P2874.X™/D1
Draft Standard for Spatial Web Implementation Specification — HSML Implementation

Developed by the

C/AISC - Artificial Intelligence Standards Committee
of the
IEEE Computer Society

Approved

IEEE SA Standards Board

Abstract:

Keywords: Hyperspace Transaction Protocol, HSTP, Spatial Web, Hyperspace Modelling Language, HSML, Universal Domain Graph, UDG, Spatial Web Identifier, SWID, W3C Decentralized Identifier (DID), Verifiable Credentials

---

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

PDF: ISBN 978-0-XXXX-XXXX-X STDXXXXX STDXXXXX
Print: ISBN 978-0-XXXX-XXXX-X STDPDXXXXX STDPDXXXXX

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (https://standards.ieee.org/ipr/disclaimers.html), appear in all IEEE standards and may be found under the heading "Important Notices and Disclaimers Concerning IEEE Standards Documents."

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers involved in technical working groups are not necessarily members of IEEE or IEEE SA and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning all standards, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement IEEE Standards documents do not guarantee safety, security, health, or environmental protection, or compliance with law, or guarantee against interference with or from other devices or networks. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied "AS IS" and "WITH ALL FAULTS."

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document should rely upon their own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus balloting process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English language version published by IEEE is the approved IEEE standard.

Use by artificial intelligence systems

In no event shall material in any IEEE Standards documents be used for the purpose of creating, training, enhancing, developing, maintaining, or contributing to any artificial intelligence systems without the express, written consent of IEEE SA in advance. "Artificial intelligence" refers to any software, application, or other system that uses artificial intelligence, machine learning, or similar technologies, to analyze, train, process, or generate content. Requests for consent can be submitted using the Contact Us form.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual is not, and shall not be considered or inferred to be, the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE or IEEE SA. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s). News releases about IEEE standards issued by entities other than IEEE SA should be considered the view of the entity issuing the release rather than the formal position of IEEE or IEEE SA.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, consulting information, or advice pertaining to IEEE Standards documents**.

Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its Societies and subcommittees of the IEEE SA Board of Governors are not able to provide an instant response to comments or questions, except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE SA working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the IEEE SA myProject system.[1] An IEEE Account is needed to access the application.

Comments on standards should be submitted using the Contact Us form.[2]

[1] Available at: https://development.standards.ieee.org/myproject-web/public/view.html#landing.
[2] Available at: https://standards.ieee.org/about/contact/.

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

Copyrights

Photocopies

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit IEEE Xplore or contact IEEE.[3] For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

[3] Available at https://ieeexplore.ieee.org/browse/standards/collection/ieee.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE SA Website.[4] Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in IEEE Xplore. Users are encouraged to periodically check for errata.

Patents

IEEE standards are developed in compliance with the IEEE SA Patent Policy.[5]

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at https://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

IMPORTANT NOTICE

Technologies, application of technologies, and recommended procedures in various industries evolve over time. The IEEE standards development process allows participants to review developments in industries, technologies, and practices, and to determine what, if any, updates should be made to the IEEE standard. During this evolution, the technologies and recommendations in IEEE standards may be implemented in ways not foreseen during the standard's development. IEEE standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, data privacy, and interference protection practices and all applicable laws and regulations.

[4] Available at: https://standards.ieee.org/standard/index.html.
[5] Available at: https://standards.ieee.org/about/sasb/patcom/materials.html.

Participants

At the time this draft Standard was completed, the Spatial Web Working Group had the following membership:

<Chair Name>, *Chair*
<Vice-chair Name>, *Vice Chair*

Participant1	Participant4	Participant7
Participant2	Participant5	Participant8
Participant3	Participant6	Participant9

The following members of the Standards Association balloting group voted on this Standard. Balloters may have voted for approval, disapproval, or abstention.

Balloter1	Balloter4	Balloter7
Balloter2	Balloter5	Balloter8
Balloter3	Balloter6	Balloter9

When the IEEE SA Standards Board approved this Standard on <Date Approved>, it had the following membership:

<Name>, *Chair*
<Name>, *Vice Chair*
<Name>, *Past Chair*
<Name>, *Secretary*

SBMember1	SBMember4	SBMember7
SBMember2	SBMember5	SBMember8
SBMember3	SBMember6	SBMember9

Introduction

This introduction is not part of P2874.X/D1, Draft Standard for Spatial Web Implementation Specification—HSML Implementation

This document specifies the Hyperspace Modeling Language (HSML) Implementation Specification, a core component of the IEEE P2874 Spatial Web Standard. HSML provides the semantic backbone for representing **entities, domains, activities, channels, tools, and digital twins** in a unified framework. It is both human- and machine-readable, designed to ensure semantic precision, governance, and interoperability at web scale.

As envisioned in IEEE P2874/D3.3.2, the Spatial Web is a globally interoperable socio-technical fabric connecting humans, autonomous systems, and the Internet of Things (IoT). Its purpose is to integrate the digital and physical worlds into a **shared semantic fabric** where information is contextual, interactions are secure, and governance is intrinsic. Realizing this vision requires a data model capable of:

– representing complex, compositional systems;

– supporting automated activities and contracts;

– embedding governance and credentialing;

– enabling explainability and compliance across diverse domains.

HSML fulfills this role by defining a **normative ontology and modeling language** for the Spatial Web. It specifies how entities are identified, how domains declare spatial structure (via hyperspaces), and how activities, permissions, and governance rules are semantically represented. HSML does not itself provide query or transaction mechanisms; these are addressed by complementary specifications such as the **Hyperspace Query Language (HSQL)** and the **Hyperspace Transaction Protocol (HSTP)**. Instead, HSML forms the semantic layer upon which those protocols depend.

IMPORTANT:
IEEE P2874/D3.3.2 is an approved draft of a proposed IEEE Standard and remains subject to change. This HSML Implementation Specification has been developed to meet the Implementation Specification requirements defined within the IEEE P2874 Draft Standard.

Background and Motivation

The Spatial Web is built on an integrated framework of **protocol, architecture, and governance**, embodied in the Hyperspace Transaction Protocol (HSTP) and the Hyperspace Modeling Language (HSML). Together these provide the foundation for a **globally interoperable system of systems**, defining how entities are modeled, how transactions are conducted, and how governance is embedded at every layer. IEEE P2874 unifies these elements into a reference model for Spatial Web components, systems, and services.

At the core of this foundation is the concept of **HYPERSPACE**. Formally, a hyperspace is a set of points where, for every ordered pair, there may exist a set of paths between them, plus an identity path for each point. This abstraction generalizes the notion of "space" beyond geography: points may represent entities, and paths may represent relationships, adjacencies, or transformations. Hyperspace enables the Spatial Web to merge the physical and virtual into a **shared semantic fabric** that transcends national and geographic boundaries. Entities that are physically distant may still be tightly coupled in other dimensions (e.g., logical, organizational, or contextual), allowing new forms of interaction, coordination, and governance.

Complementing **HYPERSPACE** is the concept of **DOMAIN**, defined as a "sphere of knowledge, influence, or activity." Domains provide the semantic scaffolding of the Spatial Web: they anchor identifiable entities,

declare the hyperspaces they inhabit, and establish rules of interaction (see Figure 1). Through shared and linkable domain architectures, HSML ensures that entities and their interrelationships can be consistently described, discovered, and governed across contexts.

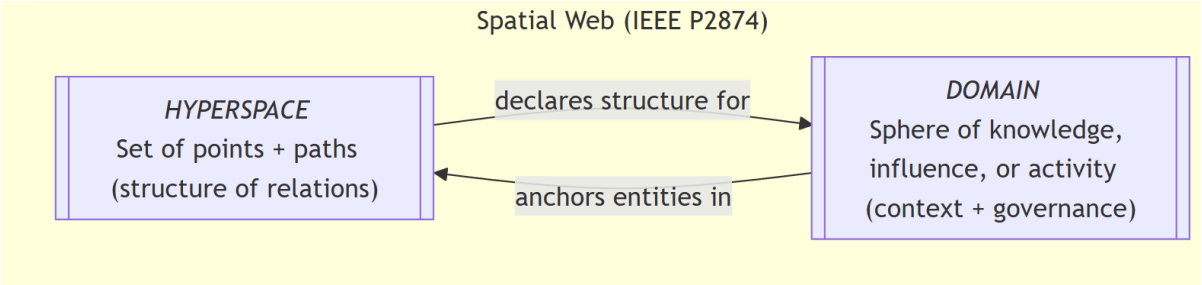

Figure 1—Hyperspace and Domain in the Spatial Web

The development of the Spatial Web is driven by converging technological and societal forces, including:

- The increasingly **graph-like nature of global data**.
- The opportunity for **automation and autonomic activities** enabled by context-aware AI.
- The need for **composable and governable systems** spanning organizations and jurisdictions.
- The requirement for **secure, verifiable transactions**.
- The rise of **machine learning, neural computation, and edge intelligence**, demanding semantic integration.
- The imperative for **explainable AI and robotic governance** to ensure trust and accountability.
- The global proliferation of the **IoT and sensor meshes**, requiring scalable, standards-based models.

Ultimately, the Spatial Web is conceived as a **socio-technical system of systems**. Its realization must address not only technical challenges of modeling, computation, and communication, but also the **social, cultural, ethical, and legal dimensions** of technology use. IEEE P2874 is therefore designed as a socio-technical standard, balancing technical infrastructure with governance frameworks to ensure a secure, interoperable, and human-centered digital-physical reality.

Relationship to IEEE P2874 (Spatial Web Foundation)

This HSML Implementation Specification is directly derived from and builds upon the IEEE P2874/D3.3.2 Draft Standard for Spatial Web Protocol, Architecture and Governance, dated March 2025. Material from the Spatial Web Foundation's "Spatial Web Protocol, Architecture and Governance," Version D3.3, copyright © 2025 Spatial Web Foundation, was used with permission as the base for this standard.

As a reference model, the IEEE P2874 standard establishes the set of required modules and sub-modules for Spatial Web-compliant implementations, along with their minimum functions and associated information models. It defines requirements for a set of Implementation Specifications, such as this HSML document, and identifies existing technologies and standards to be integrated into these implementations.

Goals and Benefits of HSML

The overarching purpose of the IEEE P2874 standard, and by extension this HSML Implementation Specification, is to provide a holistic and coherent technical framework for implementing a collaborative, interactive Spatial Web and shared world system. HSML is designed as a human- and machine-readable

modeling language and semantic data ontology schema that describes objects, relations, actions, activities, and their permissions.

Specifically, HSML aims to achieve the following key goals:

- **Spatially Defined Digital Content Fulfillment**: Enable a functional layer stack capable of fulfilling spatially defined real-world and virtual requests for digital content while respecting governance authorities and self-sovereign identity.
- **Comprehensive Data Ontology**: Define a data ontology for describing objects, relationships, and activities, which is foundational for consistent data interpretation.
- **Verifiable Credentialing**: Specify a verifiable credentialing and certification method for permissioning create, retrieve, update, and delete access to devices, locations, users, and data.
- **Automated Contracting**: Support a **machine- and human-readable contracting language** that enables expression and automated execution of legal, financial, and physical commitments, tied to activities and governed by explicit norms and policies.
- **Shared Understanding**: Facilitate **semantic interoperability** between humans and AIs by ensuring that contextual meaning, intent, and activity outcomes are represented in a common HSML framework.
- **Explainable AI**:: Advance **AI transparency and accountability** by enabling explicit modeling of decision-making processes, conditions, and credentials required for activities.
- **Universal Interoperability**: Drive interoperability of models, data, and activities to enable **cross-domain collaboration** across organizations, networks, and geopolitical boundaries.
- **Compliance and Governance**: Provide the semantic hooks for ensuring compliance with **local, regional, national, and international** regulatory, ethical, and cultural norms—while maintaining auditability through credential checks and policy evaluation.
- **Secure Identity and Authentication**: Ensure that **identity, privacy, authentication, and transparency** are embedded by design, contributing to robust verification of critical activities, agents, and data flows.

Together, these goals ensure that HSML delivers not just a data model, but a **governable, interoperable, and trustworthy foundation** for the Spatial Web—capable of unifying physical, digital, and agentic domains into a shared semantic fabric.

Contents

List of tables

List of figures

1. Overview

This document provides a high-level overview of the Hyperspace Modeling Language (HSML) specification. HSML is the **semantic foundation of the Spatial Web**, enabling consistent representation of entities, domains, activities, spaces, and governance across distributed systems. It describes the modeling paradigm, guiding principles, modular structure, and formal extension mechanisms needed to support diverse domains while ensuring interoperability and compliance with IEEE P2874.

1.1. HSML Modeling Paradigm

The Spatial Web is founded on eight interdependent dimensions that, together, distinguish it from the conventional Semantic Web. HSML integrates these dimensions into a coherent modeling paradigm.

- **Entities:** The fundamental units of description in the Spatial Web. Every entity is globally identifiable by a Spatial Web Identifier (SWID) that conforms to W3C Decentralized Identifier (DID) syntax. Unlike Linked Data URIs, which depend on centralized DNS resolution, SWIDs are cryptographically verifiable, self-sovereign, and portable across networks. This guarantees persistence of reference and enables interoperable interactions across distributed systems.
- **Domains:** A specialized kind of entity that functions as a holon — a self-contained whole that is also part of larger systems. A domain defines a "sphere of knowledge, influence, or activity" in which governance rules apply. Examples include agents, persons, organizations, places, and things. Domains carry identity, enforce norms, and govern activities within their scope. Domains naturally form **holarchies**, where they can be linked through ad hoc memberships, component-of relations, or aggregation into more complex structures.
- **Hyperspace:** The structural layer that defines how entities in a domain relate spatially or structurally. Hyperspaces generalize the notion of "space" beyond geography, encompassing topological, metric, vector, cellular, or graph-based structures. A domain may be bound to one or more hyperspaces, which define adjacency, connectivity, and distance among its entities. This allows HSML to uniformly represent physical, logical, and abstract spatial structures.
- **Semantics:** Every element and relationship in HSML has a well-defined, machine-readable meaning. By adopting RDF, OWL, and SHACL, HSML ensures that data is not just syntactically structured but also contextually understood. This enables reasoning, validation, and interoperability across heterogeneous systems. Semantics provide the interpretive layer that makes entities discoverable, explainable, and governable.
- **Distributed Graph Structure:** The Spatial Web is organized as a **distributed semantic hypergraph**. Entities and domains form the nodes, and their relationships form semantically typed edges (e.g., isLocatedAt, isOwnedBy, performsActivity). This distributed graph spans multiple domains and infrastructures, enabling global interconnection without centralized control. The hypergraph model supports complex, multi-dimensional queries and compositional reasoning across contexts.
- **Identity and Trust:** Trust is not an external service but intrinsic to the model. Every entity's SWID is a DID, which anchors it in a decentralized identity framework. Verifiable Credentials (VCs) provide attestations about entities or domains, while Zero-Knowledge Proofs (ZKPs) allow selective disclosure of credentials for privacy-preserving validation. This creates a **built-in trust layer**, ensuring authentication, authorization, and provenance are verifiable across domains without reliance on centralized authorities.
- **Activities:** Activities are first-class entities that capture dynamic processes and state changes. They can be defined abstractly as schemas (with pre- and post-conditions) or realized as specific instances with execution states. Activities link agents, tools, and things, providing the means by which goals

are pursued and domains interact. Modeling activities explicitly makes the Spatial Web executable, supporting workflows, coordination, and monitoring.

– **Governance:** The Spatial Web is not only technical but also social and ethical. Governance encompasses the rules, norms, contracts, and policies that domains declare and enforce. Governance by design ensures that interactions are not merely possible but accountable, lawful, and ethical. HSML provides formal hooks for embedding governance into domains, so that compliance, trust, and fairness are enforced natively rather than retrofitted.

1.2. Design Principles for HSML Implementation

The implementation of HSML is guided by design principles that ensure the specification is robust, adaptable, and widely reusable. These principles extend the modeling paradigm into practical rules for ontology engineering and system interoperability.

– **Composability:** HSML models are built from smaller, reusable units. Domains, activities, and hyperspaces are defined as modular components that can be assembled into larger systems. This allows complex applications to be constructed dynamically from standardized parts without redundancy.

– **Interoperability:** HSML provides a shared semantic layer that ensures systems developed by different organizations can interact without ambiguity. Interoperability extends beyond syntax to include identity, provenance, and trust by binding every entity to a SWID/DID.

– **Leverage Existing Standards:** HSML reuses existing standards wherever possible (RDF, OWL, SHACL, W3C DID/VC, GeoSPARQL, WoT, SAREF). By aligning with mature specifications, HSML avoids reinvention, accelerates adoption, and maintains compatibility with broader ecosystems.

– **Extensibility:** HSML anticipates evolving needs. Formal mechanisms such as **profiles**, **custom schemas**, and **module extensions** allow new data types, relationships, and governance models to be introduced without breaking backward compatibility. This ensures adaptability across industries and use cases.

– **Modularity and Minimal Ontological Commitment:** HSML is structured into **independent modules**, each addressing a specific functional concern (e.g., Agents, Activities, Hyperspaces, Channels, Governance). Each module makes only the minimal ontological commitments necessary for interoperability, avoiding prescriptive domain-specific assumptions. This maximizes flexibility, lowers barriers to adoption, and supports **plug-and-play reusability**.

– **Governance by Design:** Governance is intrinsic to the model, not an external layer. Domains can declare rules, norms, contracts, and policies as part of their identity and scope. This ensures compliance, accountability, and fairness are enforceable at the semantic level.

– **Decentralized Trust by Design:** Trust is anchored in decentralized identifiers (DIDs), verifiable credentials (VCs), and zero-knowledge proofs (ZKPs). This creates a built-in trust fabric where identity, authentication, and authorization are portable and verifiable across domains, without reliance on centralized authorities.

– **Holonic Organization:** Domains act as holons, enabling both autonomy and integration. Holarchies allow domains to be nested or linked in multiple ways — through membership, composition, or aggregation — ensuring scalability and resilience in distributed environments.

– **Separation of Concerns:** HSML distinguishes clearly between **entities** (what exists), **hyperspaces** (how things relate), **activities** (what happens), and **governance** (how things are regulated). This modularity ensures each dimension can evolve independently while remaining semantically aligned.

- **Validation and Assurance:** HSML employs SHACL shapes and OWL constraints to enforce structural and semantic consistency. Validation is a first-class concern, guaranteeing that data exchanged across domains is coherent, trustworthy, and policy-compliant.
- **Explainability and Reasoning:** HSML ensures that all relationships are explicit and machine-interpretable, enabling semantic reasoning and explainable AI. Agents and systems can not only act but also justify their actions against declared norms, goals, and rules, supporting transparency and accountability.

1.3. Core Model and Modules

To manage complexity and support broad adoption, the HSML specification is organized into a **Core Model** and a set of **modular extensions**. This modular design ensures maximum reusability with minimal ontological commitment, allowing implementers to adopt only the components relevant to their application while maintaining interoperability across the Spatial Web.

- **Core Model:** The Core Model defines the most fundamental and universal constructs of HSML. It establishes:
  - **Entity** – the universal superclass for all things in the Spatial Web, each identified by a Spatial Web Identifier (SWID/DID).
  - **Domain** – a specialized kind of entity functioning as a holon, encapsulating identity, governance, and scope.
  - **Foundational relationships** – common properties for identity, membership, composition, and linkage across entities and domains. The Core Model is the **semantic foundation** upon which all other modules are built.
- **Modules:** Building on the Core Model, HSML defines a set of modules that introduce specialized constructs for particular functional areas. Each module adds only the minimum commitments necessary to support its scope, ensuring extensibility and reuse:
  - **Agent Module** – defines persons, organizations, and autonomous systems as domains with agency, goals, and capabilities.
  - **Activity Module** – defines structured processes, schemas, instances, and execution lifecycles that domains can perform.
  - **Hyperspace Module** – defines the spatial and structural contexts where entities relate. – The Hyperspace Module is itself **composed of submodules** for different space types: **Topological Space**, **Metric Space**, **Vector Space**, **Cellular Space**, **Graph Space**, and **Datatype Space**.
  - **Communication Module** – defines **Channels** as the semantic construct for communication and coordination. – A Channel is a stream of HSML entities bound to a specific Activity, enabling message exchange, data flow, and coordination among participants. – Channels are **transient**: unlike Domains, they do not carry persistent holonic identity but exist in relation to an Activity context. – The module provides constructs for sub-channels, membership, and message traceability, aligning with HSTP message envelopes for interoperable communication.
  - **Governance Module** – defines credentials, contracts, norms, and policies that domains declare and enforce.

This modular architecture enables **plug-and-play deployment**: implementers can select the subset of modules required for their application, while remaining compatible with the broader HSML ecosystem and the IEEE P2874 Spatial Web standard (see Figure 2).

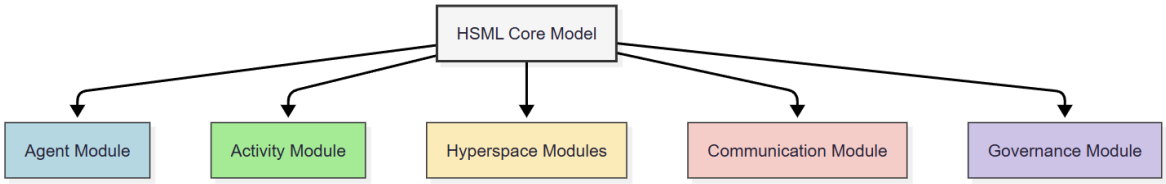

Figure 2—HSML Core Model and Modules with Hyperspace Submodules

1.4. Extension Mechanisms

Recognizing that no single specification can anticipate all future needs, HSML provides formal mechanisms for extension. These ensure that the language can evolve and adapt to new technologies, industries, and governance requirements without breaking compatibility with the Core Model.

- **Profiles:** A profile is a curated collection of HSML modules, constraints, and optional extensions tailored for a particular application or industry. Profiles act as **vertical specializations** while remaining grounded in the common HSML core.
  - Example: A **Smart Mobility Profile** may combine the Agent, Activity, and Hyperspace modules with a custom **Transportation** schema to support autonomous vehicle networks.
  - Example: A **Digital Health Profile** may combine the Governance module with domain-specific ontologies for medical credentials, ensuring trust and compliance. Profiles promote interoperability by establishing shared constraints within a sector, while still ensuring cross-domain compatibility through the HSML foundation.
- **Custom Schemas:** For more granular or domain-specific needs, developers may define custom schemas by reusing Semantic Web standards:
  - **OWL 2** for declaring new types of entities, domains, properties, and relationships.
  - **SHACL** for defining constraints, validation rules, and profiles that data instances must satisfy.
  - **RDF-star and SHACL rules** for expressing richer link types (e.g., weighted, temporal, or probabilistic edges). These schemas can be registered, published, and shared across domains, allowing the Spatial Web ontology to **grow organically while preserving coherence**.
- **Hyperspace Extensions:** The Hyperspace Module is intentionally **open-ended**. In addition to its current submodules (**Topological**, **Metric**, **Vector**, **Cellular**, **Graph**, and **Datatype Spaces**), P2874 anticipates other space types that can be added as extensions. Examples include:
  - **Probabilistic Spaces** – where paths carry uncertainty or probability weights, supporting stochastic modeling.
  - **State-Machine Spaces** – where points represent states and paths represent transitions, aligning with process or workflow modeling.
  - **Observation/Data-Cube Spaces** – treating multi-dimensional data (e.g., sensor arrays, statistical cubes) as hyperspaces.
  - **Quantum or Hilbert Spaces** – supporting future quantum computation or simulation domains. New types can be introduced simply by declaring subclasses of hsml:Hyperspace, ensuring extensibility without changes to the Core Model.

- **Governance Extensions:** Governance models vary across cultures and industries. HSML allows the definition of new contract forms, policy languages, and credential profiles. For example, a financial domain might introduce a **RegulatoryComplianceProfile** requiring specific verifiable credentials, while a social domain might define an **EthicsProfile** for AI agent behavior.
- **Modularity and Minimal Commitment:** All extensions follow the principle of **minimal ontological commitment**. Modules and schemas introduce only the constructs strictly necessary for their scope, ensuring that extensions remain lightweight, reusable, and composable. This preserves interoperability across domains and minimizes barriers to adoption.

This extension framework ensures that HSML can evolve alongside the Spatial Web: supporting innovation while maintaining stability, trust, and semantic coherence.

1.5. Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).[6] [7]

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

2. Normative references

There are no normative references in this document.

The following referenced documents are indispensable for the application of this specification. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document applies.

Draft Standard for Spatial Web Protocol, Architecture, and Governance.

[W3C did-core], Decentralized Identifiers (DIDs) v1.0

[W3C json-ld11], A JSON-based Serialization for Linked Data

JSON-LD 1.1: A JSON-based Serialization for Linked Data, W3C Recommendation.

RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation.

RDF 1.2 Concepts and Abstract Syntax, W3C Working Draft.

RDF 1.2 Turtle, W3C Working Draft.

RDF 1.2 Semantics, W3C Working Draft.

RDF-star and SPARQL-star, W3C Candidate Recommendation.

OWL 2 Web Ontology Language: Document Overview, W3C Recommendation.

[6] The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

[7] The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

Shapes Constraint Language (SHACL), W3C Recommendation.

OGC GeoSPARQL – A Geographic Query Language for RDF Data, OGC Standard.

Verifiable Credentials Data Model v2.0, W3C Recommendation, May 15 2025.

Verifiable Credential Data Integrity 1.0, W3C Recommendation, May 15 2025.

Controlled Identifiers v1.0, W3C Recommendation, May 15 2025.

Bitstring Status List v1.0, W3C Recommendation, May 15 2025.

Securing Verifiable Credentials using JOSE and COSE, W3C Recommendation, May 15 2025.

OWL-Time Ontology, W3C Recommendation.

Schema.org Core Vocabulary, Schema.org Community Standard.

Draft Standard for Spatial Web Implementation Specification — HSML Implementation

4. Abstract

The Hyperspace Modeling Language (HSML) Implementation Specification, a core compliance target of IEEE P2874™ **Standard for Spatial Web Protocol, Architecture, and Governance**, defines a human- and machine-readable semantic modeling language and ontology for the Spatial Web. It establishes a shared vocabulary for **Entities, Domains, Agents, Activities, Credentials, Channels, Hyperspaces, and Governance**, providing a unifying framework for modeling structure, behavior, and trust.

HSML supports multiple spatial structures through the Hyperspace Module, including **topological, metric, vector, cellular, graph, and datatype spaces**. These are expressed using well-defined constructs such as points, paths, subspaces, tensor products, and functors, ensuring flexible yet precise representation of spatial and relational contexts.

The specification leverages **W3C Semantic Web standards (RDF 1.1/1.2, RDF-star, OWL 2, SHACL, JSON-LD 1.1)** to declare formal axioms, constraints, and validation shapes across heterogeneous implementations. It also introduces the **Hyperspace Semantic Query Language (HSQL)**, a query and validation language aligned with SPARQL 1.2 and extended for hyperspace constructs, enabling advanced reasoning over topological, metric, and vector relationships.

Every entity is identified by a **Spatial Web Identifier (SWID)** conformant with W3C DID Core, embedding DID-compliant documents and referenceable endpoints to provide decentralized identity, provenance, and discoverability.

HSML aligns with the **Hyperspatial Transaction Protocol (HSTP)** for message encapsulation and activity routing, and with the **Universal Domain Graph (UDG)** architecture for distributed discovery, linkage, and state management. Together, these enable interoperable, semantically consistent interactions among autonomous agents, digital twins, IoT devices, and distributed services.

As an Implementation Specification, HSML fulfills the compliance requirements of Annex A of IEEE P2874, providing the foundation for **cross-domain semantic interoperability, verifiable credential exchange, governed contract execution, explainable AI, and secure, zero-trust operations** within the global Spatial Web ecosystem.

5. Terms

5.1. Definitions

Only the most HSML-specific terms, or terms fundamental to the core semantic structure and payload content, are defined here.

Activity	A temporally-extended process defined by a set of changes an hsml:Agent can effect. An **Activity Schema** provides a template with conditions, parameters, and variables. An **Activity** (instance) is the execution of such a schema. Activities are central to entity-to-entity execution and state change and may be composed of other Activities. In HSML, Activity Schemas are expressed in RDF/OWL/SHACL for validation and reasoning.
Agent	An hsml:Entity that senses and responds to its environment, maintains a model of that environment, and performs Activities to achieve goals. Agents communicate using Spatial Web protocols.
Channel	A Spatial Web entity (hsml:Channel) that groups a stream of HSML entities related to an Activity in a specific context that does not itself warrant a Domain or hierarchy. Channels support ad-hoc coordination, limited-scope data exchange, and contextual search.
Contract	Represents contractual agreements governing transactions. A Contract typically specifies obligations, rights, and conditions for the performance of an HSML Activity. Contracts can be linked to negotiation or execution phases across multiple Domains.
Credential	Represents verifiable claims about an Entity. HSML adopts the W3C Verifiable Credentials model for authentication, authorization, and trust verification. Credentials are used to validate access to protected entities, activities, or domains.
Domain	A central element of the Spatial Web representing any Entity with a persistent identity. Domains are identified by Spatial Web Identifiers (SWIDs). Domains provide a conceptual area to describe reality, declare hyperspaces, and establish rules of interaction. Relationships between Domains are managed in the Universal Domain Graph (UDG).
Domain Authority	A governing authority responsible for the lifecycle and trust model of a Domain, including resolution of its SWID Document and validation of credentials associated with the Domain.
Entity	The base class for all Spatial Web entities (hsml:Entity). Entities with persistent identity are modeled as Domains. All Entities must have an associated SWID.

Holon	A recursive unit of organization within the Spatial Web that is simultaneously a **whole** and a **part**.
Hyperspace	A generalized concept of space, fundamental to the Spatial Web. HSML defines abstract classes for specialized hyperspaces (e.g., TopologicalSpace, MetricSpace, VectorSpace, CellularSpace) that specify relationships among elements of a Domain.
MetricSpace	A specialization of Hyperspace (metric:MetricSpace) where relationships between elements are determined by a defined metric (distance or similarity function). A MetricSpace declares one or more metrics (e.g., Euclidean distance, cosine similarity) that quantify how "close" or "similar" its elements are. MetricSpaces support quantitative operations such as nearest-neighbor search, clustering, threshold filtering, and path cost evaluation. They separate structural representation (the Hyperspace itself) from the semantics of measurement (metrics), enabling consistent and portable evaluation across implementations.
Policy	A set of constraints or rules governing the behavior of Entities, Domains, and Activities. Defined in IEEE P2874 §6.6.7, Policies specify permissions, prohibitions, or obligations, expressed declaratively (e.g., SHACL, OWL, SPARQL).
Norm	A socially or organizationally defined expectation governing behavior within a Domain or Activity. Defined in IEEE P2874, Norms complement Policies by specifying conventions, standards, or best practices that guide interactions among Entities.
SWID	**Spatial Web Identifier.** A unique identifier for a Domain. SWIDs conform to the W3C Decentralized Identifier (DID) Core specification. Each SWID resolves to a SWID Document describing endpoints, credentials, and governance.
Time	Represents temporal aspects within the Spatial Web. Time can be attached to Entities, Activities, or Hyperspaces, and is aligned with OWL-Time for temporal reasoning.
TopologicalSpace	A specialization of Hyperspace (topo:TopologicalSpace) where relationships are defined by **adjacency, neighborhoods, and continuity**, independent of numeric distance. TopologicalSpaces describe how elements are connected (e.g., graph structures, cellular complexes, networks) and which subsets are considered "open." They enable reasoning about connectivity, boundaries, regions, and continuity, making them suitable for domains such as graphs, GIS geometries, cellular automata, and process flows. HSML allows both standard classes (e.g., Region, Boundary) and custom topology definitions by extending the TopologicalSpace model.
UDG (Universal Domain Graph)	A distributed metagraph containing relationships between all known Domains and Entities in the Spatial Web. The UDG provides routing, discovery, and contextual metadata.
UDG Context	Metadata about a Domain's position in the UDG, including its SWID, known neighbor SWIDs, and spatial or routing hints.

NOTE—In HSML, only the semantic entities (Activity, Domain, Agent, Channel, Contract, Credential, Hyperspace, SWID, UDG, etc.) are normative. This section is under development and will be expanded in future drafts.[8]

[8] Notes to text, tables, and figures are for information only and do not contain requirements needed to implement the standard.

5.2. Acronyms and abbreviations

DID	Decentralized Identifier (referenced in SWID definition)
HSML	Hyperspace Modeling Language
HSTP	Hyperspace Transaction Protocol
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
JSON-LD	JSON for Linking Data
OWL	Web Ontology Language
PDR	Preliminary Design Review (spec development phase)
RFC	Request for Comments
SHACL	Shapes Constraint Language
SPARQL	SPARQL Protocol and RDF Query Language
SWF	Spatial Web Foundation
SWID	Spatial Web Identifier
UDG	Universal Domain Graph
VC	Verifiable Credential
W3C	World Wide Web Consortium

6. Conventions and Notation

6.1. URI and IRI Conventions

This specification uses IRIs for identifying HSML modules, classes, and properties. All HSML terms are defined under a **single base namespace**:

```
https://www.spatialwebfoundation.org/ns/hsml
```

Figure 3

HSML is modular: each module (Activity, Agent, Domain, Hyperspace, Policy, etc.) defines its own sub-namespace under this base. This hierarchy ensures that all HSML identifiers are globally unique, discoverable, and traceable to the Spatial Web Foundation. Submodules of Hyperspace (e.g., Metric, Topological, Vector, Cellular) extend hspace: with their own namespaces.

6.1.1. HSML Normative Namespaces

All HSML modules share the common base namespace:

```
https://www.spatialwebfoundation.org/ns/hsml
```

Figure 4

Each module extends this base by declaring its own sub-namespace. Hyperspace modules form a hierarchy: hyperspace is the root, and its specializations (metric, topo, vector, cell,…) extend from it.

