Operationalizing the Semantic Graph: A Kinetic Extension Framework for the Built Environment

1. Introduction: The Ontological Gap in Digital Twin Architectures

The trajectory of digital twin technology within the built environment has historically followed a path of increasing descriptive fidelity. The initial wave of standardization focused on the normalization of data points and the classification of physical assets. This effort culminated in the widespread adoption of ontologies such as **Brick Schema** and **RealEstateCore (REC)**, which have successfully provided a lingua franca for building metadata.[1] These frameworks allow disparate systems—Lighting, HVAC, Access Control—to be queried through a unified semantic graph, enabling applications like energy analytics and fault detection to operate across heterogeneous vendor environments.[3]

However, as the industry pivots from *observation* to *operation*, a critical deficiency in these open standards has emerged: the lack of a native "kinetic" layer. While Brick and REC excel at describing the *state* of the world (e.g., "This is a VAV box serving Room 204"), they lack the semantic primitives to describe *change* (e.g., "A user is requesting to lower the temperature setpoint by 2 degrees"). In current architectures, this transactional logic is externalized—hardcoded into proprietary applications or buried within the opaque logic of Building Management Systems (BMS).[1]

In stark contrast, proprietary enterprise operating systems, most notably **Palantir Foundry**, have pioneered a "Business-Object-Action" framework. Palantir's ontology is not merely a passive map; it is an active system of record for decision-making. Through its "Kinetic Layer," Palantir models "Actions" as first-class citizens, wrapping data modifications in layers of validation, business logic, permissioning, and side-effect orchestration.[4] This allows organizations to capture the "decision lineage" of their operations—not just what the temperature *is*, but *who* changed it, *why*, and what business process authorized that change.[6]

This report provides an exhaustive analysis of the architectural divergence between the passive semantic web (Brick/REC) and the active kinetic framework (Palantir). It deconstructs the components of Palantir's Action Types—parameterization, function-backed logic, validation, and write-back—and evaluates existing semantic web standards (SHACL, ODRL, Hydra, W3C WoT) that could replicate these capabilities. Finally, it proposes the **Kinetic Building Extension (KBE)**, a comprehensive ontological extension designed to inject Palantir-style operational capabilities into the open standard ecosystem, thereby transforming the digital twin from a dashboard into an operating system.

2. The Kinetic Paradigm: Deconstructing Palantir's Action Framework

To propose a viable extension to open standards, one must first rigorously analyze the reference implementation: Palantir Foundry. Palantir's architecture distinguishes between three primary layers: the **Semantic Layer** (the digital twin of entities), the **Kinetic Layer** (the verbs and actions), and the **Dynamic Layer** (simulations and AI-driven decisions).[7]

2.1 The Anatomy of an Action Type

In Palantir Foundry, an "Action" is not a direct database update (CRUD operation). It is a sophisticated transaction object defined by an **Action Type**. An Action Type serves as a schema for a specific business intent, such as "Approve Maintenance Request" or "Override Setpoint." This schema encapsulates four critical components that distinguish it from a simple API call:

1. Parametric Inputs and Submission Criteria:
   The Action Type defines the required inputs, which are often mapped to properties of the underlying ontology objects. Crucially, these inputs are subject to "Submission Criteria"—rules that validate the legitimacy of the action before it is attempted.8 For example, an action to "Assign Ticket" might require that the ticket status is currently "Open" and that the assignee has the correct role. This validation logic moves data integrity checks from the application layer into the ontology itself.
2. Function-Backed Logic:
   While simple actions can be defined declaratively (e.g., "Update Property A with Input B"), complex enterprise workflows require arbitrary logic. Palantir supports "Function-backed Actions," where the execution logic is defined in code (typically

TypeScript).9 These functions can perform complex traversals of the graph—for instance, "If a user updates the occupancy schedule for a Floor, recursively update the schedules for all Zones located on that Floor".11 The use of decorators like @OntologyEditFunction binds this code directly to the semantic model, ensuring that the logic is versioned and managed alongside the data schema.11

