This step uses the `@context` generated by Jargon to expand each field into a fully qualified semantic identifier. It's how we turn column names into machine-readable references that align with shared vocabularies.

⚙ JSON-LD @CONTXT (FROM JARGON)

```
{
  "@context": {
    "fdic": "http://example.com/fdic",
    "schema": "https://schema.org/",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "@protected": true,
    "@version": 1.1,
    "type": {
      "@id": "@type"
    },
    "Branch": {
      "@protected": true,
      "@id": "fdic:Branch",
      "@context": {
```

> ***This is the core requirement of the Semantic Augmentation Challenge:***
> *"Show a mapping of columns in a machine-readable and processable format to common and citable resources."*

That's exactly what this step does. Using a model-generated `@context`, we resolve fields like:

- `dateEstablished → schema:foundingDate`
- `branchAddress → schema:streetAddress`
- `numOffices → xsd:integer`

This isn't just prettier JSON. It's enriched, contextual data - ready for validation, integration, querying, or reasoning by any Linked Data system.

The result below shows the fully expanded form in both JSON-LD and RDF (Turtle/N-Quads) views:

Expand

JSON-LD | RDF (N-Quads)

🎉 EXPANDED JSON-LD (LIVE)

```
Click "Expand" to see expanded JSON-LD.
```

Validate Semantic Alignment

It's not enough to say the data is enriched — this proves it was enriched correctly.

Once data has been expanded into its semantic form using JSON-LD, the question remains: **"was it expanded correctly?"** Were the right URIs applied? The right structure used? The right context referenced?

Jargon generates a second-layer validation schema — one that operates not on the raw data, but on the **expanded JSON-LD graph**. This schema confirms that:

- The correct URIs were applied to each field
- The expected data types and structures match the semantics
- The correct JSON-LD context was used during expansion

This acts as a contract: not just that the data is present, but that it is *semantically aligned* and trustworthy — without relying on SHACL or OWL.

- `providerId → schema:providerId`
- `policyType → omg2:policyType`

⚙ SEMANTIC JSON SCHEMA (FROM JARGON)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$defs": {
    "Document": {
      "type": "object",
      "required": [
        "@type"
      ],
      "properties": {
        "@type": {
          "type": "array",
          "contains": {
            "const": "fdic:Document"
          }
```

Validate Semantics

```
Click "Validate Semantics" to check the expanded JSON contains the expected semantic mappings.
```

From Model to Vocabulary

This is where the semantics become portable — described in RDF, aligned to standards, and always in sync with the model.

The Jargon models aren't only describing structure - they define meaning. This section shows the machine-readable vocabulary generated automatically from the same domain model used for transformation and validation.

The output includes both a **JSON-LD graph** and a **RDF (N-Quads)**, declaring every class and property with types, labels, domains, and ranges. It uses standard ontologies like `rdfs`, `owl`, `xsd`, and `schema.org`.

> ***This makes the semantics portable.***
> *Any system that understands RDF - whether for validation, alignment, or integration - can work with this model directly.*

Because the vocabulary comes from the same source as the schema and mappings, it stays aligned.
No duplication. No drift.

JSON-LD Vocabulary | RDF Vocabulary

JSON-LD VOCABULARY (FROM JARGON)

