edi3 UML Profile 1.0 Specification

## Introduction

edi3 semantic specifications are developed by business subject matter experts with deep experience in their relevant business domain. Graphical tools provide an effective means for non-technical modellers to express business semantics and communicate them to non-technical consumers.

UML is the most ubiquitous graphical modelling notation and is the basis for all edi3 semantic modelling. However, without guidance, different modellers are likely to use UML differently for the same purpose. Therefore this specification provides recommendations for the consistent use of UML for the representation of the data model and state lifecycle of information entities that are at the core of all edi3 semantic specifications.

## Goals

This specification will have acheived its purpose when

* All edi3 semantic specifications are consistently represented as UML class and state lifecycle diagrams.
* Models can be successfully interchanged between different UML tools.
* Open API 3.0 interface specifications and JSON-LD ontologies can be generated from the UML models.

Like all edi3 specifications, the editors have aimed for simplicity and ease of use buy the target audience (business modellers in this case). Therefore we

* focus on only two UML diagram types (class diagrams and statechart diagrams), and
* use a limited set of standard UML constructs (avoiding un-necessary use of stereotypes), and
* provide plentiful examples.

It is NOT a goal of this specification to provide guidance on best practices of information mnodelling. We assume that the reader is a capable infomration modeller that wishes to use the edi3 UML profile so that models are consistent and interchangeable.

## Status

This specification is at ![raw](data:text/html;base64,) status. It will move to ![draft](data:text/html;base64,) when at least one UML tool supports the specification as an alpha prototype.

## Dependencies

Users of this specification will typically leverage existing semantic libraries such as the UN/CEFACT Core Component Library (CCL) or other data dictionaries based on ISO-11179. Therefore this specification maps UML constructs to both ISO-11179 and UN/CEFACT CCL.

Although this specification is inteneded to be independent of specific deployment technologies, the eid3 suite is generally targeted at REST API implementation by web developers. Therefore, where necessary, we introduce concepts into the UML profile that will be necessary for the generation of complete OPEN API 3.0 specifications and JSON-LD ontologies.

## Glossary

|  |  |
| --- | --- |
| Phrase | Definition |
| CCL | UN/CEFACT Core Compomnent Library |
| UML | Unified Modelling Language |
| UOM | Unit of Measure |

## Licence

This Specification is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](http://www.gnu.org/licenses) as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This Specification is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

## Change Process

This document is governed by the [2/COSS](http://rfc.unprotocols.org/spec:2/COSS/) (COSS).

## Language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in IETF RFC 2119.

# Business domains

The highest level construct in edi3 modelling is the business domain. A domain is a business subject area that groups a number of related information resources.

* Domains are represented as a UML Package
* Domain names must be unique across edi3.org
* Domains may contain subdomains and, if so, subdomain names musg be iunique within the parent doamin.
* Subdomains are related to the parent domain using a UML package conmtainment relationship.
* Domains and subdomains must have a description

Sample

Transport Domain

Transport Domain

# Data types

Data types are represented as the “type” of a UML attribute. There are only 8 possible types:

|  |  |
| --- | --- |
| Data Type | Usage Rule |
| Text | Any string of unicode characters |
| Numeric | Any integer or floating point number |
| Binary | A binary file URL. File type is indicated by file extension which must be a valid MIME type |
| DateTime | an ISO-8601 date/time string |
| Identifier | a controlled public identifier type such as a business registration number |
| Code | a code from a controlled list such as ISO-3166 country code |
| Indicator | a yes/no, true/faule, 1/0 boolean |
| Measure | A measured value with defined UOM from UNECE-Rec-20 |

*Sample*

Data Types

Data Types

*Implementation Note*

Although simply represented at model layer, the code and identifier types MUST resolve to unambiguous values at runtime. The means by which that is achieved is described in the relevant NDR (Naming & Design Rules) specifications but will typically mean that the data element value is represented as a SchemeURI and code/identifier value tuple.

# Data elements

Every data element has globally unique name called the “dictionary entry name” of the form domain::class.property.type - for example “transport::consignment.number.identifier”

Simple data elements are represented as UML classes and properties (attributes) - eg “transport::consignment.number.identifier” Complex data elements are represented as containment relationships between UML classes - eh “transport::consignment.consignee.party”

* Class names MUST be unique within a domain.
* Attribute names MUST be unique within a class.
* Containment relationship names MUST be unique within the parent class.
* All classes, attributes, and containment relationships MUST include a description which provides a meaningful definition of the data element.
* All attributes and containment relationships MUST have a cardinality.