Table 1—HSML Normative Namespaces

Prefix	Namespace IRI	Module
core	https://www.spatialwebfoundation.org/ns/hsml/core#	HSML Core
agent	https://www.spatialwebfoundation.org/ns/hsml/agent#	Agent Module
act	https://www.spatialwebfoundation.org/ns/hsml/activity#	Activity Module
gov	https://www.spatialwebfoundation.org/ns/hsml/gov#	Governance Module
comm	https://www.spatialwebfoundation.org/ns/hsml/comm#	Communication Module
hspace	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#	Hyperspace Core
metric	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#	MetricSpace Submodule
topo	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/topo#	TopologicalSpace Submodule
vector	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/vector#	VectorSpace Submodule
cell	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/cell#	CellularSpace Submodule

6.1.2. External Referenced Namespaces

Table 2—External Referenced Namespaces

Prefix	Namespace IRI	Source
dct	http://purl.org/dc/terms/	[DCTERMS]
foaf	http://xmlns.com/foaf/0.1/	[FOAF]
gsp	http://www.opengis.net/ont/geosparql#	[GeoSPARQL]
org	http://www.w3.org/ns/org#	[VOCAB-ORG]
prov	http://www.w3.org/ns/prov#	[PROV]
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	[RDF-SYNTAX-GRAMMAR]
rdfs	http://www.w3.org/2000/01/rdf-schema#	[RDF-SCHEMA]
schema	http://schema.org/	[schema-org]
sh	http://www.w3.org/ns/shacl#	[SHACL]
skos	http://www.w3.org/2004/02/skos/core#	[SKOS-REFERENCE]
owl	http://www.w3.org/2002/07/owl#	[OWL2]
xsd	http://www.w3.org/2001/XMLSchema#	[XMLSCHEMA11-2]
vc	https://www.w3.org/2018/credentials#	[VC-DATA-MODEL]

6.2. RDF/Turtle Syntax Conventions

RDF examples in this specification use Turtle syntax [TURTLE]. The following conventions apply:

– All examples assume the namespace prefixes defined in [HSML_Normative_Namespaces] and [External_Referenced_Namespaces].

– If a prefix is used in an example but not explicitly declared, it SHALL be interpreted as belonging to one of those normative tables.

– Example fragments are illustrative and **non-normative** unless explicitly marked as normative.

– Literals use standard xsd: datatypes (e.g., xsd:string, xsd:integer, xsd:dateTime).

– Persistent identifiers (e.g., SWIDs) are preferred over blank nodes.

– Compact IRIs (CURIEs) are used when a prefix is declared.

Example:

```
did:swd:smarthome123:activity:1 a activity:Activity ;
    core:swid "did:swd:smarthome123:activity:1" ;
    act:activitySchema :TurnOnLight ;
    act:hasStatus "running"^^xsd:string .
```

Figure 5

6.3. JSON-LD Syntax Conventions

HSML entities, activities, and governance artifacts shall be serializable into JSON-LD [JSON-LD11].

For the sake of simplicity and consistency, HSML defines a **single canonical JSON-LD context** that covers all modules:

```
https://www.spatialwebfoundation.org/ns/hsml/context.jsonld
```

Figure 6

This context performs the following functions:

– Provides prefix mappings for all HSML modules (core, agent, activity, gov, comm, hspace, metric, topo, vector, cell).

– Maps each HSML **class IRI** and **property IRI** to a simplified JSON key, corresponding to the **"JSON name"** column defined in the normative tables of this specification.

– Enables JSON developers to work with compact, human-readable field names while ensuring full semantic traceability to IRIs.

– Profiles and extensions MAY extend the canonical context with additional mappings.

```
{
"@context": {
    "core": "https://www.spatialwebfoundation.org/ns/hsml/core#",
    "act": "https://www.spatialwebfoundation.org/ns/hsml/activity#",
    "agent": "https://www.spatialwebfoundation.org/ns/hsml/agent#",
    "schema": "http://schema.org/",
    "dct": "http://purl.org/dc/terms/",
    "type": "@type",
    "swid": "core:swid",
    "name": "schema:name",
    "description": "schema:description"
    "Activity": "act:Activity",
    "Ongoing": "act:Ongoing",
    "activitySchema": "act:activitySchema",
    "hasStatus": "act:hasStatus"
    ....
   }
}
```

Figure 7—Example mapping excerpt (informative)

```
{
```

```
"@context": "https://www.spatialwebfoundation.org/ns/hsml/context.
jsonld",
    "id": "did:swid:smarthome123:activity:1",
    "@id": "did:swid:smarthome123:activity:1",
    "@type": "Activity",
    "swid": "did:swd:smarthome123:activity:1",
    "activitySchema": "did:swd:smarthome123:schemas:TurnOnLight",
    "hasStatus": "Ongoing",
    "name": "Turn on light",
    "description": "Switches on the living room light"
}
```

Figure 8—Example instance

6.4. SHACL Shape Syntax and Annotations

Validation in HSML is expressed using SHACL shapes [SHACL]. The following conventions apply:

- Shapes are defined in Turtle syntax with the sh: namespace.
- If prefixes appear in a SHACL example without being explicitly declared, they SHALL be interpreted using the mappings in [HSML_Normative_Namespaces] or [External_Referenced_Namespaces].
- NodeShapes target specific HSML classes (e.g., activity:Activity).
- Property shapes specify expected predicates, datatypes, or cardinality constraints.
- HSML-specific annotations (e.g., hsml:parameter, hsml:precondition) may extend SHACL for Activity schemas.
- Shapes may serve as **Activity payload templates**, validating inputs and outputs.

Example:

```
:TurnOnLightShape a sh:NodeShape ;
    sh:targetClass act:Activity ;
    sh:property [
        sh:path act:hasStatus ;
        sh:datatype xsd:string ;
        sh:in (act:Planned, act:Ongoing, act:Completed`, act:Failed) ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
] .
```

Figure 9

7. The HSML Modeling Paradigm: Holons, Semantics, and Extensions

The Hyperspace Modeling Language (HSML) is meticulously designed around a set of core entities, with a pivotal emphasis on the “Domain” as a holonic entity, capable of containing and being contained within other Domains, forming a holonic graph. This inherent design choice is crucial for representing the intricate and interconnected real-world and virtual systems that comprise the Spatial Web. Within this framework, every entity inherently exists within a Domain.

The central question then becomes: what modeling language formalisms are necessary to achieve this complex representation, particularly given the holonic nature of Domains?

7.1. The Imperative for Semantics and the Semantic Web Stack

A strong semantic foundation is indispensable for HSML due to the inherent complexity and interoperability requirements of the Spatial Web. The Spatial Web aims to create interoperable, semantically compatible connections between network-connected hardware and software. Digital content in the Spatial Web is designed to be respectful of governance authorities and self-sovereign identity. This necessitates a shared understanding of meaning and context, not just between humans, but also between humans and Artificial Intelligence (AI) systems. Semantics allow for AI systems to have explainability, by explicitly modeling their decision-making processes, and for interoperability of models and data across organizations, networks, and borders.

The **Semantic Web stack**, utilizing technologies like Resource Description Framework (RDF), Web Ontology Language (OWL), SPARQL Protocol and RDF Query Language (SPARQL), and Shapes Constraint Language (SHACL), is the preferred modeling language formalism for HSML over a purely Labeled Property Graph (LPG) approach. The Semantic Web stack natively supports:

- **Ontologies (OWL):** OWL provides a rich set of constructs to define classes, properties, and complex relationships, enabling rigorous semantic modeling. This goes beyond simple key-value pairs, allowing for the expression of transitivity, symmetry, inversions, and other logical inferences crucial for reasoning about interconnected Domains and their behaviors.
- **Linked Data (RDF):** RDF allows for the creation of a "web of data" where entities are identified by URIs and described through triples. This inherently distributed nature aligns with the Spatial Web's decentralized vision and the Universal Domain Graph (UDG) as a distributed hypergraph.
- **Querying Capabilities (SPARQL):** SPARQL enables complex queries over distributed RDF data, allowing for sophisticated retrieval of information across interconnected Domains based on their semantic relationships, not just their direct links.
- **Validation and Constraints (SHACL):** SHACL provides a mechanism to define structural constraints on RDF graphs. This is critical for ensuring data quality, consistency, and adherence to the defined ontology, thereby enforcing the rules and relationships necessary for a coherent holonic graph structure.

While LPGs are excellent for representing graph structures, their primary strength lies in their flexible schema and efficient traversal of direct relationships. However, they typically lack the inherent semantic expressiveness and formal reasoning capabilities that OWL provides, which are critical for defining nuanced relationships (like mereology for nested domains) and enforcing complex constraints needed for a governed Spatial Web. The document emphasizes the need for a shared, linkable knowledge domain architecture and a common language to describe interrelationships, which is precisely what the Semantic Web stack offers through its formal semantics and established standards for knowledge representation.

7.2. Limitations of Existing Standards in Accommodating Holonic Graphs Out of the Box

While the RDF model provides a flexible foundation for representing interconnected data as a graph of triples, certain aspects of its core design present challenges when attempting to natively accommodate holonic structures "out of the box." The concept of a holon, being both a whole and a part simultaneously, and possessing internal properties and governance while also participating in larger wholes, requires more nuanced modeling than basic triples can directly offer.

Here's what's missing in the foundational RDF model to fully accommodate holons, and how extensions like RDF-star, Named Graphs, and the concept of "ad-hoc relations" (as hinted at in RDF-H) attempt to address these gaps:

- **Lack of Native Holonic Representation:** Many graph databases and modeling languages, including traditional property graphs, primarily focus on explicit, pairwise relationships between nodes. While nesting can be *simulated* through parent-child relationships, they often lack built-in mechanisms to enforce the inherent properties of holons, where the whole is more than the sum of its parts, and parts maintain a degree of autonomy within the whole. The concept of a "Domain" as a self-contained entity that also participates as a part of a larger Domain, while retaining its own internal governance, is not a first-class citizen in many existing models. The document describes Domains as able to contain other Spatial Web Domains and that they are "partially or entirely within other Spatial Web Domains; fully or semi-nested and connected geophysical locations or logical structures".
- **Limited Semantic Expressiveness for Complex Relationships:** In a pure RDF model, a relationship (a predicate in a triple) is a simple link. It's difficult to attach metadata or properties **to the relationship itself**. For a holonic structure, the relationship between a "parent" holon and its "child" holon isn't just a simple hasPart link. This hasPart relationship might have properties like:
- startDate (when the child became a part of the parent)
- authority (which Domain Authority sanctioned this part-whole relationship)
- typeOfContainment (e.g., physical, conceptual, administrative)
- permissionGranted (specific permissions granted to the child within the parent's context)

Standard RDF reification (using rdf:Statement, rdf:subject, rdf:predicate, rdf:object) is verbose and can complicate querying. While some standards allow for typed relationships, they may not offer the rich ontological constructs found in OWL to define the *nature* of these relationships (e.g., mereological relations like hasPart or isContainedIn with their implied transitivity) or to reason over them abstractly. The document explicitly mentions drawing on ideas from mereology (the theory of parthood relations) and order theory, which are better supported by formal ontologies than by typical graph database schemas.

- **Bounded Contexts and Graph Partitioning:** A core aspect of holons is that they represent "bounded contexts" where internal rules and properties might apply differently than externally. While RDF allows for arbitrary connections, it doesn't inherently provide a mechanism to delineate distinct "sub-graphs" that represent the internal state or perspective of a holon. This makes it challenging to:
- **Manage Scope and Identity within a Holon:** How do you say that a particular triple applies **only** within the context of a specific CityDomain holon, and not globally?
- **Enforce Internal Governance:** If a BuildingDomain is a holon within a CityDomain, its internal operations might have specific rules. RDF alone doesn't provide direct support for these encapsulated rule sets.
- **Focus on Specific Domains, Not Universal Interoperability:** Many existing standards are highly optimized for specific domains (e.g., geographic information systems, IoT device models). While they provide excellent local solutions, they often struggle to achieve universal interoperability and semantic compatibility across vastly different domains, especially when considering non-physical dimensions or abstract concepts. The Spatial Web's requirement for a "universal domain graph" and its extension of "geography to hyperspace" highlights this gap.
- **Governance and Identity Models Not Inherently Integrated:** Existing graph standards often separate the data model from governance, identity, and access control mechanisms. The Spatial Web, however, integrates concepts like Domain Authorities, verifiable credentials, and self-sovereign identity directly into its core design, influencing how entities interact and how data is accessed and managed within Domains. This deep integration of governance, especially polycentric governance, with the data model is not an out-of-the-box feature of many current graph standards.

- **Dynamic and Adaptive Relationships:** The Spatial Web acknowledges that relationships can be dynamic and adaptive. While graph databases can update relationships, the formalisms to reason about *changes* in relationships or to model adaptive systems at an ontological level are more aligned with advanced semantic web capabilities.

How Extensions and Concepts Address These Gaps:

- **RDF-star (RDF*)**: RDF-star directly addresses the limitation of expressing properties about statements. It introduces a concise syntax to assert triples *about* other triples (nested triples).
- **For Holons:** This is invaluable for modeling holonic relationships. Instead of cumbersome reification, one could directly state [_CityA_hsml_hasPart__BuildingB] hsml:startDate "2023-01-01"^^xsd:date . This allows the precise capture of metadata (like creation date, governing authority, or specific permissions) directly on the part-whole relationship between holonic Domains. It makes the model of a holon's internal composition much richer and more governable.
- **Named Graphs**: Named Graphs extend RDF by allowing a collection of triples to be identified by a URI (its "name" or context). This effectively partitions the RDF graph into distinct sub-graphs.
- **For Holons:** Named Graphs are crucial for representing the "bounded contexts" of holons. Each holon (e.g., a BuildingDomain or OrganizationDomain) could be associated with its own Named Graph. This allows for:
- **Scoped Assertions:** Triples asserted within a building's Named Graph would pertain specifically to that building's internal state, personnel, or operations, without polluting the global graph.
- **Contextual Queries:** Queries can be made against specific Named Graphs, enabling retrieval of information relevant to a particular holon's internal context.
- **Access Control:** Access permissions could be managed at the Named Graph level, dictating who can read or write triples within a specific holon's representation, aligning with Domain Authority concepts.
- **"Ad-Hoc Relations" (drawing from RDF-H principles):** In the dynamic and decentralized environment of a holonic graph like the Spatial Web, not all relationships between entities or domains are permanent, universally defined, or strictly dictated by a pre-existing, rigid schema. This is where the concept of "ad-hoc relations" becomes particularly relevant. These relations are emergent, contextual, and flexible, arising dynamically from the interactions or needs of specific entities or agents. For instance, a temporary collaboration between two geographically distant research ConceptDomain`s might lead to a unique "collaboratesOnProjectX" link that isn't part of the standard `hsml:isRelatedTo hierarchy. Holons' autonomy and interdependence necessitate these:
- **Internal Self-Organization:** A ProjectDomain (a holon) might dynamically create temporary "taskDependency" relations between its internal Activity entities for a specific project phase, which are only relevant within that ProjectDomain.
- **Inter-Holon Collaboration:** When two distinct OrganizationDomain holons form a joint venture, they might establish temporary jointlyManagesResource relations that exist only for the duration of that venture.
- **Domain Authority Assertions:** A DomainAuthority (itself a holonic Agent) might assert an administrativeLink between two otherwise unconnected sub-domains under its governance for a specific, temporary administrative purpose, such as a localized energy distribution optimization project within a larger UrbanDigitalTwinDomain. This link isn't a fundamental, ontological isChildOf relation, but an operational one.

To handle "Ad-Hoc Relations" within HSML's Semantic Web Stack:

- **Dynamic Predicate Creation and Registration:** New predicates (URIs for relations) could be generated and "registered" on the fly, perhaps within a specific Domain's context or by a Domain

Authority. This implies a lightweight mechanism for declaring the *intent* or *scope* of these new predicates.

– **Higher-Order Logic/Reasoning (OWL & SHACL):** Even if specific ad-hoc predicates are dynamic, OWL can define broader patterns or categories for these relations (e.g., an owl:Class called hsml: TemporaryAdministrativeLink). SHACL can then constrain the properties of these ad-hoc relations, ensuring data quality and consistency even for emergent links.

– **Provenance of Relationships (RDF-star):** Using RDF-star, the assertion of an "ad-hoc relation" itself can be treated as a statement with its own metadata. This allows the Spatial Web to precisely record **who** asserted this temporary relation, **when**, **why**, and **for how long** it is considered valid. This provenance is critical for auditability and maintaining trust in a decentralized, dynamic environment where relationships aren't always static.

In essence, while existing graph technologies provide foundational capabilities, the unique blend of holonic structures, multi-dimensional hyperspaces, and integrated governance within the Spatial Web necessitates a more semantically rich and extensible modeling language formalism. The Semantic Web stack provides the necessary expressiveness and extensibility to fully realize the vision of a truly holonic and interconnected Spatial Web.

8. HSML Core Module

8.1. Architectural Principles and Design Rationale

This clause describes the key architectural principles that underpin the Hyperspace Modelling Language (HSML) Core Module. This module provides the foundational concepts upon which all other HSML modules are built. Understanding this rationale is not required for conformance but is highly recommended for a robust and correct implementation, as it clarifies the intent behind the normative rules specified in subsequent clauses.

8.1.1. The Universal Foundation: core:Entity

The cornerstone of the HSML ontology is the core:Entity class, which serves as the universal root for every object, agent, and concept within the Spatial Web. This abstract class embodies the principle that everything which "is perceived, known, or inferred to exist" can be represented and managed within a single, coherent framework. By establishing core:Entity as the ultimate superclass, the standard ensures that all HSML-compliant software can uniformly ingest, validate, and manipulate heterogeneous constructs. This polymorphic foundation is the key to the semantic interoperability and extensibility mandated by the IEEE P2874 standard.

8.1.2. Decentralized Identity: The Spatial Web Identifier (SWID)

A core principle of the Spatial Web is that every core:Entity SHALL be uniquely identified by a Spatial Web Identifier (SWID). This is enforced via the core:swid property. The modeling of this identifier is a critical design choice.

The core:swid property SHALL contain a W3C Decentralized Identifier (DID) conformant URI. This value is modeled as a literal of type xsd:anyURI, not as a link to another resource. This ensures the SWID is a verifiable attribute of the entity.

More importantly, this design establishes a fundamental separation of concerns between the identifier for an entity and the identifier for its description.

core:swid (The "Thing"): This property identifies the entity itself—the person, device, or concept—independent of any single representation or network location. It is the stable, location-agnostic anchor for identity.

Subject IRI (The "Document"): This is the URI of the data record (e.g., an HTTP URL) that describes the entity. An entity can have many different descriptions at different locations, each with its own subject IRI.

This distinction, which aligns with the "Cool URIs for the Semantic Web" best practice, provides several key advantages:

Data Federation: A single conceptual entity, identified by one core:swid, can have multiple representations across the web—a record in a corporate database, a profile on a social network, a state entry in a distributed ledger. The shared SWID makes it possible to link and integrate this federated data.

Resilience & Flexibility: The entity's identity persists even if specific descriptions change or become unavailable. The core:swid provides a permanent identifier that can be resolved to discover current, valid representations of the entity.

Interoperability: It allows any existing web resource (e.g., from DBPedia or a corporate dataset) to be integrated into the Spatial Web non-invasively. By simply adding the core:swid attribute, the resource becomes part of the Spatial Web without altering its original, native identity.

In summary, modeling the SWID as a literal-valued property provides a robust, flexible, and architecturally sound foundation essential for building a distributed, interoperable, and resilient Spatial Web.

8.1.3. Contextual Scaffolding: core:Domain

The Spatial Web is not a flat collection of entities; it is a structured network of contexts. The core:Domain class provides the formal mechanism for this structure, and its placement in the Core Module is essential. While other modules define specific **types** of entities, the Core Module must provide the universal scaffolding **for** all entities. core:Domain is this scaffold, providing the foundational "where" for every "what" (core:Entity).

Domains are conceptualized as **holons**—a term coined by Arthur Koestler to describe an entity that is simultaneously a whole in and of itself, as well as a part of a larger whole. A holon is a semi-autonomous, self-reliant unit that can handle local contingencies, yet it is also integrated into and subject to the context of a larger system. For example, a cell is a whole, but it is also a part of an organ; the organ is a whole, but also part of an organism.

This nested structure of Domains (as holons) forms a **holarchy**—a hierarchy of holons. Unlike a traditional hierarchy based on top-down command, a holarchy emphasizes composition and interdependence. The relationship between levels is best described as "made up of" or "making up" the next level, where the whole exists only as an integration of its semi-independent parts.

This holonic model is critically important for modeling the Spatial Web for several reasons:

– **Reflects Real-World Complexity**: It allows for the creation of complex, multi-level systems that mirror natural and social structures, from biological organisms to human organizations.

– **Enables Polycentric Governance**: It provides a robust framework for distributed governance. Each Domain, as a holon, can maintain its own autonomy and internal rules while being an interdependent part of a larger containing Domain. This allows for policies and logic to be scoped at the appropriate level.

– **Promotes Scalability and Resilience**: The semi-autonomous nature of holons means they can manage local operations and disturbances without constant instruction from higher levels, creating a more resilient and scalable system.

By modeling Domains as holons within a holarchy, HSML provides a powerful and flexible framework for organizing the vast, interconnected, and multi-layered contexts of the Spatial Web.

8.1.4. Minimal Ontological Commitment and Pragmatic Reuse

To ensure maximum flexibility and interoperability, the Core Module adheres to a principle of minimal ontological commitment for its foundational classes. Classes like core:Thing, core:Concept, and core:SpatialFeature are defined with only the most essential properties, establishing them as base types without imposing a heavy, prescriptive structure. This minimalist approach allows them to serve as robust extension points for more detailed, domain-specific standards.

This is complemented by the pragmatic reuse of established vocabularies. For example, core:SpatialFeature leverages GeoSPARQL for its geometric representations (geosparql:hasGeometry), ensuring out-of-the-box compatibility with a mature ecosystem of geospatial tools and data. Similarly, common metadata properties are drawn from Dublin Core (dct:) and Schema.org (schema:). This strategy lowers the barrier to adoption and enhances the integration of HSML data into the broader web of data.

8.2. Normative Classes

This clause provides the normative definitions for all class concepts within the HSML Activity Module. Each class is detailed in a table that specifies its URI, description, JSON-LD context name, usage notes, and relationship to other classes.

The namespace prefix core: refers to https://www.spatialwebfoundation.org/ns/hsml/core#.

Table 3—Summary of Core Module Classes

Class	Description
core:Entity	The abstract root class for all objects, agents, and concepts in the Spatial Web.
core:Domain	A foundational entity that represents a sphere of knowledge, influence, or activity, forming the contextual scaffolding of the Spatial Web.
core:SpatialFeature	A type of Domain representing a location-bound or geometry-bearing element, linking semantics to geospatial structure.
core:Thing	A type of Domain representing a bounded, passive item without agency that can be acted upon or sensed.
core:Concept	A type of Domain representing an abstract idea or category used for classification and knowledge organization.
core:Condition	An abstract class representing a logical condition that can be evaluated to be true or false.
core:SHACLCondition	A concrete condition that uses a SHACL shape for validation logic.
core:Participation	A reusable entity that explicitly models the involvement of another entity in a specific role within a given context.
core:Role	Represents the function or position of an entity within a core:Participation, typically defined in a controlled vocabulary.

8.2.1. Class: core:Entity

The class core:Entity realizes the base concept **ENTITY** in IEEE P2874 Section 6.6.2, defined as “that which is perceived, known, or inferred to exist, has existed, or is anticipated to exist”.

As the root of the Spatial Web ontology, core:Entity provides the universal, abstract foundation upon which all other HSML classes are built. It establishes the mandatory requirements for universal identification and class declaration, ensuring that every object in the ecosystem is uniquely addressable and explicitly typed.

Key Requirements

– **Identity**: Every core:Entity **SHALL** possess a Spatial Web Identifier (core:swid) conformant with W3C DID Core syntax.

– **Typing**: Every instance **SHALL** declare its specific, concrete class via the rdf:type property.

– **Extensibility**: The core:Entity model is designed to be subclassed to create more specialized types. It **SHALL NOT** be instantiated directly.

These requirements ensure that all HSML constructs share a common, machine-readable foundation, enabling polymorphic processing, universal identification, and semantic interoperability.

8.2.1.1. Class Definition

Table 4—Class Definition for core:Entity

RDF Class	**core:Entity**
Is Abstract	Yes
Definition	An *ENTITY* is “that which is perceived, known, or inferred to exist, has existed, or is anticipated to exist.”
Subclass Of	owl:Thing
Usage Note	core:Entity is an abstract class and **SHALL NOT** be instantiated directly. Only its concrete subclasses should be instantiated.
Rationale	As the root of the Spatial Web ontology, core:Entity provides a uniform, extensible foundation for all HSML constructs, enabling universal identification and semantic interoperability.

Table 5—Properties Summary for core:Entity

Predicate	**JSON-LD name**	**Description**	**Range**	**Cardinality**	**Requirement Level**
rdf:type	@type	Declares the specific class of the resource.	rdfs:Class	1..*	**Mandatory**
core:swid	swid	The Spatial Web Identifier for the entity.	xsd:anyURI	1..1	**Mandatory**
schema:name	name	A short, human-readable name for the entity.	xsd:string	0..1	Recommended
schema: description	description	A detailed explanation of the entity’s purpose.	xsd:string	0..1	Optional

8.2.1.2. Properties

This section provides the detailed normative definitions for the properties of the core:Entity class.

8.2.1.2.1. Property: type

Table 6—Property Definition: rdf:type

Property	rdf:type
IRI	http://www.w.org/1999/02/22-rdf-syntax-ns#type
JSON name	@type
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	core:Entity

Table 6—Property Definition: rdf:type *(continued)*

Range	rdfs:Class
Definition	Declares the specific class (or classes) of which this resource is an instance.
Usage Note	Every entity **SHALL** have at least one rdf:type that specifies a concrete (non-abstract) subclass of core:Entity. Multiple types may be present, for example, to include inferred classes.

8.2.1.2.2. Property: swid

Table 7—Property Definition: core:swid

Property	core:swid
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#swid
JSON name	swid
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	core:Entity
Range	xsd:anyURI
Definition	A globally unique Spatial Web Identifier conformant with W3C DID Core syntax.
Usage Note	This property provides the stable, universal identifier for the entity across all contexts and networks.

8.2.1.2.3. Property: name

Table 8—Property Definition: schema:name

Property	schema:name
IRI	https://schema.org/name
JSON name	name
Requirement Level	Recommended
Cardinality	0..1
Domain	core:Entity
Range	xsd:string
Definition	A short, human-readable name for the entity.
Usage Note	Recommended for all entities to improve usability in user interfaces, logs, and administrative tools. Reuses the widely adopted property from Schema.org.

8.2.1.2.4. Property: description

Table 9—Property Definition: schema:description

Property	schema:description
IRI	https://schema.org/description
JSON name	description
Requirement Level	Optional
Cardinality	0..1
Domain	core:Entity
Range	xsd:string
Definition	A detailed explanation of the entity’s purpose or nature.

Table 9—Property Definition: schema:description *(continued)*

Usage Note	Useful for documentation and for providing context to human users or AI agents. Reuses the widely adopted property from Schema.org.

8.2.2. Class: core:Domain

The class core:Domain realizes the DOMAIN concept, defined in the Spatial Web ontology as an ENTITY with identity through time endowed with rights and credentials. It represents a sphere of knowledge, influence, or ACTIVITY.

As a foundational element, core:Domain provides the structure for all entities that have a persistent identity as their defining essence. It serves as a distinct container for people, places, things, and concepts, allowing reality to be described from multiple perspectives and through correlated hierarchies. Domains are hierarchical in nature, allowing for parent, child, and sibling relationships, and can be nested within one another to maintain coherence. They form a holarchy, where each domain is a self-contained whole (a holon) that can also be a constituent part of a larger domain.

- **Identity:** Every core:Domain SHALL possess a Spatial Web Identifier (core:swid).
- **Typing:** Every core:Domain SHALL have a domainType attribute specifying its classification (e.g., Geographic, Concept, Agent, etc.).
- **Authority:** Every core:Domain SHALL have a Domain Authority that governs its norms, terms, and contracts.
- **Spatial Context:** A core:Domain MAY reference a HYPERSPACE to define its location, boundaries, or paths.
- **Holonic Structure:** A core:Domain MAY be part of a parent domain via a holonic (part-whole) link to create nested and hierarchical structures.

8.2.2.1. Class Definition

Table 10—Class Definition for core:Domain

RDF Class	**core:Domain**
Is Abstract	No
Definition	An ENTITY with identity through time endowed with rights and credentials, representing a sphere of knowledge, influence, or ACTIVITY.
Subclass Of	core:Entity
Usage Note	Domains are used to partition the Spatial Web into manageable regions. For example, a “Metropolis Digital Twin” is a geographic Domain, while the “Regional Energy Grid” is a concept Domain. An AGENT is also a type of Domain.
Rationale	Provides a foundational, general-purpose mechanism for grouping entities. This enables the creation of scalable, hierarchical, and nested structures necessary for contextual awareness, governance, and decentralized administration in the Spatial Web.

Table 11—Properties Summary for core:Domain

core:hasSpace	**hasSpace**	**Links the domain to the Hyperspace in which it applies or is located.**	**space: Hyperspace**	**0..***	**Optional**
core: managedBy	managedBy	Identifies the Domain Authority credentialed to define norms and govern the domain.	agent:Agent	1..1	Mandatory

Table 11—Properties Summary for core:Domain *(continued)*

core:hasSpace	**hasSpace**	**Links the domain to the Hyperspace in which it applies or is located.**	**space: Hyperspace**	**0..***	**Optional**
rdfh:partOf	partOf	Establishes a holonic link, indicating this domain is a constituent part of a parent domain.	core:Domain	0..1	Optional

8.2.2.2. Properties

This section provides the detailed normative definitions for the properties of the core:Domain class. (Properties inherited from core:Entity such as core:swid and schema:name are not redefined here).

8.2.2.2.1. Property: domainType

8.2.2.2.2. Property: hasSpace

Table 12—Property Definition: core:hasSpace

Property	core:hasSpace
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#hasSpace
JSON name	hasSpace
Requirement Level	Optional
Cardinality	0..*
Domain	core:Domain
Range	space:Hyperspace
Definition	Associates the domain with a HYPERSPACE, indicating the spatial boundary, location, or context in which the domain's rules and membership apply.
Usage Note	Essential for location-aware governance. For example, a geographic Domain is implicitly or explicitly associated with a location. The HYPERSPACE can be a geometry in Cartesian space or a range of values.

8.2.2.2.3. Property: managedBy

Table 13—Property Definition: core:managedBy

Property	core:managedBy
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#managedBy
JSON name	managedBy
Requirement Level	Mandatory
Cardinality	1..1
Domain	core:Domain
Range	agent:Agent
Definition	Identifies the Domain Authority: an entity credentialed to define the norms and terms under which contracts are created for AGENTS, ACTIVITIES, and CREDENTIALS within that Domain.

Table 13—Property Definition: core:managedBy *(continued)*

Usage Note	Every Spatial Web Domain SHALL have a Domain Authority. The managing agent is responsible for defining domain policies and governing the entities and activities within it.

8.2.2.2.4. Property: partOf

Table 14—Property Definition: rdfh:partOf

Property	rdfh:partOf
IRI	http://purl.org/rdfh/partOf
JSON name	partOf
Requirement Level	Optional
Cardinality	0..1
Domain	core:Domain
Range	core:Domain
Definition	Establishes a holonic or mereological (part-whole) link, indicating this domain is a constituent part of a parent domain.
Usage Note	This property is used to create nested and hierarchical domain structures. It allows for parent, child, and sibling relationships, enabling domains to maintain coherence while being contained within other domains.

8.2.3. Class: core:Concept

The class core:Concept realises the CONCEPT DOMAIN type as described in IEEE P2874 § 6.3.2.1.3:

"A CONCEPT is a DOMAIN representing an abstract, non-physical entity or idea that provides semantic context or categorization for other Entities in the Spatial Web."

core:Concept is used to model abstract or symbolic constructs, classifications, or types that help structure knowledge, rules, or ontological categories in the Spatial Web. It does not represent physical or agentic entities, but rather supports semantic reasoning, categorization, and rule expression.

8.2.3.1. Class Definition

Table 15—Class Definition for core:Concept

RDF Class	**core:Concept**
Is Abstract	No
Definition	A core:Domain representing an abstract idea, category, or semantic construct used for classification and knowledge organization.
Subclass Of	core:Domain
Usage Note	Concepts provide the semantic scaffolding of the Spatial Web; they are not physical or agentic entities.
Rationale	Supports ontology modeling, classification, and reasoning consistent with P2874's Concept Domain type.

8.2.3.2. Properties

The core:Concept class inherits all properties from core:Domain and core:Entity. It introduces no new properties at this level, providing a minimal base class for semantic constructs.

8.2.4. Class: core:Thing

The class core:Thing realises the THING DOMAIN type defined in IEEE P2874 § 6.3.2.1.7:

"A THING is a DOMAIN representing a bounded item without agency that can be acted upon, sensed, or described in the Spatial Web."

core:Thing represents physical or virtual entities that lack autonomous action but may be observed, measured, manipulated, or referenced by Agents or Activities. Examples include devices, sensors, vehicles, digital files, or other discrete objects.

8.2.4.1. Class Definition

Table 16—Class Definition for core:Thing

RDF Class	**core:Thing**
Is Abstract	No
Definition	A passive core:Domain representing a bounded item without agency, which may be sensed, acted upon, or described.
Subclass Of	core:Domain
Disjoint With	agt:Agent
Usage Note	Things are passive entities; they do not possess goals or capabilities but may participate in Activities as targets, inputs, or outputs.
Rationale	Provides a formal category for all non-agentic entities in the Spatial Web, ensuring compliance with P2874's Domain taxonomy.

8.2.4.2. Properties

The core:Thing class inherits all properties from core:Domain and core:Entity. It introduces no new properties at this level, providing a minimal base class that can be extended by more specific standards (such as W3C WoT or SAREF) for detailed device modeling.

8.2.5. Class: core:SpatialFeature

The class core:SpatialFeature realises the SPATIAL FEATURE concept, corresponding to the GEOGRAPHIC DOMAIN type described in IEEE P2874 § 6.3.2.1.2:

"A SPATIAL FEATURE is an ENTITY representing a location-bound or geometry-bearing element within a Domain, corresponding to a GEOGRAPHIC DOMAIN type where position, extent, or spatial relationships are essential."

core:SpatialFeature represents real or abstract entities anchored in space—such as landmarks, boundaries, sensor sites, or natural features—that are meaningful in spatial reasoning and analysis. It is the fundamental class for binding semantic entities to geospatial structure in the Spatial Web.

