Lowering the Barrier: An Architecture for Creating, Maintaining and Executing Mappings

Pim Mulder[1], Wouter van den Berg[1], Jan Pieter Wijbenga[1] and Cornelis Bouter[1]

[1]TNO, The Netherlands

Abstract

The Data Act requires semantic assets to be published transparently and consistently, but leaves open how integration between heterogeneous participant models should be managed. We address this by implementing a mapping pipeline that treats mappings as first-class semantic assets. The pipeline introduces two profiles created in this work: SSSOM$^+$ for conceptual alignment at design time, and JSONata$^+$ for executable transformation logic at run time. A drag-and-drop mapping workflow captures annotated design intent, and an LLM-asssisted translation workflow performs elicitation, mapping generation, and validation to pass the mapping intent over to an executable translation request. We evaluated the approach in a transport order matching use case with independently modeled participant systems. The resulting architecture shows that mappings can be managed as semantic assets and that separating conceptual alignment from executable transformation logic is a workable and useful design principle for Eventual Interoperability.

Keywords

Data Act, Semantic Interoperability, Mapping Governance, SSSOM, JSONata, LLM-assisted mapping

1. Introduction

The Data Act (Regulation (EU) 2023/2854, article 33) requires that vocabularies, data structures, and classification schemes are published in a transparent and consistent way. The standardisation request M/614 relies on the Data Space Support Center (DSSC) Blueprint to operationalise this requirement. It does so mainly through shared models and gradual convergence. Although this direction is useful, it does not eliminate the problem for organisations that continue to maintain different message models.

This gap is especially visible in logistics. The sector consists of many small and medium-sized enterprises (SMEs) that work with existing software, limited IT budgets, and message schemas that were often created ad hoc for a specific customer or process. Consequently, replacing those schemas with one shared standard is often unrealistic. If these parties want to cooperate they will need to rely on an alternative manner of integration such as semantic mappings.

Our motivation is a transport-order matching use case in which carriers seek to jointly plan transport in order to minimise empty kilometres, without exposing sensitive operational data. Each carrier expresses orders in its own model, as they do not have the capacity to converge to a standard such as the Open Trip Model (OTM)[1]. As such, interoperability depends on translating between models instead of replacing them.

Colpaert's notion of Eventual Interoperability [1] offers the following perspective: organisations first model data in their own context and create alignments later when a concrete business case demands it. This shifts semantic interoperability from harmonising data models upfront to governing the connections between them. However, those connections are still rarely managed or recognised as semantic assets in their own right, either by participants, the Data Act or the DSSC.

We address this gap with an architecture that separates conceptual alignment from executable transformation logic, i.e. a distinction between mapping and transforming. The design-time side

Seventh International Workshop on Semantics for Transport and Logistics Co-located with the SEMANTiCS Conference 2026 Ghent - 15th of September 2026

pim.mulder@tno.nl (P. Mulder); wouter.vandenberg@tno.nl (W. van den Berg); jan_pieter.wijbenga@tno.nl (J. P. Wijbenga); cornelis.bouter@tno.nl (C. Bouter)

0009-0000-1471-7705 (W. van den Berg); 0009-0004-1534-0566 (J. P. Wijbenga)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]https://opentripmodel.org/

captures alignment intent in SSSOM$^+$, the run-time executes JSONata$^+$, an LLM-assisted translation component connects both. Our contribution is that we show technical feasibility in a logistics pilot and that it offers a practical way to document, version, inspect and execute mappings.

The remainder of this paper is structured as follows. Section 2 clarifies the distinction between mapping and transformation and positions our approach with respect to SSSOM and EDOAL. Section 3 presents the architecture and specifies SSSOM$^+$ and JSONata$^+$ used in our pipeline. Section 4 further explains this by walking through an end-to-end logistics example. In section 5 we evaluate the architecture through implementation to establish if this approach is technically feasible. Section 6 summarises our contribution and outlines future work and further implications for treating alignments as first-class semantic assets.