3. Side Effects and Orchestration:
   A digital twin rarely exists in isolation. It must interact with legacy systems of record (ERPs, CMMS). Palantir models these interactions as "Side Effects." When an Action is successfully committed to the ontology, it can trigger configured side effects such as sending notifications (email/SMS) or firing Webhooks to external REST APIs.8 This turns the ontology into an orchestration engine; a single user action in the twin can synchronously update SAP, notify a facility manager, and write a setpoint to a BACnet controller.
4. Write-back and Materialization:
   The mechanism of "Write-back" in Palantir is distinct from standard database writes. Edits are captured as structured events. In the legacy "Phonograph" architecture (OSv1) and the newer Object Storage V2, user edits are applied to the read-optimized data store, effectively "patching" the materialization.13 This architecture ensures that the ontology remains the single source of truth, reflecting the intended state even before the underlying raw data sources have ingested the change.

2.2 The Decision Graph and Auditability

Perhaps the most profound implication of the Kinetic Layer is the creation of a "Decision Graph." Because every change to the system is wrapped in an Action, the system generates a granular log of operations. This allows for "Decision Capture," enabling organizations to analyze the *velocity* of operations.[6] Instead of merely querying the current temperature (a state), an analyst can query "How many times was the temperature overridden due to 'Occupant Complaint' in the last month?".[15] This shift from state-analysis to behavior-analysis is the defining characteristic of the kinetic approach.

3. The Static Nature of Current Building Ontologies

While Palantir has focused on the *mechanics* of operation, the open standards community has focused on the *semantics* of classification. A detailed review of Brick Schema and RealEstateCore reveals the structural limitations that prevent them from natively supporting kinetic workflows.

3.1 Brick Schema: The Descriptive Graph

Brick Schema utilizes a class hierarchy based on RDF (Resource Description Framework) to define the physical, logical, and virtual assets of a building.[16] Its primary goal is the standardization of metadata to support portable analytics.[1]

3.1.1 The Ambiguity of "Command"

Brick defines a Command class as a type of Point.[17] Specifically, it is defined as a setting or action that directly determines the behavior of equipment (e.g., Start/Stop Command).[18] However, in the Brick ontology, a Command is fundamentally a **data point**—a "slot" where a value can be written. It does not model the *act* of writing that value.

For example, a brick:Temperature_Setpoint represents the *current value* of the setpoint. If a user wants to change this setpoint, Brick offers no vocabulary to describe that transaction. There is no construct in Brick to say "User X requested to change Setpoint Y to Value Z at Time T." Brick models the *result* of the action (the new value), but not the *action* itself.

3.1.2 Externalization of Control Logic

The Brick consortium explicitly scopes "control logic" out of the core ontology. Research indicates that Brick is designed to complement, not replace, control sequences. The **Control Description Language (CDL)**, developed by ASHRAE (Standard 231P), is the intended partner for Brick.[19] CDL uses block-diagram logic to describe how inputs are transformed into outputs.

While CDL describes *algorithmic* control (PID loops, boolean logic), it does not describe *business* transactions. A CDL sequence might describe how a VAV box modulates a damper based on temperature, but it does not describe the workflow of a facility manager issuing a "Purge Mode" command that requires approval from a supervisor. The gap remains: Brick models the nouns, CDL models the internal reflexes, but neither models the human-driven business transactions.

3.2 RealEstateCore and DTDL: The Interface Approach

RealEstateCore (REC) has increasingly aligned with Microsoft’s **Digital Twin Definition Language (DTDL)** to support Azure-based digital twins.[21] DTDL takes a slightly different approach by defining "Interfaces" that can contain Properties, Telemetry, and **Commands**.[23]

3.2.1 DTDL Commands: RPC vs. Business Logic

A DTDL Command defines a function signature—a name, a request payload schema, and a response payload schema.[23] This allows a developer to define a reboot command that takes a delay integer as input. This structure supports Remote Procedure Call (RPC) patterns, primarily for device interaction.