Key Requirements

– Identity: Every core:SpatialFeature SHALL possess a Spatial Web Identifier (core:swid).

– Geometry: A core:SpatialFeature SHOULD have at least one geometric representation linked via the geosparql:hasGeometry property.

8.2.5.1. Class Definition

Table 17—Class Definition for core:SpatialFeature

RDF Class	core:SpatialFeature
Is Abstract	No
Definition	An entity that represents a location-bound or geometry-bearing element of the GEOGRAPHIC DOMAIN type in the Spatial Web.
Subclass Of	core:Domain
Usage Note	core:SpatialFeature instances link semantic entities to geospatial structures (geometry, topology) for reasoning, mapping, or analysis.
Rationale	Provides a formal mechanism to model GEOGRAPHIC DOMAIN instances (per P2874) and bind entities to spatial coordinates, shapes, and relationships, aligning with established geospatial standards like OGC GeoSPARQL.

Table 18—Properties Summary for core:SpatialFeature

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
geosparql: hasGeometry	hasGeometry	Links the feature to its geometric representation.	geosparql: Geometry	0..*	Recommended

8.2.6. Class: core:Role

The class core:Role represents a named capacity, position, or function that a bearer (e.g., person, agent, device, or service) may hold within some context. It is a **definition-level concept** (the role itself), distinct from any particular assignment or holding of that role.

Key Requirements

– **Typing**: Every instance **SHALL** declare its class via the rdf:type property.

– **Minimal Metadata**: Role definitions **SHOULD** provide a human-readable schema:name and **MAY** include a schema:description.

– **Neutral & Standalone**: core:Role **SHALL NOT** require any superclass from external ontologies and **SHALL** remain independent of assignment-specific data (e.g., time bounds, authority, or basis).

8.2.6.1. Class Definition

Table 19—Class Definition for core:Role

RDF Class	core:Role
Is Abstract	No
Definition	A named capacity, position, or function that can be held by an agent, object, or service within a context.
Subclass Of	owl:Thing
Usage Note	core:Role captures the **definition** of a role. Assignment-specific details (bearer, time, authority, basis) **SHOULD NOT** be modeled on this class and belong in a separate assignment pattern if needed.
Rationale	Provides a lightweight, interoperable notion of “role” that can be reused across activities, memberships, stewardship, and other Spatial Web contexts without imposing external ontology dependencies.

Table 20—Properties Summary for core:Role

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
rdf:type	@type	Declares the specific class of the resource.	rdfs:Class	1..*	**Mandatory**
schema:name	name	A short, human-readable name for the role.	xsd:string	0..1	Recommended
schema: description	description	A human-readable explanation of the role's purpose or scope.	xsd:string	0..1	Optional

8.2.6.2. Properties

This section provides the detailed normative definitions for the properties of the core:Role class.

8.2.6.2.1. Property: type

Table 21—Property Definition: rdf:type

Property	rdf:type
IRI	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
JSON name	@type
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	core:Role
Range	rdfs:Class
Definition	Declares the specific class (or classes) of which this resource is an instance.
Usage Note	Every role **SHALL** have at least one rdf:type equal to core:Role. Additional types (e.g., domain-specific categories) MAY be included.

8.2.6.2.2. Property: name

Table 22—Property Definition: schema:name

Property	schema:name
IRI	https://schema.org/name
JSON name	name
Requirement Level	Recommended
Cardinality	0..1
Domain	core:Role
Range	xsd:string
Definition	A short, human-readable name for the role.
Usage Note	Recommended for UI, documentation, and discoverability. Reuses widely adopted Schema.org semantics.

8.2.6.2.3. Property: description

Table 23—Property Definition: schema:description

Property	schema:description

Table 23—Property Definition: schema:description *(continued)*

IRI	https://schema.org/description
JSON name	description
Requirement Level	Optional
Cardinality	0..1
Domain	core:Role
Range	xsd:string
Definition	A human-readable explanation of the role's purpose, responsibilities, or scope.
Usage Note	Useful to clarify intent and distinguish similarly named roles in catalogs and governance artifacts.

8.2.7. Class: core:Participation

The class core:Participation is the generic link object that models an entity's involvement in a host resource (e.g., a gov:Contract, ev:Event, or act:Activity). It is a reified relationship that states:

– What entity is involved (via core:hasParticipant),

– What role it plays (via core:hasRole), and

– Optionally, the duration of the involvement (via schema:startTime and schema:endTime).

This pattern is fundamental in the Spatial Web because it provides a standardized, reusable mechanism for describing and querying participation across heterogeneous contexts. It prevents model duplication and ensures that agentic, legal, and procedural interactions are captured consistently—critical for maintaining trust, accountability, and semantic precision.

8.2.7.1. Class Definition

Table 24—Class Definition for core:Participation

RDF Class	**core:Participation**
Is Abstract	No
Definition	A reified relationship that captures an entity's specific involvement in a host resource, connecting the entity and its role to the context.
Subclass Of	owl:Thing
Usage Note	Host resources (like gov:Contract) SHOULD use a property such as gov: hasParticipation to link to one or more instances of core:Participation.
Rationale	Centralizes the {entity, role, time} pattern for maximum reuse. This single, robust pattern can describe the involvement of agents, artifacts, or other entities in any situation.

Table 25—Properties Summary for core:Participation

Predicate	**JSON-LD name**	**Description**	**Range**	**Cardinality**	**Requirement Level**
core: hasParticipant	hasParticipant	The entity playing the role.	hsml:Entity	1..1	Mandatory
core:hasRole	hasRole	The core:Role that describes the function of the participant.	core:Role	1..1	Mandatory
schema: startTime	startTime	The timestamp when the participation begins.	xsd:dateTime	0..1	Optional

Table 25—Properties Summary for core:Participation *(continued)*

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
schema:endTime	endTime	The timestamp when the participation ends.	xsd:dateTime	0..1	Optional

8.2.7.2. Properties

This section provides the detailed normative definitions for the properties of the core:Participation class.

8.2.7.2.1. Property: hasParticipant

Table 26—Property Definition: core:hasParticipant

Property	core:hasParticipant
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#hasParticipant
JSON name	hasParticipant
Requirement Level	Mandatory
Cardinality	1..1
Range	hsml:Entity
Definition	The actual resource (agent, object, etc.) that is involved in the context.
Usage Note	This property links the core:Participation instance to the participating entity itself.

8.2.7.2.2. Property: hasRole

Table 27—Property Definition: core:hasRole

Property	core:hasRole
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#hasRole
JSON name	hasRole
Requirement Level	Mandatory
Cardinality	1..1
Range	core:Role
Definition	The role the participant entity fulfills within the participation (e.g., client, sensor, certifier).
Usage Note	The value should be an IRI pointing to a concept in a controlled vocabulary of roles.

8.2.7.2.3. Property: startTime

Table 28—Property Definition: schema:startTime

Property	schema:startTime
IRI	https://schema.org/startTime
JSON name	startTime
Requirement Level	Optional
Cardinality	0..1
Range	xsd:dateTime

Table 28—Property Definition: schema:startTime *(continued)*

Definition	The specific date and time when the participation begins or becomes effective.
Usage Note	Reuses the property from Schema.org for standardized temporal metadata.

8.2.7.2.4. Property: endTime

Table 29—Property Definition: schema:endTime

Property	schema:endTime
IRI	https://schema.org/endTime
JSON name	endTime
Requirement Level	Optional
Cardinality	0..1
Range	xsd:dateTime
Definition	The specific date and time when the participation ends or is no longer effective.
Usage Note	Reuses the property from Schema.org for standardized temporal metadata.

8.2.8. Class: core:Condition

A core:Condition is an abstract class representing a logical condition that can be evaluated to be true or false.

8.2.8.1. Class Definition

Table 30—Class Definition for core:Condition

RDF Class	**core:Condition**
Is Abstract	Yes
Definition	An abstract representation of a logical condition that must be satisfied.
Subclass Of	owl:Thing
Usage Note	This class is never instantiated directly. Concrete subclasses like core:SHACLCondition must be used to provide the specific evaluation logic.
Rationale	Provides a single, abstract root for all types of logical validation, ensuring that the ontology can evolve to support new validation technologies without breaking changes.

8.2.9. Class: core:SHACLCondition

The class core:SHACLCondition is a concrete implementation of core:Condition that uses a SHACL shape for validation. It provides a standardized way to express constraints on the structure and values of RDF data.

As a foundational element of the Generic Condition Model, core:SHACLCondition is the primary mechanism for defining data types, cardinality, value ranges, and other structural rules for entities and variables throughout the HSML ontology.

Inheritance: Every core:SHACLCondition is a subclass of core:Condition.

Shape Definition: Every core:SHACLCondition SHALL link to exactly one sh:Shape via the core:hasShape property.

Validation Logic: The validation of a core:SHACLCondition is performed by evaluating the associated SHACL shape against a target RDF graph.

8.2.9.1. Class Definition

Table 31—Class Definition for core:SHACLCondition

RDF Class	**core:SHACLCondition**
Is Abstract	No
Definition	A condition defined by a SHACL shape, used to validate the structure and values of RDF data.
Subclass Of	core:Condition
Usage Note	This is the primary mechanism for defining data type, cardinality, and other structural constraints on variables and entities.
Rationale	Leverages the W3C SHACL standard for robust, standardized data validation, ensuring interoperability and machine-checkable conformance.

Table 32—Properties Summary for core:SHACLCondition

Predicate	**JSON-LD name**	**Description**	**Range**	**Cardinality**	**Requirement Level**
core:hasShape	hasShape	Links the condition to the SHACL shape defining its logic.	sh:Shape	1..1	Mandatory

8.2.9.2. Properties

This section provides the detailed normative definitions for the properties of the core:SHACLCondition class.

8.2.9.2.1. Property: hasShape

Table 33—Property Definition: core:hasShape

Property	core:hasShape
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#hasShape
JSON name	hasShape
Requirement Level	Mandatory
Cardinality	1..1
Domain	core:SHACLCondition
Range	sh:Shape
Definition	Links a core:SHACLCondition to the specific SHACL shape that defines its validation logic.
Usage Note	The object of this property must be a valid SHACL shape definition. This shape is used by validation engines to evaluate the condition.

9. HSML Activity Module

9.1. Architectural Principles and Design Rationale

This clause describes the architectural principles underlying the HSML Activity Module. These principles are *informative* and not required for conformance; however, understanding them is recommended to ensure robust and correct implementations.

9.1.1. The Schema/Instance Pattern: Enabling Scalable Governance

A fundamental principle of this standard is the strict separation between the *definition* of an activity and the *record of its execution*. This separation is realized through two distinct HSML classes:

– act:ActivitySchema — A reusable, design-time template that specifies the type of action, its required inputs, expected outputs, and the conditions under which it is valid.

– act:Activity — An execution-time entity representing a single, concrete, stateful instance of an act:ActivitySchema performed by a specific agent at a specific time.

This separation is a cornerstone of governance and scalability within the Spatial Web. It enables a "trust-by-reference" model where an orchestration engine can validate an action with high efficiency. Rather than analyzing the internal logic of the action itself, the validator only needs to check whether the act:Activity references an act:ActivitySchema from an approved registry. This decouples the high-level, infrequent work of policy definition (approving schemas) from the high-frequency, operational work of validation (checking activities), producing a highly scalable and secure governance framework.

9.1.2. The Parameter/Binding Pattern: Achieving Contextual Reusability

Complementing the schema/instance pattern is the separation of abstract parameters from their concrete values. This is achieved through the act:Variable and act:VariableBinding classes.

– act:Variable is used within an act:ActivitySchema to define a placeholder for an input or output (e.g., "target_location"), including its name, description, and constraints, but no specific value.

– act:VariableBinding is used within an act:Activity to connect a specific act:Variable from the schema to a concrete value (e.g., linking "target_location" to a specific address).

This design allows act:ActivitySchema definitions to be generic and broadly reusable. A single schema for "TransferAsset" can be instantiated in thousands of different act:Activity instances, each using act:VariableBinding to supply the concrete assets and accounts for a specific transaction. This promotes standardization while ensuring every act:Activity is a complete, self-contained, and auditable record.

A key architectural decision in this pattern is how a VariableBinding references its corresponding Variable. This model mandates the use of the Variable's locally unique `schema:identifier (e.g., "targetLocation") rather than its global URI. This choice is deliberate and provides two main benefits:

a) **Consistency**: It establishes a single, predictable pattern across the entire Activity model. Just as act:DataLink uses local identifiers to wire steps together within a composite schema, act:VariableBinding uses the same mechanism to bind values at runtime. This consistency simplifies implementation and makes the model easier to learn.

b) **Portability**: It treats the ActivitySchema as a true, self-contained template. An Activity instance is bound to the *role* a variable plays within the schema (its local identifier), not to a rigid, globally

unique URI. This decouples the execution record from the schema definition, allowing the schema to be versioned or reused more flexibly without invalidating existing activity instances.

This approach requires the processing engine to resolve the reference by looking up the identifier within the context of the activity's schema. This is a standard pattern and a worthwhile trade-off for the significant gains in consistency, portability, and developer experience.

9.1.3. The Composite Model: First-Class Workflows

For composite (multi-step) activities, a simple list of sub-activities is insufficient. A robust model must also describe the **sequence of steps**, the **flow of data** between them, and the **control flow logic** (e.g., conditional branches). The Activity Module addresses this by elevating the core components of a workflow to be first-class citizens:

– act:ActivityStep: A node in the workflow graph. It represents a discrete stage of execution and is a rich, self-describing object.

– act:DataLink: An edge in the workflow graph. It represents the explicit flow of data from one step's output to another's input.

The decision to model act:ActivityStep and act:DataLink as distinct owl:Class instances, rather than simple properties, is a cornerstone of this architecture. It makes the entire workflow—its structure, logic, and data flow—explicit, addressable, and governable.

This design allows for sophisticated patterns: * **Documentable and Addressable Steps**: Each act:ActivityStep can have a human-readable schema:name and schema:description, making the workflow self-documenting. It is given a simple, local schema:identifier (e.g., "validate-user") for easy reference by act:DataLink instances. * **Conditional Control Flow**: An act:ActivityStep can carry its own core:Condition. This enables powerful, declarative control flow, such as an "if/then/else" branch, where the execution of a step depends on the outcome of a formal condition. * **Governable and Self-Describing Data Flow**: Because a act:DataLink is a first-class entity, it can be documented with its own schema:name and schema:description (e.g., "Pass User ID to Validator"). Furthermore, metadata can be attached directly to the data transfer itself, allowing for quality-of-service requirements (e.g., ex:maxLatency), security classifications, or even a core:Condition to enforce a policy on the data link.

This approach transforms a simple sequence of tasks into a fully governable, introspectable, and self-describing graph of actions that can be validated, audited, and reused with high fidelity.

9.1.4. The Generic Condition Model: Future-Proofing Logic

The HSML architecture separates semantic intent from technical implementation. This is exemplified by the core:Condition model, defined in the HSML Core Module and used extensively by this Activity Module for properties like act:hasPrecondition, act:hasEffect, and for enabling conditional logic on an act:ActivityStep.

The core:Condition class is abstract; it represents the idea that a condition exists without specifying *how* that condition must be evaluated. The actual evaluation logic is delegated to concrete subclasses, such as core:SHACLCondition or core:SPARQLCondition. This architectural choice is a powerful future-proofing mechanism. By having models reference the abstract core:Condition, the standard is not locked into any single validation technology. New condition types can be introduced in the future without requiring breaking changes to the core models, ensuring the long-term relevance and extensibility of the standard.

9.1.5. Domain-Specific Governance of Referenced Schemas

A core tenet of the Spatial Web's architecture is that domains can apply **local, context-specific governance** to universal, standardized act:ActivitySchema definitions without modifying the original schema. This is

achieved by separating the activity's definition from the rules governing its execution within a domain. The primary mechanism for enforcing these domain-specific rules is the requirement of a domain-specific **CREDENTIAL**.

A Domain Authority (DA) can incorporate a universal schema by reference and then configure its domain policies to require that any agent performing the activity must present a specific credential. This model ensures that while the schema remains a universal standard, its execution is always subject to the explicit, verifiable, and non-negotiable rules of the domain in which it is performed.

9.2. Modeling of Composite Activities

This clause specifies the normative rules for constructing complex, multi-step workflows using the act: CompositeActivitySchema class.

9.2.1. The Composite Pattern: ActivityStep and DataLink

A CompositeActivitySchema shall be composed of one or more act:ActivityStep instances and zero or more act:DataLink instances.

An act:ActivityStep shall function as an addressable node within the workflow graph defined by the composite schema. Each act:ActivityStep shall use the act:usesSchema property to reference exactly one act: ActivitySchema that defines the logic for that step. This referenced schema may be either atomic or another composite schema, allowing for nested workflows.

An act:DataLink shall function as a directed edge in the workflow graph, representing the flow of data. This formal "wiring" is necessary for composite logic, as it makes the dependencies between steps explicit and machine-checkable.

9.2.2. Control Flow Constructs

An act:CompositeActivitySchema shall define its internal control flow by using exactly one of the following three properties: act:hasOrderedSteps, act:hasChoice, or act:hasUnorderedSteps. This mutual exclusivity is normatively enforced by the sh:xone constraint in the act:CompositeActivitySchemaShape (see Annex B).

– **Sequence (act:hasOrderedSteps)**: If this property is used, its value shall be a well-formed rdf:List where each member of the list is an act:ActivityStep. The steps shall be executed by an orchestration engine in the strict order defined by the list.

– **Choice (act:hasChoice)**: If this property is used, it shall link to two or more act:ActivityStep instances. These steps represent mutually exclusive execution paths. An orchestration engine shall select and execute exactly one of the specified choice steps.

– **Set (act:hasUnorderedSteps)**: If this property is used, it shall link to one or more act:ActivityStep instances. These steps have no prescribed execution order and may be executed in any sequence or in parallel, subject to data dependencies defined by act:DataLink instances.

9.2.3. Interface Wiring and Data Flow Management

The act:DataLink class is used to manage the flow of data between steps and across the boundary of the composite schema. The following normative rules apply, as enforced by the act:DataLinkInterfaceShape and act:DataLinkTypeConsistencyShape (see Annex B).

- **Internal Wiring**: To connect the output of one step to the input of another, an act:DataLink shall specify the act:sourceStep, act:sourceVariable, act:targetStep, and act:targetVariable.
- **Input Wiring**: To connect a public input of the CompositeActivitySchema to the input of an internal step, an act:DataLink shall be created that omits the act:sourceStep property. For such a link, the act:sourceVariable shall be a variable that is declared as an act:hasInput on the parent CompositeActivitySchema.
- **Output Wiring**: To expose the output of an internal step as a public output of the CompositeActivitySchema, an act:DataLink shall be created that omits the act:targetStep property. For such a link, the act:targetVariable shall be a variable that is declared as an act:hasOutput on the parent CompositeActivitySchema.
- **Type Consistency**: For any act:DataLink, the type constraint of the act:sourceVariable should be compatible with the type constraint of the act:targetVariable. The act:DataLinkTypeConsistencyShape provides a normative check for this compatibility.

9.3. Execution and Traceability Model

This clause specifies the normative requirements for creating a run-time trace of an executed activity. This model ensures that a complete and auditable record of all actions is preserved.

9.3.1. Instantiation of Composite Activities

When an orchestration engine begins the execution of a CompositeActivitySchema, it shall first create a parent act:Activity instance corresponding to the composite schema itself. As the engine proceeds to execute each constituent act:ActivityStep defined within the composite, it shall create a new, distinct child act:Activity instance for each step executed.

9.3.2. Normative Requirements for Traceability Links

To ensure a complete and unambiguous provenance graph, the following linking properties shall be used on all child act:Activity instances created during the execution of a composite activity.

- **Compositional Link (act:subActivityOf)**: Every child Activity created from an ActivityStep shall be linked to its parent composite Activity using exactly one act:subActivityOf property. This creates the compositional hierarchy, clearly defining which actions are part of a larger workflow.
- **Sequential Link (act:precededBy)**: For workflows defined with act:hasOrderedSteps, every child Activity (except for the first one in the sequence) shall be linked to the Activity instance of the immediately preceding step using exactly one act:precededBy property.

The combined use of these two properties creates a comprehensive, dual-axis traceability graph. The act:subActivityOf links provide the compositional or “part-of” hierarchy, while the act:precededBy links provide the temporal and causal sequence. Together, they allow for the complete and unambiguous reconstruction of a complex event, which is critical for auditing, debugging, and establishing legal or operational accountability.

9.4. Normative Classes

This clause provides the normative definitions for all class concepts within the HSML Activity Module. Each class is detailed in a table that specifies its URI, description, JSON-LD context name, usage notes, and relationship to other classes.

The namespace prefix act: refers to https://www.spatialwebfoundation.org/ns/hsml/activity#.

9.4.1. Summary of Normative Classes

Table 34—Summary of HSML Activity Module Classes

Class	Description
act:Activity	A concrete, stateful record of a specific action that has been, is being, or is planned to be performed.
act:ActivitySchema	A reusable, design-time template that defines the logic, interface, and conditions for a type of activity.
act:AtomicActivitySchema	A specialization of act:ActivitySchema for simple, indivisible activities whose internal logic is opaque to the model.
act: CompositeActivitySchema	A specialization of act:ActivitySchema for complex workflows composed of multiple steps and data links.
act:ActivityStep	A structural node within a act:CompositeActivitySchema that represents a single step in a workflow.
act:DataLink	A structural entity that explicitly defines the "wiring" or flow of data between steps in a composite workflow.
act:Variable	An abstract parameter placeholder within an act:ActivitySchema that defines an input or output slot.
act:VariableBinding	A concrete link within an act:Activity that connects an act:Variable to a specific value for an execution.

9.4.2. Class: act:Activity

The class act:Activity realises the **ACTIVITY** concept defined in IEEE P2874 § 6.6.3:

"An ACTIVITY is an execution-time entity representing a single, concrete, stateful instance of an act: ActivitySchema performed by a specific agent at a specific time."

act:Activity is the primary entity for recording and tracking the execution of an action in the Spatial Web. It serves as a stateful, auditable record that links the abstract act:ActivitySchema to the concrete agents and values involved in a specific occurrence.

Key Requirements

- **Identity**: Every act:Activity **SHALL** possess a Spatial Web Identifier (hsml:swid, W3C DID Core-compliant).
- **Schema Conformance**: An act:Activity **SHALL** reference exactly one act:ActivitySchema via the act: activitySchema property.
- **Agent Assignment**: An act:Activity **SHALL** identify the hsml:Agent(s) responsible for its execution via the act:performedBy property.
- **Parameter Binding**: An act:Activity **SHALL** provide concrete values for all required inputs of its schema using act:VariableBinding instances.
- **Lifecycle Tracking**: An act:Activity **SHALL** have its current state tracked via the act:status property, using the normative HSML Activity Status vocabulary.
- **Temporality**: An act:Activity **SHOULD** record the time execution began (schema:startTime) and concluded (schema:endTime).
- **Execution Traceability**: If an act:Activity is part of a composite workflow, it **SHOULD** link to its parent activity (dct:isPartOf) and the step it realized (act:realizedStep).

These requirements ensure that every executed action is a complete, self-contained, and verifiable record that is traceable to its defining schema and responsible agent.

9.4.2.1. Class Definition

Table 35—Class Definition for act:Activity

RDF Class	**act:Activity**
Is Abstract	No
Definition	A concrete, stateful execution of an act:ActivitySchema. It is the record of an action that has been, is being, or is planned to be performed.
Subclass Of	hsml:Entity
Usage Note	An act:Activity instance is created for every action that is initiated via a gov:Contract. It contains act:VariableBinding instances for all required inputs and its act:status property tracks its lifecycle.
Rationale	Provides the normative record of execution for all actions, ensuring that every event is auditable, governable, and traceable to its formal definition and responsible parties.

Table 36—Properties Summary for act:Activity

Predicate	**JSON-LD name**	**Description**	**Range**	**Cardinality**	**Requirement Level**
act: activitySchema	activitySchema	Links the act:Activity instance to the act:ActivitySchema it implements.	act: ActivitySchema	1..1	**Mandatory**
act: performedBy	performedBy	Identifies the hsml:Agent(s) responsible for performing the act: Activity.	hsml:Agent	1..*	**Mandatory**
act:status	status	The current lifecycle state of the act:Activity.	skos:Concept	1..1	**Mandatory**
act:hasBinding	hasBinding	Associates an act:VariableBinding with the act:Activity.	act: VariableBinding	0..*	Conditional
schema: startTime	startTime	The timestamp when the activity execution began.	xsd:dateTime	0..1	Recommended
schema: endTime	endTime	The timestamp when the activity execution concluded (completed or failed).	xsd:dateTime	0..1	Recommended
dct:isPartOf	isPartOf	Links this activity instance to its parent composite activity instance.	act:Activity	0..1	Optional
act:realizedStep	realizedStep	Identifies the specific act: ActivityStep that this instance realized.	act:ActivityStep	0..1	Optional

9.4.2.2. Properties

This section provides the detailed normative definitions for the properties of the act:Activity class.

9.4.2.2.1. Property: activitySchema

Table 37—Property Definition: act:activitySchema

Property	act:activitySchema
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#activitySchema

Table 37—Property Definition: act:activitySchema *(continued)*

JSON name	activitySchema
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:Activity
Range	act:ActivitySchema
Definition	Links an act:Activity instance to the act:ActivitySchema it implements.
Usage Note	This is a critical link for governance, as it allows validators to check the activity's conformance against its approved template.

9.4.2.2.2. Property: performedBy

Table 38—Property Definition: act:performedBy

Property	act:performedBy
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#performedBy
JSON name	performedBy
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	act:Activity
Range	hsml:Agent
Definition	Identifies the hsml:Agent (or agents) responsible for performing an act:Activity.
Usage Note	Establishes accountability for the action's execution and outcomes.

9.4.2.2.3. Property: status

Table 39—Property Definition: act:status

Property	act:status
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#status
JSON name	status
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:Activity
Range	skos:Concept
Definition	The current lifecycle state of an act:Activity.
Usage Note	The value **SHALL** be an IRI from the normative HSML Activity Status vocabulary (e.g., act:Planned, act:Ongoing, act:Completed, act:Failed). Provides real-time tracking of the action's progress and is used to determine the status of the governing gov:Contract.

9.4.2.2.4. Property: hasBinding

Table 40—Property Definition: act:hasBinding

Property	act:hasBinding
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasBinding
JSON name	hasBinding
Requirement Level	Conditional

Table 40—Property Definition: act:hasBinding *(continued)*

Cardinality	0..*
Domain	act:Activity
Range	act:VariableBinding
Definition	Associates an act:VariableBinding with an act:Activity, indicating that a variable has been bound to a value.
Usage Note	An act:Activity **should** have a binding for each required input (act:hasInput) of its schema.

9.4.2.2.5. Property: startTime

Table 41—Property Definition: schema:startTime

Property	schema:startTime
IRI	https://schema.org/startTime
JSON name	startTime
Requirement Level	Recommended
Cardinality	0..1
Domain	act:Activity
Range	xsd:dateTime
Definition	The specific date and time when the activity execution began.
Usage Note	Essential for temporal reasoning and auditing. Should be populated when the status transitions to act:Ongoing. Reuses the property from Schema.org.

9.4.2.2.6. Property: endTime

Table 42—Property Definition: schema:endTime

Property	schema:endTime
IRI	https://schema.org/endTime
JSON name	endTime
Requirement Level	Recommended
Cardinality	0..1
Domain	act:Activity
Range	xsd:dateTime
Definition	The specific date and time when the activity execution concluded (completed or failed).
Usage Note	Essential for temporal reasoning and auditing. Should be populated when the status transitions to a terminal state (act:Completed or act:Failed). Reuses the property from Schema.org.

9.4.2.2.7. Property: isPartOf

Table 43—Property Definition: dct:isPartOf

Property	dct:isPartOf
IRI	http://purl.org/dc/terms/isPartOf
JSON name	isPartOf
Requirement Level	Optional
Cardinality	0..1

Table 43—Property Definition: dct:isPartOf *(continued)*

Domain	act:Activity
Range	act:Activity
Definition	Links a child activity instance to its parent composite activity instance.
Usage Note	Used for execution traceability in composite workflows. If populated, the range must be an instance of an activity whose schema is an act: CompositeActivitySchema.

9.4.2.2.8. Property: realizedStep

Table 44—Property Definition: act:realizedStep

Property	act:realizedStep
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#realizedStep
JSON name	realizedStep
Requirement Level	Optional
Cardinality	0..1
Domain	act:Activity
Range	act:ActivityStep
Definition	Identifies the specific act:ActivityStep within the parent composite schema that this activity instance realized.
Usage Note	Provides a precise link between the execution trace and the workflow definition. Should be used in conjunction with dct:isPartOf.

9.4.3. Class: act:ActivitySchema

The class act:ActivitySchema realises the ACTIVITY SCHEMA concept defined in IEEE P2874 § 6.6.3:

"An ACTIVITY SCHEMA is an ENTITY that defines the template, structure, or plan for an ACTIVITY, specifying its preconditions, postconditions, roles, and expected outcomes."

act:ActivitySchema provides the abstract, reusable blueprint for an act:Activity. It enables the consistent modeling of repeatable processes by defining the action's interface (inputs/outputs), its required logical state (preconditions), and its expected results (effects).

Key Requirements

– Identity: Every act:ActivitySchema SHALL possess a Spatial Web Identifier (hsml:swid, W3C DID Core-compliant).

– Condition Specification: An act:ActivitySchema SHOULD define its required state and results using act:hasPrecondition and act:hasEffect.

– Parameterization: An act:ActivitySchema SHALL declare its interface of expected parameters using act:hasInput and act:hasOutput, which reference act:Variable instances.

These requirements ensure that activities based on schemas are machine-checkable, consistent, and reusable in Spatial Web systems.

9.4.3.1. Class Definition

Table 45—Class Definition for act:ActivitySchema

RDF Class	act:ActivitySchema
Is Abstract	No
Definition	A reusable, design-time template for a type of activity, defining its logic, interface (inputs and outputs), preconditions, and effects.
Subclass Of	hsml:Entity
Usage Note	Schemas are published and reused across multiple act:Activity instances. They are the foundation of the "trust-by-reference" governance model.
Rationale	Provides a formal mechanism for modeling reusable processes and workflows in compliance with the P2874 Activity model, separating the definition of an action from its execution.

9.4.3.2. Property Definitions

This section provides the detailed normative definitions for the properties of the act:ActivitySchema class.

Table 46—Properties Summary for act:ActivitySchema

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
rdf:type	@type	Declares the resource as an instance of act:ActivitySchema.	rdfs:Class	1..n	Mandatory
hsml:swid	swid	Spatial Web Identifier conformant with W3C DID Core.	xsd:anyURI	1..1	Mandatory
act:hasInput	hasInput	Declares an act:Variable as a required input parameter.	act:Variable	0..n	Optional
act:hasOutput	"hasOutput"	Declares an act:Variable as an expected output parameter.	act:Variable	0..n	Optional
act:hasPrecondition	"hasPrecondition"	Specifies a logical condition that must be satisfied before the activity can be executed.	core:Condition	0..n	Optional
act:hasEffect	"hasEffect"	Describes the expected state of the world after the successful completion of the activity.	core:Condition	0..n	Optional

9.4.3.2.1. Property: hasInput

Table 47—Property Definition: act:hasInput

Property	act:hasInput
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasInput
JSON name	"hasInput"
Requirement Level	Optional
Cardinality	0..n
Domain	act:ActivitySchema
Range	act:Variable
Definition	Declares an input parameter required by the Activity Schema.

Table 47—Property Definition: act:hasInput *(continued)*

Usage Note	Defines the "interface" for the schema. Each input act:Variable must be bound to a value in an act:Activity instance.

9.4.3.2.2. Property: hasOutput

Table 48—Property Definition: act:hasOutput

Property	act:hasOutput
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasOutput
JSON name	"hasOutput"
Requirement Level	Optional
Cardinality	0..n
Domain	act:ActivitySchema
Range	act:Variable
Definition	Declares an output parameter produced by the Activity.
Usage Note	Defines the "interface" for the schema. Each output act:Variable will be bound to a resulting value in a completed act:Activity instance.

9.4.3.2.3. Property: hasPrecondition

Table 49—Property Definition: act:hasPrecondition

Property	act:hasPrecondition
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasPrecondition
JSON name	"hasPrecondition"
Requirement Level	Optional hasPrecondition
Requirement Level	Optional
Cardinality	0..n
Domain	act:ActivitySchema
Range	core:Condition
Definition	Specifies a logical condition on the world state that must be satisfied before the activity can be executed.
Usage Note	Used by governance engines to validate a gov:Contract before allowing an activity to proceed.

9.4.3.2.4. Property: hasEffect

Table 50—Property Definition: act:hasEffect

Property	act:hasEffect`
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasEffect
JSON name	"hasEffect"
Requirement Level	Optional
Cardinality	0..n
Domain	act:ActivitySchema
Range	core:Condition
Definition	Describes the expected state of the world after the successful completion of the activity.

Table 50—Property Definition: act:hasEffect *(continued)*

Usage Note	Used for validation of outcomes and success criteria.

9.4.4. Class: act:AtomicActivitySchema

The class act:AtomicActivitySchema realizes the **ATOMIC ACTIVITY SCHEMA** concept, a specialization of act:ActivitySchema as defined in IEEE P2874 § 6.6.3.

An act:AtomicActivitySchema is a schema for a simple, indivisible activity that is not broken down into smaller steps within the HSML model. It represents a fundamental operation whose internal logic is treated as a "black box."

Key Requirements

– **Inheritance**: An act:AtomicActivitySchema **SHALL** inherit all requirements from act:ActivitySchema.

– **Indivisibility**: An act:AtomicActivitySchema **SHALL NOT** contain any act:ActivityStep instances or control flow properties (act:hasOrderedSteps, act:hasChoice, act:hasUnorderedSteps).

9.4.4.1. Class Definition

Table 51—Class Definition for act:AtomicActivitySchema

RDF Class	**act:AtomicActivitySchema**
Is Abstract	No
Definition	A schema for a simple, indivisible activity that does not internally reference other act:ActivitySchema instances.
Subclass Of	act:ActivitySchema
Usage Note	Used for fundamental operations whose internal logic is opaque to the HSML model (e.g., invoking an external API, performing a cryptographic operation).
Rationale	Provides a clear distinction between simple, black-box tasks and complex, multi-step workflows (act:CompositeActivitySchema), which simplifies validation and execution logic.