2. Background

The DSSC Blueprint operationalises the Data Act's transparency requirement through shared models, vocabulary governance, and discovery between data spaces. That direction is useful, but it does not eliminate the need to map between message models that remain heterogeneous in practice. Our focus is on those mappings, especially for syntax-specific serialisations such as XML, JSON, YAML, and CSV that are not always expressed as Linked Data in RDF.

This is not a new problem. RDF mapping languages such as RML [2] and YARRRML [3] already address mapping to RDF. Related work also shows that functions matter. Demeester et al. [4] propose a function hub and ontology so that functions can be reused across implementations.

In their survey on Knowledge Organization Scheme mappings, Dos Reis et al. [5] show that source models evolve and that maintenance requires substantial human effort. They also note that some changes appear in naturally described items such as labels and descriptions, while others appear in formal constraints such as OWL axioms. SeMoX [6] shows how standardisation and tools can support multiple serialisations and links to global standards. BOMOS [7] offers a similar practical standardisation governance method and is already used in Dutch government settings.

Albeit from another angle, the literature on vocabulary hubs reaches the same conclusion that interoperability work is not only about defining standards, but also about continuously managing mappings between evolving models in practice. Dedecker et al. [8] argue that vocabulary hubs should support both mapping and vocabulary design by helping users discover reusable data models and avoid duplicate work. David et al. [9] propose a layered federation approach, although this only partly resolves the interoperability problem between vocabulary hubs [10]. Semantic Treehouse (STH) [11] is the vocabulary hub environment used for this paper. It is an open-source, collaborative model-management environment that supports publishing, versioning, validation, example messages, usage guidance and generators for different syntaxes. STH uses the approach: open standards in, open standards out. It is used for both larger international models and for smaller, locally tailored profiles with their own release cycles. It features a bottom-up wizard, allowing models to be derived from CSV or Excel input data that can then be mapped to a collective standard. The ~8 communities using STH provide testing grounds for validating new interoperability solutions such as this mapping approach.

The paper is grounded in design science research. The Hevner et al. guidelines [12] fit the work because we build an artifact, evaluate it in a relevant setting, and refine it iteratively. Halpin and Morgan [13] stress expressivity, clarity, examples and validation by non-technical domain experts, which matches our concern with readable mappings. Alvarez et al. [14] make a similar point when arguing for a simplified approach that remains compatible with existing standards but is better suited for SME use. For evaluation, we follow the distinction between ex ante and ex post, and between naturalistic and artificial settings, from [15]. A naturalistic setting is possible here because the STH ecosystem has a number of active communities.

This literature leads to the distinction used in the rest of the paper. One layer captures conceptual mapping intent. Another layer captures executable transformation. The next section explains the full architecture.

3. Architecture

We divide the architecture into a design-time side and a run-time side. The mapping (design-time) requires expressiveness over preciseness, whereas the the transformation (run-time) has it the other way around. The pipeline of our architecture contains three loosely coupled components: A design-time mapping editor, a run-time message translator, and a translation component that derives an executable mapping from the conceptual one.

As stated, the components are loosely coupled. Semantic Treehouse (STH) is useful for creating an SSSOM$^+$ mapping, but it is not a hard dependency of the approach. The message translator's JSONata$^+$ configuration can also be authored without using the full pipeline. However, for clarity of presentation we describe the components as one end-to-end flow.

3.1. SSSOM$^+$: Conceptual Alignment

For expressing conceptual alignment of two message models, we use SSSOM as the base model and serialise the result as RDF/Turtle. In STH, mapping designers (from here on called 'user') can create relations between fields of two message models through a drag-and-drop interface. Each relation is stored as an SSSOM mapping entry, where the relation type is expressed with a SKOS predicate, e.g. `skos:exactMatch`, `skos:closeMatch`, or `skos:relatedMatch`. Mappings can be annotated with a natural-language description of their intent. This might include lookup logic, value conversion, or pseudocode. We decided not to require a formalised mapping notation for mapping intent. This choice keeps the process inclusive, because mapping relations can be understood and reviewed without a technical background.

SSSOM was chosen because it gives a lightweight, open, and syntax-independent way to express mappings with explicit provenance, predicates and identifiers. Alternatives such as EDOAL[2] are more formal and more ontology-centric. This can be useful for ontology alignment. However, in our motivational case, the message models are created ad hoc (e.g. from CSV files) and are not necessary aligned to an ontology.

