Department of Computing and Software

Faculty of Engineering — McMaster University

Domain Information System (DIS): Specification and Automation

by

Alicia Marinache, Ridha Khedri, Wendy MacCaull

CAS Report Series **CAS-25-01-RK**

Department of Computing and Software **August 2025**
Information Technology Building
McMaster University
1280 Main Street West Hamilton, Ontario, Canada L8S 4K1

Domain Information System (DIS): Specification and Automation

Alicia Marinache[1], Ridha Khedri[1], and Wendy MacCaull[2]

[1] Department of Computing and Software, Faculty of Engineering,
McMaster University, Hamilton, Ontario, Canada

[2] Department of Mathematics, Statistics and Computer Science, Faculty of Science,
St. Francis Xavier University, Antigonish, Nova Scotia, Canada

Technical Report CAS-25-01-RK
Department of Computing and Software
McMaster University

August 18, 2025

Abstract

This report presents the formal specification and partial automation of the Domain Information System (DIS), a modular framework for building knowledge representations from structured datasets. DIS separates the data layer and the conceptual layer into two rigorously defined theories: the Domain Data View (DDV), modelled as a cylindric algebra, and the Domain Ontology (DOnt), defined through a concept monoid, Boolean lattice, and a family of rooted graphs. A mapping operator semantically aligns data instances with domain concepts. The framework is implemented in Isabelle/HOL, where each component is specified using locales that inherit from foundational mathematical structures. To support scalable engineering, we introduce a template-driven mechanism for generating DIS instances directly from datasets. A concrete instantiation, the Wine DIS, illustrates the feasibility of this approach. The result is a semi-automated, formally grounded methodology for developing evolvable, verifiable ontologies aligned with structured data.

Keywords: ontology engineering, knowledge representation, ontology specification, data theory, template-driven automation, hol, interactive theorem prover, isabelle

Contents

1 Introduction **1**

2 Related Work **2**

3 Isabelle Overview **3**
3.1 Proof Assistants and Higher-Order Logic 3
3.2 Core Concepts: Theories, Locales, and Types 3
3.3 Syntax and Proof Language 4
3.4 Benefits of Isabelle 6

4 Formalising DIS Semantics **9**

5 Template-Based Automation **11**
5.1 Template Structure 11
5.2 Automation Pipeline 12

6 Case Study: Wine DIS Specification **13**

7 Discussion **16**
7.1 Practical Implications of a DIS 16
7.2 Support for Ontology Engineering and Evolution 16
7.3 Expressivity vs. Decidability: Tradeo-ffs in Higher-Order Logic 17
7.4 Verification in Data-Aware Conceptual Modelling 17

8 Conclusion and Future Work **17**

A Appendix: DIS Templates BNF **18**

Glossary **20**

1 Introduction

Modern knowledge systems are increasingly expected to integrate complex domain understanding with structured data representations. However, many existing frameworks face challenges when aligning conceptual models with the data they are meant to represent [KVS20]. Ontologies are often developed independently of their associated datasets, and the connections between domain knowledge and data are introduced manually through mapping layers or external configurations. These subsequent mappings can lead to fragile systems where semantic misalignment is common, making updates, evolution, and reuse difficult to manage [DGLL$^+$18].

To address these challenges, we build on the DIS, a logic-based formalism that integrates structured datasets and domain conceptual knowledge [Mar16]. In DIS, a domain application is represented using two core theories: the DDV, which captures the structure and content of organised datasets, and the DOnt, which defines a conceptual abstraction over this data. These two layers are linked via a formally defined mapping operator, which ensures that the resulting ontology is grounded in the semantics of the dataset and remains stable under evolution. In doing so, DIS enables a form of knowledge engineering that is both syntactically constrained and semantically transparent.

In this work, we focus on the formal specification and automation aspects of the DIS. Our first contribution is the formalisation of the DIS syntax and semantics [MKM25] in Isabelle/HOL, a higher-order logic interactive proof assistant [Isa25]. By using Isabelle/HOL, we define a unified representation of both the data and domain knowledge layers of DIS, while enabling formal verification of key structural properties. This encoding provides a solid foundation for building consistent, extensible, and verifiable ontology specifications.

Building on this specification, our second contribution is the introduction of a template-based mechanism for automating DIS construction. We define a set of reusable templates that capture common DIS patterns, which can be instantiated using structured datasets. A parser tool processes these template specifications and automatically generates the corresponding Isabelle/HOL theories. This approach significantly reduces the effort required to create domain-specific ontologies while maintaining formal correctness.

Finally, to demonstrate the practical viability of the DIS framework and its automation pipeline, we present a detailed case study based on the Wine Ontology domain. This case study illustrates the entire process, from dataset analysis and template instantiation to the generation of an Isabelle specification, showing how domain-specific knowledge systems can be built in a reproducible and formally grounded manner.

Through these contributions, this report positions DIS not only as a theoretically sound framework for data-grounded ontology design, but also as a practical and automatable engineering process for generating modular and formally specified knowledge systems.

The rest of the paper is structured as follows. In Section 2 we present an overview of the DIS and its two formal theories and contrast it to existing ontology frameworks. In Section 3, we present the Isabelle/HOL robust environment for specifying mathematical structures. In Section 4 we provide an overview of Isabelle/HOL as specification environment and describe the DIS architecture. In Section 5 we introduce the template-based automation process for specifying a DIS, and in Section 6 we provide a case study for generating a DIS starting from an existing dataset. In Section 7 we discuss the practical implications of a formalised DIS specification and the engineering support for maintenance and modularisation. Finally, in Section 8 we present a summary of our work, along with future plans for DIS evolution.

2 Related Work

Ontology engineering has long grappled with the challenge of bridging the gap between data and domain knowledge in a coherent and scalable manner. Several established formalisms and frameworks address this problem from different perspectives, including DL, OBDA, and engineering methodologies, such as DOGMA and SEADOO. This section contrasts the DIS approach with these established lines of work, highlighting its unique contribution to modularity, semantic grounding, and specification automation.

DL [BCM+03] are a family of decidable fragments of FOL used extensively in ontology design and reasoning. DL-based systems, such as OWL and its reasoning tools, support reasoning tasks including concept satisfiability, subsumption, and consistency checking through automated tableau-based procedures.

While DL-based tools offer powerful decidability guarantees and mature tooling, they do not guarantee that an ontology is structurally aligned with the data it is meant to describe. The burden of ensuring semantic relevance between data and ontology lies with the modeller, and mappings between schema and data attributes are often applied afterwards. In contrast, DIS integrates structured data into the ontology specification process itself, making the mapping operator a first-class construct in the formalism. Every DIS instance is by construction a model of the underlying semantic theory, ensuring structural and semantic coherence from the outset.

OBDA frameworks aim to enable logical querying over datasets via an ontology layer, typically by rewriting queries over the ontology into SQL or SPARQL queries over a relational backend. While powerful, OBDA solutions often involve complex translation mechanisms and schema-ontology mappings [CCKE+17], which must be maintained manually or semi-automatically. Moreover, performance concerns arise due to the overhead introduced by these mappings and the reliance on query rewriting[XCK+18]. The DIS formalism addresses these limitations by unifying the data and ontology layers in a single coherent framework. The domain ontology is derived from the data schema through a mapping operator that captures the structural correspondence between sorts and atomic concepts. Consequently, DIS avoids expensive query translations and maintains a minimal coupling between the data and domain representations. Moreover, changes in the dataset structure or content propagate in a localised and predictable manner within a DIS instance [MKLM21].

DOGMA [STM08] distinguishes between the lexon base (data layer) and the application-specific ontologies, aiming to achieve reusability through separation of concerns. Similarly, SEADOO [GCW23] proposes a flexible and extensible architectural approach for ontology engineering, emphasising the automation of documentation and post-development activities. These methodologies share conceptual motivations with DIS, especially in promoting modularity and structured design. However, DIS takes this further by offering a formally defined logic-based framework in which each instance inherits correctness, modularity, and semantics from the underlying theory. Through its use of higher-order logic and proof automation in Isabelle/HOL, DIS provides machine-verifiable guarantees that traditional design methodologies do not.