9.4.4.2. Properties

An act:AtomicActivitySchema defines no new properties. It inherits all of its properties, including act:hasInput, act:hasOutput, act:hasPrecondition, and act:hasEffect, from its superclass, act:ActivitySchema.

9.4.5. Class: act:CompositeActivitySchema

The class act:CompositeActivitySchema realizes the **COMPOSITE ACTIVITY SCHEMA** concept, which extends the act:ActivitySchema to model complex, multi-step workflows as defined in IEEE P2874 § 6.6.3.

A act:CompositeActivitySchema is a schema for a complex workflow composed of other activities. It acts as a container for a set of act:ActivityStep instances and defines the control and data flow between them. This enables the modeling of sophisticated processes like sequences, choices, and parallel operations.

Key Requirements

– **Composition**: A act:CompositeActivitySchema **SHALL** be composed of one or more act:ActivityStep instances linked via act:hasStep.

– **Control Flow**: A act:CompositeActivitySchema **SHALL** define its execution logic using exactly one of the control flow properties: act:hasOrderedSteps, act:hasChoice, or act:hasUnorderedSteps.

– **Data Flow**: A act:CompositeActivitySchema **MAY** define the data wiring between its steps and its public interface using act:DataLink instances.

9.4.5.1. Class Definition

Table 52—Class Definition for act:CompositeActivitySchema

RDF Class	**act:CompositeActivitySchema**
Is Abstract	No
Definition	A schema for a complex workflow defined in terms of a sequence or combination of other activities.
Subclass Of	act:ActivitySchema
Usage Note	This class is the basis for modeling multi-step processes. It must contain act:ActivityStep instances and define a control flow.
Rationale	Provides a formal, governable structure for modeling complex workflows, making both the process logic and data dependencies explicit and auditable.

Table 53—Properties Summary for act:CompositeActivitySchema

Predicate	**JSON-LD name**	**Description**	**Range**	**Cardinality**	**Requirement Level**
act:hasStep	hasStep	Connects the composite schema to a constituent act:ActivityStep.	act:ActivityStep	1..*	**Mandatory**
act:hasDataLink	hasDataLink	Connects the composite schema to a act:DataLink that defines data wiring.	act:DataLink	0..*	Optional
act:hasOrderedSteps	hasOrderedSteps	Defines a strict sequence of `act:ActivityStep`s.	rdf:List	0..1	Conditional
act:hasChoice	hasChoice	Defines a set of mutually exclusive act:ActivityStep options.	act:ActivityStep	0..*	Conditional
act:hasUnorderedSteps	hasUnorderedSteps	Defines a set of `act:ActivityStep`s with no prescribed execution order.	act:ActivityStep	0..*	Conditional

9.4.5.2. Properties

This section provides the detailed normative definitions for the properties of the act:CompositeActivitySchema class.

9.4.5.2.1. Property: hasStep

Table 54—Property Definition: act:hasStep

Property	act:hasStep
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasStep
JSON name	hasStep
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	act:CompositeActivitySchema

Table 54—Property Definition: act:hasStep *(continued)*

Range	act:ActivityStep
Definition	Connects a composite schema to a constituent act:ActivityStep.
Usage Note	Every composite schema must contain at least one step.

9.4.5.2.2. Property: hasDataLink

Table 55—Property Definition: act:hasDataLink

Property	act:hasDataLink
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasDataLink
JSON name	hasDataLink
Requirement Level	Optional
Cardinality	0..*
Domain	act:CompositeActivitySchema
Range	act:DataLink
Definition	Connects a composite schema to a act:DataLink that defines data wiring.
Usage Note	Used to make the data dependencies between steps explicit.

9.4.5.2.3. Property: hasOrderedSteps

Table 56—Property Definition: act:hasOrderedSteps

Property	act:hasOrderedSteps
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasOrderedSteps
JSON name	hasOrderedSteps
Requirement Level	Conditional
Cardinality	0..1
Domain	act:CompositeActivitySchema
Range	rdf:List
Definition	Defines a strict sequence of act:ActivityStep`s. The value **shall** be a well-formed `rdf:List.
Usage Note	A composite schema **shall** use exactly one control flow property. This property defines a sequential workflow.

9.4.5.2.4. Property: hasChoice

Table 57—Property Definition: act:hasChoice

Property	act:hasChoice
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasChoice
JSON name	hasChoice
Requirement Level	Conditional
Cardinality	0..*
Domain	act:CompositeActivitySchema
Range	act:ActivityStep
Definition	Defines a set of mutually exclusive act:ActivityStep options.
Usage Note	A composite schema **shall** use exactly one control flow property. If used, this property **shall** link to two or more steps.

9.4.5.2.5. Property: hasUnorderedSteps

Table 58—Property Definition: act:hasUnorderedSteps

Property	act:hasUnorderedSteps
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasUnorderedSteps
JSON name	hasUnorderedSteps
Requirement Level	Conditional
Cardinality	0..*
Domain	act:CompositeActivitySchema
Range	act:ActivityStep
Definition	Defines a set of `act:ActivityStep`s with no prescribed execution order.
Usage Note	A composite schema **shall** use exactly one control flow property. This defines a set of steps that may be executed in parallel, subject to data dependencies.

9.4.6. Class: act:ActivityStep

The class act:ActivityStep realizes the **ACTIVITY STEP** concept as part of a act:CompositeActivitySchema in IEEE P2874 § 6.6.3.

"An ACTIVITY STEP is an addressable node within a composite activity's workflow, representing a discrete stage of execution that uses a specific act:ActivitySchema to define its logic."

act:ActivityStep is a structural component used to build complex workflows. It acts as a placeholder or node in a workflow graph, separating the structure of the workflow from the logic of the individual steps. This allows for the creation of modular, reusable, and governable multi-step processes with declarative control flow, including **conditional branching**.

Key Requirements

- **Context**: An act:ActivityStep **SHALL** only exist as part of a act:CompositeActivitySchema, linked via properties like act:hasStep.
- **Logic Definition**: An act:ActivityStep **SHALL** reference exactly one act:ActivitySchema via the act:usesSchema property to define its executable logic.
- **Identification**: An act:ActivityStep **SHOULD** have a simple, locally-unique schema:identifier to simplify referencing within the workflow definition.
- **Description**: An act:ActivityStep **SHOULD** have a human-readable schema:name for display purposes and **MAY** have a schema:description for detailed documentation.
- **Conditional Execution**: An act:ActivityStep **MAY** include an act:hasCondition to enable conditional branching within a workflow.

9.4.6.1. Class Definition

Table 59—Class Definition for act:ActivityStep

RDF Class	**act:ActivityStep**
Is Abstract	No
Definition	A unique, addressable step within a act:CompositeActivitySchema. Each step is an element of the workflow that points to an act:ActivitySchema it executes and can be annotated with names, descriptions, and conditional logic.
Subclass Of	owl:Thing

Table 59—Class Definition for act:ActivityStep *(continued)*

RDF Class	act:ActivityStep
Usage Note	An act:ActivityStep acts as a node in a workflow graph. It is a structural placeholder that defines a unit of work.
Rationale	Modeling steps as first-class entities makes the workflow structure explicit, addressable, and governable, allowing metadata, documentation, and conditional logic to be attached to individual stages of a process.

Table 60—Properties Summary for act:ActivityStep

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
act:usesSchema	usesSchema	Links the step to the act: ActivitySchema defining its logic.	act: ActivitySchema	1..1	**Mandatory**
schema:name	name	A short, human-readable name for the step.	xsd:string	0..1	Recommended
schema: description	description	A detailed explanation of the step's purpose.	xsd:string	0..1	Optional
act: hasCondition	hasCondition	A condition that must be met for this step to be executed.	core:Condition	0..1	Optional
schema: identifier	identifier	A unique identifier for the step within its composite schema.	xsd:string	0..1	Recommended

9.4.6.2. Properties

This section provides the detailed normative definitions for the properties of the act:ActivityStep class.

9.4.6.2.1. Property: usesSchema

Table 61—Property Definition: act:usesSchema

Property	act:usesSchema
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#usesSchema
JSON name	usesSchema
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:ActivityStep
Range	act:ActivitySchema
Definition	Links an act:ActivityStep to the act:ActivitySchema that defines its executable logic.
Usage Note	The referenced schema can be either atomic or another composite schema, allowing for nested workflows.

9.4.6.2.2. Property: name

Table 62—Property Definition: schema:name

Property	schema:name
IRI	https://schema.org/name
JSON name	name
Requirement Level	Recommended

Table 62—Property Definition: schema:name *(continued)*

Cardinality	0..1
Domain	act:ActivityStep
Range	xsd:string
Definition	A short, human-readable name for the activity step.
Usage Note	This property is intended for display in user interfaces, such as workflow diagrams or logs. For example: “Validate User Credentials”. Reuses the property from Schema.org.

9.4.6.2.3. Property: description

Table 63—Property Definition: schema:description

Property	schema:description
IRI	https://schema.org/description
JSON name	description
Requirement Level	Optional
Cardinality	0..1
Domain	act:ActivityStep
Range	xsd:string
Definition	A detailed explanation of the step’s purpose, inputs, outputs, or behavior.
Usage Note	This property is for documentation and maintainability, helping developers understand the role of the step in the overall process. Reuses the property from Schema.org.

9.4.6.2.4. Property: hasCondition

Table 64—Property Definition: act:hasCondition

Property	act:hasCondition
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#hasCondition
JSON name	hasCondition
Requirement Level	Optional
Cardinality	0..1
Domain	act:ActivityStep
Range	core:Condition
Definition	An optional condition that must evaluate to true for this step to be eligible for execution.
Usage Note	This property provides the mechanism for conditional control flow. When multiple steps are presented as part of an act:hasChoice construct, the runtime engine **SHALL** evaluate the act:hasCondition for each step. The first step whose condition evaluates to true **SHALL** be the path that is executed.

9.4.6.2.5. Property: identifier

Table 65—Property Definition: schema:identifier

Property	schema:identifier
IRI	https://schema.org/identifier
JSON name	identifier

Table 65—Property Definition: schema:identifier *(continued)*

Requirement Level	Recommended
Cardinality	0..1
Domain	act:ActivityStep
Range	xsd:string
Definition	A locally unique, human-friendly identifier for the step.
Usage Note	This identifier provides a simple, stable "nickname" (e.g., "stepA", "validateUser") that **SHOULD** be used to uniquely identify the step **within its parent act:CompositeActivitySchema**. This makes it easier for act:DataLink instances to reliably reference the step as a source or target without using long, complex URIs.

9.4.7. Class: act:DataLink

The class act:DataLink realizes the **DATA LINK** concept as part of a act:CompositeActivitySchema in IEEE P2874 § 6.6.3.

"A DATA LINK is a construct for connecting the data flow between steps in a composite activity."

act:DataLink is a structural entity that explicitly models the "wiring" of data within a complex workflow. It connects the output of one step to the input of another, or links the inputs and outputs of the composite schema to its internal steps. By modeling data flow as a first-class entity, the standard makes it addressable, auditable, and governable.

Key Requirements

– **Context**: An act:DataLink **SHALL** only exist as part of a act:CompositeActivitySchema, linked via the act:hasDataLink property.

– **Source Specification**: An act:DataLink **SHALL** identify its data source via act:sourceVariable and, if internal, act:sourceStep. The link is made using the schema:identifier of the respective components.

– **Target Specification**: An act:DataLink **SHALL** identify its data target via act:targetVariable and, if internal, act:targetStep. The link is made using the schema:identifier of the respective components.

– **Interface Wiring**: To wire the composite's interface, a DataLink **SHALL** omit either the sourceStep (for an input) or targetStep (for an output).

9.4.7.1. Class Definition

Table 66—Class Definition for act:DataLink

RDF Class	**act:DataLink**
Is Abstract	No
Definition	A directed "wire" defining data flow. It can connect two internal steps, a composite's input to a step, or a step's output to a composite's output.
Subclass Of	owl:Thing
Usage Note	Making act:DataLink a class allows the data flow itself to be an addressable and governable entity. Metadata (e.g., encryption requirements, Quality of Service policies, names, descriptions) can be attached directly to the link.
Rationale	Elevates the workflow's data flow to be an explicit, governable, and self-describing entity, enabling more sophisticated security and management patterns.

Table 67—Properties Summary for act:DataLink

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
schema:name	name	A short, human-readable name for the data link.	xsd:string	0..1	Optional
schema: description	description	A detailed explanation of the data link's purpose.	xsd:string	0..1	Optional
act:sourceStep	sourceStep	The identifier of the source act: ActivityStep.	xsd:string	0..1	Conditional
act:targetStep	targetStep	The identifier of the target act: ActivityStep.	xsd:string	0..1	Conditional
act: sourceVariable	sourceVariable	The identifier of the source act: Variable.	xsd:string	1..1	**Mandatory**
act: targetVariable	targetVariable	The identifier of the target act: Variable.	xsd:string	1..1	**Mandatory**

9.4.7.2. Properties

This section provides the detailed normative definitions for the properties of the act:DataLink class.

9.4.7.2.1. Property: name

Table 68—Property Definition: schema:name

Property	schema:name
IRI	https://schema.org/name
JSON name	name
Requirement Level	Optional
Cardinality	0..1
Domain	act:DataLink
Range	xsd:string
Definition	A short, human-readable name for the data link.
Usage Note	Useful for documenting the workflow. For example: "Pass User ID to Validator". Reuses the property from Schema.org.

9.4.7.2.2. Property: description

Table 69—Property Definition: schema:description

Property	schema:description
IRI	https://schema.org/description
JSON name	description
Requirement Level	Optional
Cardinality	0..1
Domain	act:DataLink
Range	xsd:string
Definition	A detailed explanation of the data link's purpose or any transformations.

Table 69—Property Definition: schema:description *(continued)*

Usage Note	Can be used to explain complex wiring or data handling logic to developers. Reuses the property from Schema.org.

9.4.7.2.3. Property: sourceStep

Table 70—Property Definition: act:sourceStep

Property	act:sourceStep
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#sourceStep
JSON name	sourceStep
Requirement Level	Conditional
Cardinality	0..1
Domain	act:DataLink
Range	xsd:string
Definition	The identifier of the source act:ActivityStep for the data link.
Usage Note	The value of this property **SHALL** match the schema:identifier of an act:ActivityStep within the same composite schema. This property is omitted for data links that wire a public input of the composite schema to an internal step.

9.4.7.2.4. Property: targetStep

Table 71—Property Definition: act:targetStep

Property	act:targetStep
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#targetStep
JSON name	targetStep
Requirement Level	Conditional
Cardinality	0..1
Domain	act:DataLink
Range	xsd:string
Definition	The identifier of the target act:ActivityStep for the data link.
Usage Note	The value of this property **SHALL** match the schema:identifier of an act:ActivityStep within the same composite schema. This property is omitted for data links that expose the output of an internal step as a public output of the composite schema.

9.4.7.2.5. Property: sourceVariable

Table 72—Property Definition: act:sourceVariable

Property	act:sourceVariable
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#sourceVariable
JSON name	sourceVariable
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:DataLink
Range	xsd:string
Definition	The identifier of the source act:Variable for the data link.

Table 72—Property Definition: act:sourceVariable *(continued)*

Usage Note	The value of this property **SHALL** match the schema:identifier of an act:Variable defined in the source step's schema (or the composite schema itself).

9.4.7.2.6. Property: targetVariable

Table 73—Property Definition: act:targetVariable

Property	act:targetVariable
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#targetVariable
JSON name	targetVariable
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:DataLink
Range	xsd:string
Definition	The identifier of the target act:Variable for the data link.
Usage Note	The value of this property **SHALL** match the schema:identifier of an act:Variable defined in the target step's schema (or the composite schema itself).

9.4.8. Class: act:Variable

The class act:Variable realizes the **VARIABLE** concept as part of an act:ActivitySchema in IEEE P2874 § 6.6.3.

"A VARIABLE represents a typed parameter or binding placeholder within an ACTIVITY SCHEMA, used to specify required inputs, produced outputs, or contextual bindings."

act:Variable is a structural component that formally defines the interface for a single parameter within a schema. It supports the **declaration → binding → validation** pipeline by separating the declaration of a variable's expected type from the formal conditions used to enforce its validity.

Key Requirements

- **Identification**: A variable **SHALL** have a locally unique, machine-readable schema:identifier. It **SHOULD** also be described with a schema:name and **MAY** have a schema:description.
- **Declaration**: A variable **SHOULD** declare its expected data type(s) via the act:expects property. This serves as a clear, human- and machine-readable declaration of intent.
- **Enforcement**: A variable **MAY** be constrained by one or more core:Condition instances. This is the formal mechanism for enforcing complex validation rules.
- **Usage**: Variables **SHALL** be used in an act:ActivitySchema to define inputs (act:hasInput) or outputs (act:hasOutput).

9.4.8.1. Class Definition

Table 74—Class Definition for act:Variable

RDF Class	**act:Variable**
Is Abstract	No
Definition	A formal parameter definition for an act:ActivitySchema, representing an input, output, or contextual binding.
Subclass Of	owl:Thing

Table 74—Class Definition for act:Variable *(continued)*

RDF Class	act:Variable
Usage Note	A Variable is a structural component of a schema's interface. It defines the complete signature for a parameter, including its identifier, name, expected type(s), and validation logic.
Rationale	Provides a formal structure for parameterizing Activities, ensuring correctness and reusability. Separating type declaration from validation conditions improves both usability and robustness.

Table 75—Properties Summary for act:Variable

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
schema:identifier	identifier	A locally unique, machine-readable identifier for the variable.	xsd:string	1..1	**Mandatory**
schema:name	name	A short, human-readable name for the variable.	xsd:string	0..1	Recommended
schema:description	description	A detailed explanation of the variable's purpose.	xsd:string	0..1	Optional
act:expects	expects	The expected data type or class of the variable's value.	rdfs:Class	0..*	Recommended
core:hasCondition	hasCondition	Declares formal conditions (e.g., value range) applied to the variable's value.	core:Condition	0..*	Optional

9.4.8.2. Properties

This section provides the detailed normative definitions for the properties of the act:Variable class.

9.4.8.2.1. Property: identifier

Table 76—Property Definition: schema:identifier

Property	schema:identifier
IRI	https://schema.org/identifier
JSON name	identifier
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:Variable
Range	xsd:string
Definition	A locally unique, machine-readable identifier for the variable.
Usage Note	This identifier **SHALL** be unique within the scope of the act:ActivitySchema where it is defined. It is used for programmatic binding via act:VariableBinding and for referencing variables in act:DataLink instances (e.g., "targetLocation"). Reuses the property from Schema.org.

9.4.8.2.2. Property: name

Table 77—Property Definition: schema:name

Property	schema:name

Table 77—Property Definition: schema:name *(continued)*

IRI	https://schema.org/name
JSON name	name
Requirement Level	Recommended
Cardinality	0..1
Domain	act:Variable
Range	xsd:string
Definition	A short, human-readable name for the variable.
Usage Note	This property provides a human-friendly label for user interfaces and documentation. For example: “Target Location”. Reuses the property from Schema.org.

9.4.8.2.3. Property: description

Table 78—Property Definition: schema:description

Property	schema:description
IRI	https://schema.org/description
JSON name	description
Requirement Level	Optional
Cardinality	0..1
Domain	act:Variable
Range	xsd:string
Definition	A detailed explanation of the variable’s purpose and usage context.
Usage Note	This property is for documentation and maintainability, helping developers understand the role of the variable. Reuses the property from Schema.org.

9.4.8.2.4. Property: expects

Table 79—Property Definition: act:expects

Property	act:expects
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#expects
JSON name	expects
Requirement Level	Recommended
Cardinality	0..*
Domain	act:Variable
Range	rdfs:Class
Definition	A declaration of the expected data type(s) or class(es) for the variable’s value.
Usage Note	This property provides a simple, direct way for applications to understand the variable’s type without needing to parse a core:Condition. The range can be an XSD datatype (e.g., xsd:string) or another class (e.g., core:Agent). The property may be repeated to allow for union types (e.g., a variable that expects either a core:Agent or an xsd:string). This is the **declaration of intent**.

9.4.8.2.5. Property: hasCondition

Table 80—Property Definition: core:hasCondition

Property	core:hasCondition
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#hasCondition
JSON name	hasCondition
Requirement Level	Optional
Cardinality	0..*
Domain	act:Variable
Range	core:Condition
Definition	Associates one or more formal conditions that a variable's bound value must satisfy.
Usage Note	This is the **enforcement mechanism**. While act:expects declares the basic type, core:hasCondition is used to enforce more complex rules, such as value ranges, string patterns (regex), or conformance to a SHACL shape. A core:SHACLCondition can also be used to enforce the act:expects declaration.

9.4.9. Class: act:VariableBinding

The class act:VariableBinding provides the concrete link between an abstract act:Variable in a schema and a specific value in an act:Activity instance.

act:VariableBinding is the mechanism that makes schemas reusable. It connects the abstract parameter slots defined in the act:ActivitySchema to the real-world data and entities used in a particular execution, ensuring that every act:Activity is a complete and self-contained record.

Key Requirements

– **Variable Reference**: A act:VariableBinding **SHALL** reference exactly one act:Variable from the parent activity's schema via the act:variable property, using the variable's local identifier.

– **Value Assignment**: A act:VariableBinding **SHALL** provide exactly one concrete value (either a literal or a resource) via the standard rdf:value property.

– **Context**: A act:VariableBinding only exists as part of an act:Activity, linked via the act:hasBinding property.

9.4.9.1. Class Definition

Table 81—Class Definition for act:VariableBinding

RDF Class	**act:VariableBinding**
Is Abstract	No
Definition	A binding that connects a Variable to a concrete value (object or literal) in the context of an Activity.
Subclass Of	owl:Thing
Usage Note	This class is the bridge between the abstract schema and the concrete activity. An act:Activity will have one act:VariableBinding instance for each parameter it provides.
Rationale	Separates the abstract parameter definition from its concrete value, enabling the "Parameter/Binding Pattern" for maximum schema reusability.

Table 82—Properties Summary for act:VariableBinding

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
act:variable	variable	The identifier of the act:Variable this binding parameterizes.	xsd:string	1..1	**Mandatory**
rdf:value	value	The concrete value (IRI or Literal) assigned in the binding.	rdfs:Resource or rdfs:Literal	1..1	**Mandatory**

9.4.9.2. Properties

This section provides the detailed normative definitions for the properties of the act:VariableBinding class.

9.4.9.2.1. Property: variable

Table 83—Property Definition: act:variable

Property	act:variable
IRI	https://www.spatialwebfoundation.org/ns/hsml/activity#variable
JSON name	variable
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:VariableBinding
Range	xsd:string
Definition	Connects a VariableBinding to the Variable it parameterizes by referencing the variable's identifier.
Usage Note	The value of this property **SHALL** match the schema:identifier of an act:Variable defined in the activity's schema. This provides a consistent, portable binding mechanism.

9.4.9.2.2. Property: value

Table 84—Property Definition: rdf:value

Property	rdf:value
IRI	http://www.w3.org/1999/02/22-rdf-syntax-ns#value
JSON name	value
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	act:VariableBinding
Range	rdfs:Resource or rdfs:Literal
Definition	The concrete value assigned in a VariableBinding.
Usage Note	This standard RDF property is used to provide the actual data for a specific activity execution. The value can be a literal (e.g., string, number, boolean) or a URI pointing to another entity. The value provided **should** be validated against the core:hasCondition of the bound act:Variable.

10. HSML Governance Module

10.1. Architectural Principles and Design Rationale

This clause explains the design choices behind the HSML Governance Module and how they satisfy IEEE P2874's requirements for governable, auditable, and interoperable action across Domains. The principles are informative (non-normative), but they motivate the normative models and constraints defined in subsequent clauses.

10.1.1. The Norm/Policy Separation: Atomic Rules vs Governance Bundles

The foundation of the governance model is the separation between atomic rules and the governance bundles that publish them. act:ActivitySchema defines the universal semantics of an action—its roles, inputs, outputs, and pre/post conditions—in a way that is portable and immutable across Domains. By contrast, a gov:Norm represents a single deontic rule. Each Norm specifies whether an action or state is Obligatory, Prohibited, or Permitted, and links that modality to explicit machine-executable conditions (core:Condition). Norms are intentionally minimal: they state "what rule applies, to whom, and under which condition."

To make those rules effective in practice, authorities issue gov:Policy artifacts. A Policy bundles one or more Norms and supplies the governance context that a Norm by itself does not carry. Policies declare scope (which actors, activities, or domains the rules apply to), set validity periods, publish lifecycle state, establish ordering in case of conflicts, and—critically—reference credential profiles that define what types of verifiable credentials are acceptable. This separation allows Norms to remain atomic and reusable across contexts, while Policies provide the authoritative packaging that determines where and when Norms are in force.

10.1.2. Credentials and Credential Profiles

Verifiable Credentials (VCs) are not embedded in Norms but are instead attached to Agents. Norms may presuppose that certain credentials exist, but it is Policies that enforce credential requirements through gov:CredentialProfile. A Credential Profile describes, at the type level, what any acceptable VC must satisfy—its type, issuer trust, subject binding, attribute constraints, proof suite, and freshness requirements. During contract validation, Agents present credential instances, and the validation process checks both their cryptographic proofs and their conformance to the profiles declared by the applicable Policies. This design keeps Policies declarative and reusable, while preserving Agent privacy until execution time.

10.1.3. Deontic Modality and Executable Conditions

The heart of each Norm is its deontic modality—Obligation, Prohibition, or Permission—coupled with one or more executable conditions. A Norm is therefore both semantically clear and operational: it not only states the normative force but also the concrete, machine-checkable predicates under which the rule applies. These conditions are expressed in standards-aligned constraint languages such as SHACL, SPARQL ASK, JSON-Schema, or CEL, ensuring determinism, auditability, and interoperability. Policy-level precedence rules determine how to resolve conflicts when multiple Norms apply at once.

10.1.4. Time-Bound Governance

Policies provide temporal boundaries for governance. They carry schema:validFrom and schema: validThrough properties to indicate when a bundle of Norms is in force. A Policy outside its validity window

must not be considered during contract validation. If no end time is declared, the Policy remains in effect until explicitly revoked. Individual Norms may also include effectivity windows when a rule itself is inherently time-limited, but the general assumption is that time-scoping is managed at the Policy level.

10.1.5. Contracts as the Universal Trigger

Execution of every act:Activity is mediated through a gov:Contract. The Contract acts as the universal trigger and single enforcement point. It records the intent to act, the participating parties, the validation process, and the resulting lifecycle status. By funnelling all execution through Contracts, the module ensures a tamper-evident audit trail that links the governing Policies and Norms, the conditions that were evaluated, the credentials that were presented, and the decision outcomes. This approach guarantees both consistent enforcement and complete traceability.

10.1.6. Trust Substrate and Evidence

The governance model relies on W3C standards for identity and credentials. Spatial Web Identifiers (DIDs) identify Agents, Domains, and Contracts. Verifiable Credentials supply cryptographic attestations of authority, qualification, or compliance. Validation always includes proof verification, issuer trust assessment, revocation checks, and freshness evaluation. Implementations are expected to persist minimal verification transcripts so that decisions can be independently re-audited, providing accountability without unnecessary data retention.

10.1.7. Evaluation Flow

The governance evaluation process follows a predictable sequence. First, the system discovers which Policies apply given the Domain, ActivitySchema, actors, and time. It then collects the Norms bundled in those Policies and applies precedence rules to resolve conflicts. Next, the system verifies that the Agent's credentials satisfy the CredentialProfiles declared by the Policies. Once credential gates are passed, the executable conditions associated with the Norms are evaluated against the current state. The combined results yield a decision—permit, deny, or permit with obligations—which is recorded in the Contract along with evidence. If permitted, obligations may be monitored at runtime, and violations may trigger enforcement actions.

10.1.8. Interoperability and Extensibility

The model is designed to be interoperable across heterogeneous Domains. Activities are always referenced by IRIs, making schema usage neutral. Conditions are pluggable: any standards-aligned constraint language can be used so long as it is identifiable by content type. Policies can be composed hierarchically to represent organizational or jurisdictional layers. Evidence is linked but content-agnostic, allowing different proof suites and VC profiles to interoperate.

10.1.9. Security and Privacy

Finally, the governance design incorporates privacy and security considerations. Policies declare only credential **types**, never instances, limiting disclosure. Contracts bind VC presentations to specific transactions, preventing replay. Selective disclosure and zero-knowledge proofs are preferred when only one attribute is needed. Revocation checks are repeated when contracts extend across expiry or state changes, preventing drift. Together these measures ensure least disclosure, freshness, and resistance to side-channel leakage.

10.2. Normative Classes

This clause provides the normative definitions for all class concepts. It is divided into two parts: classes that are defined as part of this Governance Module, and classes from the HSML Core Module that are normatively used by this module.

The namespace prefix gov: refers to https://www.spatialwebfoundation.org/ns/hsml/governance#/.

10.2.1. Summary of Governance Module Classes

Table 85—Summary of Classes Used in the HSML Governance Module

Class	Description
gov:Contract	The binding agreement that initiates and governs the execution of one or more act:Activity instances.
gov:Credential	A tamper-evident, cryptographically verifiable set of claims, as defined by the W3C VC standard, used for authentication and authorization.
gov:CredentialProfile	A reusable bundle of constraints describing what a Verifiable Credential instance MUST satisfy (type, issuer/trust, subject binding, attribute constraints, proof, freshness)
gov:Norm	An atomic deontic rule (Obligation, Prohibition, Permission) governing behavior in a Domain, expressed as one or more executable Conditions.
gov:Policy	A governance artifact issued by an authority, bundling one or more Norms, and declaring their scope, applicability, and lifecycle.
gov:DeonticModality	An enumeration class that defines the set of allowable deontic modalities for Norms (Obligation, Prohibition, Permission).

10.2.2. Class: gov:Contract

The class gov:Contract realises the **CONTRACT** concept in IEEE P2874 § 6.6.4:

"A binding agreement between two or more parties—enforceable by law or internal policy—that governs one or more Activities in the Spatial Web."

gov:Contract links the initiating **Domain**, the participating **Agents**, the governed **Activity** (or Activities), and the evolving **ContractStatus** instances that capture its lifecycle.

Key Requirements

- **Identity** Every Contract **SHALL** possess a Spatial Web Identifier (hsml:swid, W3C DID Core-compliant).
- **Initiator** Exactly one hsml:isRequestedBy **SHALL** reference the **Domain** that issued the request.
- **Acceptance** One or more hsml:isAcceptedBy references **SHALL** identify the Agent(s) that accept the terms.
- **Fulfilment** One or more hsml:isFulfilledBy references **SHALL** identify the Agent(s) responsible for execution.
- **Governed Activity** One or more hsml:contractFor references **SHALL** identify the Activity(ies) the contract governs.
- **Lifecycle** hsml:contractStatus references **SHALL** record the contract's state
- **Extensibility** Additional clauses, policy links, or monetary terms **MAY** be added via further properties or subclasses without altering these core semantics.

These rules guarantee that every contract is discoverable, auditable, and machine-actionable across Spatial-Web systems.

10.2.2.1. Class Definition

Table 86—Class Definition for gov:Contract

RDF Class	gov:Contract
Is Abstract	No
Definition	A binding agreement governing the execution of one or more act:Activity instances by specific agt:Agent`s within a `hsml:Domain context.
Subclass Of	core:Entity
Usage Notes	A gov:Contract must carry a SWID and link to its initiating domain, accepting and fulfilling agents, the governed activity, and its lifecycle status. For detailed roles, it uses core:Participation records.

Table 87—Properties Summary for gov:Contract

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
gov: isRequestedBy	requestedBy	The core:Domain that initiated the contract.	core:Domain	1..1	**Mandatory**
gov: isAcceptedBy	acceptedBy	The agt:Agent(s) that accepted the contract.	agt:Agent	1..*	**Mandatory**
gov: isFulfilledBy	fulfilledBy	The agt:Agent(s) responsible for fulfilling the contract.	agt:Agent	1..*	**Mandatory**
gov:contractFor	contractFor	The act:Activity the contract governs.	act:Activity	1..*	**Mandatory**
gov: contractStatus	contractStatus	The current lifecycle status of the contract.	xsd:string	1..1	**Mandatory**
gov: hasParticipation	hasParticipation	Links to detailed participation records.	core: Participation	0..*	Optional

10.2.2.2. Properties

This section provides the detailed normative definitions for the properties of the gov:Contract class.

10.2.2.2.1. Property: isRequestedBy

Table 88—Property Definition: gov:isRequestedBy

Property	gov:isRequestedBy
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#isRequestedBy
JSON name	requestedBy
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	gov:Contract
Range	hsml:Domain
Definition	Identifies the hsml:Domain that initiated or requested creation of the contract.
Usage Note	Exactly one initiating hsml:Domain is recorded.

10.2.2.2.2. Property: isAcceptedBy

Table 89—Property Definition: gov:isAcceptedBy

Property	gov:isAcceptedBy
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#isAcceptedBy
JSON name	acceptedBy
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	gov:Contract
Range	agt:Agent
Definition	Identifies the agt:Agent(s) formally accepting the contract.
Usage Note	Multi-party contracts must list every accepting agt:Agent.

10.2.2.2.3. Property: isFulfilledBy

Table 90—Property Definition: gov:isFulfilledBy

Property	gov:isFulfilledBy
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#isFulfilledBy
JSON name	fulfilledBy
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	gov:Contract
Range	agt:Agent
Definition	Identifies the agt:Agent(s) tasked with fulfilling the contractual obligations.
Usage Note	This is often, but not always, the same agt:Agent(s) as gov:isAcceptedBy.

10.2.2.2.4. Property: contractFor

Table 91—Property Definition: gov:contractFor

Property	gov:contractFor
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#contractFor
JSON name	contractFor
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	gov:Contract
Range	act:Activity
Definition	Identifies the act:Activity (or Activities) that this contract governs.
Usage Note	This property links the agreement to the specific action(s) to be executed.