We use the shorthand SSSOM$^+$ for this constrained profile. This does not describe a new standard, rather the annotated combination of SSSOM and RDF that stores the transformation intent in human-readable form.

3.2. JSONata$^+$: Data Transformation

The run-time configuration should be a representation that is directly executable. We chose JSONata [16] as the expression language and chose to wrap it in a configuration profile that the translator service can validate and execute. We refer to the shorthand JSONata$^+$, but it is best understood as a JSONata-based mapping schema rather than as a new transformation language.

JSONata was chosen, because it is declarative, JSON-native, and readable enough for the majority of implementers who already work with JSON-like data [17]. Compared with alternatives such as JOLT, jq, or XSLT/XPath, it gives the best balance between structural expressiveness and operational simplicity.

The selection was also pragmatic: JOLT offered limited flexibility for custom logic, jq is primarily a command-line tool and awkward to embed as a service dependency, and XML-oriented stacks such as XSLT/XPath add unnecessary complexity for a JSON-first runtime. We therefore use JSONata as the executable core and place project-specific constraints in the JSONata$^+$ profile around it.

3.3. From Design to Run-time

With both profiles in place, the remaining component translates conceptual intent into executable logic. We implemented a microservice that translates SSSOM$^+$ to JSONata$^+$ through a closed-loop workflow:

[2] https://ns.inria.fr/edoal/1.0/

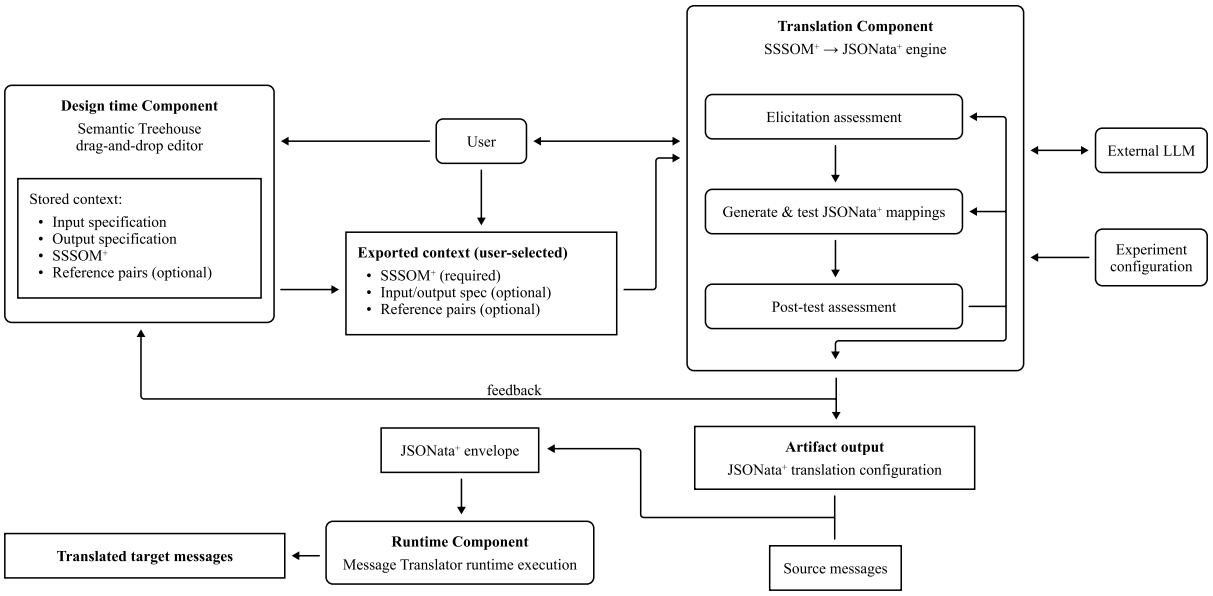

Figure 1: End-to-end mapping pipeline. A user creates a conceptual alignment in STH. The translation component performs elicitations, candidate generation and validation to derive a JSONata$^+$ configuration. The run-time translator then applies the configuration to translate source messages to the intended target format. Session outcomes and lessons learned are fed back to improve the SSSOM$^+$, input and output specification descriptions.

