Implementation Semantics

Go up to Terms and Definitions

Discussion

Implementation semantics defines meaning through executable artifacts, deployed configurations, runtime behavior, database constraints, message handlers, validation functions, policy enforcement points, and system operations.

Implementation artifacts express semantic content when their behavior preserves traceability to the defined domain meaning. That meaning comes from authoritative semantic sources such as semantics, conceptual models, schemas, type systems, rules, policies, ontologies, reports, and conformance criteria.

Implementation semantics differs from conceptual semantics. Conceptual semantics defines meaning before an architecture selects implementation technology. Implementation semantics expresses selected meaning through runtime behavior and technical artifacts.

Implementation semantics also differ from implementation details. Implementation detail describes how a system performs work. Implementation semantics identifies what the implemented behavior means with respect to the governed architecture.

Implementation semantic content includes:

- Executable code
- Runtime logic
- Configuration files
- Database constraints
- Validation functions
- Message handlers
- Event handlers
- Application Programming Interface (API) behavior
- DDS topic behavior
- Policy enforcement points
- Access-control decisions
- Transformation rules
- Serialisation and deserialization logic
- Error handling
- Logging behavior
- Conformance test outcomes
- Traceability to semantic sources

A governed architecture prevents implementation behavior from becoming the hidden source of meaning. The architecture defines the meaning first; implementation artifacts then express, enforce, validate, or operationalise that meaning.

Definition

meaning expressed through executable artifacts, deployed configurations, runtime behavior, and technical constraints within an implementation context

Source

DIDO Solutions' usage is informed by software architecture, model-driven architecture, semantic traceability, validation practices, and conformance testing practices.

Note

Implementation semantics does not replace conceptual semantics. Implementation semantics expresses selected conceptual, schema, type, rule, policy, report, or ontology meaning in executable or deployed form. A governed implementation preserves traceability from runtime behavior to the semantic source that establishes the intended meaning.

Example

An FX Demo implementation receives a proposed trade message and executes validation logic before committing the trade state.

Implementation artifact	**Behavior**	**Semantic source**	**Implementation semantic meaning**
Trade message parser	Reads `tradeDate`, `settlementDate`, `counterpartyLEI`, and `notionalAmount`	FX trade schema and conceptual model	The parser recognises the message fields as trade information elements, not as arbitrary data
Date validation function	Rejects a confirmed trade when `settlementDate` precedes `tradeDate`	FX trade lifecycle rule	The implementation enforces the meaning of a valid confirmed trade lifecycle
LEI validation function	Rejects a trade when `counterpartyLEI` lacks a valid LEI format	LEI type and validation rule	The implementation treats the value as a legal entity identifier, not as a generic string
Commit guard	Accepts a state transition only when the predecessor state identifier matches the current committed state	lifecycle semantics and state-transition rule	The implementation preserves the meaning of ordered trade-state progression
Audit log writer	Records validation result, timestamp, validator identifier, and rejection reason	Conformance semantics and governance policy	The implementation creates evidence of the semantic decision

The implementation semantics do not come from the code alone. The code expresses meaning because each implemented behavior traces to a governed semantic source. Without that traceability, the runtime behavior becomes an undocumented interpretation of the domain.