While traditional DL-based frameworks trade off expressiveness for decidability, DIS is embedded in a HOL setting. This allows the representation of rich domain knowledge and meta-theoretical reasoning that extends beyond that of DL fragments. Although HOL is undecidable in general, modern ITPs such as Isabelle/HOL make practical reasoning feasible through automation tactics, proof libraries, and engineering practices. The use of Isabelle

in DIS enables not only property verification but also the specification and validation of the construction templates and data-ontology mappings themselves.

This emphasis on verifiability by design distinguishes DIS from other knowledge representation frameworks. By tightly integrating data interpretation, concept abstraction, and reasoning infrastructure into a single formal environment, DIS offers a scalable solution for building and maintaining trustworthy knowledge systems.

3 Isabelle Overview

This section provides an overview of the Isabelle proof assistant as used in the specification and verification of DIS. The goal is to familiarise the reader with the features of Isabelle/HOL that support our formalisation efforts, from its foundational principles to its syntax, proof language, and integration capabilities. We also highlight the benefits of using Isabelle for the DIS specification and its instances.

3.1 Proof Assistants and Higher-Order Logic

A proof assistant is a software system that supports the construction and verification of formal proofs by providing a formal language and a checking engine. These tools are often based on expressive logical foundations such as HOL or dependent type theory. Isabelle is a widely-used ITP that supports a declarative proof style and provides a rich logical infrastructure built on HOL [WW07, BASCW18]. Unlike automated theorem provers, which attempt to find proofs entirely on their own, Isabelle relies on user guidance through structured proof development. This makes it particularly suitable for complex, modular, and layered systems such as the DIS.

In contrast to many decidable formalisms such as DL, Isabelle logic is undecidable, in general. However, this flexibility allows it to express a broader range of concepts, and its rich libraries and tactic-based proof system make verification of structured theories practical. Our choice of Isabelle is guided by its robust support for modular reasoning, type-safe theory development, and extensibility, factors that are essential for working with the heterogeneous and algebraic nature of DIS.

Isabelle architecture is multi-layered, with its core logic and infrastructure implemented in Standard ML. At the highest layer sits the PIDE, a robust user interface framework that supports real-time interaction with theory documents. PIDE provides essential editing capabilities such as context-aware auto-completion, inline error reporting, and semantic highlighting, allowing users to develop proofs in an intuitive and responsive environment. The overall architecture of Isabelle, including its logical infrastructure and interface layers, is illustrated on the left side of Figure 1, and the communication flow between the Isabelle core and the PIDE interface is depicted on the right side of the figure [BASCW18]. In Isabelle, users are not restricted to a single level of interaction. The system grants access to all layers of the architecture, enabling both high-level theory development and low-level customisation or extension of the logical environment.

3.2 Core Concepts: Theories, Locales, and Types

In Isabelle, formal development is organised around theories. A theory is a self-contained module that can declare types, constants, axioms, and proofs. Theories can import other

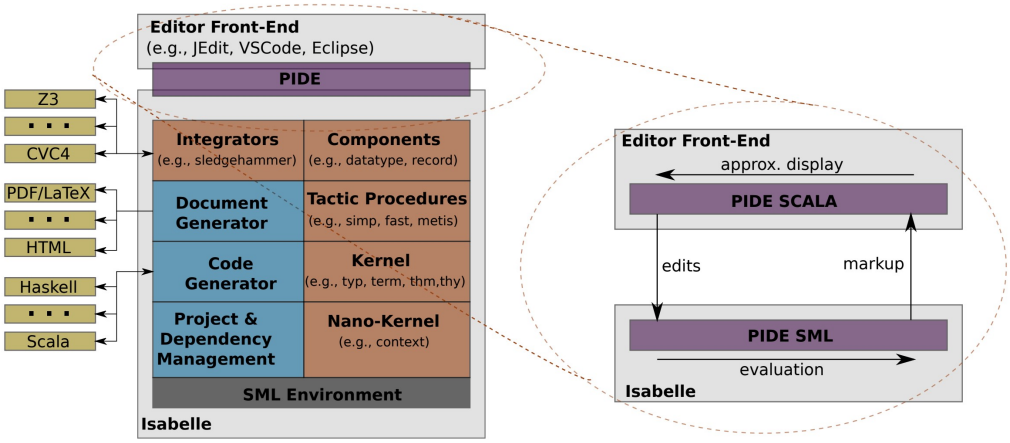

Figure 1: Isabelle system architecture [BASCW18]

theories, forming a graph of dependencies that promotes modular design.

For structuring parameterised reasoning, Isabelle provides locales, which define a logical context (with fixed parameters and assumptions) in which definitions and theorems can be stated and proved. Locales are essential in our work, as they allow the DIS theory to be specified generically and then instantiated for concrete applications such as the Wine DIS case study.

Isabelle distinguishes between types, terms, and propositions. Types are used to constrain the kind of objects (such as sets, elements, functions) under consideration. HOL type system is polymorphic, and supports higher-order functions, enabling abstract and reusable definitions.

3.3 Syntax and Proof Language

Isabelle uses a syntax that is both human-readable and machine-checkable. The formal language allows the definition of constants, inductive types, functions, and lemmas in a concise notation. The document uses specific fonts to distinguish between Isabelle keywords (such as locale, lemma, fixes, assumes, shows) and mathematical content. Table 1 provides a summary of common syntactic rules used in Isabelle/HOL.

Isabelle supports writing proofs in the structured Isar format, which promotes a clear and human-readable style of formal verification. In this declarative framework, each lemma is expressed by explicitly stating its assumptions and desired conclusion. The system then ensures that the conclusion logically follows from the given assumptions via a sequence of formally justified steps.

Table 2 summarises a set of standard proof methods commonly employed in Isabelle. These methods represent high-level strategies for navigating and discharging proof goals. Examples include auto and simp for automation and simplification, rule for applying named theorems, and induct for initiating structural or rule-based induction. Each method encapsulates a combination of lower-level tactics and is designed to streamline proof construction while preserving soundness.

Table 3 presents the foundational rules of natural deduction used in Isabelle for reasoning within higher-order logic. These rules govern the behaviour of standard logical connectives,

Rule	**Term**	**Interpretation**	**NOT**
	$f\ t\ u$	$(f\ t)\ u$	$f\ (t\ u)$
The prefix form of a function binds more strongly than anything	$f\ x + y$	$(f\ x) + y$	$f(x + y)$
Iff is expressed using equality, but equality has a higher priority	$\neg\neg P = P$	$\neg\neg(P = P)$	$(\neg\neg P) = P$
Constructs with opening delimiter and no closing one bind very weakly. E.g., *if*, *let*, *case*, $\lambda, \forall, \exists$.	$\lambda x.\ x = f$	$\lambda x.\ (x = f)$	$(\lambda x.\ x) = f$

Table 1: General syntactic rules of Isabelle

Method	**Description and Usage**
`auto`	Combines classical reasoning with simplification, and intended for proving theorems that have a lot of trivial subgoals. It proves easy subgoals and leaves the ones it cannot prove.
`best`	Legacy Isabelle method, uses best-first search, guided by a heuristic function
`blast`	A generic tableau prover, used for FOL
`fast`	Legacy Isabelle methods, uses depth-first search
`force`	Intended to completely prove the first goal. Performs an exhaustive search, thus proof attempts may take longer or diverge easily.
`rule`	A backward reasoning method that unifies the conclusion of the rule with the current subgoal
`simp`	Simplification, also known as *term rewriting*; invokes the Simplifier on the first subgoal,
`slow`	Similar to `fast`; using a broader search
`bestsimp` `fastforce` `slowsimp`	Similar to `best`, `fast`, `slow`, respectively; using the Simplifier as additional wrapper
`meson`	Implements Loveland's model elimination procedure [Lov16]
`metis`	A resolution prover

Table 2: Automated Proof Tactics in Isabelle

such as conjunction, disjunction, implication, and negation, and include core inference principles like modus ponens and proof by contradiction. The rules are categorised into *introduction* and *elimination* schemes. Introduction rules define how a connective can be established in a proof (e.g., proving $P \wedge Q$ requires independently proving P and Q), whereas elimination rules describe how one may extract or derive further information from an established connective (e.g., from $P \wedge Q$ one may conclude P or Q separately). This structured system ensures that logical derivations remain rigorous and modular, supporting the declarative and human-readable proof style of Isabelle.