1. **Elicitation assessment.** An LLM evaluates whether the provided context is sufficient to generate a correct mapping. The input, the context, always contains the SSSOM$^+$ export and may contain source and target specifications, sample messages, or reference input-output pairs. When the LLM determines that required information is missing or that a transformation choice cannot be resolved without making an assumption that could produce incorrect output, it asks focused clarification questions. The loop continues until the LLM assesses the context as sufficient.
2. **Candidate generation and testing.** Once context is sufficient, the LLM generates multiple JSONata$^+$ candidates. Each candidate is tested against a configurable suite of experiments. Each experiment has a threshold and a direction (minimise or maximise). A candidate passes when all experiments meet their threshold. The current prototype features one implemented experiment that measures syntactic validity, with a threshold of zero errors to minimise.
3. **Post-test assessment.** Before the result is offered to the user, a final LLM evaluates the test results and mapping to decide whether the passing candidate is acceptable. If not, it can decide to return to regeneration or re-elicitation to resolve remaining ambiguity.

The state of each session is persisted as a sequence of immutable events. This makes the LLM interaction inspectable and allows a user to review how the system moved from the conceptual mapping to the executable result. The full set of tasks of the user are to provide context, answer any questions and review the generated result.

4. End-to-End Example

To illustrate the pipeline, this section walks through a non-trivial end-to-end example taken from the transport order use case, based on real data. The mapping connects a carrier-specific order schema to a universal order model. Rather than describing both models in full, we introduce the relevant fields as they appear in the mapping.

For this example, we first imported source model, using the bottom-up approach from real dta, because no schema was provided. Target model was entered into the tool as fit by the use case.Next, we manually created a mapping in Semantic Treehouse and exported it as SSSOM$^+$. Listing 1 shows three of its relations. The first maps `order.orderNumber` to `company_reference` as an exact match.

Listing 1: SSSOM+ sample: a direct match, annotated derivation, and two pairwise entries for a multi-field relation.

```
<.../Mapping_orderNumber>
  a sssom:Mapping ;
  owl:annotatedSource
    <.../Order/order/orderNumber> ;
  sssom:subject_label
    "Order/order/orderNumber" ;
  owl:annotatedTarget
    <.../Order/company_reference> ;
  sssom:object_label
    "Order/company_reference" ;
  owl:annotatedProperty skos:exactMatch ;
  sssom:mapping_justification
    semapv:ManualMappingCuration ;
  owl:versionInfo "final" .

<.../Mapping_containerOwner>
  a sssom:Mapping ;
  owl:annotatedSource
    <.../Order/order/container/number> ;
  sssom:subject_label
    "Order/order/container/number" ;
  owl:annotatedTarget
    <.../Order/container_owner> ;
  sssom:object_label
    "Order/container_owner" ;
  owl:annotatedProperty skos:broadMatch ;
  sssom:mapping_justification
    semapv:ManualMappingCuration ;
  owl:versionInfo "final" ;
  rdfs:comment "ISO 6346 container number.
    Use the first four characters
    as the container owner." .
```

```
<.../Mapping_dateToOrigin>
  a sssom:Mapping ;
  owl:annotatedSource
    <.../Order/order/stops/stop/date> ;
  sssom:subject_label
    "Order/order/stops/stop/date" ;
  owl:annotatedTarget
    <.../Order/stops_time_window/origin> ;
  sssom:object_label
    "Order/stops_time_window/origin" ;
  owl:annotatedProperty skos:relatedMatch ;
  sssom:mapping_justification
    semapv:ManualMappingCuration ;
  owl:versionInfo "final" ;
  rdfs:comment "Date of the stop.
    For origin: use the first stop
    (lowest sequence number)." .

<.../Mapping_timeToOrigin>
  a sssom:Mapping ;
  owl:annotatedSource
    <.../Order/order/stops/stop/time> ;
  sssom:subject_label
    "Order/order/stops/stop/time" ;
  owl:annotatedTarget
    <.../Order/stops_time_window/origin> ;
  sssom:object_label
    "Order/stops_time_window/origin" ;
  owl:annotatedProperty skos:relatedMatch ;
  sssom:mapping_justification
    semapv:ManualMappingCuration ;
  owl:versionInfo "final" ;
  rdfs:comment "Time of the stop.
    For origin: use the first stop
    (lowest sequence number)." .
```

