

# A Vision on Algebraic Flows for Declarative Resource Descriptions

Jitse De Smet<sup>1</sup>, Ruben Verborgh<sup>1</sup> and Ruben Taelman<sup>1</sup>

<sup>1</sup>IDLab, Department of Electronics and Information Systems, Ghent University – imec.

## Abstract

In the age of agentic systems, correct autonomous data management is increasingly critical. Agents need to discover what resources exist and how to interact with them, but existing interface description frameworks that could provide such support either leave too much open to interpretation or fail to model underlying resources, which are not directly exposed. To address this gap, we propose a declarative description that describes underlying resources and their transformations by modeling interfaces as algebraic flows that declaratively define how underlying resources are exposed, derived, and manipulated. We detail our vision by walking through an example use case that showcases the various problems agents have when interacting with data interfaces and incrementally present our solution of modeling how data flows within the system. Our approach shows that interface developers can express their systems using familiar query languages, while enabling autonomous agents to reason about the consequences of actions and determine required operations to manipulate resources safely and predictably. This makes deterministic interactions with complex, asymmetric information interfaces possible. Looking ahead, we aim to formalize this approach into a theoretically sound framework with a proof-of-concept implementation. Additionally, we identify promising avenues for integrating policy-aware semantics and privacy-preserving interface disclosure.<sup>2</sup>

## Keywords

Autonomous Data Management, Interface Semantics, Interface Description, Asymmetric Interfaces, Modeling Underlying Resources

## 1. Introduction

Agentic systems whether powered by stochastic language models or deterministic query engines – require a precise understanding of both the semantics and the consequences of the actions they perform. When these systems interact with decentralized data sources, they need to know: 1. What does it *mean* to write data here? 2. What *state changes* will this action trigger? 3. How do the endpoints of an interface reflect those changes after completion? In practice, today’s interface descriptions rarely answer these questions fully. Descriptions are often partial, human-readable, and inconsistently maintained. As a result, autonomous agents operate with brittle assumptions and incomplete models of the systems they interact with.

One key insight from previous research is that symmetric interfaces (e.g. LDP [1]) – where the manipulation of one exposed resource does not have effects on the manipulation of another, and where CRUD operations happen on the same exposed resource, meaning updates are not performed on some underlying resource (e.g. a database) have difficulties supporting complex access policies [2]. With that regard, asymmetric interfaces, that is interfaces which depend on underlying resources, where the manipulation of one resource can have consequences on the manipulation of another, and data manipulation methods can be used to enact consistency boundaries, are a more promising fit. Unfortunately, the inclusion of underlying resources introduces a new challenge, as write semantics become opaque, agents no longer know what resources are manipulated and how - when performing a single write. This semantic gap prevents agents from reliably reasoning over their own actions. At the same time, the same underlying resources can be exposed in different ways, such as through different fragmentations or interaction methods, each having their own benefits and drawbacks [3,4]. Moreover, in dynamic environments these choices of fragmentation and interaction methods can change [5], mirroring the elasticity of cloud infrastructure. In this paper we use the term *resources* to mean both *exposed resources* and *underlying resources*.

To manage these complexities, we propose a framework that algebraically **describes** (e.g. relational or SPARQL algebra) **the relationship** between write operations, resource manipulations, and the exposed resource representations using shape descriptions (e.g. SPARQL ASK [6], ShEx [7]).

<sup>2</sup> Canonical version: <https://interface-data-description-hyperagents-2025.jitsedesmet.be/>

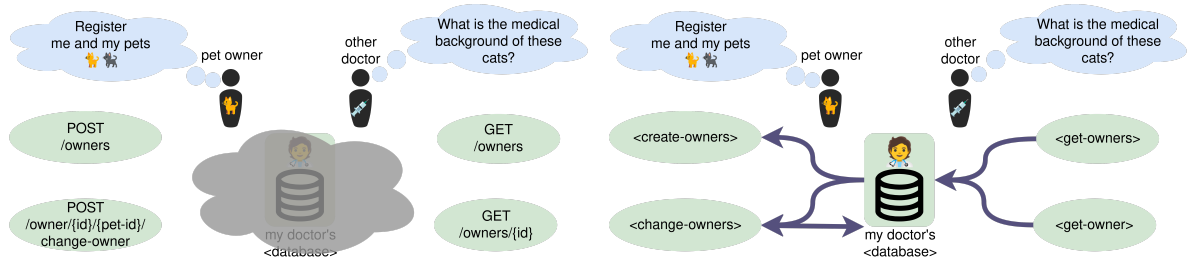
The Second International Workshop on Hypermedia Multi-Agent Systems (HyperAgents 2025), in conjunction with the 28th European Conference on Artificial Intelligence (ECAI 2025), October 26, 2025, Bologna, Italy