Finally, Table 4 collects common proof abbreviations and tactical shorthand frequently seen in Isabelle developments. Familiarity with these abbreviations significantly enhances readability and writing efficiency in larger formalisations. Together, these tables serve as a compact yet comprehensive guide to the core syntactic and semantic elements underpinning the natural deduction and interactive proof mechanisms of Isabelle.

Proof construction is interactive: the user incrementally applies proof methods such as **simp**, **auto**, or **blast** to transform or discharge proof obligations. The Isabelle IDE provides real-time feedback throughout this process, guiding the user through remaining subgoals. Once all goals have been resolved, the system reports "No subgoals!", indicating that the proof has been successfully completed. This structured workflow supports both rigorous formalisation and accessible documentation of the reasoning process.

3.4 Benefits of Isabelle

The decision to formalise the DIS framework within Isabelle/HOL was shaped by several structural and semantic advantages of the tool. The support that Isabelle offers for modular reasoning through theories and locales aligns naturally with the layered architecture of DIS, where reusable algebraic components need to be specified once and instantiated across various domain applications.

The specification style in Isabelle encourages explicit, type-safe definitions, which ensures that errors in the logical construction of DIS instances are caught early in the development process. By leveraging Isabelle built-in tactics and rich theory libraries, the properties of the DIS operators and mappings are automatically verified during the construction of any instance. Although Isabelle operates under higher-order logic and is not decidable in general, its combination of user-guided proofs and tactic automation makes it effective for verifying structured algebraic systems such as DIS.

Isabelle supports the extraction of executable code from verified specifications, which will enable the future usage of DIS instances in functional programming environments. This capability complements the use of template-driven automation for generating DIS instances and reinforces our goal of bridging formal specification with practical deployment.

Finally, Isabelle extensive infrastructure, including the jEdit-based IDE, offers robust interactive proof support, continuous feedback, and cross-referencing tools that significantly improve the user experience during formalisation. For a framework like DIS that relies on composability and semantic precision, these features provide a well-suited environment for both foundational development and application-level instantiation.

Isabelle is supported by a jEdit-based IDE, which includes syntax highlighting, real-time proof checking, tooltips, and hyperlinks for navigation across definitions. The IDE provides integration with ML, allowing users to define custom proof methods and extend Isabelle behaviour. This extensibility is key to supporting the algebraic structures required by DIS, such

Rule	Name	Safe	Isabelle Representation
$\frac{P \quad Q}{P \wedge Q}$	conjI	Y	$[\![?\mathtt{P};\ ?\mathtt{Q}]\!] \Longrightarrow ?\mathtt{P} \wedge ?\mathtt{Q}$
$\frac{P \wedge Q \quad [\![\mathtt{P};\ \mathtt{Q}]\!] \Longrightarrow \mathtt{R}}{R}$	conjE	Y	$[\![?\mathtt{P} \wedge ?\mathtt{Q}; [\![?\mathtt{P}; ?\mathtt{Q}]\!] \Longrightarrow \mathtt{R}]\!] \Longrightarrow ?\mathtt{R}$
$\frac{P}{P \vee Q} \quad \frac{Q}{P \vee Q}$	disjI1/2	N	$[\![?\mathtt{P}]\!] \Longrightarrow ?\mathtt{P} \vee ?\mathtt{Q} \quad [\![?\mathtt{Q}]\!] \Longrightarrow ?\mathtt{P} \vee ?\mathtt{Q}$
$\frac{P \vee Q \quad P \Longrightarrow R \quad Q \Longrightarrow R}{R}$	disjE	Y	$[\![?\mathtt{P} \vee ?\mathtt{Q};\ ?\mathtt{P} \Longrightarrow ?\mathtt{R};\ ?\mathtt{Q} \Longrightarrow ?\mathtt{R}]\!] \Longrightarrow ?\mathtt{R}$
$\frac{P \Longrightarrow Q}{P \longrightarrow Q}$	impI	Y	$[\![?\mathtt{P} \Longrightarrow ?\mathtt{Q}]\!] \Longrightarrow ?\mathtt{P} \longrightarrow ?\mathtt{Q}$
$\frac{P \longrightarrow Q \quad P \quad Q \Longrightarrow R}{R}$	impE	N	$[\![?\mathtt{P} \longrightarrow ?\mathtt{Q};\ ?\mathtt{P};\ ?\mathtt{Q} \Longrightarrow ?\mathtt{R}]\!] \Longrightarrow ?\mathtt{R}$
$\frac{P \Longrightarrow False}{\neg P}$	notI	Y	$[\![?\mathtt{P} \Longrightarrow \mathtt{False}]\!] \Longrightarrow \neg ?\mathtt{P}$
$\frac{\neg P \quad P}{Q}$	notE	N	$[\![\neg ?\mathtt{P};\ ?\mathtt{P}]\!] \Longrightarrow ?\mathtt{Q}$
$\frac{P\ a}{\forall x.\ P\ x}$	allI	Y	$[\![\bigwedge \mathtt{a}.\ ?\mathtt{P}\ \mathtt{a}]\!] \Longrightarrow \forall \mathtt{x}.\ ?\mathtt{P}\ \mathtt{x}$
$\frac{\forall x.\ P\ x \quad P[t/x] \Longrightarrow R}{R}$	allE	N	$[\![\forall \mathtt{x}.\ ?\mathtt{P}\ \mathtt{x};\ ?\mathtt{P}\ ?\mathtt{x} \Longrightarrow ?\mathtt{R}]\!] \Longrightarrow ?\mathtt{R}$
$\frac{P\ a}{\exists x.\ P\ x}$	exI	N	$?P\ ?a \Longrightarrow \exists x.\ ?P\ x$
$\frac{\exists x.\ P\ x \quad \begin{array}{c}[P] \\ \vdots \\ Q\end{array}}{Q}$	exE	Y	$[\![\exists \mathtt{x}.\ ?\mathtt{P}\ \mathtt{x};\ \bigwedge \mathtt{x}.\ ?\mathtt{P}\ \mathtt{x} \Longrightarrow ?\mathtt{Q}]\!] \Longrightarrow ?\mathtt{Q}$
$\frac{P \Longrightarrow Q \quad Q \Longrightarrow P}{P = Q}$	iffI	Y	$[\![?\mathtt{P} \Longrightarrow ?\mathtt{Q};\ ?\mathtt{Q} \Longrightarrow ?\mathtt{P}]\!] \Longrightarrow ?\mathtt{P} = ?\mathtt{Q}$
$\frac{P = Q \quad [\![\mathtt{P} \longrightarrow \mathtt{Q}; \mathtt{Q} \longrightarrow \mathtt{P}]\!] \Longrightarrow \mathtt{R}}{R}$	iffE	Y	$[\![?\mathtt{P} = ?\mathtt{Q}; [\![?\mathtt{P} \longrightarrow ?\mathtt{Q}; ?\mathtt{Q} \longrightarrow ?\mathtt{P}]\!] \Longrightarrow \mathtt{R}]\!] \Longrightarrow \mathtt{R}$
$\frac{P \longrightarrow Q \quad P}{Q}$	mp	Y	$[\![?\mathtt{P} \longrightarrow ?\mathtt{Q};\ ?\mathtt{P}]\!] \Longrightarrow ?\mathtt{Q}$
$\frac{\neg P \Longrightarrow False}{P}$	ccontr	Y	$[\![\neg ?\mathtt{P} \Longrightarrow \mathtt{False}]\!] \Longrightarrow ?\mathtt{P}$