However, DTDL Commands are stateless definitions of capability. They do not encompass the rich metadata of a Palantir Action Type. There is no native support in DTDL for:

- **Preconditions/Validation:** DTDL schemas define types (e.g., "Must be an Integer") [23], but they do not define business rules (e.g., "Must be less than the high-limit setpoint of the parent AHU").
- **Side Effects:** There is no declarative way to state that invoking a DTDL command should trigger a webhook.
- **Orchestration:** DTDL does not model multi-step workflows or state transitions.

Consequently, while REC/DTDL provides a "hook" to invoke code, the business logic remains hidden behind that hook, opaque to the ontology.

4. Theoretical Foundations for a Kinetic Extension

To synthesize a Palantir-like framework within the open Semantic Web stack, we cannot simply invent new concepts from scratch. We must leverage and harmonize existing, disparate standards that address specific components of the "Kinetic" layer. The proposed extension will act as a unifying fabric for these standards.

4.1 The Structure of Action: W3C Web of Things (WoT) & Hydra

The most robust open standard for modeling interaction capabilities is the **W3C Web of Things (WoT) Thing Description**. WoT defines the concept of an **ActionAffordance**.[25] An ActionAffordance models the invocation of a process that may manipulate state or trigger a physical action.

Crucially, WoT introduces the separation of the *affordance* (the potential to act) from the *implementation*. It allows for the definition of input and output data schemas using JSON Schema, and it includes flags for idempotent and safe interactions.[1]

Similarly, the **Hydra Core Vocabulary** defines hydra:Operation, which represents a request that can be performed on a resource.[28] Hydra explicitly models the expects (input) and returns (output) of an operation, as well as the HTTP method (GET, POST) required to invoke it.[30]

Insight: Both WoT and Hydra provide the *syntax* for defining an action's interface. However, they lack the *semantics* for business rules. They tell a client *how* to construct a request, but not *whether* that request makes business sense.

4.2 The Logic of Validation: SHACL (Shapes Constraint Language)

Palantir's "Submission Criteria" and parameter validation find their open-standard equivalent in **SHACL**.[31] SHACL is typically used to validate the "shape" of an RDF graph (e.g., checking that every Person has a Name).

However, advanced Semantic Web patterns utilize SHACL to validate **Action Payloads**.[31] By defining a sh:NodeShape that corresponds to the input parameters of an action, one can enforce rigorous constraints:

- **Datatype Constraints:** "The setpoint must be a decimal."
- **Range Constraints:** "The value must be between 18 and 26" (sh:minInclusive, sh:maxInclusive).
- **Pattern Constraints:** "The ID must match a specific Regex."
- **Logical Constraints:** "If Mode is 'Occupied', then Setpoint must be defined."

Research into "Extended SHACL Validators" suggests that SHACL can be used not just for static checks, but for verifying the preconditions of an update operation.[33] This directly maps to Palantir's parameter validation logic.

4.3 The Semantics of Governance: ODRL (Open Digital Rights Language)

Palantir places a heavy emphasis on permissioning—ensuring that only authorized users can execute specific actions on specific objects.[35] In the Semantic Web, the **ODRL Information Model** provides the vocabulary for this governance.[36]

ODRL models **Policies**, which consist of **Permissions**, **Prohibitions**, and **Duties**. An ODRL Policy can state: "The Facility Manager (Party) has Permission to execute the 'Override' Action on the 'VAV Box' Asset, provided they satisfy the Duty of 'Logging the Reason'".[37] Integrating ODRL allows the ontology to express Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) rules declaratively.

4.4 The Semantics of Transaction: REA (Resources, Events, Agents)

To model the "Business" aspect of the framework, we look to the **REA Ontology**. Originally an accounting model, REA describes **Economic Events** where **Agents** exchange **Resources**.[39]

The core insight of REA is the **Duality** relationship: every economic event (e.g., giving a resource) is linked to a reciprocal event (e.g., receiving cash).[41] When applied to building operations, this pattern models the causality of actions: an "Energy Consumption Event" (Resource decremented) corresponds to a "Comfort Provision Event" (Service incremented). This theoretical lineage provides the backbone for modeling complex transactions that involve trade-offs, crucial for advanced energy optimization workflows.