✉ jitse.desmet@ugent.be (J. De Smet); ruben.verborgh@ugent.be (R. Verborgh); ruben.taelman@ugent.be (R. Taelman)

🆔 0009-0002-6513-5013 (J. De Smet); 0000-0002-8596-222X (R. Verborgh); 0000-0001-5118-256X (R. Taelman)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



**Fig. 1: The two figures provide a visual representation of the interface for our use case. Left is a classical interface description, describing the exposed resources as endpoints but not describing how data flows between them, while right shows a description that also describes the flow of data between the exposed resources, through underlying resources. The interface exposes a pet doctors database containing owners with their pets. Resources are modeled in green and are identified with an IRI. Ovals are exposed resources and the arrows depict explicitly modeled dependencies between resources. This description allows the agent of our pet owner Bob to register himself and his pets to the system, and it also allows the agent of another doctor to get the medical history of Bob’s cats when he has an emergency doctor visit.**

Throughout this paper we will use Fig. 1 as an example use case to motivate our approach. Our use case is an interface around a pet doctors database registering pet-owners and their pets. We include four endpoint families: 1. **<post-owner>**: registers a new owner and their pets in the system. 2. **<change-owner>**: transfers a pet from one registered owner to another. 3. **<get-owners>**: retrieves a list of URIs representing all registered owners. 4. **<get-owner>**: fetches an individual owner along with their associated pets + pet data. Important to note is that our system *explicitly* models the underlying <database> resource, meaning other systems can now reference the underlying resource ‘all owners and their pets registered in the system’, and deduce what endpoints manipulate and expose it.

This paper presents a vision for a future in which interfaces formally describe their underlying resources and how to manipulate them, creating rich, composable, and formal interface semantics. Our goal is to stimulate discussion on how such descriptions can support autonomous operation, improve agent reliability, and pave the way for more interoperable and adaptive knowledge infrastructures. While the proposed direction remains conceptual, we offer initial ideas and mappings that illustrate how this approach could be realized in practice. In the next section we review related work, discussing various existing ways to describe data interfaces. In Section 3 we detail our vision, and we conclude our paper in Section 4.

## 2. Related Work

Existing web interface description frameworks fall into several distinct families, each optimized for different usage scenarios. However, most fall short in their ability to formally and completely describe how data flows from write operations to corresponding reads or how internal state transitions occur within the system, across endpoints.

**RPC and Syntactic Interface Specifications** - Remote Procedure Call (RPC) style specification languages such as WADL [8] and OpenAPI/Swagger [9] primarily offer syntactic descriptions of web APIs. These specifications define endpoints, parameters, and payload schemas in formats like XML (WADL) or JSON/YAML (OpenAPI). Although they support URI templates [10] and detail input-output structures, they do not capture semantic relationships between operations, nor do they support hypermedia or model the consequences of state modifications. OpenAPI is widely used for generating documentation and client code, but it only supports syntactic interoperability, requiring human developers or intelligent stochastic agents to infer the meaning of operations.

Similarly, AsyncAPI [11] extends the RPC-style description model to event-based APIs, offering a syntax for defining message-driven architectures. It describes publish/subscribe channels, message schemas, and communication protocols (e.g., MQTT, WebSocket), which makes it well-suited for heterogeneous environments. However, like OpenAPI, AsyncAPI lacks the ability to model how emitted messages impact application state. Messages are specified structurally, but not semantically; the relationship between an event and the resulting system state is left implicit.

Our solution will convey how the underlying resources are modified by describing the interaction between write operations, exposed resources and non-exposed resources (e.g. an underlying database).