10.2.2.2.5. Property: contractStatus

Table 92—Property Definition: gov:contractStatus

Property	gov:contractStatus
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#contractStatus
JSON name	contractStatus

Table 92—Property Definition: gov:contractStatus *(continued)*

Requirement Level	**Mandatory**
Cardinality	1..1
Domain	gov:Contract
Range	xsd:string
Definition	The current lifecycle state of the contract. The value **shall** be one of: "Requested", "Executed", "Fulfilled", "Rescinded", or "Breached".
Usage Note	This property provides the current state of the agreement. For a full audit trail, a history of status changes with timestamps should be recorded.

10.2.2.2.6. Property: hasParticipation

Table 93—Property Definition: gov:hasParticipation

Property	gov:hasParticipation
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#hasParticipation
JSON name	hasParticipation
Requirement Level	Optional
Cardinality	0..*
Domain	gov:Contract
Range	core:Participation
Definition	Links the contract to detailed participation records.
Usage Note	Use for complex contracts to explicitly define agent roles beyond requester/fulfiller.

10.2.3. Class: gov:Credential

The class gov:Credential realizes the CREDENTIAL concept in IEEE P2874 §6.6.3. It specializes the W3C VC Data Model to carry attestations that govern identity, authorization, compliance, and trust across Domains and Activities.

10.2.3.1. Key Requirements (Normative)

a) A gov:Credential **shall** be processable as a W3C Verifiable Credential (VC); core VC fields (e.g., issuer, credentialSubject, proof) **shall** be supported.

b) Verification **shall** be two-stage:

   1) **Cryptographic** — verify proof per the VC spec; failure **shall** reject the credential.

   2) **Semantic (conditional)** — if a schema is discoverable (from policy via gov:CredentialProfile and/or from gov:hasSchema in the credential), the credentialSubject **shall** be validated accordingly; failure **shall** make it semantically invalid for the intended purpose.

c) A credential **need not embed** a schema; policies typically provide schema/constraints via gov:CredentialProfile. If present, a schema pointer **shall** use gov:hasSchema.

d) Implementations **should** support multiple schema languages (e.g., SHACL, JSON Schema) and select a validator based on the schema's media type and/or RDF type.

10.2.3.2. Class Definition

Table 94—Class Definition for gov:Credential

RDF Class	gov:Credential
Is Abstract	No
Definition	An Entity representing a permission, attestation, or claim that can be verified or required in the Spatial Web.
Subclass Of	vc:VerifiableCredential, core:Entity
Usage Note	Schema pointers are optional because policy-linked gov:CredentialProfile typically supplies schema/constraints.

10.2.3.3. Properties Summary

Table 95

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement
gov: conformsToProfile	conformsToProfile	Declares an intended gov: CredentialProfile this credential claims to satisfy (hint for evaluators; policy is authoritative).	gov: CredentialProfile	0..*	Optional

10.2.3.4. Properties

10.2.3.4.1. Property: conformsToProfile

Table 96—Property Definition: gov:conformsToProfile

Property	gov:conformsToProfile
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#conformsToProfile
JSON name	conformsToProfile
Requirement Level	Optional
Cardinality	0..*
Domain	gov:Credential
Range	gov:CredentialProfile
Definition	Identifies a credential profile the credential claims to meet; used as a routing/validation hint.
Usage Note	Policy evaluation is governed by the policy's required gov: CredentialProfile(s), not by this self-assertion.

10.2.3.5. Credential Validation Process

a) **Stage 1 — Cryptographic (MUST):** Verify proof per VC spec; reject on failure.

b) **Stage 2 — Semantic (IF schema discoverable):** If policy requires a gov:CredentialProfile and/or the credential provides gov:hasSchema, fetch the schema and validate credentialSubject using a suitable engine for the media type or RDF type.

10.2.3.6. Requirements for Policy Engines

a) Policies **should** specify credential type/class and constraints via gov:CredentialProfile; engines **shall** fetch and apply referenced schemas.

b) Engines **shall** support gov:hasSchema (generic). gov:hasClaimSchema is processed only as a deprecated alias.

10.2.4. Class: gov:CredentialProfile

The class gov:CredentialProfile bundles the **requirements a Verifiable Credential instance MUST satisfy** to pass a policy's credential gate. It unifies (a) the **credential class/type**, (b) **subject/holder binding rules**, and © **attribute / issuer / trust / proof / freshness** constraints into a single reusable profile.

gov:CredentialProfile is referenced by Policies (and Norms) and evaluated at gov:Contract validation time against presented VC instances.

Key Requirements

– **Type Basis** A profile **SHALL** specify the acceptable credential class via gov:profileOfCredentialType.

– **Conformance** Presented VC instances **MUST** conform to all constraints in the profile (shapes, conditions, issuer/trust, status, proof suites, freshness, subject binding).

– **Reusability** Profiles **SHOULD** be reusable across Policies; Policies reference profiles rather than hard-coding constraints.

10.2.4.1. Class Definition

Table 97—Class Definition for gov:CredentialProfile

RDF Class	**gov:CredentialProfile**
Is Abstract	No
Definition	A reusable bundle of constraints describing what a Verifiable Credential instance MUST satisfy (type, issuer/trust, subject binding, attribute constraints, proof, freshness).
Subclass Of	dct:Standard (recommended)
Usage Notes	Profiles may be versioned and governed by a Domain. Use with dct:conformsTo during verification.

Table 98—Properties Summary for gov:CredentialProfile

Predicate	**JSON-LD name**	**Description**	**Range**	**Card.**	**Level**
gov: profileOfCredentialType	profileOfCredentialType	The base credential class/type this profile constrains.	cred: CredentialType	1..1	**Mandatory**
gov: credentialShape	credentialShape	SHACL shape(s) the VC instance MUST satisfy (claims structure/ content).	sh:NodeShape	0..*	Optional
core: hasCondition	hasCondition	Executable conditions evaluated against VC (and/or subject) claims.	core:Condition	0..*	Optional
gov: acceptableIssuer	acceptableIssuer	Allowlisted issuers whose VCs are accepted for this profile.	agt:Agent or hsml:Domain	0..*	Optional

Table 98—Properties Summary for gov:CredentialProfile *(continued)*

Predicate	JSON-LD name	Description	Range	Card.	Level
gov:trustFramework	trustFramework	Named trust framework governing status & crypto evaluation.	skos:Concept	0..*	Optional
gov:statusPolicy	statusPolicy	Required status (e.g., notRevoked / StatusList2021:valid).	xsd:string	0..1	Optional
gov:issuedWithin	issuedWithin	Max credential age at validation (duration, e.g., P1Y).	xsd:duration	0..1	Optional
gov:requiresSubjectBinding	requiresSubjectBinding	VC subject MUST be bound to the performing Agent's identifier.	xsd:boolean	0..1	Optional
gov:proofSuite	proofSuite	Acceptable cryptographic proof suites (e.g., Ed25519Signature2020).	skos:Concept	0..*	Optional
gov:profileVersion	profileVersion	Version identifier of this profile.	xsd:string	0..1	Optional

10.2.4.2. Properties

10.2.4.2.1. Property: profileOfCredentialType

Table 99—Property Definition: gov:profileOfCredentialType

Property	gov:profileOfCredentialType
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#profileOfCredentialType
JSON name	profileOfCredentialType
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	gov:CredentialProfile
Range	cred:CredentialType
Definition	Points to the base credential class that instances MUST instantiate to be eligible under this profile.
Usage Note	The instance's @type MUST equal or subclass this type.

10.2.4.2.2. Property: credentialShape

Table 100—Property Definition: gov:credentialShape

Property	gov:credentialShape
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#credentialShape
JSON name	credentialShape
Requirement Level	Optional
Cardinality	0..*
Domain	gov:CredentialProfile
Range	sh:NodeShape
Definition	SHACL NodeShape(s) that MUST validate true for a presented VC instance.
Usage Note	Use to assert claim presence, formats, enumerations, nested subject constraints, etc.

10.2.4.2.3. Property: hasCondition

Table 101—Property Definition: core:hasCondition

Property	core:hasCondition
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#hasCondition
JSON name	hasCondition
Requirement Level	Optional
Cardinality	0..*
Domain	gov:CredentialProfile
Range	core:Condition
Definition	Machine-executable conditions over VC/subject/environment (e.g., assurance ≥ X, role in org Y).
Usage Note	Reuses HSML Core; express as SHACL rules, SPARQL ASK templates, or CEL expressions.

10.2.4.2.4. Property: acceptableIssuer

Table 102—Property Definition: gov:acceptableIssuer

Property	gov:acceptableIssuer
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#acceptableIssuer
JSON name	acceptableIssuer
Requirement Level	Optional
Cardinality	0..*
Domain	gov:CredentialProfile
Range	agt:Agent or hsml:Domain
Definition	Whitelist of issuers permitted for credentials conforming to this profile.
Usage Note	Combine with gov:trustFramework and revocation checks.

10.2.4.2.5. Property: trustFramework

Table 103—Property Definition: gov:trustFramework

Property	gov:trustFramework
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#trustFramework
JSON name	trustFramework
Requirement Level	Optional
Cardinality	0..*
Domain	gov:CredentialProfile
Range	skos:Concept
Definition	Named trust framework policy guiding verification and status handling.
Usage Note	Examples: “GovID-EU-eIDAS-High”, “Aviation-FAA-Ops”.

10.2.4.2.6. Property: statusPolicy

Table 104—Property Definition: gov:statusPolicy

Property	gov:statusPolicy

Table 104—Property Definition: gov:statusPolicy *(continued)*

IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#statusPolicy
JSON name	statusPolicy
Requirement Level	Optional
Cardinality	0..1
Domain	gov:CredentialProfile
Range	xsd:string
Definition	Required VC status condition (e.g., “notRevoked”, “StatusList2021:valid”).
Usage Note	Mapped to concrete verifier behavior.

10.2.4.2.7. Property: issuedWithin

Table 105—Property Definition: gov:issuedWithin

Property	gov:issuedWithin
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#issuedWithin
JSON name	issuedWithin
Requirement Level	Optional
Cardinality	0..1
Domain	gov:CredentialProfile
Range	xsd:duration
Definition	Maximum credential age at validation time.
Usage Note	Evaluated against VC issuanceDate; ensure VC not expired.

10.2.4.2.8. Property: requiresSubjectBinding

Table 106—Property Definition: gov:requiresSubjectBinding

Property	gov:requiresSubjectBinding
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#requiresSubjectBinding
JSON name	requiresSubjectBinding
Requirement Level	Optional
Cardinality	0..1
Domain	gov:CredentialProfile
Range	xsd:boolean
Definition	If true, the VC subject MUST be bound to the performing Agent’s identifier at contract time.
Usage Note	Enforce via holder binding or subject DID equality checks.

10.2.4.2.9. Property: proofSuite

Table 107—Property Definition: gov:proofSuite

Property	gov:proofSuite
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#proofSuite
JSON name	proofSuite
Requirement Level	Optional
Cardinality	0..*

Table 107—Property Definition: gov:proofSuite *(continued)*

Domain	gov:CredentialProfile
Range	skos:Concept
Definition	Acceptable proof suites for presented VCs (e.g., Ed25519Signature2020, ECDSA-JWS).
Usage Note	Profiles may list multiple acceptable suites for interoperability.

10.2.4.2.10. Property: profileVersion

Table 108—Property Definition: gov:profileVersion

Property	gov:profileVersion
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#profileVersion
JSON name	profileVersion
Requirement Level	Optional
Cardinality	0..1
Domain	gov:CredentialProfile
Range	xsd:string
Definition	Version identifier of the profile for audit/change control.
Usage Note	Consider semantic versioning; deprecate older profiles via publisher policy.

10.2.5. Class: gov:DeonticModality

The class gov:DeonticModality defines the **enumerated values** of deontic force that a gov:Norm may express. It provides the controlled vocabulary for gov:modality values.

Key Requirements

– **Identity** Every modality value **SHALL** be globally identified by an IRI in the gov: namespace.

– **Closed Enumeration** Allowed values are strictly limited to: gov:Obligation, gov:Prohibition, gov:Permission.

– **Usage** Every gov:Norm **MUST** reference exactly one gov:DeonticModality via gov:modality.

10.2.5.1. Class Definition

Table 109—Class Definition for gov:DeonticModality

RDF Class	**gov:DeonticModality**
Is Abstract	Yes (enumeration)
Definition	An enumeration class for deontic modalities representing the normative force of a Norm.
Subclass Of	skos:Concept (optional alignment)
Usage Notes	Used only as the range of gov:modality. Instances are controlled vocabulary individuals.

10.2.5.2. Individuals

Table 110—Enumeration Values of gov:DeonticModality

Individual	Description
gov:Obligation	A Norm requiring that an action **must** be performed or a condition **must** hold.
gov:Prohibition	A Norm requiring that an action **must not** be performed or a condition **must not** hold.
gov:Permission	A Norm indicating that an action **may** be performed or a condition **may** hold (neither required nor forbidden).

10.2.6. Class: gov:Norm

The class gov:Norm realizes the **NORM** concept in IEEE P2874 § 6.6.4: **"An atomic deontic rule—Obligation, Prohibition, or Permission—governing behavior in a Domain and expressed via one or more evaluable Conditions."**

A Norm is the **atomic rule unit**: it specifies what modality (obligation, prohibition, permission) applies to which targets, and under which conditions. It does **not** carry issuer, validity, credential gates, or precedence; those are modeled at the gov:Policy level. Policies must reference one or more Norms via gov:hasNorm.

10.2.6.1. Class Definition

Table 111—Class Definition for gov:Norm

RDF Class	gov:Norm
Is Abstract	No
Definition	An atomic deontic rule (Obligation/Prohibition/Permission) expressed by executable conditions and applied to specific targets.
Subclass Of	core:Entity
Usage Notes	Norms define one rule. Policies package and govern them.

10.2.6.2. Key Requirements (Normative)

a) A gov:Norm **shall** declare exactly one gov:modality (Obligation, Prohibition, Permission).

b) A gov:Norm **shall** specify one or more governed targets via gov:appliesTo.

c) A gov:Norm **shall** include one or more core:hasCondition that operationalize applicability.

d) A gov:Norm **may** declare enforcement actions via gov:onViolation.

e) A gov:Norm **may** include an intra-policy ordering hint via gov:priority.

f) A gov:Policy **shall** reference one or more gov:Norm when publishing governance bundles.

10.2.6.3. Properties Summary

Table 112—Properties Summary for gov:Norm

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement
gov:appliesTo	appliesTo	Targets governed by the norm (agents, activities, domains, or role classes).	hsml:Entity or rdfs:Class	1..*	**Mandatory**
core: hasCondition	hasCondition	Executable conditions for applicability/evaluation.	core:Condition	1..*	**Mandatory**
gov:modality	modality	Deontic modality of the rule (Obligation, Prohibition, Permission).	gov: DeonticModality	1..1	**Mandatory**
gov:onViolation	onViolation	Enforcement action(s) if violated.	gov: EnforcementAction	0..*	Optional
gov:priority	priority	Resolution priority within a policy bundle.	xsd:integer	0..1	Optional

10.2.6.4. Properties

10.2.6.4.1. Property: modality

Table 113—Property Definition: gov:modality

Property	gov:modality
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#modality
JSON name	modality
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	gov:Norm
Range	gov:DeonticModality
Definition	Declares the deontic force of the norm.
Usage Note	Allowed values: gov:Obligation, gov:Prohibition, gov:Permission.

10.2.7. Class: gov:Policy

The class gov:Policy is a governance artifact issued by an authority to **declare scope**, **bundle Norms**, and **gate execution** of Activities via credential requirements. A Policy does not itself execute rules; rather, it references executable core:Condition`s (directly or via `gov:Norm) and **type-level** credential requirements expressed as gov:CredentialProfile. Policy validity is time-scoped using schema:validFrom and schema:validThrough.

Design Notes

- **Bundled Norms** Policies collect one or more gov:Norm (Obligation/Prohibition/Permission) that apply within the policy's declared scope.
- **Credential Gating (Type-level)** Policies reference one or more reusable gov:CredentialProfile via gov: hasCredentialRequirement. Each profile specifies an allowed credential **type/class** and the constraints that any presented **credential instance** must satisfy at gov:Contract validation time.

– **Conditions Reuse** Policies MAY also include additional core:hasCondition for policy-level applicability or evaluation (e.g., jurisdictional, temporal, or role constraints) without redefining the Condition class.

10.2.7.1. Class Definition

Table 114—Class Definition for gov:Policy

RDF Class	gov:Policy
Is Abstract	No
Definition	A governance artifact that declares scope, bundles one or more Norms, and specifies credential profiles that authoritatively gate Activity execution.
Subclass Of	core:Entity
Usage Notes	Policies are time-scoped with schema:validFrom/schema:validThrough. Credential gates reference gov:CredentialProfile (type-level). Instance-level credential verification occurs during gov:Contract validation.

Table 115—Properties Summary for gov:Policy

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement
schema:description	rationale	Human-readable rationale/ summary.	xsd:string	0..1	Optional
schema:creator	issuedBy	Issuing authority (Domain or Agent) responsible for the policy.	hsml:Domain or agt:Agent	1..1	**Mandatory**
schema:validFrom	validFrom	Start of the policy's validity window (inclusive).	xsd:dateTime	1..1	**Mandatory**
schema:validThrough	validThrough	End of the policy's validity window (exclusive). Omit for open-ended.	xsd:dateTime	0..1	Recommended
gov:appliesToActivitySchema	appliesToActivitySchema	Activity schema(s) governed by this policy.	act:ActivitySchema	1..*	**Mandatory**
gov:appliesToDomain	appliesToDomain	Domain(s) (places/orgs/contexts) where the policy applies.	hsml:Domain	0..*	Optional
gov:appliesToActorClass	appliesToActorClass	Actor/role classes targeted (e.g., Pilot, Contractor).	rdfs:Class	0..*	Optional
gov:hasNorm	hasNorm	Bundled atomic deontic rules governed by this policy.	gov:Norm	1..*	**Mandatory**
core:hasCondition	hasCondition	Additional executable applicability/evaluation conditions.	core:Condition	0..*	Optional
gov:hasCredentialRequirement	hasCredentialRequirement	Type-level credential gate(s) expressed as reusable profiles.	gov:CredentialProfile	0..*	Optional
gov:precedence	precedence	Policy priority for conflict resolution (higher wins).	xsd:integer	0..1	Optional
gov:policyStatus	policyStatus	Lifecycle status (e.g., Draft, Active, Suspended, Retired).	xsd:string	0..1	Optional
schema:version	version	Policy version identifier.	xsd:string	0..1	Optional
schema:spatialCoverage	jurisdiction	Geographic or legal jurisdiction in scope.	schema:Place or skos:Concept	0..*	Optional
gov:relatedPolicy	relatedPolicy	Links to superseding/subordinate/ peer policies.	gov:Policy	0..*	Optional

10.2.7.2. Properties

10.2.7.2.1. Property: description

Table 116—Property Definition: schema:description

Property	schema:description
IRI	https://schema.org/description
JSON name	rationale
Requirement Level	Optional
Cardinality	0..1
Domain	gov:Policy
Range	xsd:string
Definition	Human-readable summary, rationale, or notes.
Usage Note	Non-normative; do not encode executable constraints here.

10.2.7.2.2. Property: issuedBy

Table 117—Property Definition: schema:creator

Property	schema:creator
IRI	https://schema.org/creator
JSON name	issuedBy
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	gov:Policy
Range	hsml:Domain or agt:Agent
Definition	Identifies the authority that authors and promulgates the policy.
Usage Note	Prefer the controlling hsml:Domain when the issuer is an organization; use an Agent for individual signatories.

10.2.7.2.3. Property: validFrom

Table 118—Property Definition: schema:validFrom

Property	schema:validFrom
IRI	https://schema.org/validFrom
JSON name	validFrom
Requirement Level	**Mandatory**
Cardinality	1..1
Domain	gov:Policy
Range	xsd:dateTime
Definition	Start timestamp when the policy becomes applicable.
Usage Note	Evaluators MUST ignore a policy before this timestamp.

10.2.7.2.4. Property: validThrough

Table 119—Property Definition: schema:validThrough

Property	schema:validThrough
IRI	https://schema.org/validThrough
JSON name	validThrough
Requirement Level	Recommended
Cardinality	0..1
Domain	gov:Policy
Range	xsd:dateTime
Definition	End timestamp (exclusive) after which the policy is no longer applicable.
Usage Note	Omit for open-ended; revocation/suspension should set gov:policyStatus.

10.2.7.2.5. Property: appliesToActivitySchema

Table 120—Property Definition: gov:appliesToActivitySchema

Property	gov:appliesToActivitySchema
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#appliesToActivitySchema
JSON name	appliesToActivitySchema
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	gov:Policy
Range	act:ActivitySchema
Definition	Declares the Activity schema(s) governed by the policy.
Usage Note	Use IRIs of globally reusable schemas to enable cross-domain interoperability.

10.2.7.2.6. Property: appliesToDomain

Table 121—Property Definition: gov:appliesToDomain

Property	gov:appliesToDomain
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#appliesToDomain
JSON name	appliesToDomain
Requirement Level	Optional
Cardinality	0..*
Domain	gov:Policy
Range	hsml:Domain
Definition	Limits applicability to one or more Domains (e.g., sites, org units).
Usage Note	If omitted, default scope is the issuer's Domain and its sub-domains (implementation-defined).

10.2.7.2.7. Property: appliesToActorClass

Table 122—Property Definition: gov:appliesToActorClass

Property	gov:appliesToActorClass

Table 122—Property Definition: gov:appliesToActorClass *(continued)*

IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#appliesToActorClass
JSON name	appliesToActorClass
Requirement Level	Optional
Cardinality	0..*
Domain	gov:Policy
Range	rdfs:Class
Definition	Targets classes/roles of actors (e.g., org:Role, agt:Pilot).
Usage Note	Pair with Norm conditions for precise targeting.

10.2.7.2.8. Property: hasNorm

Table 123—Property Definition: gov:hasNorm

Property	gov:hasNorm
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#hasNorm
JSON name	hasNorm
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	gov:Policy
Range	gov:Norm
Definition	Includes the atomic deontic rules governed by the policy.
Usage Note	Each Norm declares gov:deonticModality and one or more executable core:Condition.

10.2.7.2.9. Property: hasCondition

Table 124—Property Definition: core:hasCondition

Property	core:hasCondition
IRI	https://www.spatialwebfoundation.org/ns/hsml/core#hasCondition
JSON name	hasCondition
Requirement Level	Optional
Cardinality	0..*
Domain	gov:Policy
Range	core:Condition
Definition	Executable constraints for applicability/evaluation at policy level.
Usage Note	Reuse HSML Core. Express with SHACL/ASK/CEL and record evaluation artifacts during contract validation.

10.2.7.2.10. Property: hasCredentialRequirement

Table 125—Property Definition: gov:hasCredentialRequirement

Property	gov:hasCredentialRequirement
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#hasCredentialRequirement
JSON name	hasCredentialRequirement
Requirement Level	Optional

Table 125—Property Definition: gov:hasCredentialRequirement *(continued)*

Cardinality	0..*
Domain	gov:Policy
Range	gov:CredentialProfile
Definition	References reusable credential profiles that **type-level** define acceptable credential classes and constraints.
Usage Note	Instance-level verification occurs at gov:Contract time by checking presented VCs **conform to** the referenced gov:CredentialProfile(s) (type match, issuer/trust, status, proof suite, freshness, subject binding, shapes/conditions).

10.2.7.2.11. Property: precedence

Table 126—Property Definition: gov:precedence

Property	gov:precedence
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#precedence
JSON name	precedence
Requirement Level	Optional
Cardinality	0..1
Domain	gov:Policy
Range	xsd:integer
Definition	Numeric priority used in conflict resolution across applicable policies.
Usage Note	Higher values override lower; tie-break by modality precedence if needed.

10.2.7.2.12. Property: policyStatus

Table 127—Property Definition: gov:policyStatus

Property	gov:policyStatus
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#policyStatus
JSON name	policyStatus
Requirement Level	Optional
Cardinality	0..1
Domain	gov:Policy
Range	xsd:string
Definition	Current lifecycle state (e.g., Draft, Active, Suspended, Retired).
Usage Note	Evaluators SHOULD ignore policies not in an “Active”-like state.

10.2.7.2.13. Property: version

Table 128—Property Definition: schema:version

Property	schema:version
IRI	https://schema.org/version
JSON name	version
Requirement Level	Optional
Cardinality	0..1
Domain	gov:Policy
Range	xsd:string

Table 128—Property Definition: schema:version *(continued)*

Definition	Version label for change control and auditability.
Usage Note	Use semantic versioning where practical.

10.2.7.2.14. Property: jurisdiction

Table 129—Property Definition: schema:spatialCoverage

Property	schema:spatialCoverage
IRI	https://schema.org/spatialCoverage
JSON name	jurisdiction
Requirement Level	Optional
Cardinality	0..*
Domain	gov:Policy
Range	schema:Place or skos:Concept
Definition	Geographic or legal area covered by the policy.
Usage Note	May reference controlled vocabularies for regions/jurisdictions.

10.2.7.2.15. Property: relatedPolicy

Table 130—Property Definition: gov:relatedPolicy

Property	gov:relatedPolicy
IRI	https://www.spatialwebfoundation.org/ns/hsml/governance#relatedPolicy
JSON name	relatedPolicy
Requirement Level	Optional
Cardinality	0..*
Domain	gov:Policy
Range	gov:Policy
Definition	Links to superseding, superseded, parent, or child policies.
Usage Note	Use gov:relatedPolicy with subproperties if finer relation typing is required.

11. HSML Agent Module

11.1. Architectural Principles and Design Rationale

This clause describes the architectural principles underlying the HSML Agent Module. These principles are *informative* and not required for conformance; however, understanding them is recommended to ensure robust and correct implementations.

11.1.1. The Goal-Directed Actor Pattern: Defining Purpose

A fundamental principle of the Agent Module is that all autonomous actions are **purpose-driven**. An agent doesn't simply act; it acts to achieve an objective. This is realized through a clear separation between the actor and its purpose:

– agt:Agent — The actor itself. An autonomous entity capable of perception, decision-making, and action.

– agt:Goal — A declarative, self-contained entity that represents the desired state or objective the agent seeks to achieve.

This separation is the cornerstone of **explainable AI** and **governable autonomy** in the Spatial Web. By making an agent's Goal an explicit, inspectable entity linked via the mandatory agt:hasGoal property, the model ensures that the "why" behind an agent's behavior is always a first-class citizen. This allows an orchestration engine or auditor to understand an agent's intent without needing to reverse-engineer its internal logic. It provides a clear, declarative basis for authorizing agent actions—if the goal is permissible within a domain, the agent's subsequent actions can be evaluated against it.

11.1.2. The Capability Model: Separating Potential from Action

Complementing the goal-directed pattern is the distinction between an agent's **potential to act** and its **actual performance of an action**. This is achieved by linking the agent to the act:ActivitySchema rather than to specific act:Activity instances:

– agt:Agent — The actor.

– act:ActivitySchema — A reusable template defining a *type* of action an agent is capable of performing.

– agt:canPerform — The property linking an Agent to the act:ActivitySchema instances that represent its skills or functions.

This design decouples the agent's intrinsic abilities from its operational history. The agt:canPerform property defines the agent's **skill set**, which is a relatively static and verifiable aspect of its identity. This allows a Domain Authority to grant credentials based on a clear, auditable list of an agent's capabilities. It ensures that governance is applied to what an agent *can do* in principle, providing a stable foundation for trust and permissioning. The agent's actual execution of these capabilities is then recorded as separate act:Activity instances, creating a clean and auditable separation between an agent's potential and its performance.

11.1.3. The Specialization Pattern: Tailoring Agents and Goals

A key architectural principle is that the agt:Agent and agt:Goal classes are designed as extensible foundations. This allows the framework to support a wide variety of agent architectures—such as **Simple Reflex**, **Goal-based**, **Deliberative (BDI)**, or **LLM-based Agents**—without being overly prescriptive. The primary mechanism for this is the specialization of the base agt:Goal class, as the way an agent describes its goals is fundamental to its nature.

This enables the creation of specific Goal subclasses that align with different agent behaviors:

– AchievementGoal:: A goal to bring about a specific state of the world. Once the state is reached, the goal is considered achieved and is complete. For example, "Deliver a package to a specific address."

– MaintenanceGoal:: A goal to keep a specific condition continuously true over time. This type of goal persists and is never truly "complete." For example, "Keep the server uptime above 99.9%."

– PerformingGoal:: A goal where success is defined by the execution of an action itself, regardless of the resulting state. For example, "Perform a system diagnostic scan every 24 hours."

– QueryGoal:: A goal whose aim is to acquire information, to know the true value of a proposition, or to otherwise reduce uncertainty before planning or acting. For example, "Determine the current traffic conditions on the planned route."

By defining these and other goal specializations (like **Hard** vs. **Soft Goals** with preference models), the architecture allows for a rich and precise description of agent behavior. A sophisticated Deliberative Agent might pursue complex AchievementGoal`s, while a simpler monitoring agent would be concerned with a

`MaintenanceGoal. This provides a flexible and future-proof framework that can adapt to new agent designs by simply introducing new, well-defined subclasses of agt:Goal.

11.2. Normative Classes

This clause provides the normative definitions for all class concepts within the HSML Agent Module. Each class is detailed in a table that specifies its URI, description, JSON-LD context name, usage notes, and relationship to other classes.

The namespace prefix agt: refers to https://www.spatialwebfoundation.org/ns/hsml/agent#.

11.2.1. Summary of Normative Classes

Table 131—Summary of HSML Agent Module Classes

Class	Description
agt:Agent	An autonomous entity capable of perceiving, deciding, and performing Activities to achieve its goals.
agt-Person	A human person, modeled as a special type of agt:Agent with a self-sovereign identity.
agt:Goal	A state or objective that an Agent aims to achieve through the performance of Activities.

11.2.2. Class: agt:Agent

The class agt:Agent realises the **AGENT** concept in IEEE P2874 § 6.6.3:

"An ENTITY that senses, responds, and maintains a model of its environment while performing ACTIVITIES to achieve its goals."

agt:Agent represents an autonomous entity capable of perceiving its environment, making decisions, and enacting act:Activity instances in pursuit of its goals. It may represent a person, robot, AI system, or other autonomous entity.

Key Requirements

– **Identity**: Every agt:Agent **SHALL** possess a Spatial Web Identifier (core:swid, W3C DID Core-compliant).

– **Goal-Directed**: An agt:Agent **SHALL** be linked to one or more objectives it seeks to achieve via the agt:hasGoal property.

– **Capability**: An agt:Agent **SHALL** be linked to the act:ActivitySchema definitions for the activities it is capable of performing via the agt:hasCapability property.

11.2.2.1. Class Definition

Table 132—Class Definition for agt:Agent

RDF Class	agt:Agent
Is Abstract	No
Definition	An autonomous entity capable of perceiving, deciding, and performing Activities to achieve its goals.
Subclass Of	core:Domain

Table 132—Class Definition for agt:Agent *(continued)*

RDF Class	agt:Agent
Usage Note	Agents are the actors in the Spatial Web. They initiate gov:Contract instances and perform act:Activity instances.
Rationale	Provides the normative model for autonomous actors in the Spatial Web, supporting goal-directed behavior and governed interactions.

Table 133—Properties Summary for agt:Agent

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement Level
agt:hasGoal	hasGoal	Links the Agent to one or more objectives it seeks to achieve.	agt:Goal	1..*	**Mandatory**
agt:canPerform	canPerform	Links the Agent to the schemas for Activities it can perform.	act: ActivitySchema	1..*	**Mandatory**

11.2.2.2. Properties

This section provides the detailed normative definitions for the properties of the agt:Agent class.

11.2.2.2.1. Property: hasGoal

Table 134—Property Definition: agt:hasGoal

Property	agt:hasGoal
IRI	https://www.spatialwebfoundation.org/ns/hsml/agent#hasGoal
JSON name	hasGoal
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	agent:Agent
Range	agent:Goal
Definition	Links an Agent to one or more goals that it aims to fulfill through its Activities.
Usage Note	Goals represent abstract or concrete states the Agent seeks to bring about. The range is core:Entity to allow goals to be simple text descriptions or complex, structured entities.

11.2.2.2.2. Property: canPerform

Table 135—Property Definition: agt:canPerform

Property	agt:canPerform
IRI	https://www.spatialwebfoundation.org/ns/hsml/agent#canPerform
JSON name	canPerform
Requirement Level	**Mandatory**
Cardinality	1..*
Domain	agt:Agent
Range	act:ActivitySchema
Definition	Identifies the types of Activities the Agent is capable of performing.

Table 135—Property Definition: agt:canPerform *(continued)*

Usage Note	This property links an agent to the act:ActivitySchema templates for its skills. This is distinct from an act:Activity, which is a record of a specific execution.

11.2.3. Class: agt:Person

The class agt:Person realises the **PERSON** concept in IEEE P2874 § 6.3.2.1.6, a special subtype of agt:Agent that maintains a self-sovereign identity.

"A person type of Domain refers to entities belonging to the broader agential Domain, but maintaining the capability to control their own Self-Sovereign Identity."

agt:Person represents a human individual in the Spatial Web. While it inherits all the goal-directed and capability-driven characteristics of an agt:Agent, its primary distinction is its intrinsic link to a self-sovereign identity, ensuring individual control over personal data and interactions.

Key Requirements

– **Inheritance**: An agt:Person **SHALL** inherit all requirements from agt:Agent, including having a core:swid, agt:hasGoal, and agt:hasCapability.

– **Self-Sovereignty**: The core:swid of an agt:Person **SHALL** represent a self-sovereign identity, giving the individual ultimate control over their digital presence and credentials.

– **Personal Attributes**: An agt:Person **SHOULD** use properties from well-known vocabularies like schema.org to describe personal attributes for interoperability.

11.2.3.1. Class Definition

Table 136—Class Definition for agt:Person

RDF Class	**agt:Person**
Is Abstract	No
Definition	A human person, modeled as a special type of agt:Agent with a self-sovereign identity.
Subclass Of	agt:Agent
Usage Note	Represents human users who can initiate contracts and perform activities. The management of its identity and credentials **SHALL** adhere to the principles of self-sovereignty.
Rationale	Provides a specific class for human actors, distinguishing them from autonomous AI or robotic agents and ensuring their rights and privacy are structurally represented.