Table 3: Natural deduction rules in Isabelle

Abbreviation	**Stands for**
proof	**proof** (`rule` `elim−rules` `intro−rules`)
proof-	Suppress all rule matching
this	Previous proposition
then	**from this**
thus	**then show** **from this show**
hence	**then have**
with *facts*	**from** *facts* **this**
.	**by assumption**
..	**by** (`rule` `elim−rules` `intro−rules`)
...	Schematic term variable, refers to right hand side of last expression
have $\mathtt{P}_1$ ... **moreover have** $\mathtt{P}_2$ ... ⋮ **moreover have** $\mathtt{P}_n$ ... **ultimately show**...	**have** $\mathtt{p}_1$: $\mathtt{P}_1$ ... **have** $\mathtt{p}_2$: $\mathtt{P}_2$ ... ⋮ **have** $\mathtt{p}_n$: $\mathtt{P}_n$ ... **from** $\mathtt{p}_1$ ... $\mathtt{p}_n$ **show**...
?**thesis**	Current goal (the enclosing **show** or **have** statement)

Table 4: Isar proof abbreviations in Isabelle

as Boolean and cylindric algebras. The IDE also supports session management, allowing for large collections of interrelated theories, such as the core DIS and its instances, to be built incrementally. Errors and warnings are reported inline, making the proof process transparent and accessible.

4 Formalising DIS Semantics

The DIS implementation is structured around Isabelle locales, which serve as parameterised modules representing each layer of the framework. The DDV is formalised as a diagonally free cylindric algebra, where data elements (called s-datas) are sets of sorted tuples (called s-datums), and each tuple is a collection of sort–value pairs (called s-values). The sorts correspond to attributes in the dataset and are interpreted as finite sets of values. At this level, data operators such as reduction, extension, flooring, and raising are defined and verified for their algebraic properties, such as idempotency, commutativity, and structural preservation.

On the ontology side, the DOnt layer is modelled as a Boolean lattice of concepts, augmented with rooted graphs to encode semantic constraints and contextual hierarchies. These are represented as named sets over the same universe of sorts, and operations on them (such as composition or restriction) are also formally defined and verified. The key connection between DDV and DOnt is made through the mapping operator, a mapping from s-data structures to ontological concepts that preserves semantic consistency and enables meaningful inference. The overall design of the DIS theory is presented in Figure 2.

The diagram represents the layered and compositional design of the DIS specification. At its foundation are reusable formal theories that extend core Isabelle theories, depicted at the top of the diagram. These reusable generic theories model sorted data, algebraic operations, and basic data transformations. They are implemented independently of any specific domain, allowing them to be reused across multiple applications. On top of this foundation, the DIS core combines the data-side (Domain Data View) and the domain knowledge side (Domain Ontology), specifying how they relate through a mapping operator. This mapping serves as a bridge between raw structured data and conceptual knowledge, aligning them in a semantically consistent way.

In practice, this architecture enables modular verification, partial automation, and scalability. Each of these components is implemented as an Isabelle theory file and encapsulated in its corresponding locale. This supports independent reasoning and reuse, as well as it allows concrete DIS instances, such as the Wine DIS case study presented in Section 6, to inherit the entire structure and correctness of the generic theory. Through this design, the DIS formalisation guarantees that any instantiation of DIS adheres to its semantic constraints by construction, without requiring post hoc validation or external tooling.

Beyond correctness, this formal infrastructure opens the door to semi-automation, discussed in more detail in Section 5. By designing generic specification templates and defining reusable Isabelle constructs, domain engineers can generate DIS specifications directly from structured datasets, with most of the proof machinery handled automatically. The native support for tactic-based proof automation, locale inheritance, and executable code extraction further facilitates this process, enabling rapid development of domain-specific DIS instances that are correct, traceable, and ready for reasoning. To support this automation, we introduce a template-driven mechanism that generates domain-specific DIS specifications in Isabelle directly from structured datasets.

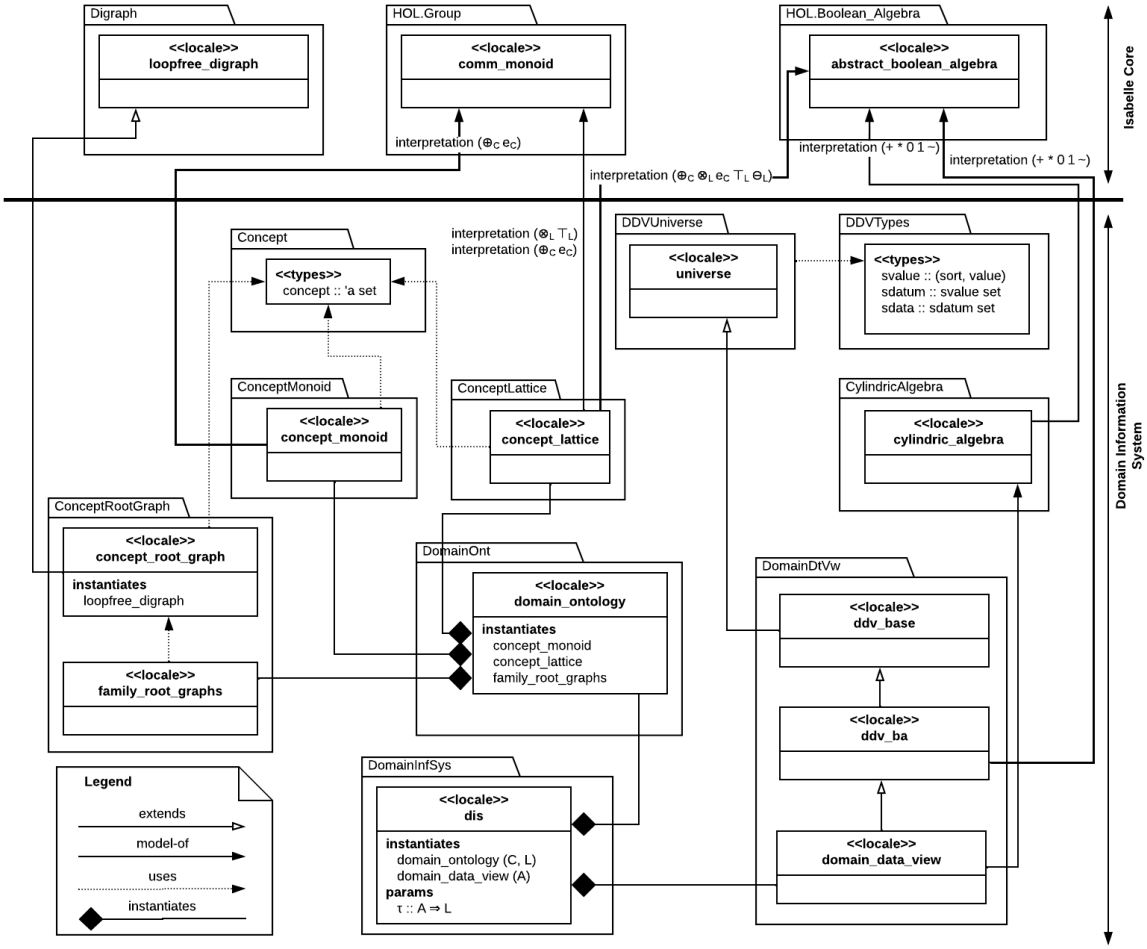

Figure 2: DIS Specification in Isabelle: Design Overview

The complete Isabelle/HOL implementation of the DIS generic theory, including its locale hierarchy, data and ontology specification layers, and mapping formalisation, is available in the accompanying GitHub repository [DIS25]. This serves both as a formal reference and a reusable basis for instantiating domain-specific DIS models.

