Type Semantics

Go up to Terms and Definitions

Discussion

Type Semantics defines meaning through types, type names, type membership, type compatibility, type constraints, permitted operations, and type-system rules.

A type identifies a semantic category of values, objects, model elements, messages, events, or expressions. A type system defines the set of types and the rules that govern classification, use, compatibility, and interpretation.

Type Semantics differs from datatype semantics. Datatype Semantics constrains value space, representation, and permitted operations for data values. Type Semantics constrains domain interpretation by identifying what kind of thing a value, object, message, event, or model element represents.

Type Semantics also differs from schema semantics. Schema semantics constrains information elements, structures, relationships, permitted values, and validation rules within a representation context. Type Semantics establishes semantic categories and compatibility rules that schemas, models, APIs, databases, messages, and implementation artifacts express or enforce.

Type Semantic content includes:

- Type names
- Type definitions
- Type membership
- Type compatibility
- Type constraints
- Subtypes
- Supertypes
- Type hierarchies
- Permitted operations
- Assignment rules
- Conversion rules
- Validation rules
- Domain-specific value categories
- Interface type contracts
- Message type contracts
- Traceability to Domain meaning

A governed architecture preserves traceability from Type Semantics to the conceptual model, vocabulary, schema, ontology, rule set, report model, API contract, implementation artifact, or conformance test that establishes or validates the intended meaning.

Definition

meaning defined through type membership, type compatibility, type constraints, permitted operations, and type-system rules within a defined context

Source

DIDO Solutions usage is informed by type-system practice, software architecture, schema-language practice, modeling practice, semantic modeling practice, and model-driven architecture principles.

Note

Type Semantics does not reduce to Datatype Semantics. A datatype constrains representation. A type constrains domain interpretation.

For example, `LEI`, `ISIN`, `CurrencyCode`, `JurisdictionCode`, and `TradeIdentifier` often use a string datatype. Type Semantics prevents an implementation from treating those values as interchangeable strings.

Example

An FX architecture defines several semantic types.

Semantic type	Representative datatype	Type Semantic meaning
`LEI`	`string`	legal entity identifier governed by LEI rules
`ISIN`	`string`	financial instrument identifier governed by ISIN rules
`CurrencyCode`	`string`	currency identifier governed by currency-code rules
`CurrencyPair`	`string`	ordered pair of currencies exchanged in a trade
`TradeIdentifier`	`string`	identifier assigned to a trade
`NotionalAmount`	`decimal`	amount used to calculate settlement obligations
`ExchangeRate`	`decimal`	rate used to exchange one currency for another
`TradeDate`	`date`	date on which the trade occurs
`SettlementDate`	`date`	date on which settlement occurs

The datatype `string` constrains representation for `LEI`, `ISIN`, `CurrencyCode`, `CurrencyPair`, and `TradeIdentifier`. Type Semantics distinguishes the domain meaning and permitted use of each type.

The datatype `decimal` constrains the representation for `NotionalAmount` and `ExchangeRate`. Type Semantics prevents a notional amount from being treated as an exchange rate.

The datatype `date` constrains the representation for `TradeDate` and `SettlementDate`. Type Semantics distinguishes the trade occurrence date from the settlement occurrence date.

A schema, API, DDS topic, database table, or validation function expresses Type Semantics when it preserves these distinctions and traces each type to the governed domain meaning.