**Tool Invocation and Agent-Oriented APIs** - The Model Context Protocol (MCP) [12] represents a newer class of interface which is self-descriptive and aimed at AI agent interoperability. Since MCP allows external tools to advertise available resources (readable data) and tools (invokable operations) to large language models (LLMs), enabling structured interaction via natural language, MCP could be used as a data interface. While this model does conceptually align reads and writes with corresponding resources and tools, it lacks formal semantics. Descriptions in MCP are purely textual, and while this enables flexible reasoning by LLMs, it does not guarantee unambiguous interpretations of system behavior or consequences of actions. As a result, agents cannot reliably infer the downstream effects of invoking a particular tool. The solution we present tackles the reliability problem by algebraically describing the relationship.

**Semantic Web Descriptions** - In the context of the Semantic Web, interface description languages aim to embed operational semantics into Linked Data. The Hydra Core Vocabulary [13] allows API providers to annotate resources with affordances - like descriptions of possible state transitions. This enables clients to discover valid actions dynamically through hypermedia controls. While Hydra provides an elegant model for describing RESTful interactions, it fundamentally assumes symmetry between inputs and outputs. It does not support asymmetric interfaces, for example for changing the owner of a pet in Fig. 1. Therefore, while Hydra enables simple autonomous interactions (e.g., resource creation or deletion), it cannot express the full semantics of data flows [14].

OWL-S [15] provides a semantically rich framework for describing Web services, including preconditions, effects, and complex process models. It supports abstract reasoning about resource modifications, making it theoretically capable of modeling service composition.

RESTdesc [16] extends ideas similar to Hydra using Notation3 (N3) [17], enabling interface contracts through logical implications. Although more expressive than Hydra, RESTdesc similarly lacks a formal model of the data lifecycle; it does not provide mechanisms for tracing how data flows through write-to-read transitions across endpoints.

To our knowledge all semantic web descriptions thus lack direct mechanisms for describing concrete modifications to underlying resources (e.g., <database> in Fig. 1). As such, they are not well-suited to applications requiring fine-grained reasoning about how specific API calls modify underlying resources.

**Query-Based Interfaces** - Interfaces like SPARQL Protocol [18] and GraphQL [19] expose an underlying dataset as an HTTP-based query endpoint allowing clients to select the specific data they require at run-time. GraphQL will allow you to describe different available resources and manipulations, but similar to syntactic interface descriptions, the description of the resources is done through natural language leaving room for interpretation error. The SPARQL protocol on the other hand, is, in the absence of further policies, inherently symmetrical, meaning the shape of the data one can read matches that which can be written. This limits their use when determinism is needed, or when multiple or underlying resources are required, additionally their interaction method comes at significant costs for the data provider [20]. Our solution enables (and encourages) the semantic definition of multiple resources and additionally allows you to use multiple interaction methods.

### 3. Algebraic Resource Descriptions

In this section we will explain our vision by walking through the pet doctor example (Fig. 1). Within our example definitions we use SPARQL algebra [6] over RDF [21] - due to RDF's strengths in representing linked, web-native data using IRIs - the broader approach is algebra-agnostic. Any algebra that can express data transformations in a way that preserves semantic meaning and exposes IRIs as first-class citizens could be used (e.g. RML algebra [22] seems particularly useful).

We also emphasize that the description is there to help clients and should only be correct in regard to the behaviour of the described system, it is thus decoupled from the underlying implementation. For example, an RDF-based description may describe a JSON API backed by a relational database. Despite this abstraction, much like a blueprint, the algebraic description is sufficient to generate a server implementation, illustrating the expressive power of the model.

**Pet registration** - We start by focussing on Bob, the proud owner of two cats. To protect his cats, he requests his agent to register them to the local pet doctor. Since Bob just moved to a new city, the interface of the local pet doctor's system is not yet known to Bob's agent. To make Bob's agent discover the capabilities of the local pet doctor's system, the agent first **request a description of the interface** of the local doctor. Within this description the agent finds how the <post-owners> endpoint manipulates the doctors' database. The agent discovers the **interaction method** (e.g. a POST to /owners), the **resource representation** (e.g. Turtle [23]), and the **expected data shape** (e.g. a SPARQL ASK [6] query see Fig. 2). The interaction method and resource representation allow for out of band communication, since the agent either supports them or does not - meaning the agent knows