The second maps `order.container.number` to `container_owner` as a broad match and annotates it to explain that the first four characters of a container number describe the owner of the container. The third and fourth entries show how SSSOM+ handles a relation that involves multiple source fields. The origin time window requires both a stop date and a stop time, each is recorded as a separate entry with an annotation to identify what it is needed for in this multisource relation.

In our example, this SSSOM+ export, together with the specification of the source and target is given to the translation component. In this iteration, the system determined that there was insufficient context. As such, it asked focused questions such as “What must the city lookup use as the key?” and “What must the mapping do when the city lookup fails?”.

Following elicitation, the engine repeatedly tried to generate and assess a correct mapping. After the third attempt, it achieved a mapping without any syntactic errors that was also sufficient for the post-assessor. Listing 2 shows a selection of the generated fields. The `company_reference` is a direct copy of `order.orderNumber`. The `container_owner` is derived with `$substring()`. The origin coordinates are resolved by looking up the lowercased city name in an embedded table. The coordinate reference system is injected as a constant. The time window boundaries are computed by concatenating the stop’s `date` and `time` fields into an ISO datetime string, then offsetting by a configured number of milliseconds.

5. Implementation and Evaluation

We implemented the proposed architecture in the context of the order matching use case described in the introduction. The goal of the implementation was to determine whether the architecture is technically possible and whether it makes mapping implementations more manageable in practice.

The design-time component was implemented and was used to define a complete mapping from one order model to the target specification. STH has generated the SSSOM+ mappings for this. The run-time translator is also operational and has processed more than 1,000,000 samples in the pilot without run-time failures. Encountered issues were caused by mistakes in manually authored mappings rather than failures in the translator itself.

The separate components were tested, but the fully integrated pipeline is to be tested as future work. The implemented translation component was verified through ad hoc testing and showed that it can be

Listing 2: JSONata⁺ sample: selected fields from the generated mapping.

```
{ "onMissing": "error",
  "tables": {
    "cities_lat": { "rotterdam": "51.9225", "antwerpen": "51.2213", ... },
    "cities_lon": { "rotterdam": "4.4776",  "antwerpen": "4.4117",  ... }
  },
  "fields": [
    { "target": "company_reference",
      "source": "order.orderNumber" },
    { "target": "container_owner",
      "expression": "$substring(order.container.number, 0, 4)" },
    { "target": "stops_locations.origin.crs",
      "constant": "WGS84" },
    { "target": "stops_locations.origin.lat",
      "expression": "$number($lookup('cities_lat', $lowercase(order.stops[0].location.address.city)))" },
    { "target": "stops_time_window.origin.start",
      "expression": "$fromMillis($toMillis(order.stops[0].date & 'T' & order.stops[0].time) - 3600000)" }
  ]
}
```

used to translate a non-trivial mapping into a transformation configuration. However, repeated runs with equivalent inputs do not yet follow a stable path, so the robustness should be further increased for production use.

Even with that limitation, the pilot yielded useful design lessons. Lookup tables are not exceptional but recurring, often re-used among multiple field mappings. SKOS relations without additional annotations do not provide enough information to define meaningful executable transformations. Additional annotations were expressed as semi-formal code and were shown to be effective, without forcing users to learn one specific formalisation. Also, error handling and missing-value behavior should be configurable and explicit. These insights have been incorporated in the architecture described in this paper.

6. Conclusion and Future Work

The Data Act and related standards (cf. Mandate 614) require semantic transparency through the publication of semantic assets in a consistent manner. But in reality it is hard to manage the connections between those inevitably heterogeneous semantic assets. This problem is especially relevant for SMEs, for example within the logistics domain. We proposed an architecture in which conceptual alignment is captured in SSSOM⁺, executable logic is expressed in JSONata⁺, and an LLM-assisted component connects both through elicitation, generation and validation.