*Sample*

Consignment

Consignment

# Relationships

The edi3 UML profile includes four standard UML class diagram relationship types, all of which have the standard UML meaning.

* The UML generalisation relationship is used to inicate sub-types or specialisations of an information resource. For example, a “Vessel” is a type of “Transport Means”.
* The UML aggregation relationship is used to indicate a containment relationship between two classes where the contained class may be re-used by other classes. For example “Address” is a frequently encountered structure and may be contained in several different parent structures.
* The UML composition relationship is used to indicate a containment relationship between two classes where the contained class is not re-used by any other class. For example the “consignment” class may contain the “consignment item” class - which is unlikely to be re-used by any class other than “consignment”.
* The UML directed relationship is used to indicate a (non containment) relationship between two classes within or across domains. The relationship is used to indicate discoverable links between separate web resources (eg a consignment resource might reference the bill of lading document as a link URL)

*Sample*

copnsignment item

copnsignment item

*Implementation note*

Although the modelling activity should be largely independent of implementation technology, it is helpful for the modeller to understand the implementation impact of modelling choices - particularly with regard to the choice of relationship type between classes.

|  |  |
| --- | --- |
| Relationship type | Open API 3.0 implementation |
| generalisation | will be “flattened” at implementation layer so that the JSON schema for the sub-type contains all properties of the parent type. In effect, the generalisation is a model layer concept only and has no equivalent at schema layer |
| aggregation | re-usable schema definition that is referenced by the containing schema |
| composition | single tree structure at schema layer with no separate referenceable schema object for the contained class |
| directed relationship | URL link to a separarate web resource. Note that, unlike other relationship types, the directed relationship operates at instance level not schema level - aty schema level it is just a string type that, at instance level contains the URL of the related web resource |

# Web Resource

A web resource represents a URL addressable object that has a defined information model and a state lifecyle. In tradtional EDI modelling, a resource correlates most closely to a business document or message.

A web resource is modelled most like a UML interface. A web resource

* MUST be modelled as a UML class with stereotype <<resource>>
* MUST be named using a noun, not a verb.
* MUST be the target of a UML <<realisation>> relationship where the source is a UML class hierarchy that represents the resource information model.
* MUST include a styate machine model that defines the behaviour of the reource object (see more on this in the next section)
* MUST include two mandatory UML attributes
  + id:string - represnting the unique identifier of the resource instance (usually a GUID)
  + status:code - representing the current state of the resource instance
* MUST include one or more UML operations that define the allowed actions on the resource which SHOULD be named as http verbe (GET / POST / PATCH etc).
* MAY be the target of one or more UML aggregation relationships where the sources MUST be another <<resource>> class. This is used to define sub-resource relationships.
* MAY be the source of one of UML association relationshoips where the targets MUST be other <<resource>>class. This is used to represent links to independent but related resources such as the invoice for a shipment or the bill of lading for a consignment.

*sample*

*implementation notes*

* The UML <<resource>> type will map to a RESTful API web resource such as https://api.acme.com/v1/consignments/{123456789}. The mapping rules for Open API 3.0 specification generation are defined in the Open API 3.0 NDR specification.
* Resopurce models are deliberately kept simple and so do not attempt to model things like different structures for different types of PATCH operations or a list of filters allowed in GET collection actions. These things are likely to vary by implementation. Models and generated OpenAPI specifications published by edi3.org represent only the minimum conformant behaviour for an implemention.

# Resource state lifecycle

Each <<resource>> class MUST have an embedded UML state machine model that describes the state lifecycle of a resource. Resource state lifecvycles are important because

* They describe the set of allowed states and allowed transitions between states. For example an invoice resource may transition through received, approved, disputed, amended, and paid. The precise meanings of each state can have signifcant legal or commercial significance (eg financing an approved invoice will have less risk and hence lower cost than financing a received invoice)
* Under edi3 technical specification guidelines, every resource owner SHOULD publish an event to all authorised subscribers whenever a resource transitions from one state to another. So the state lifecycle model tells susbsriber what they can listen for.

The UML profile for state lifecycles is very simple as it is just a minimal UML state machine

* MUST iunclude an initial and final state and at least one simple state.
* MUST include at least one state transition into and out of each state.
* state names MUST be unique within one state lifecycle and represent the list of allowed values for the “status” attribute of the parent resource.
* All transitions MUST have a UML trigger that describes the business event that triggered the transition.
* triggers SHOULD include a target operation that links to the operation (POST/PATCH etc) of the resource action that triggered the event.

*sample*