Digital Twins Definition Language

Agenda

- Brief introduction of DTDL v2
- Why DTDL?
- Some experiences
- Collaboration and interoperability
- Questions and discussion

Digital Twins

- **Physical devices** deployed in physical environments
- **Physical entities** (buildings, streetlights, storage tanks, etc.)
- **Logical entities** (time zones, manufacturers, etc.)
- **Semantics** (an element is a measurement (as opposed to a state) and a temperature measured in degrees Celsius)

- **Systems of digital twins:** composition of multiple entities, i.e. the system's graph
  - ex. North Seattle Warehouse #W321

What is Digital Twins Definition Language?

- **Open source modeling language** for IoT devices, digital twins & systems of digital twins
- Enables **contracts** for separating specification from consumption
  - Plug & Play for devices enables them to be used (and reused) across solutions
  - Digital Twins ontologies enable reuse across interrelated sets of digital twins
- Enables contract enforcement and model-driven clients
- Implementation:
  - **DTDL metamodel**: RDF classes + SHACL constraints
  - **DTDL model**: RDF triples conforming to the DTDL metamodel
  - Programming language-agnostic
  - Intended to be developer-friendly for a broad set of developers

Simple DTDL Example

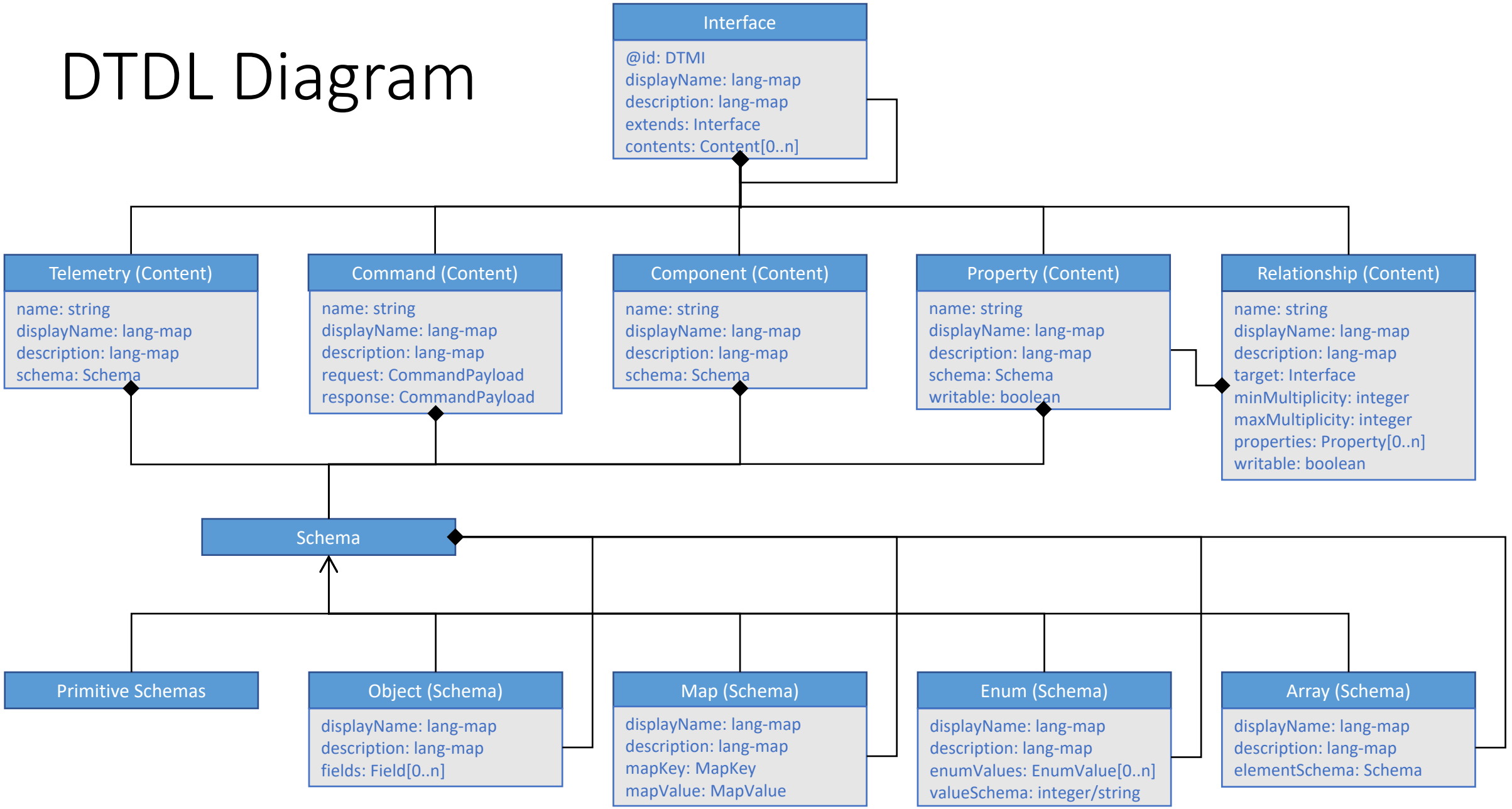
```
{
  "@id": "dtmi:com:example:MyModel;1",
  "@type": "Interface",
  "contents": [
    {
      "@type": "Property",
      "name": "name",
      "schema": "string"
    },
    {
      "@type": "Telemetry",
      "name": "temperature",
      "schema": "double"
    },
    {
      "@type": "Command",
      "name": "update"
    }
  ],
  "@context": "dtmi:dtdl:context;2"
}
```