5. Proposal: The Kinetic Building Extension (KBE)

Based on the analysis of the gaps in Brick/REC and the capabilities of Palantir, this report proposes the **Kinetic Building Extension (KBE)**. KBE is a modular ontology designed to be imported alongside Brick or RealEstateCore. It introduces a semantic layer for defining, executing, logging, and governing business-logic-driven actions.

5.1 Core Class Hierarchy

The extension creates a clear separation between the *definition* of an action (the template) and the *execution* of an action (the instance). This mirrors the ActionType vs. Action distinction in Palantir [15] and the Operation vs. Execution distinction in Hydra/WoT.

Table 1: Core KBE Class Structure

Class	**Subclass Of**	**Description**	**Palantir Equivalent**
kbe:ActionDefinition	owl:Class	The template for a business operation. Defines inputs, logic, and side effects.	Action Type
kbe:ActionExecution	prov:Activity	A reified instance of an action being attempted or completed. Used for the audit log.	Action Instance
kbe:SideEffect	owl:Thing	A definition of external consequences triggered by the action.	Side Effect
kbe:TransitionRule	sh:NodeShape	Logic defining valid state transitions and preconditions.	Submission Criteria

kbe:Function	duv:Usage	A pointer to executable logic (code) that backs the action.	Function

5.2 Modeling the Action Definition

The kbe:ActionDefinition acts as the central node that orchestrates the kinetic behavior. It binds the target object (from Brick) to the validation logic (SHACL) and the execution logic (Function).

Key Properties:

- kbe:targetsType: Specifies the Brick/REC class this action applies to (e.g., brick:VAV).
- kbe:hasInputShape: Links to a sh:NodeShape that defines the required parameters (payload).
- kbe:isBackedBy: Links to a kbe:Function resource (URI to the code).
- kbe:triggers: Links to kbe:SideEffect resources.
- kbe:hasPolicy: Links to an odrl:Policy defining permissions.

5.2.1 Example: "Adjust Setpoint" Action

The following Turtle syntax demonstrates how an "Adjust Setpoint" action would be defined using KBE, integrating SHACL for validation and ODRL for permissioning.

Code snippet

```
@prefix brick: <https://brickschema.org/schema/Brick#>.
@prefix kbe: <https://example.org/kbe#>.
@prefix sh: <http://www.w3.org/ns/shacl#>.
@prefix odrl: <http://www.w3.org/ns/odrl/2/>.

# --- Action Definition ---
```

```
:AdjustSetpointAction a kbe:ActionDefinition ;
    rdfs:label "Adjust Temperature Setpoint" ;
    kbe:targetsType brick:Temperature_Setpoint ;
    kbe:hasInputShape :SetpointAdjustmentInputShape ;
    kbe:isBackedBy <urn:function:validate-and-adjust-setpoint> ;
    kbe:triggers :NotifyFacilityManager ;
    kbe:hasPolicy :ManagerOnlyPolicy.

# --- Input Validation (SHACL) ---
:SetpointAdjustmentInputShape a sh:NodeShape ;
    sh:property ;
    sh:property.

# --- Permission Policy (ODRL) ---
:ManagerOnlyPolicy a odrl:Set ;
    odrl:permission.
```

5.3 Implementation of "Function-Backed" Logic in RDF

One of the challenges in the Semantic Web is representing procedural logic. Palantir solves this with TypeScript functions. KBE formalizes this by treating the **Function** as a referenceable resource, similar to the approach in **FNO (Function Ontology)**.

The kbe:isBackedBy property points to a URI that identifies the executable code (e.g., an AWS Lambda ARN, an Azure Function URL, or a reference to a script in a repository). This allows the ontology to remain "clean" while acknowledging that complex logic (e.g., looping through linked equipment) must be handled by a computational engine.

To support the "loop" logic described in Palantir's documentation (e.g., updating all objects linked to a starting object [10]), the external function would accept the kbe:ActionExecution RDF subgraph as input, parse the targets, traverse the Brick relationships (e.g., brick:feeds), and generate the necessary updates.