Through implementation and use case validation we showed that this architecture is technically feasible and that mappings can be treated as explicit semantic assets. We validated the separate components with real data and schemas from a logistics use case, also performing run-time translation. The consistency and robustness of the system should be further increased, by further strengthening the experimentation suite used for the candidate testing, i.e. input-output-model validation, coverage checks, synthetic test runs. The treatment of multi-field relations should be improved using native SSSOM fields. The evaluation of the pipeline itself should be strengthened to include more formal testing so that further improvements surface more easily. Finally, the pipeline should be combined into an integrated user experience so that there is one integrated mapping environment for a user, rather than three separate components.

Acknowledgments

AI tools were used to support spell checking, improve consistent word usage, and to generate the source vocabulary used in the end-to-end example. The work presented in this paper was done in the context

of the TNO Project Future Proof Smart Logistics. We acknowledge the efforts of the logistics service providers in providing data and schemas.

References

[1] P. Colpaert, Eventual interoperability, 2026. URL: https://pietercolpaert.be/interoperability/2026/01/08/eventual-interoperability, blog post. Accessed: 2026-07-09.

[2] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, The rml ontology: A community-driven modular redesign after a decade of experience in mapping heterogeneous data to rdf, in: T. R. Payne, V. Presutti, G. Qi, M. Poveda-Villalón, G. Stoilos, L. Hollink, Z. Kaoudi, G. Cheng, J. Li (Eds.), The Semantic Web – ISWC 2023, Springer Nature Switzerland, Cham, 2023, pp. 152–175.

[3] D. Van Assche, T. Delva, P. Heyvaert, B. De Meester, A. Dimou, Towards a more human-friendly knowledge graph generation & publication, in: Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice, volume 2980, CEUR Workshop, 2021.

[4] B. De Meester, T. Seymoens, A. Dimou, R. Verborgh, Implementation-independent function reuse, Future Generation Computer Systems 110 (2020) 946–959. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19303723. doi:https://doi.org/10.1016/j.future.2019.10.006.

[5] J. dos Reis, C. Pruski, C. Reynaud, State-of-the-art on mapping maintenance and challenges towards a fully automatic approach, Expert Systems with Applications 42 (2015) 1465–1478. doi:10.1016/j.eswa.2014.08.047.

[6] A. Schmitz, C. Pauken, R. Kottmann, M. A. Wimmer, Semox: Standardizing standardization – a semantic-driven approach for aligning data standards, in: I. Lindgren, M. P. Rodríguez Bolívar, M. Janssen, E. Loukis, F. Mureddu, P. Panagiotopoulos, G. Viale Pereira, E. Tambouris (Eds.), Electronic Government, Springer Nature Switzerland, Cham, 2026, pp. 196–212.

[7] E. Folmer, Bomos: Management and development model for open standards, in: Handbook of research on e-business standards and protocols: Documents, data and advanced web technologies, IGI Global Scientific Publishing, 2012, pp. 102–128.

[8] J. R. M. Ruben Dedecker, P. Colpaert, Raising the Role of Vocabulary Hubs for Semantic Data Interoperability in Dataspaces, in: Extended Semantic Web Conference (ESWC 2026): The Fourth International Workshop on Semantics in Dataspaces, May 10-11, 2026, Dubrovnik, Croatia, Dubrovnik, Croatia, 2026.

[9] R. David, V. Alexiev, P. Ivanov, W. van den Berg, Federated vocabulary hubs as a foundation for semantic layers in data spaces, in: Proceedings of the Third International Workshop on Semantics in Dataspaces (SDS 2025) co-located with the Extended Semantic Web Conference (ESWC 2025), Portorož, Slovenia, 2025.

[10] R. García Castro, M. Poveda Villalón, On the need for ontology hub interoperability, in: The Fourth International Workshop on Semantics in Dataspaces, CEUR-WS, 2026. URL: https://oa.upm.es/96528/, ontology Engineering Group OEG.