their own capabilities. The shape description allows the agent to infer what information it should provide to the endpoint to facilitate a successful interaction. The description also provides the agent with knowledge on the **consequences of a successful interaction** by providing an **algebraic relationship between the endpoint** and, among others, the database resource (e.g. using SPARQL Update, see Fig. 3). By utilizing this interface description, the agent can 1. infer that it should POST Bob's data and that of the cats, according to the shape to /owners in Turtle format. 2. verify the consequences are what is expected and that Bob's cat won't suddenly appear up for sale.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ex: <http://example.org/>
ASK {
  ?owner a ex:owner ;
    ex:name ?ownerName ;
    ex:pet ?pet ;
  FILTER(datatype(?ownerName) = xsd:string) .
  ?pet a ex:pet ;
    ex:name ?petName ;
    ex:age ?petAge .
  FILTER(datatype(?petName) = xsd:string &&
    datatype(?petAge) = xsd:positiveInteger) .
}
```

**Fig. 2: ShEx shape of a single owner.**  
**<database> contains many owners.**

```
INSERT {
  GRAPH <database> {
    ?ownerBound a ex:owner ;
      ex:name ?name ; ex:pet ?petBound .
    ?petBound a ex:pet ;
      ex:name ?petName ; ex:age ?age .
  }
} WHERE {
  ?resource ex:name ?name ;
  BIND(UUID() AS ?ownerUuid) .
  ?resource ex:pet ?pet .
  BIND(UUID() AS ?petUuid) .
  ?pet ex:name ?petName ;
    ex:age ?age ;
  BIND(URI(CONCAT(ex:owners, '/', ?ownerUuid))
    AS ?ownerBound) .
  BIND(URI(CONCAT(?ownerBound, '/', ?petUuid))
    AS ?petBound) .
}
```

**Fig. 3: Declarative description to the agent of how**  
**<post-owners> manipulates the <database>.**

**Requesting pet information** - Next in our use case, disaster strikes and one of the cats get sick, to make matters worse our doctor is on holiday. Bob visits another doctor, and now that doctor requires the medical history of Bob's cat stored on the system of our doctor. The other doctor thus tasks their agent to retrieve the files of Bob's sick cat. Since this agent needs to read data, it would technically be feasible to query all data of our doctor's system blindly, and filter on what applies to Bobs cat. This approach is currently often taken but comes with significant client and server overhead - our agent is smart and requests the interface description of our doctor's system. From this description it notices the existence of the database, containing owners and pets. Since the database resource is not directly exposed, it looks at other, derived resources, finding 1. <get-owners>: list of IRIs of the owner in the system, and 2. <get-owner>: parameterized resources containing the info of owner X and their pets. Each of those resources algebraically describes how they are derived from the database (e.g. using SPARQL CONSTRUCT [6] queries, see Fig. 4 and Fig. 5). When describing parameterized resources, one could use a tentative variable in the query that will be bound at a later point. Since the resources are exposed, the description contains just like before the **interaction method** and **representation**. Using the interaction methods you can link the resource parameters to parts of the interaction method, as such using the URI template [10] /owners/{id} we can bind the 'id' variable to a variable in the query. Next, we hypothesize that the agent can use the algebraic descriptions and the input shapes from the manipulation endpoints to **compute the shape of the data they can expect** when calling the read-endpoints. Using the expected shapes and parameter definitions, the agent will infer that it can request both Bob's data and that of his cats when Bob provides his identifier.

```

CONSTRUCT { [] ex:item ?owner . } WHERE {
  GRAPH <database> {
    ?owner a ex:owner .
  }
}

```

**Fig. 4: Declarative description to the agent of how <get-owners> is derived from the <database>.**