5.4 Modeling Side Effects and Orchestration

Palantir's ability to trigger notifications and webhooks [8] is modeled in KBE via the kbe:SideEffect class hierarchy.

- kbe:NotificationEffect: Defines a template for human alerts.
  - Properties: kbe:recipientRole, kbe:messageTemplate, kbe:channel (Email, SMS).
- kbe:WebhookEffect: Defines a machine-to-machine call.
  - Properties: kbe:targetEndpoint, kbe:httpMethod, kbe:payloadTemplate.

By linking these effects to the ActionDefinition via the kbe:triggers property, the ontology explicitly documents the downstream consequences of an action. This provides transparency that is often missing in hard-coded integration layers.

6. Executing the Kinetic Graph: Workflow and State Machines

The definition of an action is only half the battle; the execution and state management form the operational core. Palantir actions often enforce business workflows (e.g., "Request" -> "Approval" -> "Execution"). To support this, KBE incorporates **Finite State Machine (FSM)** principles, drawing on research into OWL-based FSMs.[42]

6.1 The Action Execution Life-cycle

When a user or system initiates an action, a kbe:ActionExecution instance is created. This instance is a **reified transaction**. It tracks the lifecycle of the request through status properties.

States of Execution:

1. **Pending:** The action has been requested but not yet validated or approved.
2. **Validated:** SHACL constraints have been satisfied.
3. **Approved:** ODRL policies and manual approvals (if required) are satisfied.
4. **Executing:** The backing function is running.
5. **Completed:** The state change has been committed, and side effects triggered.
6. **Failed:** An error occurred (captured in kbe:failureReason).

This structure allows for the construction of a semantic audit log. By querying the kbe:ActionExecution graph, one can reconstruct the history of operations: "Show me all *Failed* executions on *Chillers* in the last 24 hours."

6.2 Transition Rules and Guards

Complex workflows require "Guards"—logic that prevents an action from occurring unless the system is in a specific state. For example, a "Close Work Order" action should only be valid if the Work Order is currently "In Progress."

KBE models these guards using kbe:TransitionRule. These rules can be implemented as:

- **SHACL Constraints:** Validating that the brick:hasTag or rec:status property of the target object matches a required value.
- **SPARQL ASK Queries:** More complex guards that check relationships (e.g., "Cannot turn off the Chiller if any downstream AHU is requesting cooling").

This aligns with the research on modeling State Machines in ontologies, where transitions are restricted by conditions defined on the source state.[44]

6.3 The Write-Back Mechanism

Palantir's "Write-back" capability ensures that ontology edits propagate to the underlying system.[13] In the open standard context, this requires a mapping layer. KBE introduces kbe:WriteBackMapping to translate high-level semantic actions into protocol-specific commands.

Mapping Structure:

- **Input:** The parameter from the Action (e.g., proposedValue).
- **Protocol:** The communication standard (e.g., BACnet, Modbus, API).
- **Address:** The specific register or endpoint (derived from brick:hasAddress).
- **Priority:** For BACnet, the write priority (e.g., 8 for Operator Override).

This allows an "Ontology Engine" to act as a translation layer. When the :AdjustSetpointAction reaches the **Executing** state, the engine reads the WriteBackMapping, constructs the appropriate BACnet WriteProperty service request, and sends it to the device.

7. Use Case Deep Dives

To demonstrate the utility of KBE, we examine two critical use cases where standard ontologies fail and the kinetic extension succeeds.

7.1 Maintenance Workflows: Beyond the Work Order

Research into maintenance management highlights a semantic gap between "Maintenance Requests" (unstructured, user-generated) and "Work Orders" (structured, technician-focused).[45] Standard building ontologies often lack the "Process" view required to link these two.

The KBE Approach:

1. **The Action:** A user initiates a kbe:CreateMaintenanceRequest action.
2. **Validation:** SHACL validates that the request is linked to a valid brick:Location or brick:Equipment.
3. **Logic:** The backing function uses Natural Language Processing (NLP) logic (as suggested in [47]) to classify the request text (e.g., "It's too hot") into a category (e.g., "HVAC/Temperature").
4. **State Transition:** The function creates a rec:WorkOrder object in the ontology with status "Triage."
5. **Side Effect:** A kbe:WebhookEffect pushes the new Work Order to the corporate CMMS (e.g., Maximo) to ensure synchronization.