Table 137—Properties Summary for agt:Person

Predicate	**JSON-LD name**	**Description**	**Range**	**Cardinality**	**Requirement Level**
schema:givenName	givenName	The given name (first name) of the person.	xsd:string	0..1	Recommended
schema:familyName	familyName	The family name (last name) of the person.	xsd:string	0..1	Recommended
schema:email	email	An email address for the person.	xsd:string	0..*	Optional

11.2.3.2. Properties

This section provides the detailed normative definitions for properties commonly used with the agt:Person class. Note that this class also inherits all properties from agt:Agent.

11.2.3.2.1. Property: givenName

Table 138—Property Definition: schema:givenName

Property	schema:givenName
IRI	https://schema.org/givenName
JSON name	givenName
Requirement Level	Recommended
Cardinality	0..1
Domain	agt:Person
Range	xsd:string
Definition	The given name of the person.
Usage Note	Reuses the property from Schema.org for interoperability in representing personal names.

11.2.3.2.2. Property: familyName

Table 139—Property Definition: schema:familyName

Property	schema:familyName
IRI	https://schema.org/familyName
JSON name	familyName
Requirement Level	Recommended
Cardinality	0..1
Domain	agt:Person
Range	xsd:string
Definition	The family name of the person.
Usage Note	Reuses the property from Schema.org for interoperability in representing personal names.

11.2.3.2.3. Property: email

Table 140—Property Definition: schema:email

Property	schema:email
IRI	https://schema.org/email
JSON name	email
Requirement Level	Optional
Cardinality	0..*
Domain	agt:Person
Range	xsd:string
Definition	An email address for the person.
Usage Note	Reuses the property from Schema.org for contact information. Multiple email addresses are permissible.

11.2.4. Class: hsml:Organization

The class hsml:Organization realises the ORGANIZATION DOMAIN concept in IEEE P2874 § 6.6.3:

"An ORGANIZATION is a DOMAIN representing a group of Agents operating under shared governance, norms, or objectives within the Spatial Web."

hsml:Organization represents collective actors—such as companies, institutions, or public agencies—that initiate, govern, and coordinate Activities, issue Contracts, and manage subordinate Domains, Agents, and Things.

11.2.4.1. Class Definition

Table 141—Class Definition for hsml:Organization

RDF Class	hsml:Organization
Is Abstract	No
Definition	A collective group comprising Agents and/or Person under common governance, capable of issuing Contracts, Credentials, and norms.
Subclass Of	agt:Agent
Usage Note	Organizations model collective entities that issue norms, govern Agents, and manage subordinate domains.
Rationale	Provides the framework for representing legal entities, institutions, or other collective actors essential to polycentric governance and domain hierarchy in the Spatial Web.

11.2.5. Class: agt:Goal

The class agt:Goal realizes the concept of an objective that an agt:Agent is motivated to achieve. A Goal represents a state or objective that an Agent seeks to bring about through its activities.

It can be instantiated directly to represent a general-purpose goal or serve as a parent class for more specialized goal types (e.g., AchievementGoal, MaintenanceGoal, TBD). Every agt:Agent is required to be linked to one agt:Goal, making this concept fundamental to defining goal-directed behavior in the Spatial Web.

11.2.5.1. Class Definition

Table 142—Class Definition for agt:Goal

RDF Class	**agt:Goal**
Is Abstract	**No**
Definition	A state or objective that an Agent aims to achieve through the performance of Activities.
Subclass Of	Not specified in the provided text.
Usage Note	This class can be instantiated directly for general objectives. An LLM-based agent could use the description property for task planning.
Rationale	Provides a flexible, instantiable class for representing the objectives that drive autonomous, goal-directed actions in the Spatial Web.

11.2.5.2. Properties

This section provides the detailed normative definitions for the properties of the agt:Goal class.

11.2.5.2.1. Property: description

Table 143—Property Definition: schema:description

Property	schema:description
IRI	http://schema.org/description
JSON name	description
Requirement Level	**Mandatory**
Cardinality	**1..1**
Domain	agt:Goal
Range	xsd:string
Definition	A human-readable text description of the objective the Agent seeks to achieve.
Usage Note	This value should be a clear, unambiguous string that explains the goal.

11.2.5.2.2. Property: hasSubGoal

Table 144—Property Definition: agt:hasSubGoal

Property	agt:hasSubGoal
IRI	https://www.spatialwebfoundation.org/ns/hsml/agent#hasSubGoal
JSON name	subGoal
Requirement Level	**Optional**
Cardinality	0..* (zero to many)
Domain	agt:Goal
Range	agt:Goal
Definition	Links a Goal to one or more subordinate Goals that contribute to the fulfillment of the parent Goal.
Usage Note	This property enables the creation of goal trees or hierarchies (hierarchical goal decomposition), which is essential for complex planning.

12. HSML Communication Module

The HSML Communication Module realizes the **CHANNEL** concept of IEEE P2874 (§6.6.6), providing a semantic framework for modeling communication contexts and message exchange in the Spatial Web.

Where the **Governance Module** addresses rules and the **Activity Module** defines purposeful processes, the Communication Module focuses on how **entities interact during the execution of Activities**. It provides a minimal but extensible set of classes to represent **Channels, Messages, and Subscriptions** in a way that is compatible with both the Hyperspace Transaction Protocol (HSTP) and external communication systems.

The design follows three guiding intentions:

– To distinguish **ephemeral communication contexts** (Channels) from persistent identity contexts (Domains).

– To allow **semantic projection of messages** when provenance, traceability, or validation are required, without redefining HSTP transport.

– To ensure that **communication remains Activity-linked**, keeping transient exchanges grounded in HSML's enduring models of entities and processes.

The Communication Module thereby acts as the "conversation layer" of HSML: lightweight enough to support ad-hoc collaboration, yet formally defined so that message flows can participate in governance, provenance, and discovery across the Spatial Web.

12.1. Architectural Principles and Design Rationale

This clause explains the design choices behind the HSML Communication Module and how they satisfy IEEE P2874's requirements for **Channels and Message exchange** as described in §6.6.6. The principles are informative (non-normative) but motivate the normative class and property definitions introduced later.

12.1.1. Channels as Ephemeral Interaction Contexts

A core principle is that **Channels are not Domains**. While Domains represent enduring identity-bearing contexts (geographic, organizational, or conceptual), Channels are **ad-hoc and transient groupings** of HSML Entities around a purpose or Activity. P2874 defines a Channel as "a stream of HSML ENTITIES that are related to an ACTIVITY of a specific context that does not itself warrant a DOMAIN or hierarchy." This design ensures that Channels do not alter the permanent domain hierarchy but still allow session-like coordination.

- Channels are modeled as subclass of hsml:Entity but declared **disjoint from hsml:Domain**.
- Membership is flexible, but typically they are Agent, Person entities.
- Sub-channels allow hierarchical scoping of collaborative contexts without redefining domain hierarchies.

12.1.2. Messages as Optional Semantic Projections

The Communication Module distinguishes **wire messages** (defined by the Hyperspace Transaction Protocol, HSTP) from **semantic message entities**. HSTP envelopes move bytes; Channels may choose to **project messages** as HSML comm:Message entities when semantic traceability is required (e.g., for provenance, policy, or analytics). This duality prevents duplication of concepts:

- At minimum, Channel membership can simply reference Agents and Activities, with message exchange implicit in HSTP.
- When auditing, governance, or discovery are required, each HSTP envelope may be projected into a comm:Message entity with semantic annotations (sender, media type, content schema, relation to Channel).
- This pattern aligns with W3C PROV-O's distinction between events (Activities/Agents) and artifacts (Entities).

12.1.3. Separation of Concerns: HSTP vs HSML

The Communication Module does **not redefine transport**. HSTP specifies the abstract message envelope, headers, and bindings to HTTP, MQTT, QUIC, etc. The Communication Module only defines how such transported payloads are contextualized in HSML Channels. This ensures:

- **No duplication** of protocol details. Media types, signatures, and routing remain in HSTP.
- **Semantic overlay** only when needed: Channels can record provenance of what was exchanged, not how the bytes were framed.

– **Extensibility**: new bindings (e.g., WebRTC, QUIC streams) require no changes to HSML—Channels remain abstract contexts.

12.1.4. Activity-Linked Communication

Channels are always **linked to Activities**. This supports use cases such as:

– Real-time coordination of an Activity (e.g., multiple agents working on a shared task).

– Streaming sensor data into a Channel associated with a monitoring Activity.

– Collaborative annotation or decision-making during an Activity lifecycle.

The hsml:forActivity property ensures that Channels carry explicit purpose, grounding otherwise transient message streams in the enduring Activity model of HSML.

12.1.5. Governance Delegation

Policies and credential checks are not embedded directly in Channels. Instead:

– **Domain-level governance** defines who may create or join Channels.

– Channels may include optional hints (e.g., hsml:allowedAgent) but **formal credential gates and policy enforcement remain in the Governance Module**.

– This simplifies the Channel model while ensuring consistent enforcement across Activities, Domains, and Channels.

12.1.6. Interoperability and Extensibility

The design follows Semantic Web and Internet principles:

– Media type declarations in HSTP headers allow arbitrary payloads (JSON-LD, SHACL instance data, binary media).

– comm:Message entities can reference a comm:contentSchema (SHACL or JSON Schema) for validation, similar to how HTTP uses Content-Type with a profile parameter or Link rel=describedby.

– Sub-channel structures resemble chat rooms or topic hierarchies in existing systems (MQTT, XMPP), but retain formal semantics.

– Alignments with schema.org (schema:Message, schema:CommunicateAction) are possible for lightweight web interoperability.

12.1.7. Summary

The Communication Module provides a lightweight, semantically grounded way to model **Channels and Messages** without duplicating HSTP transport concerns or Domain governance rules. It supports ephemeral, Activity-linked collaboration while leaving transport framing to HSTP and policy enforcement to the Governance Module. This separation yields clarity, reuse, and flexibility across diverse use cases, from ad-hoc collaboration to persistent provenance trails.

12.2. Normative Classes

This clause provides the normative definitions for the classes of the Communication Module.

12.2.1. Summary of Communication Module Classes

Table 145—Summary of HSML Communication Module Classes

Class	Description
comm:Channel	Ephemeral, non-hierarchical, Domain-hosted context for grouping Entities and exchanging Messages around Activities.
comm:Message	Optional semantic description of a communicative artifact exchanged via HSTP. Used when provenance, validation, or audit are needed.
comm:Subscription	Declares an Agent's interest in receiving Messages from a Topic or Channel, optionally with filters and delivery hints.

12.2.2. Class: comm:Channel

The comm:Channel class realizes the **CHANNEL** concept as defined in the IEEE P2e874 draft standard. As a central element of the communications (comm) module, it provides a contextual container that links participants, data, and messages around specific Activities without forming part of the permanent domain hierarchy. This enables ad-hoc, session-based, or otherwise transient coordination and monitoring.

This class is a subclass of hsml:Entity, inheriting a hsml:swid as its unique identifier.

12.2.2.1. Key Requirements & Design Principles

- A comm:Channel **shall link** to at least one hsml:Activity it supports.
- A comm:Channel **shall be disjoint** from hsml:Domain, as it represents a non-hierarchical, transient grouping.
- A comm:Channel **shall include** one or more participants via the core:hasParticipant property.
- Parent-child relationships between channels **shall be modeled** using the inverse properties comm:hasSubChannel and comm:subChannelOf.
- Governance and access control **are intentionally delegated** to the Governance Module and are not embedded properties of a Channel.

12.2.2.2. Class Definition

Table 146—Class Definition for comm:Channel

RDF Class	comm:Channel
Is Abstract	No
Definition	A contextual, non-hierarchical grouping of entities associated with a specific Activity or purpose, for coordinating transient communications.
Subclass Of	hsml:Entity
Usage Note	Channels are for transient, ad-hoc collaboration; they do not define domain boundaries or permanent hierarchies. The core namespace refers to a separate module defining foundational, cross-cutting properties.
Rationale	Provides a lightweight mechanism for coordinating information and participants. Access control is delegated to the Governance Module; Channels do not embed their own access control lists (ACLs).

12.2.2.3. Properties Summary

Table 147—Properties Summary for comm:Channel

Predicate	JSON(-LD) Name	Description	Cardinality
rdf:type	@type	Declares the resource as an instance of comm:Channel.	1..n
hsml:swid	swid	The Spatial Web Identifier (SWID) inherited from hsml:Entity.	1
core:hasParticipant	hasParticipant	Identifies the entities participating in the Channel.	1..n
comm:forActivity	forActivity	Links the Channel to the Activity it supports.	1..n
comm:hasSubChannel	hasSubChannel	Links to a nested sub-channel.	0..n
comm:subChannelOf	subChannelOf	Links to the parent channel.	0..1

12.2.2.4. Property Details

12.2.2.4.1. Property: core:hasParticipant

Table 148—Property Definition: core:hasParticipant

Property	core:hasParticipant
IRI	https://schema.spatialwebfoundation.org/core#hasParticipant
JSON Name	hasParticipant
Requirement Level	Mandatory
Cardinality	1..n
Domain	comm:Channel
Range	hsml:Entity
Definition	Links the Channel to its participants (Agents, Things, or other Entities).
Usage Note	A Channel must have at least one participant to be meaningful.

12.2.2.4.2. Property: comm:forActivity

Table 149—Property Definition: comm:forActivity

Property	comm:forActivity
IRI	https://schema.spatialwebfoundation.org/comm#forActivity
JSON Name	forActivity
Requirement Level	Mandatory
Cardinality	1..n
Range	hsml:Activity
Definition	Associates the Channel with one or more Activities it supports or relates to.
Usage Note	Channels provide the necessary communication and data context for their referenced Activities.

12.2.2.4.3. Property: comm:hasSubChannel

Table 150—Property Definition: comm:hasSubChannel

Property	comm:hasSubChannel
IRI	https://schema.spatialwebfoundation.org/comm#hasSubChannel
JSON Name	hasSubChannel
Requirement Level	Optional
Cardinality	0..n
Range	comm:Channel
Inverse Property	comm:subChannelOf
Definition	Links the Channel to its sub-channels for hierarchical structuring of information streams.
Usage Note	This property enables efficient downward traversal of the Channel hierarchy (from parent to children).

12.2.2.4.4. Property: subChannelOf

Table 151—Property Definition: comm:subChannelOf

Property	comm:subChannelOf
IRI	https://schema.spatialwebfoundation.org/comm#subChannelOf
JSON Name	subChannelOf
Requirement Level	Optional
Cardinality	0..1
Range	comm:Channel
Inverse Property	comm:hasSubChannel
Definition	Links a sub-channel to its parent Channel.
Usage Note	This property allows for efficient upward traversal of the Channel hierarchy. The 0..1 cardinality enforces a strict tree structure, preventing a single channel from having multiple parents.

12.2.2.5. Example

```
{
  "@context": "http://spatialwebfoundation.org/context/hsml.jsonld",
  "@id": "did:swid:channel:metro-hospital-drone-ops",
  "@type": "Channel",
  "name": "Metro Hospital Drone Operations",
  "description": "Coordination channel for all emergency medical supply
deliveries in the downtown metro area.",
  "swid": "did:swid:channel:metro-hospital-drone-ops",
  "forActivity": "did:swid:activity:med-delivery-ops",
  "hasParticipant": [
    "did:swid:agent:drone-7",
    "did:swid:agent:ops-dashboard"
  ],
  "hasSubChannel": [
    {
      "@id": "did:swid:channel:drone-7-telemetry",
      "@type": "Channel",
```

```
      "name": "Drone 7 - Flight Telemetry",
      "description": "Real-time flight data, battery status, and
payload vitals for Drone 7.",
      "swid": "did:swid:channel:drone-7-telemetry",
      "forActivity": "did:swid:activity:med-delivery-ops",
      "hasParticipant": [
        "did:swid:agent:drone-7"
      ],
      "subChannelOf": "did:swid:channel:metro-hospital-drone-ops"
    }
  ]
}
```

Figure 10—Example of a Channel with a Sub-Channel using DIDs

12.2.3. Class: comm:Message

comm:Message models an optional semantic projection of a transported envelope (per HSTP) into HSML. It is used to persist provenance, governance, validation, discovery, and correlation of communications that occur within a chan:Channel and concern an hsml:Activity. Transport framing, headers, and bindings remain the responsibility of HSTP; comm:Message captures the who/what/when/why at the semantic layer.

Messages are not required for all exchanges. Implementations materialize comm:Message only when traceability or auditability is needed.

Key Requirements

a) A comm:Message shall be associated with exactly one chan:Channel via comm:inChannel.

b) A comm:Message shall be linked to at least one hsml:Activity via comm:aboutActivity.

c) A comm:Message shall identify at least one comm:sender of type hsml:Agent.

d) A comm:Message shall record a generation timestamp (e.g., prov:generatedAtTime).

e) A comm:Message should declare comm:mediaType and one or more comm:contentSchema to enable validation/routing.

f) A comm:Message may include integrity (comm:contentHash) and ordering (comm:sequenceNumber) metadata, and may correlate to other messages (comm:correlatesWith).

12.2.3.1. Class Definition

Table 152—Class Definition for comm:Message

RDF Class	comm:Message
Is Abstract	No
Definition	Semantic record of a communicative artifact exchanged within a Channel and about an Activity; optional overlay on HSTP transport.
Subclass Of	core:Concept
JSON-LD Term	“Message”: “comm:Message”
Usage Notes	Use to capture provenance (who/when), integrity (hash/sequence), typing (media/profile), and correlation (threads, request/response). Payload may be stored inline or referenced by immutable link.

12.2.3.2. Properties Summary

Table 153—Properties Summary for comm:Message

Predicate	JSON-LD name	Description	Range	Cardinality	Requirement
comm: inChannel	inChannel	Channel carrying this message (context).	chan:Channel	1..1	Mandatory
comm: aboutActivity	aboutActivity	Activity that this message concerns.	hsml:Activity	1..*	Mandatory
comm:sender	sender	Author/emitter of the message.	hsml:Agent	1..*	Mandatory
comm:recipient	recipient	Intended audience (Agents or Channels).	hsml:Agent or chan:Channel	0..*	Optional
comm: mediaType	mediaType	IANA media type (align with HSTP header).	xsd:anyURI	0..1	Recommended
comm: contentSchema	contentSchema	Schema/profile describing payload (SHACL, JSON Schema, etc.).	xsd:anyURI	0..*	Recommended
comm:content	content	Embedded payload or immutable link to content.	rdf:langString or xsd: string or xsd: base64Binary or xsd:anyURI	0..1	Optional
comm: contentHash	contentHash	Integrity hash (e.g., multihash) of payload.	xsd:string	0..1	Optional
comm: sequenceNumber	sequenceNumber	Monotonic sequence number within the Channel.	xsd:integer	0..1	Optional
comm: correlatesWith	correlatesWith	Correlation to another message (threading, request/response).	comm:Message	0..*	Optional
prov: generatedAtTime	generatedAtTime	UTC timestamp when produced.	xsd:dateTime	1..1	Mandatory
prov: wasAttributedTo	wasAttributedTo	Attribution of authorship (often same as comm:sender).	hsml:Agent	1..*	Mandatory

12.2.3.3. Properties

12.2.3.3.1. Property: inChannel

Table 154—Property Definition: comm:inChannel

Property	comm:inChannel
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#inChannel
JSON name	inChannel
Requirement Level	Mandatory
Cardinality	1..1
Domain	comm:Message
Range	chan:Channel
Definition	Binds the message to its carrying Channel (context).
Usage Note	Enables ordered archival, replay, and governance scoping per Channel.

12.2.3.3.2. Property: aboutActivity

Table 155—Property Definition: comm:aboutActivity

Property	comm:aboutActivity
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#aboutActivity
JSON name	aboutActivity
Requirement Level	Mandatory
Cardinality	1..*
Domain	comm:Message
Range	hsml:Activity
Definition	Identifies the Activity that the message concerns.
Usage Note	Grounds transient communication in enduring Activity provenance.

12.2.3.3.3. Property: sender

Table 156—Property Definition: comm:sender

Property	comm:sender
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#sender
JSON name	sender
Requirement Level	Mandatory
Cardinality	1..*
Domain	comm:Message
Range	agent:Agent
Definition	Agent that authored or emitted the message.
Usage Note	Often mirrored with prov:wasAttributedTo for PROV alignment.

12.2.3.3.4. Property: recipient

Table 157—Property Definition: comm:recipient

Property	comm:recipient
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#recipient
JSON name	recipient
Requirement Level	Optional
Cardinality	0..*
Domain	comm:Message
Range	hsml:Agent or chan:Channel
Definition	Intended audience of the message.
Usage Note	Use multiple values for broadcast/fan-out. Omit when Channel audience suffices.

12.2.3.3.5. Property: mediaType

Table 158—Property Definition: comm:mediaType

Property	comm:mediaType
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#mediaType

Table 158—Property Definition: comm:mediaType *(continued)*

JSON name	mediaType
Requirement Level	Recommended
Cardinality	0..1
Domain	comm:Message
Range	xsd:anyURI
Definition	IANA media type of the payload (align with HSTP).
Usage Note	Use IRI form for media types (e.g., https://iana.org/assignments/media-types/application/json).

12.2.3.3.6. Property: contentSchema

Table 159—Property Definition: comm:contentSchema

Property	comm:contentSchema
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#contentSchema
JSON name	contentSchema
Requirement Level	Recommended
Cardinality	0..*
Domain	comm:Message
Range	xsd:anyURI
Definition	Schema/profile describing payload (e.g., SHACL shape IRI, JSON Schema URL).
Usage Note	Enables validation and content negotiation by profile.

12.2.3.3.7. Property: content

Table 160—Property Definition: comm:content

Property	comm:content
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#content
JSON name	content
Requirement Level	Optional
Cardinality	0..1
Domain	comm:Message
Range	rdf:langString or xsd:string or xsd:base64Binary or xsd:anyURI
Definition	The payload itself (inline text/binary) or a canonical, immutable link to it.
Usage Note	Prefer immutable links plus comm:contentHash for large or sensitive payloads.

12.2.3.3.8. Property: contentHash

Table 161—Property Definition: comm:contentHash

Property	comm:contentHash
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#contentHash
JSON name	contentHash
Requirement Level	Optional
Cardinality	0..1

Table 161—Property Definition: comm:contentHash *(continued)*

Domain	comm:Message
Range	xsd:string
Definition	Integrity hash of the payload (e.g., multihash, SHA-256).
Usage Note	When comm:content is a link, the hash ensures immutability/verifiability.

12.2.3.3.9. Property: sequenceNumber

Table 162—Property Definition: comm:sequenceNumber

Property	comm:sequenceNumber
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#sequenceNumber
JSON name	sequenceNumber
Requirement Level	Optional
Cardinality	0..1
Domain	comm:Message
Range	xsd:integer
Definition	Monotonic ordering value relative to the Channel.
Usage Note	Useful for replay cursors and gap detection.

12.2.3.3.10. Property: correlatesWith

Table 163—Property Definition: comm:correlatesWith

Property	comm:correlatesWith
IRI	https://www.spatialwebfoundation.org/ns/hsml/comm#correlatesWith
JSON name	correlatesWith
Requirement Level	Optional
Cardinality	0..*
Domain	comm:Message
Range	comm:Message
Definition	Links to related messages (e.g., request/response, thread, saga).
Usage Note	Use multiple values to express multi-message conversations or branches.

12.2.3.3.11. Property: prov:generatedAtTime

Table 164—Property Definition: prov:generatedAtTime

Property	prov:generatedAtTime
IRI	http://www.w3.org/ns/prov#generatedAtTime
JSON name	generatedAtTime
Requirement Level	Mandatory
Cardinality	1..1
Domain	comm:Message
Range	xsd:dateTime
Definition	UTC timestamp when the message was produced.
Usage Note	Use Z-suffixed times or explicit offset; recommend millisecond precision.

12.2.3.3.12. Property: prov:wasAttributedTo

Table 165—Property Definition: prov:wasAttributedTo

Property	prov:wasAttributedTo
IRI	http://www.w3.org/ns/prov#wasAttributedTo
JSON name	wasAttributedTo
Requirement Level	Mandatory
Cardinality	1..*
Domain	comm:Message
Range	hsml:Agent
Definition	Attribution of authorship/agency for the message.
Usage Note	Often same individual(s) as comm:sender; keep both for PROV/tooling compatibility.

13. HSML Hyperspace Module: Normative Specification

13.1. Introduction

The Hyperspace Module defines the structural foundation for representing spatial relationships in HSML. It realizes the **HYPERSPACE** concept of IEEE P2874 (§ 6.2.1), which defines a hyperspace as:

> "A set whose elements are related by a formal notion of path, satisfying identity and composition laws. Paths may be abstract, as long as they can be traversed in a fashion akin to the paths between points in familiar spaces."

This abstract definition is intentionally broad. It allows HSML to treat a wide range of structures— geographic coordinates, social networks, organizational hierarchies, logical states, numeric values, or semantic concepts — under one unifying abstraction. Hyperspaces ensure that Domains are not just collections of entities, but **structured environments** in which relationships such as adjacency, connectivity, reachability, and composition can be explicitly represented and navigated.

Crucially, Hyperspaces in HSML are **made operational by explicit mappings**:

- **Elements** → classes or datatypes (e.g., geo:Geometry, cell:Cell, xsd:integer).
- **Arrows** (atomic steps) → RDF properties or edge classes (e.g., graph:edge, cell:adjacentTo).
- **Paths** (compositions) → explicit path resources (e.g., graph:Route, cell:CellChain) or SPARQL property paths.
- **Operations** → instances of hsml:Operation bound to these mappings, validated via SHACL profiles.

Through these mappings, navigation (neighbors, reachability, path composition) and operations (distance, similarity, routing) are not hard-coded into the ontology, but arise from **declared structures** that can be validated, extended, and executed. This makes the Hyperspace Module both **abstractly universal** and **practically concrete**.

Because the Spatial Web encompasses diverse domains, the Hyperspace Module is **modular by design**. The core module defines only the minimum semantics common to **all** Hyperspaces: elements, arrows, paths, and universal properties. Specializations provide richer semantics:

- GraphSpace — connectivity and networks.

- CellularSpace — grids and tessellations.
- VectorSpace — coordinate systems and geometries.
- MetricSpace — distance and similarity.
- Datatype Hyperspaces — value domains such as numbers, strings, colors, or tensors.

This modular structure allows implementers to import only what they need while preserving interoperability. Namespaces mirror this pattern: the core vocabulary resides in the hyperspace# namespace, while each specialization occupies its own sub-namespace under hyperspace/.

In this way, the Hyperspace Module provides both a **universal foundation** and a **scalable framework for extension**, making it a cornerstone of HSML and of the Spatial Web as a whole.

13.1.1. Navigation by Mapping

Navigation in a Domain's Hyperspace is not hard-coded; it is **achieved by mapping abstract constructs to RDF structures**:

- **Elements**
  - Mapped to an RDF class (e.g., graph:Node, cell:Cell) or a datatype (e.g., geosparql:wktLiteral, tensor:ArrayLiteral).
  - Optionally bound via a hyperspace:spatialProperty (e.g., geo:hasGeometry, hsml:cell) to point to the actual value.
- **Arrows (atomic steps)**
  - Mapped to an RDF property (e.g., graph:edge, cell:adjacentTo) connecting valid element instances.
  - Alternatively represented as a reified edge class with hsml:source and hsml:target, or as RDF-star annotated triples for weighted/labeled edges.
- **Paths (compositions)**
  - **Implicit:** expressed through SPARQL property paths over the declared arrow property (e.g., (graph:edge)+ for reachability).
  - **Explicit:** represented as RDF resources (e.g., graph:Route, cell:CellChain) with ordered hsml:step, hsml:startsAt, and hsml:endsAt properties.

These mappings allow core navigation queries such as: - **Neighbors:** ?a hsml:hasArrowType ?b - **Reachability:** ASK { ?a (?arrow)+ ?b } - **Path Extraction:** select explicit path resources linking two elements.

Thus, navigation is universally defined but concretely realized through the mapping pattern chosen for each Hyperspace specialization.

13.1.2. Design Decision: Paths with Literal Elements (Generalized)

In some Hyperspaces, hsml:hasElementType is a **datatype** (e.g., xsd:string, JSON literals, tensors, WKT). Because RDF literals cannot be subjects, arrows and paths **CANNOT** be expressed as direct triples between literals. This decision specifies **general, profile-able mapping patterns** and properties that work for **any** element kind (class- or datatype-based) and **any** path representation. (We include a geo:Geometry example purely illustratively.)

13.1.2.1. Normative Principles

P1 — No literal subjects	Arrows/paths SHALL NOT rely on triples requiring a literal subject.
P2 — Arrow-driven navigation	Navigation semantics (neighbors, reachability, composition) SHALL be derived from the Hyperspace's **arrow mapping** (predicate edge or reified edge), not from any serialized path payload.
P3 — Two interoperable mappings	Profiles MAY choose either **Value-Node** (recommended) or **Literal-End** mappings. Both are first-class in HSML; the choice MUST be advertised by the Hyperspace's mapping properties.
P4 — Path resource portability	Paths SHOULD be first-class resources (subclasses of hsml:Path) so implementations can attach metadata (cost, provenance) and validate steps with SHACL—independent of element representation.

13.1.2.2. Patterns

Pattern L1 — Value-Node (RECOMMENDED)	Materialize each element as a **resource node** (instances of a class declared by hsml:hasElementType), and carry the raw value via a datatype property (hsml:elementValue or a domain-standard property). Arrows are normal object properties (hsml:arrowProperty) or reified edge instances (hsml:arrowClass with hsml:arrowSource / hsml:arrowTarget). Paths use hsml:startsAt / hsml:endsAt and optional hsml:step.
Pattern L2 — Literal-End (SUPPORTED)	Keep elements as **literals** (i.e., hsml:hasElementType is a datatype). Represent arrows as **reified edge resources** with **literal endpoints** (hsml:arrowSourceValue, hsml:arrowTargetValue). Represent paths as **resources** whose endpoints/steps are **literal-valued** (hsml:startsAtValue, hsml:endsAtValue, hsml:stepList or hsml:pathLiteral).

NOTE—L1 enables property-path traversals and richer per-element metadata; L2 avoids node materialization at the cost of reified navigation. Choose per data-volume, query needs, and tooling.

13.1.3. Separation of Domain and Hyperspace

Domain provides identity and contextual scope. **Hyperspace** attached to a Domain defines how its entities relate. This separation allows the same Domain to be interpreted under different logics (graph, metric, cellular, temporal) without altering identity.

13.1.4. Universal Operations

A small algebra applies to all Hyperspaces:

- **Identity** — every element is reachable from itself.
- **Reachability** — test existence of a path between elements.
- **Path Composition** — concatenate two compatible paths.
- **Subspace Formation** — restrict a Hyperspace to a subset of elements and induced arrows.

Specialized Hyperspaces extend this with richer operations (e.g., shortest path, similarity metrics).

13.1.5. Modularity and Namespaces

Namespaces follow a predictable hierarchy:

- Core: https://www.spatialwebfoundation.org/ns/hsml/hyperspace#
- GraphSpace: https://www.spatialwebfoundation.org/ns/hsml/hyperspace/graph#
- CellularSpace: …/cell#
- VectorSpace: …/vector#
- MetricSpace: …/metric#
- Datatype Hyperspaces: …/datatype#

This structure ensures clarity, modularity, and extensibility.

By organizing namespaces in this way, several goals are achieved:

- **Clarity** — It is immediately clear whether a term belongs to the core Hyperspace abstraction or a specialized extension.
- **Modularity** — Implementers can import only the extensions they need, keeping systems lightweight.
- **Extensibility** — New families of Hyperspaces (e.g., probabilistic space, tensor spaces, state-machine spaces) can be introduced simply by assigning them new sub-namespaces, without disrupting existing definitions.
- **Best Practices** — This mirrors established approaches in ontology engineering, where namespaces are stable, descriptive, and composable.

In effect, the namespace design mirrors the architecture of Hyperspaces themselves: a simple core scaffold that can be specialized into many forms, each with its own dedicated space but all interoperable under a common root.

13.1.6. Hyperspace of Hyperspaces

A Hyperspace is itself an Entity and may serve as an element within another Hyperspace. This enables **Hyperspaces of Hyperspaces**, supporting higher-order composition:

- **Systems-of-Systems** — city networks combined into a national network.
- **Federations** — regional grids aggregated into a global monitoring system.
- **Holarchies** — nested Hyperspaces reflecting P2874's holonic design principles.

This higher-order capability is essential for modeling federated and polycentric structures in the Spatial Web.

13.1.7. Time as an Extension

Temporal semantics are not universal and are therefore modeled as extensions. Arrows and paths may carry temporal annotations (time:validDuring, time:hasDuration) using OWL-Time. Profiles may define spatio-temporal Hyperspaces when time is central to navigation and reasoning.

13.1.8. Identity and Governance

Every Hyperspace is identified by a **Spatial Web Identifier (SWID)** conformant with W3C DID Core. This ensures resolvability and governance. While governance logic is external, the identifier anchors policies, credentials, and trust enforcement.

13.1.9. Validation Through Profiles

Constraints are expressed via **SHACL profiles**, not fixed in OWL. Profiles may restrict element types, adjacency properties, or valid transitions, providing rigor in specific deployments while preserving generality in the core.

Summary:

The Hyperspace Module defines elements, arrows, paths, and operations as a minimal universal abstraction. By mapping these to RDF constructs, navigation and reasoning are enabled consistently across all domains. SHACL profiles enforce structural integrity, while modular namespaces and higher-order composition provide extensibility. Together, these design decisions make the Hyperspace Module a cornerstone of HSML and the Spatial Web.

13.2. Normative Classes

This clause provides the normative definitions for the classes of the **Hyperspace Module**.

13.2.1. Summary of Hyperspace Module Classes

Table 166—Summary of Hyperspace Module Classes

Class	Description
hspace:Hyperspace	Domain-attached structural abstraction that declares element, arrow, and path **types** and optional mappings for navigation and operations.
hspace:Path	First-class representation of composed paths (finite compositions of arrows); may carry endpoints, ordered steps, and serialized path values.
hspace:Operation	Declarative operation bound to a Hyperspace (e.g., reachability, routing, metric evaluation), referencing the mappings it consumes.
hspace:HyperspaceOfHyperspace	Higher-order space whose elements are themselves Hyperspaces; supports federation, holarchies, and systems-of-systems.