```

CONSTRUCT {
  ?id a ex:owner ;
  ?ownerP ?ownerO ;
  ?ownerO ?petP ?petO ;
} WHERE {
  GRAPH <database> {
    ?id a ex:owner ;
    ?ownerP ?ownerO ;
    OPTIONAL { ?ownerO ?petP ?petO . }
  }
}

```

**Fig. 5: Declarative description to the agent of how <get-owner> is derived from the <database> using parameter ‘id’.**

**Pet transfer** - Now to end our story in a positive note, Bob decides to buy a third cat. He buys the cat from Alice who is already registered at the same doctor, and asks his agent to register this change of ownership. Once again, the agent requests the description, this time discovering the <change-owners> endpoint. Like before, the endpoint has a required shape description (Fig. 8), a resource representation, an interaction method, and a query that updates the database (Fig. 9). Similar to the parameterized resource before, the update query of <change-owner> contains variables that will be bound at a later stage, the variable in question being ‘id’ and ‘pet-id’ from the URI template /owner/{id}/{pet-id}/change-owner. It is essential to note that the explicit modeling of the <database> plays an essential role in our ability to describe this operation. The change ownership modification shows the definition of a **consistency boundary** that would not be possible to model using symmetric interfaces.

**Need for algebraic descriptions** - To conclude we hypothesize that our algebraic descriptions enable agents to: 1. **Discover endpoints** providing views over given resources, and how to query them, 2. **infer how modifications can be enacted to those resources**, by tracing flows back to the corresponding write endpoints, and 3. **how these endpoints can be used** through interaction methods and required shapes. In this way, our model enables agents to reason not just about what data resources exist, but how they came to be, how they can be modified, and what such modifications would entail.

## 4. Conclusion

We propose a new approach to data interface descriptions by formally linking reads and writes through algebraic flows and explicitly modeling underlying resources. As autonomous agents become more prevalent, machine-understandable interface semantics are essential, especially for heterogeneous and asymmetric systems.

By introducing algebraic data flows, we propose a foundation for explicitly modeling read and write operations over both exposed and underlying resources. This shift enables agents to reason not only about what data is available, but also how it was derived and how it can be meaningfully modified. This approach builds on skills developers already use to write queries, requiring only a shift in mindset: modeling underlying resources explicitly and exposing interface semantics.

Future work includes formalizing the system with a defined vocabulary and algorithms, exploring policy-aware semantics for fine-grained control, and developing privacy-preserving flows where interface descriptions can be selectively disclosed. The authors next step will be to tackle the first point and create a proof-of-concept implementation.

## Acknowledgements

Jitse De Smet is a predoctoral fellow of the Research Foundation – Flanders (FWO) (1SB8525N). The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10) Ruben Taelman is a postdoctoral fellow of the Research Foundation – Flanders (FWO) (1202124N).

```
PREFIX ex: <http://example.org/>
ASK {
  ?owner ex:new-owner ?someIri .
  FILTER( isIri(?someIri) ) .
}
```

**Fig. 8: Declarative description of what data shape the agent should provide when communication with <change-owners>.**