```
{
  "@context": {
    "fdic": "http://example.com/fdic",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "owl": "http://www.w3.org/2002/07/owl#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "schema": "https://schema.org/",
    "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#"
  },
  "@graph": [
    {
      "@id": "fdic:Document",
      "@type": "rdfs:Class",
```

Handling Change Over Time

Data evolves. Models drift. This shows how Jargon manages semantic change without losing control.

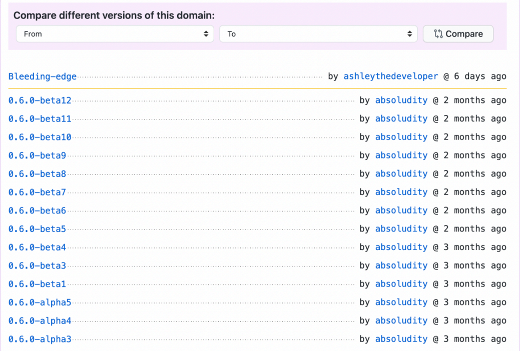
As datasets evolve, metadata can easily fall out of sync. Columns change. Definitions shift. Without versioning, even well-modeled data can become misleading.

This demonstration shows a snapshot in time, but the platform behind it - Jargon - is designed for ongoing change:

- Every domain model is versioned, with immutable references for each release.
- Mappings, schemas, and contexts are generated from the model, so they stay current by design.
- When updates are needed, Jargon provides diff views, suggestions, validation rules, and release workflows to manage change.

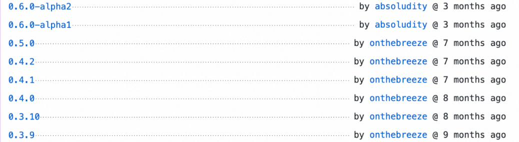
Jargon acts as a semantic version control system. Keeping metadata aligned as the meaning of data evolves.

These screenshots are taken from the UNTP Core domain, an active project used by the United Nations Transparency Protocol team. They show how Jargon supports versioning, collaboration, and semantic reuse in real-world modeling use case.



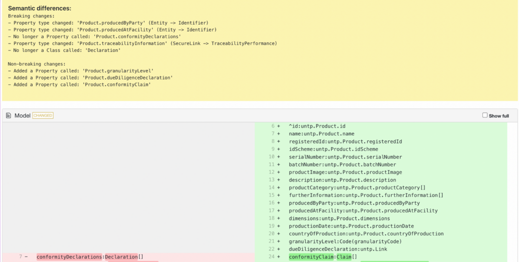
Compare different versions of this domain:

Bleeding-edge by ashleythedeveloper @ 6 days ago
0.6.0-beta12 by absoludity @ 2 months ago
0.6.0-beta11 by absoludity @ 2 months ago
0.6.0-beta10 by absoludity @ 2 months ago
0.6.0-beta9 by absoludity @ 2 months ago
0.6.0-beta8 by absoludity @ 2 months ago
0.6.0-beta7 by absoludity @ 2 months ago
0.6.0-beta6 by absoludity @ 2 months ago
0.6.0-beta5 by absoludity @ 2 months ago
0.6.0-beta4 by absoludity @ 3 months ago
0.6.0-beta3 by absoludity @ 3 months ago
0.6.0-beta1 by absoludity @ 3 months ago
0.6.0-alpha5 by absoludity @ 3 months ago
0.6.0-alpha4 by absoludity @ 3 months ago
0.6.0-alpha3 by absoludity @ 3 months ago



```
0.6.0-alpha2 ........................ by absoludity @ 3 months ago
0.6.0-alpha1 ........................ by absoludity @ 3 months ago
0.5.0 ............................... by onthebreeze @ 7 months ago
0.4.2 ............................... by onthebreeze @ 7 months ago
0.4.1 ............................... by onthebreeze @ 7 months ago
0.4.0 ............................... by onthebreeze @ 8 months ago
0.3.10 .............................. by onthebreeze @ 8 months ago
0.3.9 ............................... by onthebreeze @ 9 months ago
```

Version history for a published model



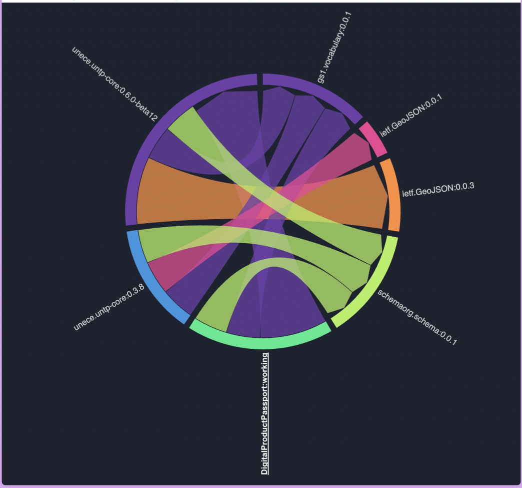
Semantic differences:

```
Breaking changes:
- Property type changed: 'Product.producedByParty' (Entity -> Identifier)
- Property type changed: 'Product.producedAtFacility' (Entity -> Identifier)
- No longer a Property called: 'Product.conformityDeclarations'
- Property type changed: 'Product.traceabilityInformation' (SecureLink -> TraceabilityPerformance)
- No longer a Class called: 'Declaration'

Non-breaking changes:
- Added a Property called: 'Product.granularityLevel'
- Added a Property called: 'Product.dueDiligenceDeclaration'
- Added a Property called: 'Product.conformityClaim'
```