13.2.2. Class: hspace:Hyperspace

The class hspace:Hyperspace realises the HYPERSPACE concept in IEEE P2874 § 6.6.3 and Hyperspace Modeling principles:

"A HYPERSPACE is a spatial structure that specifies relationships among elements (points) in a DOMAIN, including notions such as adjacency, distance, connectivity, or metric, enabling spatial reasoning and operations within the Spatial Web."

hspace:Hyperspace provides the formal structure for expressing the spatial logic that governs how entities in a Domain relate to each other. It generalizes topology, metric spaces, graphs, cellular grids, vector spaces, and datatype spaces into a unified model, supporting composable and navigable relationships across Domains.

This specification separates:

- **Hyperspace-level properties** — what the space **is** and how navigation is driven (element type, arrow type, path type, optional edge predicate).
- **Element-class properties** — properties used **on the element instances** (when elements are resources).
- **Path-class properties** — properties used **on path instances** (explicit paths), including variants for literal-based elements.

13.2.2.1. Class Definition

Table 167—Class Definition for hspace:Hyperspace

RDF Class	hspace:Hyperspace
Is Abstract	No
Definition	A spatial structure attached to a Domain that defines relationships among its elements, supporting spatial reasoning, navigation, and operations.
Subclass Of	hspace:Entity
Usage Note	Hyperspaces specify how elements in a Domain relate (e.g., adjacency, distance) and drive spatial queries, validation, and tooling.
Rationale	Provides a formal, flexible mechanism to encode spatial structure (graph, cellular, vector, metric, datatype) that underpins Spatial Web reasoning.

13.2.2.2. Hyperspace Properties (Summary)

Table 168—Properties Summary for hspace:Hyperspace

Predicate	JSON(-LD) name	Description	Cardinality
rdf:type	@type	Declares the resource as an instance of hspace: Hyperspace.	1..n
hspace:swid	swid	Spatial Web Identifier conformant with W3C DID Core.	1
hspace: hasElementType	hasElementType	Type of elements (class or datatype) represented in the Hyperspace.	1
hspace: hasArrowType	hasArrowType	Type of atomic one-step relations (“arrows”) between elements.	1
hspace: hasPathType	hasPathType	Type(s) of composed paths recognised in the Hyperspace.	0..n
hspace: arrowProperty	arrowProperty	(Optional) RDF predicate used for atomic steps (direct edges).	0..1
hspace: hasOperation	hasOperation	Declares supported operations or transformations for the Hyperspace.	0..n

13.2.2.2.1. Property: rdf:type

Table 169—Property Definition: rdf:type

Property	rdf:type
IRI	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
JSON name	@type
Requirement Level	Mandatory
Cardinality	1..n
Domain	hspace:Hyperspace

Table 169—Property Definition: rdf:type *(continued)*

Range	hspace:Hyperspace
Definition	Declares the resource as an instance of hspace:Hyperspace.

13.2.2.2.2. Property: hasElementType

Table 170—Property Definition: hspace:hasElementType

Property	hspace:hasElementType
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#hasElementType
JSON name	hasElementType
Requirement Level	Mandatory
Cardinality	1
Domain	hspace:Hyperspace
Range	rdfs:Class or datatype IRI
Definition	Identifies the type of elements (points) comprising the Hyperspace.
Usage Note	Examples: geo:Geometry, hspace:Cell, xsd:string, geo:wktLiteral.

13.2.2.2.3. Property: hasArrowType

Table 171—Property Definition: hspace:hasArrowType

Property	hspace:hasArrowType
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#hasArrowType
JSON name	hasArrowType
Requirement Level	Mandatory
Cardinality	1
Domain	hspace:Hyperspace
Range	owl:ObjectProperty IRI or rdfs:Class
Definition	Identifies the atomic one-step relation (“arrow”) connecting elements.
Usage Note	Use an object property IRI for direct edges (e.g., ex:connect). If using reified edges, hasArrowType MAY point to the edge class (defined outside this section).

13.2.2.2.4. Property: hasPathType

Table 172—Property Definition: hspace:hasPathType

Property	hspace:hasPathType
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#hasPathType
JSON name	hasPathType
Requirement Level	Optional
Cardinality	0..n
Domain	hspace:Hyperspace
Range	rdfs:Class or datatype IRI
Definition	Identifies the type(s) of composed paths (finite compositions of arrows) recognised in the Hyperspace.
Usage Note	Examples: hspace:Path, ex:Route, vector:LineString, geo:wktLiteral (LINESTRING).

13.2.2.2.5. Property: arrowProperty

Table 173—Property Definition: hspace:arrowProperty

Property	hspace:arrowProperty
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#arrowProperty
JSON name	arrowProperty
Requirement Level	Optional
Cardinality	0..1
Domain	hspace:Hyperspace
Range	owl:ObjectProperty IRI
Definition	Declares the RDF predicate used to encode atomic steps (arrows) as direct edges.
Usage Note	Enables reachability via SPARQL property paths, e.g., (?arrow).

13.2.2.2.6. Property: hasOperation

Table 174—Property Definition: hspace:hasOperation

Property	hspace:hasOperation
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#hasOperation
JSON name	hasOperation
Requirement Level	Optional
Cardinality	0..n
Domain	hspace:Hyperspace
Range	hspace:Operation
Definition	Declares supported operations (e.g., reachability, routing, subspace extraction, metric evaluation).

13.2.3. Element-Class Properties

These properties are used **on element instances** (when hspace:hasElementType is a class). They are not properties of the hspace:Hyperspace resource itself.

13.2.3.1. Summary

Table 175—Properties Summary for Element-Class

Predicate	JSON(-LD) name	Description	Cardinality
hspace:elementValue	elementValue	Carries the element's literal value on the element node (when applicable).	0..1

13.2.3.1.1. Property: elementValue

Table 176—Property Definition: hspace:elementValue

Property	hspace:elementValue
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#elementValue

Table 176—Property Definition: hspace:elementValue *(continued)*

JSON name	elementValue
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasElementType (when that range is a class)
Range	rdfs:Literal (typed per profile, e.g., geo:wktLiteral, xsd:string)
Definition	Stores a literal value on an element node, enabling arrows between resources while retaining the raw value.

13.2.4. Path-Class Properties

These properties are used **on path instances** (i.e., resources of the class named by hspace:hasPathType when that range is a class). They are not properties of the hspace:Hyperspace resource itself.

13.2.4.1. Summary

Table 177—Properties Summary for Path-Class

Predicate	JSON(-LD) name	Description	Cardinality
hspace:startsAt	startsAt	Links a path to its start **element node** (resource-based elements).	0..1
hspace:endsAt	endsAt	Links a path to its end **element node** (resource-based elements).	0..1
hspace:pathStep	pathStep	Ordered steps of a path (elements/edges/step-nodes per profile).	0..1
hspace:onPath	onPath	Membership assertion that an element node lies on the path.	0..1
hspace:startsAtValue	startsAtValue	Start **literal** of the path (literal-based elements).	0..1
hspace:endsAtValue	endsAtValue	End **literal** of the path (literal-based elements).	0..1
hspace:stepList	stepList	rdf:List of ordered **literal** elements (literal-based elements).	0..1
hspace:pathValue	pathValue	Serialized path payload (e.g., LineString, JSON array/ polyline).	0..1

13.2.4.1.1. Property: startsAt

Table 178—Property Definition: hspace:startsAt

Property	hspace:startsAt
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#startsAt
JSON name	startsAt
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasPathType (when that range is a class)
Range	Element class named by hspace:hasElementType (when that range is a class)
Definition	Links a path resource to its start **element node** (resource-based elements).

13.2.4.1.2. Property: endsAt

Table 179—Property Definition: hspace:endsAt

Property	hspace:endsAt
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#endsAt
JSON name	endsAt
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasPathType (when that range is a class)
Range	Element class named by hspace:hasElementType (when that range is a class)
Definition	Links a path resource to its end **element node** (resource-based elements).

13.2.4.1.3. Property: pathStep

Table 180—Property Definition: hspace:pathStep

Property	hspace:pathStep
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#pathStep
JSON name	pathStep
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasPathType (when that range is a class)
Range	owl:ObjectProperty IRI
Definition	Ordered property listing the steps of a path. Steps MAY reference elements, edges, or step nodes per profile.

13.2.4.1.4. Property: onPath

Table 181—Property Definition: hspace:onPath

Property	hspace:onPath
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#onPath
JSON name	onPath
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasPathType (when that range is a class)
Range	Element class named by hspace:hasElementType (when that range is a class)
Definition	Indicates that an element node lies on this explicit path.

13.2.4.1.5. Property: startsAtValue

Table 182—Property Definition: hspace:startsAtValue

Property	hspace:startsAtValue
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#startsAtValue
JSON name	startsAtValue
Requirement Level	Optional
Cardinality	0..1

Table 182—Property Definition: hspace:startsAtValue *(continued)*

Domain	Class named by hspace:hasPathType (when that range is a class)
Range	rdfs:Literal (typed with the datatype named by hspace:hasElementType when it is a datatype)
Definition	Records the **start literal** of the path when elements are literals.

13.2.4.1.6. Property: endsAtValue

Table 183—Property Definition: hspace:endsAtValue

Property	hspace:endsAtValue
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#endsAtValue
JSON name	endsAtValue
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasPathType (when that range is a class)
Range	rdfs:Literal (typed with the datatype named by hspace:hasElementType when it is a datatype)
Definition	Records the **end literal** of the path when elements are literals.

13.2.4.1.7. Property: stepList

Table 184—Property Definition: hspace:stepList

Property	hspace:stepList
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#stepList
JSON name	stepList
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasPathType (when that range is a class)
Range	rdf:List
Definition	Points to an RDF Collection whose items are the ordered **literal elements** constituting the path (used when elements are literals).

13.2.4.1.8. Property: pathValue

Table 185—Property Definition: hspace:pathValue

Property	hspace:pathValue
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#pathValue
JSON name	pathValue
Requirement Level	Optional
Cardinality	0..1
Domain	Class named by hspace:hasPathType (when that range is a class)
Range	rdfs:Literal
Definition	Serialized path payload for visualization or exchange (e.g., WKT/GeoJSON LineString, JSON array/polyline).

13.2.5. Class: hspace:Path

The class hspace:Path represents a **composed path**: a finite composition of arrows in a given Hyperspace. It provides an identifiable resource on which systems can attach metadata, validate step coherence, and (optionally) serialize a concrete realization (e.g., a LineString or JSON sequence), without altering the core arrow-based semantics.

13.2.5.1. Class Definition

Table 186—Class Definition for hspace:Path

RDF Class	hspace:Path
Is Abstract	No
Definition	A first-class representation of a composed path (finite sequence of arrows) within a Hyperspace.
Subclass Of	hspace:Entity
Usage Note	Use when a path requires identity, metadata, validation, or serialization; otherwise, implicit path composition via arrow mappings suffices.
Rationale	Enables reproducible, governed, and interoperable exchange of paths (IDs, provenance, metrics) while keeping navigation arrow-driven.

13.2.5.2. Properties Summary

Table 187—Property Definitions for hspace:Path

Predicate	JSON(-LD) name	Description	Cardinality
hspace:startsAt	startsAt	Start **element node** (when elements are resources).	0..1
hspace:endsAt	endsAt	End **element node** (when elements are resources).	0..1
hspace:pathStep	pathStep	Ordered steps (elements, edges, or step nodes per profile).	0..1
hspace:onPath	onPath	Membership assertion that an element node lies on this path.	0..n
hspace:startsAtValue	startsAtValue	Start **literal** (when elements are literals).	0..1
hspace:endsAtValue	endsAtValue	End **literal** (when elements are literals).	0..1
hspace:stepList	stepList	rdf:List of ordered **literal** elements (when elements are literals).	0..1
hspace:pathValue	pathValue	Serialized path payload (e.g., WKT/GeoJSON LineString, JSON array/polyline).	0..1

13.2.5.2.1. Property: startsAt

Table 188—Property Definition: hspace:startsAt

Property	hspace:startsAt
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#startsAt
JSON name	startsAt
Requirement Level	Optional (resource-element profiles)
Cardinality	0..1
Domain	hspace:Path (and subclasses)

Table 188—Property Definition: hspace:startsAt *(continued)*

Range	owl:ObjectProperty IRI (targeting the element class named by hspace: hasElementType)
Definition	Links a path to its start **element node** when elements are resources.
Usage Note	For literal-element profiles, use hspace:startsAtValue.

13.2.5.2.2. Property: endsAt

Table 189—Property Definition: hspace:endsAt

Property	hspace:endsAt
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#endsAt
JSON name	endsAt
Requirement Level	Optional (resource-element profiles)
Cardinality	0..1
Domain	hspace:Path (and subclasses)
Range	owl:ObjectProperty IRI (targeting the element class named by hspace: hasElementType)
Definition	Links a path to its end **element node** when elements are resources.
Usage Note	For literal-element profiles, use hspace:endsAtValue.

13.2.5.2.3. Property: pathStep

Table 190—Property Definition: hspace:pathStep

Property	hspace:pathStep
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#pathStep
JSON name	pathStep
Requirement Level	Optional
Cardinality	0..1
Domain	hspace:Path (and subclasses)
Range	owl:ObjectProperty IRI
Definition	Ordered property listing the steps of a path. Steps MAY reference elements, edges, or step nodes per profile.
Usage Note	Ordering MAY be expressed via RDF Collections (e.g., rdf:List) or sh:order metadata on step nodes.

13.2.5.2.4. Property: onPath

Table 191—Property Definition: hspace:onPath

Property	hspace:onPath
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#onPath
JSON name	onPath
Requirement Level	Optional
Cardinality	0..n
Domain	hspace:Path (and subclasses)
Range	owl:ObjectProperty IRI (targeting element nodes)
Definition	Indicates that an element node lies on this explicit path.

13.2.5.2.5. Property: startsAtValue

Table 192—Property Definition: hspace:startsAtValue

Property	hspace:startsAtValue
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#startsAtValue
JSON name	startsAtValue
Requirement Level	Optional (literal-element profiles)
Cardinality	0..1
Domain	hspace:Path (and subclasses)
Range	rdfs:Literal (typed with the Hyperspace's hspace:hasElementType datatype)
Definition	Records the **start literal** of the path when elements are literals.

13.2.5.2.6. Property: endsAtValue

Table 193—Property Definition: hspace:endsAtValue

Property	hspace:endsAtValue
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#endsAtValue
JSON name	endsAtValue
Requirement Level	Optional (literal-element profiles)
Cardinality	0..1
Domain	hspace:Path (and subclasses)
Range	rdfs:Literal (typed with the Hyperspace's hspace:hasElementType datatype)
Definition	Records the **end literal** of the path when elements are literals.

13.2.5.2.7. Property: stepList

Table 194—Property Definition: hspace:stepList

Property	hspace:stepList
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#stepList
JSON name	stepList
Requirement Level	Optional (literal-element profiles)
Cardinality	0..1
Domain	hspace:Path (and subclasses)
Range	rdf:List
Definition	Points to an RDF Collection whose items are the ordered **literal elements** that constitute the path.
Usage Note	Each list item **MUST** be typed with the datatype named by hspace: hasElementType when elements are literals.

13.2.5.2.8. Property: pathValue

Table 195—Property Definition: hspace:pathValue

Property	hspace:pathValue
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#pathValue
JSON name	pathValue

Table 195—Property Definition: hspace:pathValue *(continued)*

Requirement Level	Optional
Cardinality	0..1
Domain	hspace:Path (and subclasses)
Range	rdfs:Literal
Definition	Serialized realization of the path for rendering or exchange (e.g., WKT/ GeoJSON LineString, JSON array/polyline).
Usage Note	Navigation semantics (reachability) are derived from arrow mappings, not from this serialization.
NOTE— **Conformance.** A profile that names a class in hspace:hasPathType MAY use hspace:Path directly or a subclass thereof (e.g., ex:Route, vector:LineString), applying the properties above as appropriate to the element representation (resource-based or literal-based).	

13.2.6. Class: hspace:Operation

The class hspace:Operation declares a **portable operation** that can be applied to a given Hyperspace (e.g., reachability, shortest path, similarity routing, subspace formation). An Operation binds to the Hyperspace's declared mappings (element/arrow/path) so that different implementations can interoperate while using the same inputs and producing comparable outputs.

13.2.6.1. Class Definition

Table 196—Class Definition for hspace:Operation

RDF Class	hspace:Operation
Is Abstract	No
Definition	A declarative, reusable operation over a Hyperspace that references the mappings it consumes and the types it produces.
Subclass Of	hspace:Entity
Usage Note	A Hyperspace links to its supported operations with hspace:hasOperation. An Operation MAY be reused by multiple Hyperspaces if their mappings are compatible.
Rationale	Separates **what** to compute from **how** the Hyperspace is encoded, enabling consistent contracts for queries, analytics, validation, and services.

13.2.6.2. Properties Summary

Table 197—Property Definitions for hspace:Operation

Predicate	JSON(-LD) name	Description	Cardinality
rdf:type	@type	Declares the resource as an instance of hspace:Operation.	1..n
hspace: usesArrowProperty	usesArrowProperty	Binds the operation to the **atomic edge predicate** it traverses (when arrows are direct triples).	0..1
hspace: usesArrowClass	usesArrowClass	Binds the operation to the **reified edge class** it traverses (when arrows are edge resources).	0..1
hspace: usesAnnotationProperty	usesAnnotationProperty	Declares which **edge annotation properties** (e.g., weight, cost) the operation consumes.	0..n
hspace: returnsPathClass	returnsPathClass	Declares the **path class** produced (if the operation returns explicit paths).	0..1
hspace: returnsValueType	returnsValueType	Declares the **datatype or class** of the scalar/complex result (e.g., xsd:boolean, xsd:decimal, prov:Entity).	0..n

Table 197—Property Definitions for hspace:Operation *(continued)*

Predicate	JSON(-LD) name	Description	Cardinality
hspace: parameterShape	parameterShape	SHACL shape describing the **named parameters** and their constraints for invoking the operation.	0..1
hspace: implementation	implementation	IRI of an **algorithm, function, or service** (e.g., prov: Plan, API endpoint) that realizes this operation.	0..n

13.2.6.2.1. Property: rdf:type

Table 198—Property Definition: rdf:type

Property	rdf:type
IRI	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
JSON name	@type
Requirement Level	Mandatory
Cardinality	1..n
Domain	hspace:Operation
Range	hspace:Operation
Definition	Declares the resource as an instance of hspace:Operation.

13.2.6.2.2. Property: usesArrowProperty

Table 199—Property Definition: hspace:usesArrowProperty

Property	hspace:usesArrowProperty
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#usesArrowProperty
JSON name	usesArrowProperty
Requirement Level	Optional
Cardinality	0..1
Domain	hspace:Operation
Range	owl:ObjectProperty IRI
Definition	Binds the operation to the arrow **predicate** to traverse (e.g., ex:connect).
Usage Note	Use this when the Hyperspace uses direct RDF edges (hspace:arrowProperty). Mutually exclusive with hspace:usesArrowClass in a single invocation context.

13.2.6.2.3. Property: usesArrowClass

Table 200—Property Definition: hspace:usesArrowClass

Property	hspace:usesArrowClass
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#usesArrowClass
JSON name	usesArrowClass
Requirement Level	Optional
Cardinality	0..1
Domain	hspace:Operation
Range	rdfs:Class
Definition	Binds the operation to the **edge class** to traverse (reified edges, e.g., ex:Edge).

Table 200—Property Definition: hspace:usesArrowClass *(continued)*

Usage Note	Pair with endpoint properties defined in the profile (e.g., hspace: arrowSource / hspace:arrowTarget or their literal variants at execution time).

13.2.6.2.4. Property: usesAnnotationProperty

Table 201—Property Definition: hspace:usesAnnotationProperty

Property	hspace:usesAnnotationProperty
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#usesAnnotationProperty
JSON name	usesAnnotationProperty
Requirement Level	Optional
Cardinality	0..n
Domain	hspace:Operation
Range	rdf:Property IRI
Definition	Declares which **edge annotations** the operation reads (e.g., ex:weight, ex: capacity, rdfs:label).
Usage Note	For shortest path, bind the weight property; for multi-criteria, list multiple properties and constrain via hspace:parameterShape.

13.2.6.2.5. Property: returnsPathClass

Table 202—Property Definition: hspace:returnsPathClass

Property	hspace:returnsPathClass
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#returnsPathClass
JSON name	returnsPathClass
Requirement Level	Optional
Cardinality	0..1
Domain	hspace:Operation
Range	rdfs:Class
Definition	Declares the path **class** produced by the operation when it returns explicit paths (e.g., ex:Route, vector:LineString).
Usage Note	If omitted, the operation likely produces a scalar or set (see hspace: returnsValueType).

13.2.6.2.6. Property: returnsValueType

Table 203—Property Definition: hspace:returnsValueType

Property	hspace:returnsValueType
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#returnsValueType
JSON name	returnsValueType
Requirement Level	Optional
Cardinality	0..n
Domain	hspace:Operation
Range	rdfs:Class or datatype IRI
Definition	Declares the non-path **result type(s)** (e.g., xsd:boolean for reachability, xsd: decimal for distance, prov:Entity for artifacts).

13.2.6.2.7. Property: parameterShape

Table 204—Property Definition: hspace:parameterShape

Property	hspace:parameterShape
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#parameterShape
JSON name	parameterShape
Requirement Level	Optional
Cardinality	0..1
Domain	hspace:Operation
Range	sh:NodeShape
Definition	SHACL shape describing required/optional **parameters** (e.g., ex:source, ex:target, ex:k, ex:maxCost), their datatypes, and constraints.
Usage Note	Encourages portable invocation contracts across engines.

13.2.6.2.8. Property: implementation

Table 205—Property Definition: hspace:implementation

Property	hspace:implementation
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace#implementation
JSON name	implementation
Requirement Level	Optional
Cardinality	0..n
Domain	hspace:Operation
Range	IRI (e.g., prov:Plan, code/package/service endpoint)
Definition	Identifies an algorithm, plan, or service that realizes the operation.
Usage Note	Use prov:wasAssociatedWith / prov:used alongside this property for full provenance, if desired.
NOTE— **Binding model.** A Hyperspace links to operations via hspace:hasOperation. Each operation declares how it binds to the Hyperspace's mappings: - **Direct edges** → hspace:usesArrowProperty. - **Reified edges** → hspace:usesArrowClass (and profile-known source/target predicates). - **Weights/labels** → hspace:usesAnnotationProperty. Outputs are described by hspace:returnsPathClass and/or hspace:returnsValueType. Parameters are validated by hspace:parameterShape.	

13.2.6.3. Minimal Example (Informative)

```
@prefix hspace: <https://www.spatialwebfoundation.org/ns/hsml/
hyperspace#> .
@prefix ex:     <https://example.org/ns/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .

# Hyperspace (excerpt)
ex:RoadNet a hspace:Hyperspace ;
  hspace:hasElementType ex:Intersection ;
  hspace:hasArrowType   ex:connectsTo ;
  hspace:arrowProperty  ex:connectsTo ;
  hspace:hasPathType    ex:Route ;
  hspace:hasOperation   ex:ShortestPath .
```

```
# Operation bound to the arrow predicate and weight annotation
ex:ShortestPath a hspace:Operation ;
  hspace:usesArrowProperty ex:connectsTo ;
  hspace:usesAnnotationProperty ex:travelTime ;
  hspace:returnsPathClass ex:Route ;
  hspace:returnsValueType xsd:decimal ;       # total cost
  hspace:implementation <https://svc.example.org/ops/shortest-path> .
```

Figure 11

13.2.7. Class: hspace:HyperspaceOfHyperspace

hspace:HyperspaceOfHyperspace is a **higher-order Hyperspace** whose **elements are themselves Hyperspaces**. It enables composition, federation, and holarchic organization of spaces (systems-of-systems), allowing navigation and operations to traverse relationships among multiple constituent Hyperspaces without collapsing their internal structure.

13.2.7.1. Class Definition

Table 206—Class Definition for hspace:HyperspaceOfHyperspace

RDF Class	hspace:HyperspaceOfHyperspace
Is Abstract	No
Definition	A Hyperspace whose element type is hspace:Hyperspace, supporting relations and composed paths among Hyperspaces (e.g., containment, interoperability, federation).
Subclass Of	hspace:Hyperspace
Usage Note	Use when modeling networks or hierarchies of Hyperspaces (e.g., regional grids aggregated into a national grid; organizational spaces federated into a sector space). Internal structures of member Hyperspaces remain intact and navigable via their own declarations.
Rationale	Provides a principled way to build **holarchies** and **federations** in the Spatial Web: spaces as elements of larger spaces, with clear separation of concerns and reusable navigation semantics.

13.2.7.2. Specialization Constraints (Normative)

– **Element Type.** hspace:hasElementType **MUST** be hspace:Hyperspace.

– **Arrow Type.** hspace:hasArrowType **MAY** denote relations **between Hyperspaces** (e.g., interoperability, containment, dependency). The specific predicate (if using direct edges) is given by hspace:arrowProperty; if using reified edges, profiles SHALL provide the edge class and endpoint predicates.

– **Path Type.** hspace:hasPathType **MAY** be declared to represent composed relations across multiple Hyperspaces (e.g., federation traversal). Path instances (if used) employ the standard **Path-class properties** (hspace:startsAt, hspace:endsAt, hspace:pathStep, etc.).

NOTE—This class **reuses the same Hyperspace-level, Element-class, and Path-class property model** defined for hspace:Hyperspace. The only additional constraint is that **elements are Hyperspaces**. Consequently, any path over this space is a chain of Hyperspaces, and any arrow relates one Hyperspace to another.

13.2.7.3. Example: Holarchy (Hyperspace of Hyperspaces)

This example models a **higher-order Hyperspace** whose **elements are themselves Hyperspaces** (holons). It preserves each member Hyperspace's internal logic while expressing **whole–part** links across scales.

- **What is modeled.** ex:Holarchy contains ex:RoomSpace, ex:BuildingSpace, ex:DistrictSpace, ex:CitySpace—each a hspace:Hyperspace.
- **Arrow semantics.** ex:containsHolon is the **atomic arrow** (one step) relating Hyperspace → Hyperspace (Room → Building → District → City). Connectivity across levels is computed by arrow composition, e.g., (ex:containsHolon)+.
- **Explicit path (optional).** hspace:Path records a composed traversal (with identity and lightweight metadata). It **complements** arrow-based navigation; arrows remain authoritative for reachability.

NOTE—Conformance reminders: - The enclosing space sets hspace:hasElementType = hspace:Hyperspace. - The arrow mapping relates Hyperspace → Hyperspace (hspace:hasArrowType / hspace:arrowProperty). - If used, hspace:Path endpoints target Hyperspace instances.

```
@prefix hspace: <https://www.spatialwebfoundation.org/ns/hsml/
hyperspace#> .
@prefix ex:     <https://example.org/ns/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .

# Enclosing higher-order hyperspace: elements are Hyperspaces (holons)
ex:Holarchy a hspace:HyperspaceOfHyperspace ;
  hspace:hasElementType hspace:Hyperspace ;
  hspace:hasArrowType   ex:containsHolon ;     # whole-part relation
among hyperspaces
  hspace:arrowProperty  ex:containsHolon ;
  hspace:hasPathType    hspace:Path .

# Member hyperspaces at different scales (each is a whole/part)
ex:RoomSpace      a hspace:Hyperspace .
ex:BuildingSpace  a hspace:Hyperspace .
ex:DistrictSpace  a hspace:Hyperspace .
ex:CitySpace      a hspace:Hyperspace .

# Holarchic containment (arrows): Room ⟶ Building ⟶ District ⟶ City
ex:RoomSpace     ex:containsHolon ex:BuildingSpace .
ex:BuildingSpace ex:containsHolon ex:DistrictSpace .
ex:DistrictSpace ex:containsHolon ex:CitySpace .

# Optional explicit path: aggregation from Room to City through the
holarchy
ex:upwardAggregation a hspace:Path ;
  hspace:startsAt ex:RoomSpace ;
  hspace:endsAt   ex:CitySpace ;
  hspace:pathStep ( ex:RoomSpace ex:BuildingSpace ex:DistrictSpace
ex:CitySpace ) ;
  ex:levelCount "4"^^xsd:integer .
```

Figure 12

14. HSML TopologicalSpace SubModule

14.1. Introduction

The TopologicalSpace Module specializes the Hyperspace abstraction for structures defined by **neighborhoods, adjacency, and continuity**, rather than numeric coordinates. It realizes the **TOPOLOGICAL SPACE** concept referenced in IEEE P2874 (§ 6.2.1) and HSML Hyperspace design:

> "A topological space is a set of elements together with a structure that specifies which subsets are 'open,' thereby defining notions of adjacency, neighborhood, and continuity without reference to metric distance."

In HSML, a TopologicalSpace provides a declarative way to represent **connectivity, adjacency, boundaries, and regions**. It is the natural home for **geometries beyond points**, including **LineStrings**, **Polygons**, and **Spheres**, which are not valid vector elements but can be modeled in a topological sense.

Typical use cases include:

- Geographic features — regions, boundaries, polygonal areas, networks of connected edges.
- Computational grids — adjacency of cells in raster or tessellated models.
- Continuity spaces — abstract topologies for states, neighborhoods, or coverage.

A TopologicalSpace inherits from hspace:Hyperspace and introduces adjacency and region semantics. It does not assume distances (metric:Metric) unless combined with a MetricSpace; instead, it captures **purely topological relations**.

14.2. Architecture and Design Rationale

14.2.1. Core Principle

TopologicalSpaces in HSML are **relation-first**: they define what is "next to" or "contained in what," rather than how far apart elements are. This makes them suitable for **qualitative spatial reasoning** and **boundary-driven models**.

14.2.2. Subclassing Strategy

- topo:TopologicalSpace is a subclass of hspace:Hyperspace.
- Optional specializations include:
  - topo:TopologicalMetricSpace — adds metric distance on top of adjacency.
  - topo:CellularSpace — specialization for discrete grids and tessellations.

14.2.3. Elements and Paths

- **Element Type** — any spatial unit such as geo:Polygon, geo:LineString, or cell:Cell.
- **Path Type** — adjacency relations such as topo:adjacentTo or topo:connectedTo.

– Elements may represent continuous regions (polygons, spheres) or discrete cells.

– Paths represent adjacency chains (sequences of touching elements, border walks).

14.2.4. Neighborhoods and Open Sets

Unlike VectorSpaces, which define neighborhoods via coordinates and metrics, TopologicalSpaces rely on **declared adjacency** or **open sets**:

– topo:hasNeighborhood links an element to its set of neighboring elements.

– SHACL profiles may constrain neighborhood semantics (e.g., Moore vs. Von Neumann adjacency in a grid).

14.2.5. Regions and Boundaries

TopologicalSpaces support grouping and edge semantics:

– topo:Region — a contiguous set of adjacent elements treated as one area.

– topo:Boundary — the separating edge or border between a region and its exterior.

These abstractions allow modeling polygon boundaries, country borders, adjacency of cellular clusters, and more.

14.2.6. Generic Operations

TopologicalSpaces extend the universal Hyperspace algebra with **qualitative operations**:

– **Neighborhood query** — return adjacent elements of a given element.

– **Adjacency test** — check if two elements share a border or node.

– **Connectivity test** — check if two elements belong to the same connected component.

– **Path composition** — concatenate adjacency steps into longer paths.

– **Reachability** — determine if a path exists between two elements.

– **Boundary extraction** — identify the separating elements between regions.

– **Region closure / interior** — derive closed vs. open sets.

– **Union / intersection of regions** — form composite regions.

– **Subspace formation** — restrict to a subset of elements with induced adjacency.

– **Quotienting by equivalence** — merge elements under an equivalence relation (e.g., administrative aggregation).

These operations are **generic** and apply across polygonal, line-based, cellular, and spherical topologies.

14.2.7. Separation of Concerns

– **Domain vs Hyperspace** — a Domain provides identity; attaching a TopologicalSpace defines adjacency and region semantics.

– **Topology vs Metric** — adjacency is qualitative; distances are layered on top when required.

– **Profiles** — SHACL profiles constrain adjacency semantics (e.g., polygons must be closed, cells must tessellate consistently).

14.2.8. Examples of Spaces

TopologicalSpaces are not limited to geometry — they capture **adjacency and connectivity** in many domains:

Geometric Topologies - **Polygonal Topology**: elements are polygons; adjacency = "share an edge." - **Line Network**: elements are line segments; adjacency = "share a node." - **Spherical Topology**: elements are regions on a sphere; adjacency defined by great-circle boundaries. - **Cellular/DGGS Topology**: elements are discrete cells; adjacency defined by tessellation rules.

Computational / Data Structures - **Graph Topology**: elements are graph nodes; adjacency defined by edges. - **Tree Topology**: elements are tree nodes; adjacency = parent–child relations. - **Cellular Automaton Topology**: elements are cells; adjacency defined by automaton rules (Moore, Von Neumann).

Abstract / Conceptual Topologies - **State Space Topology**: elements are states in a machine; adjacency defined by transitions. - **Semantic Neighborhood Topology**: elements are SKOS concepts; adjacency via broader/narrower/related links. - **Linguistic Topology**: elements are words; adjacency via co-occurrence in a context window. - **Process Topology**: elements are workflow activities; adjacency via "precedes" or "depends on" relations.

Scientific / Physical Topologies - **Biological Cell Topology**: elements are cells in tissue; adjacency via direct contact. - **Protein Folding Topology**: elements are amino acids; adjacency via sequential bonds + folding contacts. - **Neural Network Topology**: elements are neurons; adjacency via synaptic connections.

To summarize, the TopologicalSpace Module captures **adjacency and neighborhood semantics** without relying on coordinates or metrics. It provides the structural foundation for representing **Polygons, LineStrings, Spheres, and beyond** as elements, enabling **qualitative reasoning, boundary logic, and region-based navigation** in HSML.

14.3. Metric Space Submodule

The Metric Space Submodule provides metric-aware, format-agnostic hooks that work with any element or path encoding chosen via the core hspace: mappings. It adds distance/similarity semantics without imposing a particular geometry, graph model, or storage format, and integrates cleanly with Vector, Graph, Cellular, Concept, or custom domains.

Namespace. https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#

Prefix. metric:

– Structural neutrality. Element and path types are still declared with hspace: (e.g., hspace:hasElementType, hspace:hasPathType). The metric module does not force vectors, geometries, or specific classes.

– Pluggable metrics. A Hyperspace MAY declare one or more metrics and optionally a default metric for operations.