```
DELETE {
  GRAPH <database> {
    ?origOwner ex:pet ?origPet .
    ?origPet a ex:pet ;
      ex:age ?age ;
      ex:name ?name .
  }
} INSERT {
  GRAPH <database> {
    ?newOwner ex:pet ?movedIri .
    ?movedIri a ex:pet ;
      ex:age ?age ;
      ex:name ?name .
  }
} WHERE {
  # From received data:
  ?o ex:new-owner ?newOwner .
  BIND(URI( CONCAT(ex:owners, '/', ?id) )
    AS ?origOwner) .
  BIND(URI( CONCAT(?origOwner, '/', ?petId) )
    AS ?origPet) .
  BIND(URI( CONCAT(?newOwner, '/', ?petId) )
    AS ?movedIri) .
  # Go look in the current state of the database
  # for whether the owner and pet exist.
  GRAPH <database> {
    ?origOwner ex:pet ?origPet .
    ?origPet a ex:pet ;
      ex:age ?age ;
      ex:name ?name .
    ?newOwner a ex:owner ;
  }
}
```

**Fig. 9: Declarative description to the agent of how <change-owners> manipulates the <database>.**

## Declaration On Generative AI

During the preparation of this work, the author(s) used Chat-GPT-4 in order to: Paraphrase and reword, improve writing style, perform grammar and spelling checks, and critically review the text written by the authors to highlight essential information. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

## References

- [1] Speicher, S., Arwe, J., Malhotra, A.: Linked Data Platform 1.0. W3C, <https://www.w3.org/TR/2015/REC-ldp-20150226/> (2015).
- [2] Dedeker, R., Slabbinck, W., Wright, J., Hochstenbach, P., Colpaert, P., Verborgh, R.: What's in a Pod? A Knowledge Graph Interpretation For The Solid Ecosystem. In: Saleem, M. and Ngomo, A.-C.N. (eds.) Proceedings of the QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs co-located with 21st International Semantic Web Conference (ISWC 2022), Hangzhou, China, 23-27 October 2022. pp. 81–96. CEUR-WS.org (2022).
- [3] Verborgh, R., Sande, M.V., Hartig, O., Herwegen, J.V., Vocht, L.D., Meester, B.D., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. J. Web Semant. 37-38, 184–206 (2016). doi:10.1016/J.WEBSEM.2016.03.003
- [4] Hartig, O., Letter, I., Pérez, J.: A Formal Framework for Comparing Linked Data Fragments. In: d'Amato, C., Fernández, M., Tamma, V., Lécué, F., Cudré-Mauroux, P., Sequeda, J.F., Lange, C., and Heflin, J. (eds.) The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I. pp. 364–382. Springer (2017). doi:10.1007/978-3-319-68288-4\_22

- [5] Azzam, A., Taelman, R., Polleres, A.: Towards Cost-Model-Based Query Execution over Hybrid Linked Data Fragments Interfaces. In: Harth, A., Presutti, V., Troncy, R., Acosta, M., Polleres, A., Fernández, J.D., Parreira, J.X., Hartig, O., Hose, K., and Cochez, M. (eds.) *The Semantic Web: ESWC 2020 Satellite Events - ESWC 2020 Satellite Events*, Heraklion, Crete, Greece, May 31 - June 4, 2020, Revised Selected Papers. pp. 9–12. Springer (2020). doi:10.1007/978-3-030-62327-2\_2
- [6] Harris, S., Seaborne, A., Prud’hommeaux, E.: SPARQL 1.1 Query Language. W3C, <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/> (2013).
- [7] Baker, T., Prud’hommeaux, E.: Shape Expressions (ShEx) 2.1 Primer. <https://shex.io/shex-primer-20191009/> (2019).
- [8] Hadley, M.: Web Application Description Language. W3C, <https://www.w3.org/submissions/wadl/> (2009).
- [9] OpenAPI Specification. <https://swagger.io/specification/>
- [10] Gregorio, J., Fielding, R.T., Hadley, M., Nottingham, M., Orchard, D.: URI Template. RFC. 6570, 1–34 (2012). doi:10.17487/RFC6570
- [11] AsyncAPI. <https://www.asyncapi.com/en>
- [12] MCP: Model Context Protocol. <https://modelcontextprotocol.io/>
- [13] Lanthaler, M., Guetl, C.: Hydra: A Vocabulary for Hypermedia-Driven Web APIs. *Proceedings of the WWW2013 Workshop on Linked Data on the Web*, Rio de Janeiro, Brazil, 14 May, 2013. 996, (2013).
- [14] Taelman, R., Verborgh, R.: Declaratively Describing Responses of Hypermedia-Driven Web APIs. In: Corcho, Óscar, Janowicz, K., Rizzo, G., Tidli, I., and Garijo, D. (eds.) *Proceedings of the Knowledge Capture Conference, K-CAP 2017*, Austin, TX, USA, December 4–6, 2017. pp. 34:1–34:4. ACM (2017). doi:10.1145/3148011.3154467
- [15] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., others: OWL-S: Semantic markup for web services. W3C member submission. 22, (2004).
- [16] Verborgh, R., Steiner, T., Deursen, D.V., Coppens, S., Vallés, J.G., de Walle, R.V.: Functional descriptions as the bridge between hypermedia APIs and the Semantic Web. In: Alarcón, R., Pautasso, C., and Wilde, E. (eds.) *Third International Workshop on RESTful Design, WS-REST ’12*, Lyon, France, April 16, 2012. pp. 33–40. ACM (2012). doi:10.1145/2307819.2307828
- [17] Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. <https://www.w3.org/TeamSubmission/n3/>
- [18] Feigenbaum, L., Todd Williams, G., Grant Clark, K., Torres, E.: SPARQL 1.1 Protocol. W3C, <https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/> (2013).
- [19] GraphQL. <https://graphql.org/>
- [20] Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.-Y.: SPARQL Web-Querying Infrastructure: Ready for Action? In: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J.X., Aroyo, L., Noy, N.F., Welty, C., and Janowicz, K. (eds.) *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference*, Sydney, NSW, Australia, October 21–25, 2013, *Proceedings, Part II*. pp. 277–293. Springer (2013). doi:10.1007/978-3-642-41338-4\_18
- [21] Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and Abstract Syntax. W3C, <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/> (2014).
- [22] Oo, S.M., Hartig, O.: An Algebraic Foundation for Knowledge Graph Construction (Extended Version). CoRR. abs/2503.10385, (2025). doi:10.48550/ARXIV.2503.10385
- [23] Carothers, G., Prud’hommeaux, E.: RDF 1.1 Turtle. W3C (2014).