5 Template-Based Automation

While the formal specification of a DIS in Isabelle ensures correctness and modularity, manually constructing such specifications can be time-consuming and error-prone, especially for users who are not familiar with HOL or proof assistants. To address this limitation, we introduce a template-based automation pipeline that generates complete, structured Isabelle/HOL specifications from domain-specific data and ontology descriptions. This approach enhances scalability, reduces manual workload, and enables domain experts to focus on conceptual modelling without needing expertise in formal verification.

By encapsulating the specification patterns into high-level templates and processing them through a dedicated parser and code generation engine, the system supports a semi-automated ontology engineering pipeline. The generated Isabelle theories remain readable, reusable, and extensible, thus preserving the formal semantics defined by the DIS theory while supporting practical deployment across domains.

5.1 Template Structure

To support automated specification of a DIS, the framework relies on four structured templates that capture all components necessary to instantiate a domain-specific DIS in Isabelle. Each template corresponds to a layer or subcomponent of the DIS architecture and together they offer a clear separation of concerns: from basic universe definition to data abstraction, domain conceptualisation, and full system integration.

The Universe template defines the set of sorts, which are not themselves a formal component of a DIS but serve as foundational vocabulary for both the DDV and the datascape concept of the DIS. Each sort represents an attribute of the dataset and is associated with a finite set of values. This template provides an enumeration of sorts and their contents, from which other components derive their structure. Its separation allows the DDV and DIS to reuse and align with a consistent domain vocabulary.

The Domain Data View template builds on the universe and describes the DDV structure. It specifies set of s-datums and the set of s-datas (i.e., the carrier set of the cylindric algebra), from the universe sorts. This information is used to build the cylindric algebra structure underpinning the data layer.

The Domain Ontology template defines the conceptual structure of the domain. It includes a number of elements, such as the Boolean lattice of atomic and composite concepts, derived from universe sorts, the family of rooted graphs that structure relationships among those concepts, and (optionally) the named concepts, expressed in terms of the Boolean lattice or graph layers. While structurally distinct from the DDV, the DOnt template is built from the same vocabulary, ensuring alignment and interoperability.

The final template represents the complete DIS instance. It integrates the DDV and DOnt specifications and provides the mapping between data sorts and atomic ontology concepts. Beyond simple mapping, this template captures how the DDV and DOnt interoperate as a unified system. It specifies the naming and typing of mapping operator, allows the user to

define datascape concept, and parameterises the final Isabelle instantiation. It is through this template that the entire system becomes executable within the Isabelle framework.

Together, these four templates provide a structured yet flexible way to fully define a DIS instance. Their modularity ensures clarity of specification and facilitates partial reuse and automation, while their integration supports the generation of semantically coherent, verifiable Isabelle/HOL theories. The templates used to instantiate DIS components are defined using an extended BNF grammar, allowing for both structured input and modular parsing logic. The general structure of each template is presented in Figure 3. The corresponding codebase, including template definitions, parsers, and Isabelle generators, is available on GitHub [DIS25], alongside the generic theory and the Wine DIS case study.

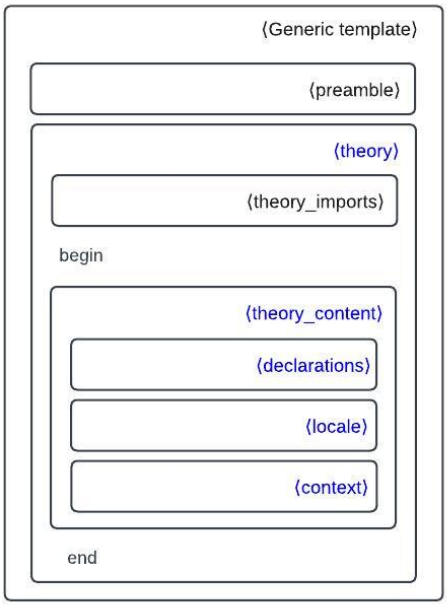

Figure 3: Generic DIS Template

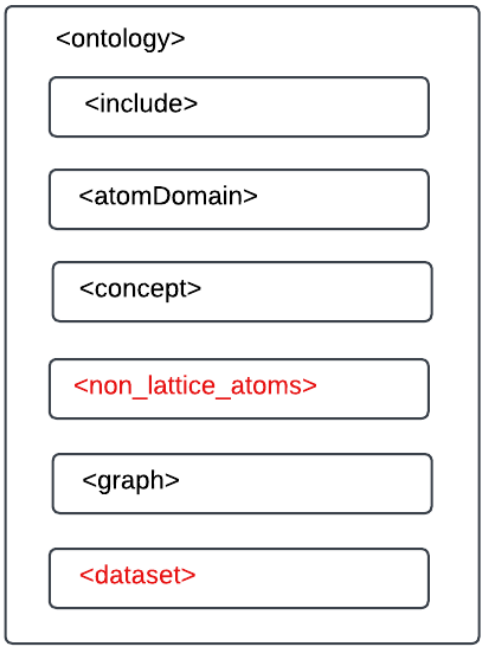

Figure 4: DIS Transformation XML

5.2 Automation Pipeline

To reduce the manual overhead associated with generating DIS specifications in Isabelle/HOL, we provide a fully automated pipeline that transforms structured inputs into four modular theory files. These files correspond to the DIS components described in Section 5.1: the universe, domain data view, domain ontology, and the encompassing DIS specification. The pipeline is designed to be extensible and modular, enabling domain experts to generate verified specifications without direct interaction with Isabelle syntax.

At the heart of the automation is a transformation tool that takes as input a structured XML file, enriched with domain and dataset-specific information. This XML can be initially created using the DISEL editor [WCAK22], which provides an intuitive interface for defining atoms, concepts, and graphs relevant to the ontology. As shown in Figure 4, the XML schema extends DISEL output by introducing additional DIS-specific components. Elements rendered in black originate from the DISEL editor (e.g., $\langle atomDomain \rangle$ and $\langle concept \rangle$), while those in

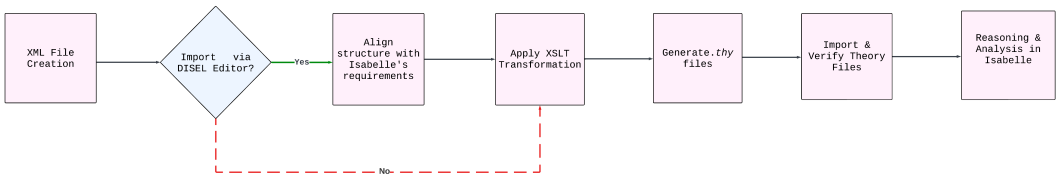

Figure 5: Workflow of the Transformation of a DIS ontology into an Isabelle Theory Using DISEL Tool

blue are currently manually introduced by the user (e.g., $\langle non_lattice_atoms \rangle$ and $\langle dataset \rangle$). Some information (such as the $\langle dataset \rangle$) could, in the future, be lifted automatically from the dataset. Metadata elements, such as the DIS name and target file paths, are captured manually.

Once the XML is prepared, an XSLT transformation is applied to generate the corresponding Isabelle theory files. The XSLT structure mirrors the extended BNF described in Section 5.1, ensuring that the generated theories conform to the syntactic and semantic expectations of Isabelle/HOL. The four theory files produced contain all necessary specifications, declarations, and type constraints, guaranteeing structural validity by design.