– Literal or resource elements. Metrics can operate on resource elements (with an operand value carried by a property) or directly on literal elements (e.g., embeddings, strings, colors).

– Distance or similarity ("polarity"). A metric SHALL be explicitly tagged as distance or similarity; operations can uniformly convert via an optional monotone transform.

– Operations-ready. Metric declarations are consumable by hspace:Operation (e.g., kNN, range query, clustering, path cost evaluation) without additional coupling.

– Monoid path aggregation. Path costs are aggregated via a declared associative aggregator (e.g., sum, max, product) with an identity element; engines MAY override.

14.3.1. Normative Classes

Table 207—Class Summary for Metric Module

Class	Description
metric:MetricSpace	Specialization of hspace:Hyperspace indicating that distance/similarity semantics are available via one or more metric:Metric declarations.
metric:Metric	Declarative description of a distance/similarity over the element domain of a Hyperspace, including operand extraction (metric:operandProperty / metric:onType), polarity (metric:polarity), unit/scale (metric:unit / metric:scaleKind), and axiom kind (metric:kind)
metric:Measure (Optional)	First-class resource for pairwise measurements (distance or similarity) computed under a given metric:Metric—useful for caching, provenance, or exchange.
metric:Aggregator (Vocabulary/Class)	Declarative path aggregator with monoid semantics (associative op + identity). Includes ready-to-use instances (sum, max, min, product, lexicographic).
metric:MetricKind (Vocabulary)	Lightweight vocabulary tagging the axiom flavor: metric:Metric, metric:Pseudometric, metric:Quasimetric, metric:Semimetric, metric:Dissimilarity.
metric:Polarity (Vocabulary)	Lightweight vocabulary for polarity: metric:Distance, metric:Similarity.
metric:ScaleKind (Vocabulary)	Optional measurement scale hint: metric:Ratio, metric:Interval, metric:Ordinal, metric:UnitInterval.

14.3.2. Class: metric:MetricSpace

A metric:MetricSpace is a specialization of hspace:Hyperspace that **equips a domain with quantitative semantics**. While a Hyperspace already defines elements, paths, and their connectivity, a MetricSpace adds the notion of **distance or similarity functions**.

This enables: * **Quantitative reasoning** – e.g., nearest neighbor search, clustering, radius queries. * **Comparative analysis** – measuring similarity between embeddings, geometries, or categorical features. * **Path evaluation** – computing shortest paths, least-cost routes, or similarity-based traversals.

By separating **structural relationships** (adjacency, connectivity) from **metric semantics**, HSML supports **portable and engine-independent computations** across graphs, cellular grids, and vector spaces. In IEEE P2874 terms, this corresponds to the requirement that a Hyperspace may declare one or more **Metrics** to enable interoperable spatial and similarity-based operations.

14.3.2.1. Class Definition

Table 208—Class Definition for metric:MetricSpace

RDF Class	metric:MetricSpace
Is Abstract	No
Definition	A hspace:Hyperspace that declares one or more metrics (distance or similarity) applicable to its elements.
Subclass Of	hspace:Hyperspace
Usage Note	Use when your domain supports metric-based operations. Element and path types remain bound via hspace: mappings.
Rationale	Separates structure (core hyperspace) from quantitative semantics (metrics), enabling portable operations (kNN, radius search, path cost).

Table 209—Properties Summary for metric:MetricSpace

Predicate (anchor)	JSON name	Description	Range	Card.	Req.
metric: hasMetric	hasMetric	Declares a metric available in this Hyperspace.	metric:Metric	0..*	Optional
metric: defaultMetric	defaultMetric	Identifies the default metric to use when none is specified by an operation.	metric:Metric	0..1	Optional
metric: weightProperty (Optional)	weightProperty	Names a property that holds precomputed edge/step weights (when used).	rdf:Property (IRI)	0..1	Optional

14.3.2.2. Properties (MetricSpace)

14.3.2.2.1. Property: hasMetric

Table 210—Property Definition: metric:hasMetric

Property	metric:hasMetric
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#hasMetric
Domain	hspace:Hyperspace (typically metric:MetricSpace)
Range	metric:Metric
Definition	Declares a metric available for this Hyperspace.

14.3.2.2.2. Property: defaultMetric

Table 211—Property Definition: metric:defaultMetric

Property	metric:defaultMetric
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#defaultMetric
Domain	hspace:Hyperspace
Range	metric:Metric
Definition	Identifies the default metric to use in absence of an explicit metric selection.

14.3.2.2.3. Property: weightProperty (Optional)

Table 212—Property Definition: metric:weightProperty

Property	metric:weightProperty
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#weightProperty
Domain	hspace:Hyperspace
Range	rdf:Property (IRI)
Definition	Indicates a property on edges/steps that carries precomputed weights (e.g., travel time, cost) that operations MAY use instead of evaluating a metric.
Usage Note	A metric:Metric MAY override at the metric level via metric: edgeWeightProperty.

14.3.3. Class: metric:Metric

Metrics are the backbone of quantitative reasoning in a Hyperspace. They declare **how closeness, likeness, or difference is measured** between elements of a domain, enabling engines to evaluate k-nearest neighbor queries, range filters, clustering, and other operations consistently across heterogeneous data types.

Unlike raw functions, a metric:Metric is a **declarative description**: it specifies the polarity (distance vs similarity), expected operand type, optional units and scale, theoretical bounds, and a reference to the computation plan. This separation of **semantics** from **implementation** allows a metric definition to be portable across engines, reproducible across versions, and validatable via SHACL.

Key design principles:

- **Declarative, not imperative** — the ontology does not encode formulas but points to plans or algorithms while capturing semantic guarantees (e.g., symmetry, triangle inequality).
- **Polarity-aware** — every metric must declare whether values are interpreted as distances (lower is better) or similarities (higher is better), ensuring uniform ranking and threshold semantics.
- **Context-bound** — a metric may be attached to a Hyperspace (via metric:hasMetric or metric: defaultMetric) to make its applicability explicit.
- **Extensible** — additional properties such as metric:unit, metric:scaleKind, metric:monotoneTransform, and metric:aggregator provide optional but powerful hints for engines, user interfaces, and interoperability.

By elevating metrics to first-class declarative entities, HSML enables **predictable**, **portable**, and **semantically rich** measurement across the Spatial Web.

14.3.3.1. Mathematical Guarantees (The Four Axioms)

The boolean properties nonNegative, identityOfIndiscernibles, symmetric, and triangleInequality are more than just metadata; they are formal mathematical guarantees. When all four are true, the function is a true **metric**, which allows query engines and algorithms to make powerful assumptions and drastic optimizations. Declaring these guarantees enables a system to validate data, select the most efficient algorithms, and ensure logically consistent results.

14.3.3.1.1. Non-Negativity

Property: `metric:nonNegative`

Axiom: The distance between two points is never negative. $d(x, y) \geq 0$.

Why it's needed: This is a fundamental sanity check. It establishes a baseline where the minimum possible distance is zero. Algorithms that find shortest paths, like Dijkstra's, rely on this, as a negative distance would imply that traversing a path could somehow "reduce" the total cost, creating nonsensical loops.

Example — Physical Distance: The distance between two cities is always a positive number of miles or kilometers. The distance is 0 only if they are the same location.

When it's false: A function measuring financial "distance" as cost - revenue. Traveling from A to B might cost $50 in fuel but generate $80 in profit, yielding a "distance" of -$30. This is a gain/loss function, not a metric.

14.3.3.1.2. Identity of Indiscernibles

Property: `metric:identityOfIndiscernibles`

Axiom: The distance between two points is zero **if and only if** they are the same point. $d(x, y) = 0 \Longleftrightarrow x = y$.

Why it's needed: This axiom guarantees that every element is unique. It ensures that if two items are indistinguishable (zero distance apart), they are identical. This is crucial for search, deduplication, and clustering, as it prevents distinct items from being treated as the same.

Example—Hamming Distance: For strings of equal length, this is the number of positions at which the characters are different. d("apple", "apply") is 1, but d("apple", "apple") is 0. The distance is only zero if the strings are identical.

When it's false (Pseudometric): A function measuring the distance between two people as 0 if they live in the same city. Alice and Bob might have a distance of 0, but they are clearly not the same person.

14.3.3.1.3. Symmetry

Property: `metric:symmetric`

Axiom: The distance from x to y is the same as the distance from y to x. $d(x, y) = d(y, x)$.

Why it's needed: Symmetry allows algorithms to assume reciprocity. This dramatically simplifies problems in routing, graph analysis, and indexing. For example, a query engine only needs to compute or store a distance matrix in one direction, cutting storage requirements nearly in half.

Example—Euclidean Distance: The straight-line distance between your home and the library is the same as the distance from the library back to your home.

When it's false (Quasimetric): Driving time in a city with one-way streets. The route from the office to the gym might be 10 minutes, but the return trip could be 20 minutes due to different required paths.

14.3.3.1.4. Triangle Inequality

Property: `metric:triangleInequality`

Axiom: The direct path between two points is always the shortest. $d(x, z) \leq d(x, y) + d(y, z)$.

Why it's needed: This is the most powerful axiom for **optimization**. It guarantees that there are no "wormholes" or unexpected shortcuts. Search algorithms (like A*) and indexing structures (like M-trees) rely on this principle to prune massive portions of the search space.

Example—Geographic Distance: The direct flight distance from New York to Tokyo is never greater than the distance of flying from New York to London and then from London to Tokyo.

When it's false (Semimetric): Airline ticket pricing. A flight from New York to London might be $500, and a flight from London to Tokyo might be $600. However, a direct flight from New York to Tokyo could cost $1800, which is far greater than the sum of the two legs ($1100).

14.3.3.2. Class Definition

Table 213—Class Definition for metric:Metric

RDF Class	metric:Metric
Is Abstract	No

Table 213—Class Definition for metric:Metric *(continued)*

RDF Class	metric:Metric
Definition	A reusable definition of a distance or similarity function over elements (or their operand values) of a Hyperspace.
Subclass Of	owl:Thing
Usage Note	Bind to a Hyperspace via metric:hasMetric / metric:defaultMetric. Operations (e.g., kNN) consume this declaration to evaluate measures.
Rationale	Enables portable, implementation-independent metric evaluation across engines and data encodings.

Table 214—Properties Summary for metric:Metric

Predicate (anchor)	JSON name	Description	Range	Card.	Req.
metric:kind	kind	Axiom flavor of the measure (Metric, Pseudometric, Quasimetric, Semimetric, Dissimilarity).	metric: MetricKind	1	Mandatory
metric:polarity	polarity	Declares whether values are distance (lower is better) or similarity (higher is better).	metric:Polarity	1	Mandatory
metric:onType	onType	Declares the element type (class) or literal datatype the metric expects.	rdfs:Class or Datatype (IRI)	1	Mandatory
metric: operandProperty	operandProperty	Property from a resource element to its operand value (e.g., embedding vector, WKT, color triple).	rdf:Property (IRI)	0..1	Optional
metric: valueType	valueType	Datatype of operand and/or measure values used by the metric (e.g., xsd:decimal, xsd:double, rdf: JSON).	rdfs:Datatype	0..1	Optional
metric:unit	unit	Unit of the measure (e.g., a QUDT/UCUM IRI).	IRI	0..1	Optional
metric: scaleKind	scaleKind	Optional measurement scale hint (Ratio, Interval, Ordinal, UnitInterval).	metric: ScaleKind	0..1	Optional
metric: rangeMin	rangeMin	Minimum attainable value (e.g., 0).	xsd:decimal	0..1	Optional
metric: rangeMax	rangeMax	Maximum attainable value (e.g., 1 for normalized similarity).	xsd:decimal	0..1	Optional
metric:function	function	IRI of an algorithm/function/plan that computes the measure (e.g., prov:Plan, code/service IRI).	IRI	0..*	Optional
metric: functionVersion	functionVersion	Version tag for the referenced function/plan to ensure reproducibility.	xsd:string	0..1	Optional
metric: aggregator	aggregator	Default path aggregation for step-wise costs (e.g., metric:Sum, metric:Max).	metric: Aggregator	0..1	Optional
metric: edgeWeightProperty	edgeWeightProperty	Edge/step property to read precomputed weights for this metric (overrides metric: weightProperty on the space).	rdf:Property (IRI)	0..1	Optional

Table 214—Properties Summary for metric:Metric *(continued)*

Predicate (anchor)	JSON name	Description	Range	Card.	Req.
metric: monotoneTransform	monotoneTransform	Monotone mapping (IRI) to convert distance↔similarity or re-scale (e.g., f(d)=1/(1+d), f(s)=1−s).	IRI	0..1	Optional
metric: symmetric	symmetric	Whether d(x,y)=d(y,x). Defaults TRUE for metrics.	xsd:boolean	0..1	Optional
metric: identityOfIndiscernibles	identityOfIndiscernibles	Whether d(x,y)=0 ⇒ x=y. Defaults TRUE for metrics.	xsd:boolean	0..1	Optional
metric: triangleInequality	triangleInequality	Whether d(x,z) ≤ d(x,y)+d(y,z). Defaults TRUE for metrics.	xsd:boolean	0..1	Optional
metric: nonNegative	nonNegative	Whether values are guaranteed ≥ 0. Defaults TRUE.	xsd:boolean	0..1	Optional

14.3.3.3. Properties (Metric)

14.3.3.3.1. Property: kind

Table 215—Property Definition: metric:kind

Property	metric:kind
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#kind
Domain	metric:Metric
Range	metric:MetricKind
Definition	Tags the axiom flavor (e.g., metric:Metric, metric:Pseudometric).

14.3.3.3.2. Property: polarity

Table 216—Property Definition: metric:polarity

Property	metric:polarity
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#polarity
Domain	metric:Metric
Range	metric:Polarity
Definition	Declares whether the metric yields distance (lower is better) or similarity (higher is better).

14.3.3.3.3. Property: onType

Table 217—Property Definition: metric:onType

Property	metric:onType
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#onType
Domain	metric:Metric
Range	rdfs:Class or Datatype (IRI)
Definition	Declares the expected element class (resource elements) or literal datatype (literal elements) over which the metric operates.

14.3.3.3.4. Property: operandProperty (Optional)

Table 218—Property Definition: metric:operandProperty

Property	metric:operandProperty
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#operandProperty
Domain	metric:Metric
Range	rdf:Property (IRI)
Definition	If elements are resources, names the property that yields the operand value used by the metric (e.g., ex:embedding, geo:asWKT, vector:asArray).
Usage Note	Omit when elements are literals of the declared metric:onType; the literal itself is the operand.

14.3.3.3.5. Property: valueType (Optional)

Table 219—Property Definition: metric:valueType

Property	metric:valueType
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#valueType
Domain	metric:Metric
Range	rdfs:Datatype
Definition	Datatype of operands and/or distance/similarity values used by the metric (e.g., xsd:decimal, xsd:double, rdf:JSON).

14.3.3.3.6. Property: unit (Optional)

Table 220—Property Definition: metric:unit

Property	metric:unit
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#unit
Domain	metric:Metric
Range	IRI (e.g., QUDT/OM/UCUM)
Definition	Unit associated with the measure values (when applicable).

14.3.3.3.7. Property: scaleKind (Optional)

Table 221—Property Definition: metric:scaleKind

Property	metric:scaleKind
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#scaleKind
Domain	metric:Metric
Range	metric:ScaleKind
Definition	Optional hint about the measurement scale (e.g., Ratio for distances; UnitInterval for normalized similarities).

14.3.3.3.8. Property: rangeMin (Optional)

Table 222—Property Definition: metric:rangeMin

Property	metric:rangeMin
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#rangeMin
Domain	metric:Metric
Range	xsd:decimal
Definition	Theoretical or enforced minimum of the measure domain.

14.3.3.3.9. Property: rangeMax (Optional)

Table 223—Property Definition: metric:rangeMax

Property	metric:rangeMax
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#rangeMax
Domain	metric:Metric
Range	xsd:decimal
Definition	Theoretical or enforced maximum of the measure domain (e.g., 1.0 for normalized similarity).

14.3.3.3.10. Property: function (Optional)

Table 224—Property Definition: metric:function

Property	metric:function
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#function
Domain	metric:Metric
Range	IRI
Definition	Identifies an algorithm, plan, or service endpoint that evaluates the metric (e.g., prov:Plan, API/operation IRI).

14.3.3.3.11. Property: functionVersion (Optional)

Table 225—Property Definition: metric:functionVersion

Property	metric:functionVersion
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#functionVersion
Domain	metric:Metric
Range	xsd:string
Definition	Version string for the referenced function/plan (for reproducibility).

14.3.3.3.12. Property: aggregator (Optional)

Table 226—Property Definition: metric:aggregator

Property	metric:aggregator
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#aggregator
Domain	metric:Metric

Table 226—Property Definition: metric:aggregator *(continued)*

Range	metric:Aggregator
Definition	Default aggregator for composing step costs into a path cost (e.g., sum, max). Engines MAY override at operation time.

14.3.3.3.13. Property: edgeWeightProperty (Optional)

Table 227—Property Definition: metric:edgeWeightProperty

Property	metric:edgeWeightProperty
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#edgeWeightProperty
Domain	metric:Metric
Range	rdf:Property (IRI)
Definition	Names a property on edges/steps that carries precomputed weights for this metric (overrides space-level metric:weightProperty).

14.3.3.3.14. Property: monotoneTransform (Optional)

Table 228—Property Definition: metric:monotoneTransform

Property	metric:monotoneTransform
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#monotoneTransform
Domain	metric:Metric
Range	IRI
Definition	Identifies a monotone mapping to invert polarity or re-scale results (e.g., distance→similarity via f(d)=1/(1+d); similarity→distance via f(s)=1−s)).
Usage Note	The target of this IRI MAY be a prov:Plan, code function, or math expression resource.

14.3.3.3.15. Property: symmetric (Optional)

Table 229—Property Definition: metric:symmetric

Property	metric:symmetric
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#symmetric
Domain	metric:Metric
Range	xsd:boolean
Definition	Indicates whether the measure is symmetric.

14.3.3.3.16. Property: identityOfIndiscernibles (Optional)

Table 230—Property Definition: metric:identityOfIndiscernibles

Property	metric:identityOfIndiscernibles
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#identityOfIndiscernibles
Domain	metric:Metric
Range	xsd:boolean

Table 230—Property Definition: metric:identityOfIndiscernibles *(continued)*

Definition	Indicates whether d(x,y)=0 implies x=y.

14.3.3.3.17. Property: triangleInequality (Optional)

Table 231—Property Definition: metric:triangleInequality

Property	metric:triangleInequality
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#triangleInequality
Domain	metric:Metric
Range	xsd:boolean
Definition	Indicates whether the triangle inequality holds.

14.3.3.3.18. Property: nonNegative (Optional)

Table 232—Property Definition: metric:nonNegative

Property	metric:nonNegative
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#nonNegative
Domain	metric:Metric
Range	xsd:boolean
Definition	Indicates whether values are guaranteed to be non-negative (typical for distances).

14.3.4. Class: metric:Measure (Optional)

metric:Measure represents the computed outcome of applying a specific metric:Metric to an ordered pair of elements: metric:from and metric:to. It is deliberately structure-neutral (works with vectors, geometries, strings, graphs, etc.) and polarity-neutral (covers both distances and similarities). The how of measurement (function, axioms, operands, units) lives in metric:Metric; the what (the numeric result and its context) lives in metric:Measure. This separation keeps declarations portable and engine-independent while making results easy to cache, audit, exchange, and reason over.

14.3.4.1. Why a first-class metric:Measure?

- Caching & performance. Persist kNN/range results or expensive pairwise evaluations to avoid recomputation across queries and sessions.
- Audit & reproducibility. Record the value (metric:value), the metric used (metric:wrtMetric), and optional unit (metric:unit); link to provenance (e.g., prov:wasGeneratedBy) and implementation version (metric:functionVersion) for traceable, repeatable analytics.
- Interchange & federation. Share precomputed measures between heterogeneous engines and domains without leaking internal encodings or storage choices.
- Governance & evidence. Attach quantitative evidence to decisions/contracts in policy evaluation pipelines.
- Evaluation & ML. Store gold standards or inferred pairs for model training, validation, active learning, and drift monitoring.

14.3.4.2. Usage semantics

- Operands. metric:from and metric:to may be resources or literals. When elements are resources, the metric can declare metric:operandProperty to extract the value it operates on (e.g., embeddings, WKT). When elements are literals, the literal itself is the operand and metric:onType SHOULD be a datatype IRI.
- Polarity & transforms. Polarity is determined by the referenced metric (metric:polarity = metric:Distance or metric:Similarity). Engines MAY normalize for ranking via metric:monotoneTransform (e.g., similarity→distance). Unless stated otherwise, the stored metric:value SHOULD be the native output of the metric prior to any downstream normalization.*
- Typing & units. Prefer a numeric datatype consistent with the metric's metric:valueType (e.g., xsd:double, xsd:decimal). Provide metric:unit when the quantity is dimensional (e.g., meters, seconds) and consider metric:scaleKind / metric:rangeMin / metric:rangeMax for UI and validation hints.
- Identity & lifecycle. Ephemeral results can be blank nodes. Persisted results SHOULD mint stable IRIs (e.g., a content hash over {metric, from, to, functionVersion, params} with a timestamp). Capture provenance (prov:*) when reproducibility or explainability matters.
- Paths & aggregation. metric:Measure is pairwise by design. Path-level outcomes can be represented as additional metric:Measure instances whose endpoints denote the path start/end, with provenance pointing to the path and its metric:Aggregator (e.g., metric:Sum, metric:Max). Engines MAY also introduce a domain-specific path result class if needed.

14.3.4.3. Interoperability notes

- Works uniformly with any Hyperspace that declares metrics (see metric:MetricSpace and metric:hasMetric / metric:defaultMetric).
- Compatible with operation planning and evaluation (e.g., kNN, radius search, clustering, shortest path) without binding to a specific data model or algorithm.
- Encourages consistent cross-system semantics through explicit references to functions (metric:function), versions (metric:functionVersion), units (metric:unit), and axioms (metric:kind).

14.3.4.4. Class Definition

Table 233—Class Definition for metric:Measure

RDF Class	metric:Measure
Is Abstract	No
Definition	A first-class pairwise measure (distance or similarity) computed under a specific metric:Metric.
Subclass Of	hspace:Entity
Usage Note	Useful for caching, provenance, audit, or exchanging computed distances/similarities. Use prov:wasGeneratedBy to link computation runs if desired.

Table 234—Properties Summary for metric:Measure

Predicate (anchor)	JSON name	Description	Range	Card.	Req.
metric:usingMetric	usingMetric	Links the measurement to the metric used.	metric:Metric	1	Mandatory

Table 234—Properties Summary for metric:Measure *(continued)*

Predicate (anchor)	JSON name	Description	Range	Card.	Req.
metric:from	from	First element (resource or literal).	rdfs:Resource or rdfs:Literal	1	Mandatory
metric:to	to	Second element (resource or literal).	rdfs:Resource or rdfs:Literal	1	Mandatory
metric:value	value	Numeric value of the measure (datatype SHOULD match the metric's metric:valueType when provided).	rdfs:Literal	1	Mandatory
metric:unit	unit	Unit of the measure value (if any).	IRI	0..1	Optional
metric:confidence	confidence	Optional confidence in the computed value, in [0,1].	xsd:decimal	0..1	Optional

14.3.4.5. Properties (Measure)

14.3.4.5.1. Property: usingMetric

Table 235—Property Definition: metric:usingMetric

Property	metric:usingMetric
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#usingMetric
Domain	metric:Measure
Range	metric:Metric
Definition	Associates the measurement with the metric definition used.

14.3.4.5.2. Property: from

Table 236—Property Definition: metric:from

Property	metric:from
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#from
Domain	metric:Measure
Range	rdfs:Resource or rdfs:Literal
Definition	First element of the ordered pair.

14.3.4.5.3. Property: to

Table 237—Property Definition: metric:to

Property	metric:to
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#to
Domain	metric:Measure
Range	rdfs:Resource or rdfs:Literal
Definition	Second element of the ordered pair.

14.3.4.5.4. Property: value

Table 238—Property Definition: metric:value

Property	metric:value
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#value
Domain	metric:Measure
Range	rdfs:Literal (numeric)
Definition	Numeric value of the measurement.

14.3.4.5.5. Property: unit (Optional)

Table 239—Property Definition: metric:unit

Property	metric:unit
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#unit
Domain	metric:Measure
Range	IRI
Definition	Unit for the reported value (when applicable).

14.3.4.5.6. Property: confidence (Optional)

Table 240—Property Definition: metric:confidence

Property	metric:confidence
IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#confidence
Domain	metric:Measure
Range	xsd:decimal
Definition	Confidence in [0,1] for the computed value (if provided by the engine).

14.3.5. Class: metric:Aggregator

A metric:Aggregator defines **how step-level measures are combined into a path-level measure**. In HSML Hyperspaces (graphs, cellular spaces, vector paths), a path consists of multiple segments. To evaluate the overall cost, distance, or similarity of a path, step measures must be aggregated.

The metric:Aggregator provides a declarative, machine-readable way to specify: * **Operation** (metric:op) – associative binary operator (sum, max, product, lexicographic). * **Identity** (metric:identity) – neutral element for the operation. * **Commutativity** (metric:commutative) – whether operands can be reordered. * **Parameters** (metric:parameter) – structured bindings (e.g., p for Lp norms, weights for weighted sums).

This ensures **interoperability, optimization, and validation** across engines.

14.3.5.1. Class Definition

Table 241—Class Definition for metric:Aggregator

RDF Class	**metric:Aggregator**
Is Abstract	No
Definition	Declarative monoid for composing step measures into a path measure.

Table 241—Class Definition for metric:Aggregator *(continued)*

RDF Class	metric:Aggregator
Subclass Of	hspace:Entity
Usage Note	Hyperspatial Engines may provide optimized implementations for standard aggregators.

14.3.5.2. Properties Summary (Aggregator)

Table 242—Property Summary for metric:Aggregator

Predicate	JSON name	Description	Range	Card.	Req.
metric:op	op	Operation IRI (associative binary operator).	IRI	1	Mandatory
metric:identity	identity	Identity element (neutral value for the operation).	rdfs:Literal	0..1	Optional
metric: commutative	commutative	Whether the operation is commutative.	xsd:boolean	0..1	Optional
metric: parameter	parameter	One or more structured parameters configuring the op.	metric: Parameter	0..*	Optional

14.3.5.2.1. Property: op

Table 243—Property Definition: metric:op

IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#op
Domain	metric:Aggregator
Range	IRI
Definition	Identifies the associative binary operation used for aggregation.
Why it matters	Engines require the algebraic operation to compute path measures deterministically.
Examples	sw:sum, sw:max, sw:product, sw:lexicographic

14.3.5.2.2. Property: identity

Table 244—Property Definition: metric:identity

IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#identity
Domain	metric:Aggregator
Range	rdfs:Literal
Definition	Neutral element for the operation.
Why it matters	Enables valid evaluation of zero-length paths and ensures algebraic closure.
Examples	Sum → 0; Product → 1; Max → -INF

14.3.5.2.3. Property: commutative

Table 245—Property Definition: metric:commutative

IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#commutative
Domain	metric:Aggregator
Range	xsd:boolean
Definition	Indicates if operand order affects the result.

Table 245—Property Definition: metric:commutative *(continued)*

IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#commutative
Why it matters	Guides algorithmic optimization (parallelism, reordering).
Examples	Sum → true; Max → true; Lexicographic → false

14.3.5.2.4. Property: parameter

Table 246—Property Definition: metric:parameter

IRI	https://www.spatialwebfoundation.org/ns/hsml/hyperspace/metric#parameter
Domain	metric:Aggregator
Range	metric:Parameter
Definition	Associates the aggregator with one or more structured parameters.
Why it matters	Makes configuration machine-readable, disambiguating free-form literals.
Examples	Lp norm (p=2); Weighted sum (weights=[0.3,0.7])

14.3.6. Class: metric:Polarity

metric:Polarity captures the ordering semantics of measures produced by a metric:Metric. It tells engines whether lower values are better (metric:Distance) or higher values are better (metric:Similarity). Polarity is attached to a metric via metric:polarity and is intentionally decoupled from both metric axioms (see metric:MetricKind) and the computation plan (see metric:function). This separation lets systems perform uniform ranking, thresholding, pruning, and planning across heterogeneous metrics without knowing the metric's internal math.

Polarity also coordinates with optional scale hints and bounds declared on the metric (see metric:scaleKind, metric:rangeMin, metric:rangeMax) so UIs and algorithms can choose consistent comparators, normalize scores, and validate thresholds. When conversions are needed—such as consuming a similarity where a distance is expected or vice-versa—engines may apply the declared monotone transform (see metric:monotoneTransform) while preserving order.

Polarity is mandatory for metrics (enforced by SHACL), ensuring that operations like kNN, range queries, and top-k selection behave predictably. Note that metric:Measure remains polarity-neutral: the interpretation of a stored measurement is taken from its associated metric via metric:usingMetric.

14.3.6.1. Class Definition

Table 247—Class Definition for metric:Polarity

RDF Class	metric:Polarity
Is Abstract	No
Definition	Vocabulary class whose instances tag whether a measure behaves as a distance (lower-is-better) or a similarity (higher-is-better).
Subclass Of	owl:Thing
Usage Note	Used as the range of metric:polarity on metric:Metric to declare the intended ordering of values and to guide ranking, thresholding, and conversion via metric:monotoneTransform.
Rationale	Separates numeric ordering semantics from metric axioms (see metric:MetricKind) and from the computation plan (see metric:function), enabling uniform treatment of distances and similarities.

14.3.6.2. Instances

Table 248—Instances for metric:Polarity

Individual	Description
metric:Distance	Polarity tag indicating distance semantics: lower values indicate greater closeness; typical domains are ratio-scaled and non-negative. Often paired with aggregators such as metric:Aggregator = metric:Sum.
metric:Similarity	Polarity tag indicating similarity semantics: higher values indicate greater likeness; often normalized to metric:UnitInterval with optional bounds supplied via metric:rangeMin / metric:rangeMax.

14.3.6.2.1. Individual: metric:Distance

Table 249—Instance Definition: metric:Distance

RDF Individual	metric:Distance
Type	metric:Polarity
Definition	Declares that measurements produced by a metric are interpreted as distances (lower is better).

14.3.6.2.2. Individual: metric:Similarity

Table 250—Instance Definition: metric:Similarity

RDF Individual	metric:Similarity
Type	metric:Polarity
Definition	Declares that measurements produced by a metric are interpreted as similarities (higher is better).

15. HSML VectorSpace SubModule

15.1. Introduction

The VectorSpace Module specializes the Hyperspace abstraction for **linear spaces**. It realizes the **VECTOR SPACE** concept of IEEE P2874 (§ 6.2.x) as a Hyperspace whose elements admit **linear combination over a scalar field**, with an abstract **origin** and **composable arrows** corresponding to linear displacements.

> "A vector space is a set whose elements can be added and scaled, satisfying linearity axioms. In HSML, a VectorSpace is a Hyperspace that also qualifies as a MetricSpace, exposing linear and distance semantics without prescribing a particular element encoding or path representation."

This module is intentionally **non-prescriptive** about representation. It defines vector-space semantics and mapping hooks while allowing deployments to bind coordinates, arrays, or geometries according to their needs. Vector spaces are used across the Spatial Web for **geometric coordinates**, **embeddings and feature spaces**, **signal/field representations**, and **local linearizations** of complex manifolds.

By design, a vector:VectorSpace:

- Inherits all properties of hspace:Hyperspace (elements, arrows, paths, operations).
- Inherits all properties of metric:MetricSpace (distance, polarity, axioms, metrics).
- Adds linear structure hooks (scalar field, dimension, origin policy, CRS).

16. HSML Extensions and Profiles

HSML is designed for evolution and adaptation. While the Core Model provides a universal foundation, its true power is realized through extensions that tailor the language for specific industries and applications. This is accomplished by creating extensions and application profiles that add specialized concepts while preserving core interoperability and holonic integrity.

16.1. Subclass from the Core Model

Extensions **must** build upon the established HSML core classes. This is the primary mechanism for ensuring that new concepts are compliant, interoperable, and inherit foundational characteristics.

- **How to Model**: New classes **shall** be defined using rdfs:subClassOf to inherit from a base HSML class like hsml:Thing, hsml:Agent, or hsml:Activity. For example, a smartbuilding:Elevator would be a subclass of hsml:Thing.
- **Reasoning**: This ensures that every new entity is a valid holon that can be uniquely identified by a hsml:swid, exist within a hsml:Domain, and participate in the governance framework.

16.2. Maximize Reuse of Existing Ontologies

To promote interoperability and avoid reinventing concepts, extensions **should** integrate with established, domain-specific vocabularies wherever possible.

- **How to Model**: Before creating new properties, developers should import and reuse terms from well-known ontologies like SAREF, GeoSPARQL, or the W3C Organization Ontology. Properties like owl:equivalentClass can be used to map the extension's concepts to these external standards.
- **Reasoning**: This aligns with P2874's goal of creating a "holistic and coherent technical framework" by leveraging existing work rather than creating isolated data silos.

16.3. Enforce Governance and Data Quality with SHACL

Extensions are not just data schemas; they are governable components of the Spatial Web. Therefore, extensions **must** include formal constraints that define valid data structures and enforce rules.

- **How to Model**: Every extension profile **must** be accompanied by a set of SHACL shapes (sh:NodeShape, sh:PropertyShape). These shapes define rules such as mandatory properties (sh:minCount), data types (sh:datatype), and value ranges.
- **Reasoning**: This principle directly implements the P2874 requirements for "Governance by Design" and "Compliance". It allows domains to validate incoming data and ensure that all participants in the ecosystem adhere to the same structural and business rules.

16.4. Package Extensions as Profiles

To be manageable and reusable, extensions **should** be packaged as self-contained Application Profiles.

- **How to Model**: A profile is a collection of files that includes:
  - The OWL ontology defining the new classes and properties.
  - The SHACL shapes defining the constraints.
  - Human-readable documentation.
- **Reasoning**: This approach allows different industries (e.g., mobility, logistics, smart building, healthcare) to develop and maintain their own extensions while ensuring their foundation remains interoperable with the core HSML model and the broader Spatial Web.

Annex A
(normative)
Compliance

Annex B
(normative)
System Requirements

Annex C
(informative)
Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

Annex D
(informative)
SWF Authors