```
                                                        6 +   ^id:untp.Product.id
                                                        7 +   name:untp.Product.name
                                                        8 +   registeredId:untp.Product.registeredId
                                                        9 +   idScheme:untp.Product.idScheme
                                                       10 +   serialNumber:untp.Product.serialNumber
                                                       11 +   batchNumber:untp.Product.batchNumber
                                                       12 +   productImage:untp.Product.productImage
                                                       13 +   description:untp.Product.description
                                                       14 +   productCategory:untp.Product.productCategory[]
                                                       15 +   furtherInformation:untp.Product.furtherInformation[]
                                                       16 +   producedByParty:untp.Product.producedByParty
                                                       17 +   producedAtFacility:untp.Product.producedAtFacility
                                                       18 +   dimensions:untp.Product.dimensions
                                                       19 +   productionDate:untp.Product.productionDate
                                                       20 +   countryOfProduction:untp.Product.countryOfProduction
                                                       21 +   granularityLevel:Code(granularityCode)
                                                       22 +   dueDiligenceDeclaration:untp.Link
  7 -   conformityDeclarations:Declaration[]              24 +   conformityClaim:Claim[]
 10 -   traceabilityInformation:untp.SecureLink[]         27 +   traceabilityInformation:TraceabilityPerformance
```

Semantic diff and model change viewer



Visual dependency graph for reuse and alignment

Want to explore it live? Visit the **UNTP Core domain** to see how shared vocabularies are managed across multiple projects.

Bringing It All Together

Not just a one-off demo — a repeatable system where models, mappings, and semantics stay in sync

This approach isn't just about one dataset. It's about a repeatable, model-driven pipeline that keeps everything aligned:

Jargon defines the meaning.
Every class, property, and rule comes from a domain model. No scattered config. No guesswork.

The CSV holds the raw facts.
No markup or preprocessing. Just data.

Jargon generates everything else.
- A declarative `mapping.json`
- A `schema.json` for validation
- A JSON-LD `@context` for semantics

The pipeline can run anywhere - even in your browser.

It's powered by the model, not by scripts or custom code.

The output is semantically enriched.
The final JSON aligns with shared vocabularies like Schema.org, FIBO, and GeoSPARQL.

> ***Metadata isn't bolted on. It's shaped by the model itself.***
> *This is how Jargon answers the challenge: a semantic pipeline that begins with a model and ends with trusted, machine-readable information - not just data, but semantics that run.*

One More Thing…

Yes, Jargon even generates API designs. The same model that structures and enriches the data also powers real-world integration.

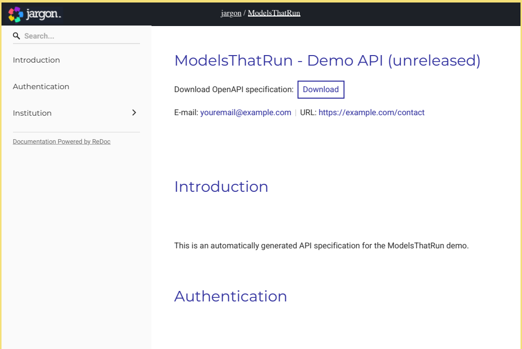
All of this is possible because of the underlying model. But what if you wanted to build a real system on top of it?

> ***Jargon also generates a full OpenAPI specification****, including request and response shapes, semantic field names, and example payloads.*

This same model can:

- Validate data using JSON Schema
- Provide semantic context using JSON-LD
- Define integration contracts using OpenAPI
- Track versions and changes across all of it

This isn't just about mapping. It's infrastructure built on models. The outputs are consistent, governed, and ready to run.



ModelsThatRun - Demo API (unreleased)

Download OpenAPI specification: Download

E-mail: youremail@example.com | URL: https://example.com/contact

Introduction

This is an automatically generated API specification for the ModelsThatRun demo.

Authentication

🚀 **Open the Full API Specification**

What Just Happened?

You've just walked through a complete semantic transformation - from a flat CSV file to structured, machine-readable JSON-LD.

What makes it different is *how* it was done:

- The model wasn't hardcoded. It was built visually, once, and used everywhere.
- Semantics were part of the model from the start.
- The mapping to JSON was declarative and driven by structure.
- Expansion used a generated `@context`, not handwritten rules.

Every artifact - the mapping, schema, context, and output - came from a single source of truth. Nothing is stitched together. Nothing can go out of sync.

This isn't just format conversion. It's semantics that run. And it shows how models can become real, working parts of systems - with immediate feedback and no ambiguity.

> *For years, we built models that were too complicated for people to use - and still not useful to machines.*
>
> *This demonstration flips that. The model is simple enough to understand, and structured enough to run.*
>
> *It doesn't replace semantic modeling - it builds on it and applies it where it matters most.*
> *By making semantics meaningful for machines.*
>
> *By making them Run.*

Reflections

This demonstration was built to answer a technical challenge, which it did by applying structured modeling — but the unspoken, deeper challenge behind the competition is a strategic one.

We've spent decades drawing models, publishing standards, and writing specs. But somewhere along the way, modeling lost its footing. It became too abstract for developers, too brittle for change, and too removed from the systems it was meant to serve.

The results are clear:

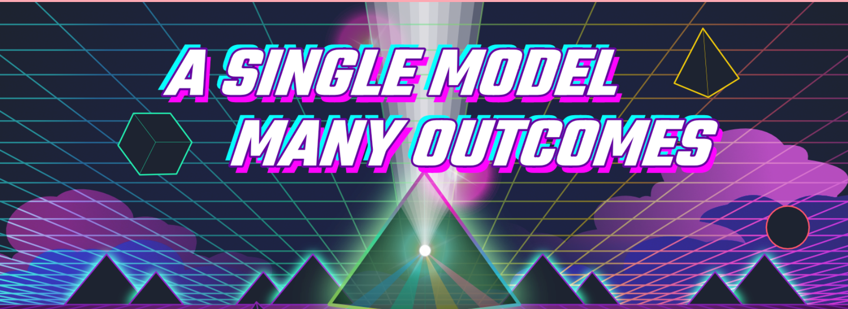
- The last two generations of developers never picked it up.
- Startups rejected it outright.
- Standards have struggled to keep pace with the speed of real software delivery.

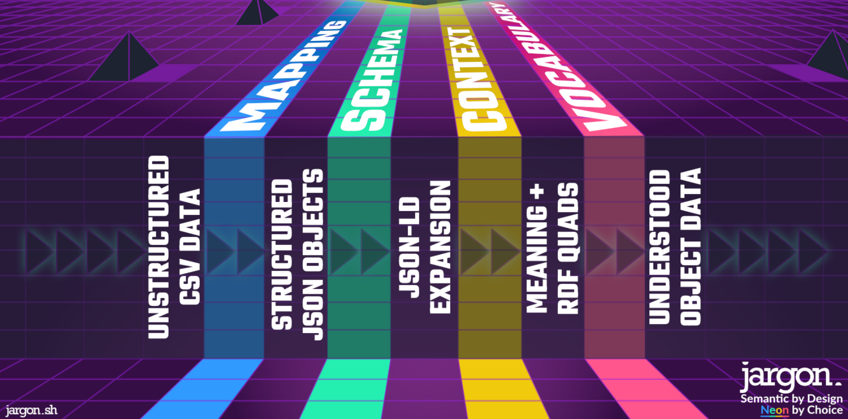
This approach shows another way:

- The model isn't documentation — it's the engine.
- The outputs aren't placeholders — they run in real pipelines, with tools teams already use.
- Semantics aren't layered on — they emerge naturally from structure, meaning, and context.

No triple store. No OWL. No SHACL.
Just a single, live model — powering every transformation, validation, and API contract.
Executable. Traceable. Governable.

> *It looks like modeling.*
> *It feels like modeling.*
>
> *But it runs.*
>
> *This isn't a reinvention of UML.*
> *It's a quiet demonstration of what modeling could have become — and maybe still can.*
>
> *If this feels familiar, like something you once tried to do but didn't quite land, that's because it is. I built it to show that it's possible. Not in theory. In practice. And this submission is the proof.*
>
> *If the time feels right to try again — I'd welcome the conversation.*
> *`alastair [at] jargon [dot] sh`*





UNSTRUCTURED CSV DATA

MAPPING

STRUCTURED JSON OBJECTS

SCHEMA

JSON-LD EXPANSION

CONTEXT

MEANING + RDF QUADS

VOCABULARY

UNDERSTOOD OBJECT DATA

jargon.

Semantic by Design
Neon by Choice

jargon.sh