- An interface with id `dtmi:com:example:MyModel;1`
  - A “name” property
  - A “temperature” telemetry
  - An “update” command

DTDL Basics

- Interface
  - Digital twin type
- Property
  - State of a digital twin
  - Read only or writable
  - Implies synchronization for remote entities
- Telemetry
  - Measurement of a digital twin
  - Implies timestamped stream of measurements
- Command
  - Method on a digital twin
  - Implies “time to start” and “time to complete” for remote entities or async operations
- Component
  - Value type containment in a digital twin
  - No circular references
- Relationship
  - Reference type containment in a digital twin
  - 1:n collection
  - Circular references allowed
- Schema
  - Data type definition
  - Standard primitive schemas provided for strings, numbers, dateTime, etc.
  - Complex schema types for arrays, enums, objects, and maps
- Inheritance
  - Interface subclassing
- Semantic Type
  - Annotations on DTDL elements (additional “semantics”)
  - Example: measurement type and unit

DTDL Diagram



Why?

- Why DTDL and not modeling language X?
  - Device modeling AND digital twin modeling
  - Relationships
  - Linking
  - “Semantic types”
- Why JSON-LD and not JSON or something else?
  - We wanted JSON (programming language agnostic)
  - We realized we wanted reuse and linking
  - We wanted “semantic types”
- Why not RDF triples for entities?
  - Production-grade libraries aren’t there for customers to process JSON-LD or triples
  - Broadly speaking, developers aren’t ready for triples
  - We didn’t want to force a JSON-LD processor to have to run on every entity
- Why not OWL?
  - We wanted to keep things simple for developers
  - We want developers to build their solutions, not necessarily create ontologies

Our experience with JSON-LD

- It’s JSON!
- Well-written spec with enough tools and libraries to learn and use
- Programming language support
  - Great .NET support
  - Good support for the basics in JavaScript
  - We haven’t dug deeply into Java yet
- Open-world assumption
  - Not natural for many things, so we use SHACL and SHACL concepts internally
  - Good for some things, like semantic types
- SHACL covers a lot of what we need
- SPARQL is useful, but performance doesn’t meet our requirements

Interoperability with other specs

- DTDL is analogous to OWL or OPC UA information model
- In many cases, reasonable transformation to/from DTDL models
- DTDL may provide “extension types” extensibility
  - Elements from other specs can be added to DTDL model elements
  - Example: special id (“i=1234”) not allowed in DTDL name

```
{
  "@id": "dtmi:com:example:MyModel;1",
  "@type": "Interface",
  "contents": [
    {
      "@type": [ "Property", "OPCUAProperty" ],
      "name": "name",
      "schema": "string",
      "nodeId": "i=1234"
    }
  ],
  "@context": "dtmi:dtdl:context;2"
}
```

Collaboration and Interoperability

- Areas for collaboration
  - Conversions to/from other specs
  - OWL ontology interoperability
  - Extensibility, including “semantic types”
  - Protocol bindings
  - Cross-language open source software stacks
    - Production grade
    - Tooling ready
  - More!
- Digital Twin Consortium
  - Consortium to influence digital twin technology
  - Also a place to collaborate on DTDL

Questions/Discussion

Resources

- Digital Twins Definition Language: https://github.com/Azure/opendigitaltwins-dtdl
- Digital Twin Consortium: https://www.digitaltwinconsortium.org/
- Brian Crawford: briancr@microsoft.com
- Elio
- Benjamin Cabé: becabe@microsoft.com / @kartben