This workflow transforms the "Maintenance Request" from a simple data entry into an orchestrated process that leverages the semantic graph for context (location, equipment type) and enforces business rules automatically.

7.2 Semantic Fault Correction

Current Fault Detection and Diagnosis (FDD) systems identify anomalies but stop at the alert. They use Brick to find the sensors, but rely on human operators to act.[1]

The KBE Approach:

1. **The Trigger:** An FDD algorithm detects a "Stuck Damper" fault.
2. **The Kinetic Response:** The algorithm automatically instantiates a kbe:ActionExecution for "Recalibrate Damper."
3. **Governance:** An ODRL policy checks if "Automated Remediation" is permitted for this asset class during occupied hours.
4. **Execution:** If permitted, the action executes, triggering the write-back to the BAS to cycle the damper.
5. **Audit:** The entire sequence is logged as a kbe:ActionExecution linked to the specific brick:Damper.

This closes the loop between *sensing* (Brick) and *acting* (KBE), enabling the "Decision Orchestration" capabilities found in Palantir.[8]

8. Strategic Implications and Implementation

8.1 The "Headless" Building Operating System

Adopting KBE moves the "Building Operating System" (BOS) logic out of proprietary application code and into the ontology. This creates a "Headless" BOS. Any application—a mobile app, a chatbot, an optimization script—can interact with the building by instantiating ActionExecutions. They do not need to know the underlying BACnet addresses or the specific API of the CMMS; they simply need to know the semantic definition of the action. This decoupling is the essence of the "Kinetic" architecture.

8.2 Semantic Governance as Code

By embedding ODRL policies directly into the action definitions, the enterprise achieves "Governance as Code." Policy changes (e.g., "Revoke all override permissions for third-party contractors") can be implemented by updating the ODRL graph, instantly propagating the restriction to every application that uses the KBE framework. This replicates the granular security model of Palantir Foundry [9] using open standards.

8.3 Data Interoperability and Migration

The use of SHACL and RDF ensures that these action definitions are portable. An organization can define a standard "Corporate Action Library" (e.g., standardized setpoint adjustments, standardized maintenance procedures) and deploy it across a global portfolio of buildings, regardless of the underlying hardware. This standardization of *process* is the logical next step after the standardization of *data* provided by Brick and REC.

9. Comparative Summary

Feature	**Standard Brick/REC/DTDL**	**Palantir Foundry**	**Proposed KBE Extension**
Primary Paradigm	Descriptive (State)	Kinetic (Action)	Kinetic (Action via RDF)
Interaction Model	API/RPC Endpoints	Action Types	kbe:ActionDefinitio n
Input Validation	Type Checking (Basic)	Submission Criteria (Rules)	SHACL Shapes (Complex constraints)
Business Logic	External Application Code	Function-backed (TypeScript)	Referenceable Function URIs
Governance	App-level RBAC	Object/Action Permissions	ODRL Policies
Audit Trail	Database Logs (Unstructured)	Decision Graph	kbe:ActionExecutio n Graph
Interoperability	High (Open	Low (Proprietary)	High (Extends

	Standards)		Brick/REC)

10. Conclusion

The operational gap between the static rigor of Brick Schema and RealEstateCore and the dynamic utility of Palantir Foundry represents the next frontier in digital twin development. While Palantir has proven the value of "binding" logic to data through its proprietary Action framework, the open semantic web has largely restricted itself to describing the physical world.

The proposed **Kinetic Building Extension (KBE)** bridges this gap by introducing the *Action* as a semantic primitive. By synthesizing the structural rigor of **W3C WoT Affordances**, the validation power of **SHACL**, the governance model of **ODRL**, and the transactional transparency of Palantir’s **Decision Graph**, KBE allows open standard ontologies to support rich, transactional business frameworks.