Figure 5 illustrates the overall workflow of the automation pipeline, from XML creation to code generation. The process includes creating or editing the XML file using the DISEL editor or a compatible text editor, optionally extending or correcting the XML content to ensure type completeness and dataset coverage, running the XSLT transformation to produce the Isabelle theories, and reviewing or extending the generated Isabelle files (e.g., by adding markup or comments), if needed. The system generates a fully populated Isabelle theory instantiating the DIS for the specific domain. The resulting theory is human-readable and supports further refinement and reasoning within Isabelle.

As shown in Figure 6, the DISEL interface plays a key role in bootstrapping the DIS development process, especially for domain experts unfamiliar with formal specification languages. In conjunction with the automation pipeline, it ensures that ontology construction and data integration within the DIS framework can scale efficiently while remaining formally verifiable. The full automation scripts and a set of template examples are available on the GitHub repository referenced in Section 4. This resource includes the generation tool, extended BNF specifications, and sample theory files for both generic DIS structures and concrete domain instances.

6 Case Study: Wine DIS Specification

Finally, the Wine DIS represents a concrete instantiation of the abstract DIS framework. It imports the generic DIS theories and populates them with domain-specific content, e.g., sorts, values, and concepts extracted from the wine dataset. Through this separation, the system ensures that all domain-specific elements inherit the semantic guarantees and structural rigour defined in the core DIS theory.

In practice, the Wine DIS comprises four Isabelle/HOL theory files, Universe, DDV, DOnt, and DIS, each corresponding to a component of the DIS architecture. The design of the Wine DIS specification is detailed in Figure 7, with the the generic DIS theories shown at the top, and the specific Wine DIS components at the bottom.

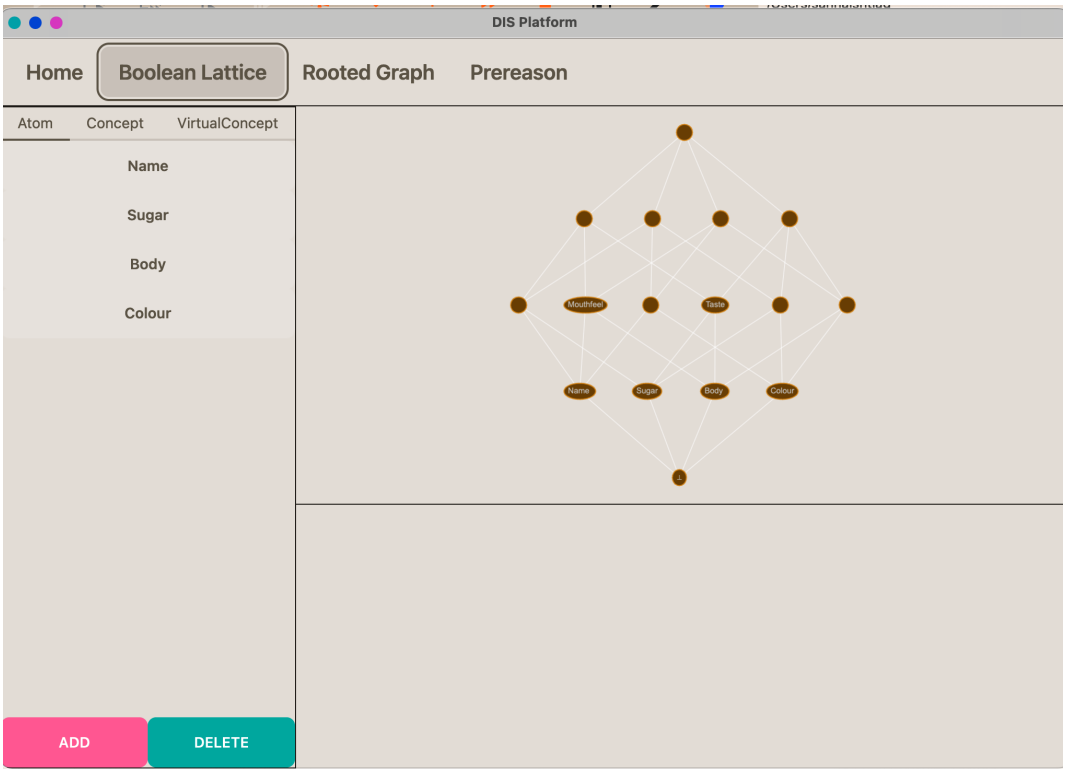

Figure 6: DISEL Editor Screenshot Capturing the Lattice of the Wine Ontology

From the sample datatset provided in Figure 5, the domain expert defines the types (sorts, s-datums, s-datas, and concepts) and specifies the universe $\mathcal{U}$ using dataset attributes such as `name`, `colour`, `sugar`, and `body`. These sorts are mapped to atomic concepts within the Boolean lattice of the domain ontology.

The DDV component, depicted in Figure 8, is derived from the generator set, which is lifted directly from the dataset. The DOnt component includes atomic concepts, named concepts, and a family of rooted graphs encoding domain relations (e.g., `region` is related to `estate`, which in turn is related to `name`). The mapping operator τ, defined via the operators `sort2atom`, establishes the semantic bridge between the DDV and DOnt layers.

Once assembled, the final Wine DIS locale encapsulates all components and supports formally verified reasoning tasks, including classification, subsumption, satisfiability, and inference. Datascape concepts, such as `whiteWine`, which is exemplified in Listing 1, are defined by filtering lattice concepts through data conditions, enabling queries grounded in both semantics and evidence.

```
definition whiteWine :: "sdata set ⇒ dataconcept" where
    "whiteWine X = {a | a. a ∈ X ∧ τW a = wine}"
```

Listing 1: Wine DIS Specification: WhiteWine datascape concept

The full Wine DIS implementation is available open-source at [DIS25]. This repository includes all theory files, dataset examples, and template outputs required to regenerate and reason on the full specification.