[11] J. B. Bootsma, J. Wijbenga, L. Oosterheert, M. Stornebrink, W. van den Berg, Establishing semantic interoperability across data spaces: a solution for sharing vocabularies, Technical Report 2024 - R10911, TNO, the Netherlands, 2024. URL: https://publications.tno.nl/publication/34642521/4YA9tB/TNO-2024-R10911.pdf.

[12] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, Management Information Systems Quarterly 28 (2004) 75–105.

[13] T. Halpin, T. Morgan, Information Modeling and Relational Databases, 3rd ed., Morgan Kaufmann, 2024.

[14] D. Alvarez-Coello, A. Bekan, S. Schleemilch, D. Wilms, S. Lawrence, T. Guild, A. Sayegh, C. Podalakuru, S. Schildt, A. Achtzehn, S2dm: a simplified semantic data modeling approach, in: Sixth International Workshop on a Semantic Data Space for Transport, co-located with the 21st Interna-

tional Conference on Semantic Systems (SEMANTiCS 2025), Vienna, Austria, 2025. URL: https://semantic-transportation.github.io/sem4tra-kg-website/2025/papers/Sem4Tra25_paper_5.pdf.

[15] J. Venable, J. Pries-Heje, R. Baskerville, A comprehensive framework for evaluation in design science research, in: K. Peffers, M. Rothenberger, B. Kuechler (Eds.), Design Science Research in Information Systems. Advances in Theory and Practice, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 423–438.

[16] JSONata.org, JSONata documentation, https://docs.jsonata.org/overview, 2021. URL: https://docs.jsonata.org/overview, version 2.2.0. Accessed: 2026-06-16.

[17] Stack Overflow, 2025 developer survey: Technology — most popular technologies, 2025. URL: https://survey.stackoverflow.co/2025/technology#most-popular-technologies-language-language, accessed: 2026-06-16.

A. Description of Custom Profiles

Table 1
SSSOM⁺ fields used in our pipeline.

Field	Description
`sssom:mapping_set_id`, `sssom:mapping_set_title`, `sssom:mapping_date`	Identify and version the exported mapping set.
`owl:annotatedSource`, `owl:annotatedTarget`	Point to the source and target schema elements that participate in the mapping.
`sssom:subject_label`, `sssom:object_label`	Preserve human-readable paths for display and prompting.
`owl:annotatedProperty`	Records the mapping predicate, e.g. `skos:exactMatch`, `skos:closeMatch`, or `skos:relatedMatch`.
`sssom:mapping_justification`	States the provenance of the relation, in our examples manual curation.
`owl:versionInfo`	Captures mapping status, e.g. draft or final.
`rdfs:comment`	Stores the human-readable transformation intent used by the translation component, such as lookup behavior, sequence selection, or value derivation.

Table 2
JSONata⁺ service request and mapping profile fields.

Field	Description
Service request	
`messages`	One or more source messages to be translated.
`mapping`	The JSONata⁺ mapping configuration to apply.
`options`	Runtime options such as output format (JSON, YAML, XML, CSV) and error behavior.
Mapping configuration	
`version`	Declares the mapping schema version.
`tables`	Embeds named lookup tables accessible through `$lookup()`.
`onMissing`	Missing source field: `"error"` (default) triggers `onError`; `"ignore"` omits field; `"null"` writes null. Per-field overridable.
`onError`	Failed expression: default drops message as errored. `action: "skip"` omits the field; `action: "fallback"` writes a configured value. Per-field overridable.
`functions`	Inline JS arrow functions callable as `$name()` in any expression; also loadable globally at startup.
`fields[*].target`	Target path to write in the output message, using dot notation for nested objects.
`fields[*].source`	Copies a value directly from the source path.
`fields[*].expression`	JSONata expression for derived values, reshaping, or conversions.
`fields[*].constant`	Injects a fixed target value.
`preTransform.filters`	JSONata filter expressions applied to input messages before field mapping; excluded messages are recorded with a reason.
`preTransform.logic`	JSONata expression that reshapes the input before field mapping.
`postTransform.filters`	JSONata filter expressions applied to translated messages before output.
`postTransform.logic`	JSONata expression applied to the assembled target object after field mapping.