This extension empowers organizations to build non-proprietary, "read-write" digital twins. It enables the transition from passive monitoring to active orchestration, ensuring that the digital twin is not just a reflection of the building, but the central nervous system that governs its operation. The integration of these kinetic capabilities into the open standard ecosystem is not merely an enhancement; it is a necessity for the realization of autonomous, intelligent building operations.

Works cited

1. Brick _ Metadata schema for portable smart building applications, accessed November 18, 2025, https://brickschema.org/papers/Brick-AppliedEnergy-2018-Balaji.pdf
2. INTRODUCTION - RealEstateCore, accessed November 18, 2025, https://www.realestatecore.io/introduction/
3. Brick Schema, accessed November 18, 2025, https://brickschema.org/
4. Understanding Palantir's Ontology: Semantic, Kinetic, and Dynamic Layers Explained, accessed November 18, 2025, https://pythonebasta.medium.com/understanding-palantirs-ontology-semantic-kinetic-and-dynamic-layers-explained-c1c25b39ea3c
5. Overview • Ontology - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/ontology/overview/
6. Foundry Ontology - Palantir, accessed November 18, 2025, https://www.palantir.com/platforms/foundry/foundry-ontology/

7. Ontology Palantir - notes - follow the idea - Obsidian Publish, accessed November 18, 2025, https://publish.obsidian.md/followtheidea/Content/AI/Ontology+Palantir+-+notes
8. Action types • Side effects • Overview - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/action-types/side-effects-overview/
9. Action types • Function-backed Actions • Overview - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/action-types/function-actions-overview/
10. Function consumption • Use functions in the platform - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/functions/use-functions/
11. Action types • Function-backed Actions • Getting started - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/action-types/function-actions-getting-started/
12. Action types • Side effects • Webhooks - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/action-types/webhooks/
13. Object edits and materializations - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/object-edits/materializations/
14. Ontology architecture - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/object-backend/overview/
15. Action basics • API Reference - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/api/ontology-resources/actions/action-basics//
16. Brick: Towards a Unified Metadata Schema For Buildings - Computer Science, accessed November 18, 2025, https://cseweb.ucsd.edu/~dehong/pdf/buildsys16-paper.pdf
17. Modeling Data Sources - Brick Ontology Documentation, accessed November 18, 2025, https://docs.brickschema.org/brick/timeseries.html
18. Point type definitions · Issue #32 · BrickSchema/Brick - GitHub, accessed November 18, 2025, https://github.com/BrickSchema/Brick/issues/32
19. The Brick Schema, Data Portability, and Independent Data Layers for Smart Buildings | by Erik Paulson | Medium, accessed November 18, 2025, https://medium.com/@erik_paulson/the-brick-schema-data-portability-and-independent-data-layers-for-smart-buildings-2dc048efbf3b
20. Shepherding Metadata Through the Building Lifecycle - Brick Schema, accessed November 18, 2025, https://brickschema.org/papers/shepherding2020fierro.pdf
21. DTDL or SHACL - RealEstateCore Developer Pages, accessed November 18, 2025, https://dev.realestatecore.io/docs/DTDL-or-SHACL
22. DTDL models - Azure Digital Twins | Microsoft Learn, accessed November 18, 2025, https://learn.microsoft.com/en-us/azure/digital-twins/concepts-models
23. Digital Twins Definition Language (DTDL) | opendigitaltwins-dtdl - Azure documentation, accessed November 18, 2025, https://azure.github.io/opendigitaltwins-dtdl/DTDL/v2/DTDL.v2.html
24. Understand IoT Plug and Play device models - Azure - Microsoft Learn, accessed November 18, 2025, https://learn.microsoft.com/en-us/azure/iot/concepts-modeling-guide
25. Web of Things (WoT) Thing Description (TD) Ontology - W3C, accessed