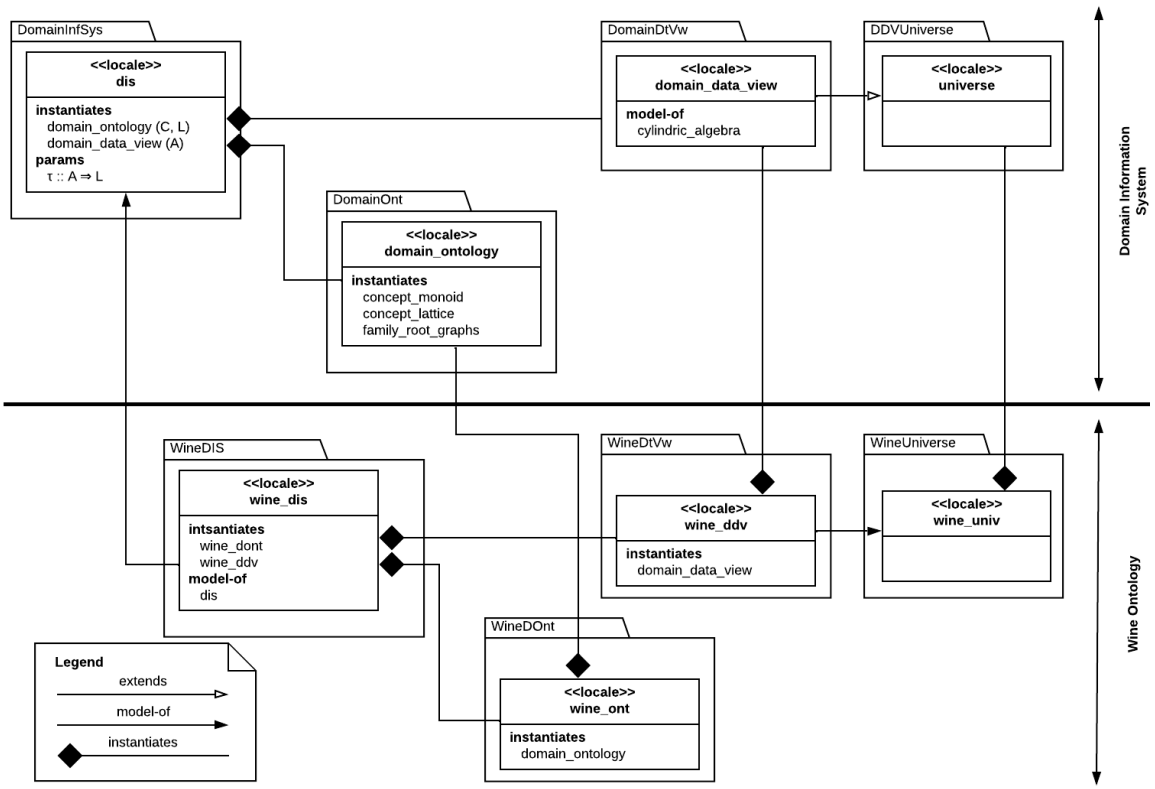

Figure 7: Wine DIS Specification: Design Overview

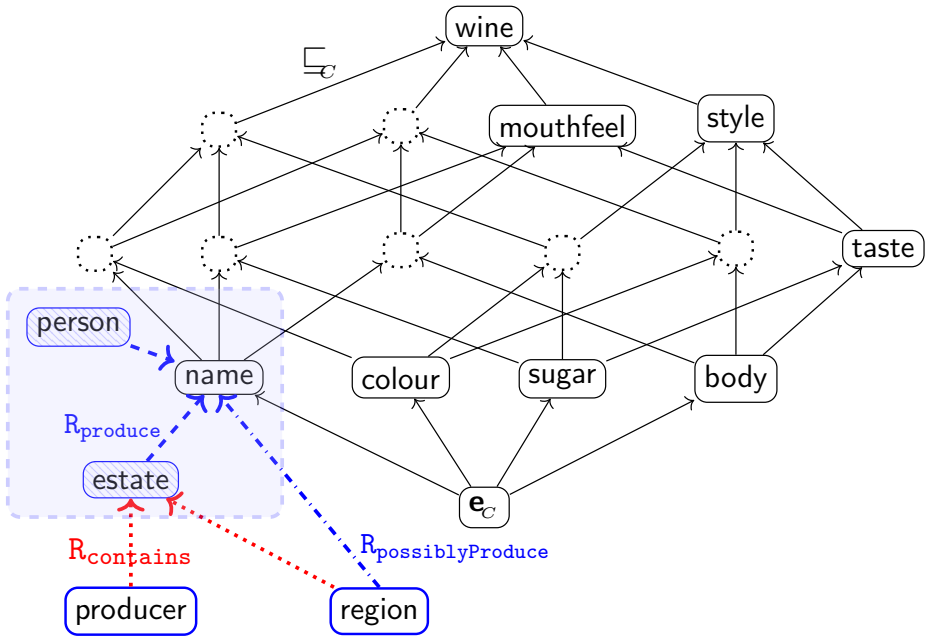

Figure 8: Wine Domain Ontology

Name	Colour	Sugar	Body
Merlot	Red	Dry	Full
Chardonnay	White	Dry	Medium
Vidal	Rose	Sweet	Fruity
Magliocco	Red	Dry	Full
...	...	...	...

Table 5: Wine dataset

7 Discussion

This section reflects on the significance of the DIS framework by examining its practical implications, its support for ontology engineering and evolution, the expressivity-decidability trade-off inherent in its formalisation, and the role of logical verification in data-aware conceptual modelling.

7.1 Practical Implications of a DIS

The formalisation of DIS in Isabelle/HOL enables the construction of data-grounded ontologies that are machine-verifiable. In practice, this provides a solid foundation for the deployment of knowledge systems in domains where data is rich, structured, and evolving, such as clinical research, environmental monitoring, or transportation planning.

By enforcing a clean separation between the data layer (DDV) and the conceptual layer (DOnt), and providing an explicit semantic mapping via the operator τ, DIS avoids the pitfalls of traditional monolithic ontology engineering. The formal semantics ensure that concepts are not merely heuristically mapped to data attributes but are grounded in set-theoretic and algebraic reasoning. This enables high-confidence integration of knowledge and data, an essential requirement in regulated or safety-critical environments.

7.2 Support for Ontology Engineering and Evolution

The layered structure of DIS translates directly into engineering benefits. Modularity at the theory level, through Isabelle locales, maps to modular maintainability in implementation. Changes in data content (e.g., new or modified data records) affect only the DDV and its mapping; any rules about the existing datascape concepts are automatically verified. Changes in domain semantics (e.g., new domain relations) impact only the DOnt; as with the data content changes, the rules that involve datascape concepts are verified automatically. This localised update model supports scalable ontology evolution with traceable impact analysis.

Moreover, DIS supports modularisation through its rooted graph constructs. As demonstrated in prior work [LKM19], through view traversal operations, modules can be extracted systematically, and the resulting knowledge loss can be measured algebraically by evaluating the kernel of the extracted module. This provides a foundation for the construction of privacy- or task-specific modules, which are key to modern KRR applications.

Finally, the template-driven generation of DIS instances (discussed in Section 5) provides a semi-automated pathway for instantiating DIS components. This significantly reduces manual engineering effort while preserving semantic correctness, enabling domain experts to operate within a constrained and safe design space.

7.3 Expressivity vs. Decidability: Tradeo-ffs in Higher-Order Logic

The use of HOL offers expressive power unmatched by the FOL fragments commonly used in OWL or OBDA settings. DIS can represent algebraic and relational structures (e.g., monoids, lattices, rooted graphs), and define operators such as cylindrification over abstract carrier sets. However, this expressivity comes at the cost of theoretical undecidability. Unlike DL-based reasoning engines, which rely on tractable FOL fragments, the Isabelle/HOL proof assistant provides no decidability guarantees for arbitrary theories.

Nonetheless, DIS implementation in Isabelle/HOL remains tractable in practice for two reasons. First, the modular separation between data and concept layers reduces proof complexity by confining reasoning to specific modules. Second, the automation tactics in Isabelle/HOL (such as `auto`, `blast`, `metis`) can efficiently handle most routine reasoning tasks such as classification, subsumption, and satisfiability. This is because the search space is finite and small, due to the design of the DIS components, specifically the carrier sets of both Boolean structures, as well as the monoid of concepts.

This trade-off is justified when correctness is paramount, and when ontologies evolve alongside dynamic data. The cost of undecidability is offset by the gain in confidence and maintainability.

7.4 Verification in Data-Aware Conceptual Modelling

DIS bridges a key gap in conceptual modelling: ensuring that domain concepts reflect and remain aligned with evolving data. Formal verification plays a central role in this. By grounding both the data theory (via cylindric algebra) and the domain ontology (via Boolean lattices and rooted graphs), and by expressing their interaction through formal mappings, DIS enables rigorous verification of alignment, consistency, and reasoning properties.

This verification extends to design intent, such as ensuring that lattice and rooted graphs concepts are always supported by actual data, checking that rooted graphs respect declared concept dependencies, and verifying that data-driven concepts (e.g.,datascape concept `whiteWine`) conform to domain axioms (e.g., “no wine is both red and white"). In this sense, verification in DIS establishes the foundation for the engineering of semantically faithful, evolvable, and auditable knowledge systems.

8 Conclusion and Future Work

This report details the specification and automation of the DIS theory, a modular and formally grounded approach to knowledge representation over structured data. The DIS theory, formalised in Isabelle/HOL, separates concerns between the data layer and the conceptual layer by modelling the DDV as a cylindric algebra and the DOnt as a combination of a concept monoid, Boolean lattice, and a family of rooted graphs. These two layers are linked via a mapping operator that aligns data instances with conceptual categories, enabling rigorous, machine-verifiable reasoning.