November 18, 2025, https://www.w3.org/2019/wot/td
26. Web of Things (WoT) Thing Description 1.1 - W3C, accessed November 18, 2025, https://www.w3.org/TR/wot-thing-description11/
27. Web of Things (WoT) Thing Description 1.1 - W3C, accessed November 18, 2025, https://www.w3.org/TR/wot-thing-description/
28. accessed November 18, 2025, http://t-code.pl/blog/2020/12/hydra-shacl-interoperability/#:~:text=The%20Hydra%20vocabulary%20defines%20a,or%20entire%20class%20of%20resources.
29. Hydra and SHACL - a perfect couple - part 1 - Tomasz Pluskiewicz, accessed November 18, 2025, https://t-code.pl/blog/2020/12/hydra-shacl-interoperability/
30. Hydra for Hypermedia APIs: Benefits, Components, and Examples, accessed November 18, 2025, https://nordicapis.com/hydra-for-hypermedia-apis-benefits-components-and-examples/
31. What Is SHACL | Ontotext Fundamentals, accessed November 18, 2025, https://www.ontotext.com/knowledgehub/fundamentals/what-is-shacl/
32. Transforming SHACL Shape Graphs into HTML Applications for Populating Knowledge Graphs - MDPI, accessed November 18, 2025, https://www.mdpi.com/2673-6470/5/4/56
33. xpSHACL: Explainable SHACL Validation using Retrieval-Augmented Generation and Large Language Models - arXiv, accessed November 18, 2025, https://arxiv.org/html/2507.08432v1
34. SHACL Validation under Graph Updates (Extended Paper) - arXiv, accessed November 18, 2025, https://arxiv.org/html/2508.00137v1
35. List Action Types • API Reference - Palantir, accessed November 18, 2025, https://palantir.com/docs/foundry/api/ontology-resources/action-types/list-action-types//
36. ODRL Information Model 2.2 - W3C, accessed November 18, 2025, https://www.w3.org/TR/odrl-model/
37. Evaluation and Comparison Semantics for ODRL, accessed November 18, 2025, https://arxiv.org/html/2509.05139v1
38. ODRL V2.2 Implementation Best Practices - W3C on GitHub, accessed November 18, 2025, https://w3c.github.io/odrl/bp/
39. Towards a Common Ontology for Business Models - CEUR-WS.org, accessed November 18, 2025, https://ceur-ws.org/Vol-200/10.pdf
40. Understanding the Resource-Event-Agent (REA) Conceptual Model - Blog: Digital Financial Reporting (using XBRL), accessed November 18, 2025, http://xbrl.squarespace.com/journal/2016/9/27/understanding-the-resource-event-agent-rea-conceptual-model.html
41. Resources, Events, Agents - Wikipedia, accessed November 18, 2025, https://en.wikipedia.org/wiki/Resources,_events,_agents_(accounting_model)
42. Towards an Ontology for UML State Machines - ResearchGate, accessed November 18, 2025, https://www.researchgate.net/publication/289867778_Towards_an_Ontology_for_UML_State_Machines

43. Factory Data Model, accessed November 18, 2025, https://virtualfactory.gitbook.io/vlft/kb/fdm
44. An excerpt of OWL model of transition in a state machine. - ResearchGate, accessed November 18, 2025, https://www.researchgate.net/figure/An-excerpt-of-OWL-model-of-transition-in-a-state-machine_fig2_220940671
45. Automated Priority Assignment of Building Maintenance Tasks Using Natural Language Processing and Machine Learning | Journal of Architectural Engineering | Vol 29, No 3 - ASCE Library, accessed November 18, 2025, https://ascelibrary.org/doi/abs/10.1061/JAEIED.AEENG-1516
46. Where do we start? Guidance for technology implementation in maintenance management for manufacturing - PubMed Central, accessed November 18, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11494710/
47. How NLP is Shaping Maintenance Management - NuShift, accessed November 18, 2025, https://nushift.ai/how-nlp-is-shaping-maintenance-management/
48. Natural Language Processing Model for Managing Maintenance Requests in Buildings, accessed November 18, 2025, https://www.semanticscholar.org/paper/Natural-Language-Processing-Model-for-Managing-in-Bouabdallaoui-Lafhaj/680eca5254024d71c90a82ef47b7a075462a01c7