A template-driven mechanism has been developed to assist with the construction of DIS specifications from structured datasets. This mechanism reduces the manual effort typically associated with ontology engineering by partially automating the generation of sorts, universe elements, lattice atoms, and their mappings. In addition, integration of DIS engineering with GUI-based tool DISEL, enables domain experts to define rooted graphs and conceptual

mappings through intuitive interfaces rather than code. The approach was validated through the construction of a concrete instantiation, the Wine DIS, which demonstrates how real-world datasets can be lifted into a formally specified, semantically aligned ontology. Reasoning tasks such as classification, satisfiability, and inference were implemented within Isabelle/HOL, confirming the tractability and expressiveness of the formalisation in practice.

Looking ahead, several extensions are planned to enhance the practicality and reach of the DIS approach. A key direction involves fully automating the pipeline from structured dataset to DIS specification. This includes the derivation of the DDV component, the generation of Boolean lattice structure, and the construction of the mapping operator, all without requiring manual intervention. In addition, the DISEL tool can be extended to add datascape concepts and axioms that involve them, which would further increase the usability of a DIS specification engineering. We aim to evaluate the DIS engineering process across a broader range of application domains and datasets, with the goal of confirming its generality, extensibility, and utility in varied real-world contexts.

Through these developments, DIS provides a unified foundation for data-aware ontology engineering, one that balances formal rigour, automation, and usability, and supports the construction of reliable and evolving knowledge systems grounded in structured data.

A Appendix: DIS Templates BNF

This section introduces the syntax used for DIS templates, defined using an extended BNF. Each template is expressed as a series of production rules, where each rule specifies how a nonterminal symbol can be expanded into sequences of terminal symbols (i.e., literal content) or additional nonterminals.

Nonterminal symbols appear within angle brackets $\langle \dots \rangle$, optional elements are enclosed in square brackets $[\dots]$, and repetition is indicated with superscripts: a superscript $*$ denotes zero or more occurrences, and $+$ denotes one or more. The overall grammar is designed to be readable, concise, and machine-processable while remaining interpretable by users unfamiliar with theorem proving environments.

In addition to standard Isabelle syntax, DIS templates support two types of special constructs, tokens and code delimiters. Tokens are written between hash characters $\# \dots \#$, and act as placeholders. These are replaced with concrete values during parsing. Code delimiters are enclosed between ! markers, and provide control structures that direct the generation process. They come in two forms: list and conditional delimiters. List delimiters start with `!foreach #token#!` and end with `!eforeach!`, iterating over the token values. Conditional delimiters start with `!if condition!` and end with `!eif!`, and include content only if the specified condition holds. To avoid verbosity, a shorthand construct called `!listgen!` is provided. This macro generates list-like constructs with optional separators and custom item formatting. It takes three arguments: a mandatory `#token#`, an optional separator `sep=sepchar`, and an optional list item `lstitem=item`. The default separator is a comma, and the default list item is the token itself. The generic expression `!listgen #token# sep=sepchar lstitem=item!` expands to

```
!foreach #token#!
    !if 'position neq first! sepchar !eif! item
!eforeach!
```

The template-generated Isabelle theories may also include markup and comment directives. Isabelle supports informal content using document comments and markup commands. Document comments are enclosed in $(* \ldots *)$, which are ignored during PDF generation. Markup commands, such as section, subsection, or text, are embedded using Isabelle special markup symbols $<< \cdots >>$). These commands enrich the generated documentation and are rendered in the final PDF output.

Once a DIS template is parsed and instantiated into an Isabelle theory, the resulting file can be compiled to generate a user-readable PDF. At this point, the ontology expert is encouraged to add comments and organisational markup using Isabelle commands. However, it is important to note that any manual edits will be overwritten if the template is regenerated. For persistent additions, users should maintain a separate layer of documentation or finalise comments after all parsing is complete. This foundational syntax ensures consistency across all four DIS templates.

Glossary

B

BNF Backus–Naur form. 18

D

DDV Domain Data View. 1, 9, 11, 13, 14, 16–18

DIS Domain Information System. i, 1–4, 6, 9–19

DL Description Logic. 2, 3, 17

DOGMA Developing Ontology-Grounded Methods and Applications. 2

DOnt Domain Ontology. 1, 9, 11, 13, 14, 16, 17

F

FOL First Order Logic. 2, 5, 17

H

HOL Higher Order Logic. 2–4, 11, 17

I

Isabelle/HOL generic Higher-Order Logic proof assistant Isabelle. 1–4, 6, 11–13, 16–18

Isar Intelligible semi-automated reasoning). 4

ITP Interactive Theorem Prover. 2, 3

K

KRR Knowledge Representation and Reasoning. 16

M

ML Meta Language. 3, 6

O

OBDA Ontology-based Data Access. 2, 17

OWL Web Ontology Language. 2, 17

P

PIDE Prover IDE. 3

S

SEADOO SEmi-Automatic Design Of Ontologies. 2

SPARQL SPARQL Protocol and RDF Query Language. 2

SQL Structured Query Language. 2

References

[BASCW18] Achim D Brucker, Idir Ait-Sadoune, Paolo Crisafulli, and Burkhart Wolff. Using the isabelle ontology framework. In International Conference on Intelligent Computer Mathematics, pages 23–38. Springer, 2018.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. The description logic handbook: Theory, implementation and applications. Cambridge university press, 2003.

[CCKE+17] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. Semantic Web, 8(3):471–487, 2017.

[DGLL+18] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. Using Ontologies for Semantic Data Integration, pages 187–202. Springer International Publishing, Cham, 2018.

[DIS25] DIS specification. `https://github.com/aliciam2025/dis/tree/main`, 2025. Accessed: June 11, 2025.

[GCW23] Michael Grüninger, Amanda Chow, and Janette Wong. Semiautomatic design of ontologies. In IFIP Working Conference on The Practice of Enterprise Modeling, pages 143–158. Springer, 2023.

[Isa25] Isabelle: A generic interactive proof assistant. `http://isabelle.in.tum.de/index.html`, 2025. Accessed: April. 10, 2025.

[KVS20] Konstantinos I Kotis, George A Vouros, and Dimitris Spiliotopoulos. Ontology engineering methodologies for the evolution of living and reused ontologies: status, trends, findings and recommendations. The Knowledge Engineering Review, 35, 2020.

[LKM19] Andrew LeClair, Ridha Khedri, and Alicia Marinache. Toward measuring knowledge loss due to ontology modularization. In Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 2: KEOD, pages 172–184. INSTICC, SciTePress, 2019.

[Lov16] Donald W Loveland. Automated theorem proving: A logical basis. Elsevier, 2016.

[Mar16] Alicia Marinache. On the structural link between ontologies and organised data sets. Master's thesis, McMaster University, Hamilton, ON, Canada, 2016.

[MKLM21] Alicia Marinache, Ridha Khedri, Andrew LeClair, and Wendy MacCaull. DIS: A data-centred knowledge representation formalism. In 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), pages 1–8. IEEE, 2021.

[MKM25] Alicia Marinache, Ridha Khedri, and Wendy MacCaul. DIS From Theory to Semantics (Technical Report). Technical report, McMaster University, Computer and Software Department, 06 2025.

[STM08] Peter Spyns, Yan Tang, and Robert Meersman. An ontology engineering methodology for dogma. Applied Ontology, 3(1-2):13–39, 2008.

[WCAK22] Yijie Wang, Yihai Chen, Deemah Alomair, and Ridha Khedri. DISEL: A Language for Specifying DIS-Based Ontologies. In Knowledge Science, Engineering and Management (KSEM), volume 13369 of Lecture Notes in Artificial Intelligence, pages 155–171. Springer, 2022.

[WW07] Makarius Wenzel and Burkhart Wolff. Building formal method tools in the isabelle/isar framework. In International Conference on Theorem Proving in Higher Order Logics, pages 352–367. Springer, 2007.

[XCK+18] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: a survey. In Proceedings of the 27th International Joint Conference on Artificial Intelligence, pages 5511–5519. AAAI Press, 2018.