DDI-CDI mapping to RDF

# List of issues

1. Association names: Are unique names good (long vs short names)? Can the same names be defined once and reused?
	1. RESOLUTION: unique predicate names because it is a more correct mapping and the same association names in UML are more a convention
2. Definitions: new line – readability. Better multiple lines?
	1. PREVIOUSRESOLUTION: replace \n with “”” plus multiple lines
	2. RESOLUTION: **no changes** because the current solution is more robust for programs. The readability goal has secondary priority.
3. IRIs: we had already some email exchange on this. It could be based either on the xmi:id value or just on DDI-CDI::classifierName. I would like to discuss again pro and cons.
	1. Decision: unique name of class and data type
	2. DECISION approved
4. Protégé: show structure of ontology?
	1. A list of all classes could be confusing, would the package structure make sense?
	2. RESOLUTION: don’t use package structure for having visual hierarchies in Protege. Hierarchy of files (according to package structure) is shown in the P address bar. Selections by file/package are possible.
5. Options for visualizations: lov, others
6. JSON-LD: do class/datatype definitions make sense?
	1. PAC will review
7. JSON-LD comments: /\* or // or nothing?
	1. How to capture: generated by and based on?
	2. RESOLUTION: move the items outside of the context. Related: add logic in the program to avoid the workaround for comma. Comments don’t exist in JSON-LD.
		1. DONE
8. JSON-LD: I focused only on the default generator not on the modular one. I thought the former would be more important in the first place. Can both ways make sense?
	1. Are both approaches important?
	2. RESOLUTION: modular approach doesn’t scale. Advantage of modular approach is less redundancy.
		1. DONE: tests are OK
9. Onto: organizational packages (DDICDIModels, DDICDILibrary, Classes...). What should be achieved?
	1. PAC and JW will review.
		1. DONE
10. UML UnlimitedNatural: xsd:nonNegativeInteger or xsd:double?
	1. RESOLUTION: might be more robust to use xsd:integer.
		1. DONE: But **let xsd:nonNegativeInteger stay** until it will cause problems because it is more correct.
11. Super-/sub class relationships / inheritance?
	1. PAC review → DONE
12. How to indicate the DDI-CDI persistent identifier as identifier? Identifier::ddiIdentifier or eachClass::identifier?
	1. Model change: isID for the attribute identifier of each class, attributes Identifier::ddiIdentifier, InternationalRegistrationDataIdentifier::dataIdentifier,InternationalRegistrationDataIdentifier::registrationAuthorityIdentifier, InternationalRegistrationDataIdentifier::versionIdentifier
	2. RESOLUTION: if UML isID then owl:hasKey
	E.g.
	cdi:Identifier a owl:Class; owl:hasKey (cdi:ddiIdentifier).
	cdi:DdiIdentifier a owl:Class owl:hasKey (cdi:p1 cdi:p2 cdi:p3).
13. Are the generated onto and JSON-LD files valid syntax? The lexer of the pygment formatter in Sphinx complains.
	1. DONE Seems to work now after some fixes
14. How to make use of trace relationships in the model: XSD data types, RDF langstring, DCTERMS fields? Realized by UML trace in the model but DCTERMS fields are in the model just attributes. Trace is not possible for attributes. Better specifying equivalence in the properties file? Also the UML primitives? However, not all are totally equivalent like string vs language string. Most DCTERMS fields are defined more complex in CDI. Possible configuration example: XMLSchemaDataTypes::XsdAnyUri = XMLSchema::anyURI, another option could be the emulation of hashes in OCL sets for the mapping purpose.
	1. Which approaches could make sense? rdfs:seeAlso, rdfs:isDefinedBy, rdfs:subClassOf, owl:equivalentClass, rdfs:subPropertyOf, owl:equivalentProperty?
	2. URIs for items in Context? Like XMI#DCTerms-alternative?
	3. RESOLUTION: the relations captured by the UML trace relation are encoded as rdfs:subClassOf / rdfs:subPropertyOf respectively.
	4. RESOLUTION: we don’t use the UML trace relation in the generator because it is too vague. It has mainly documentary purposes.
	Instead, we **create a TTL file containing all the corresponding axioms for aligning with other vocabularies**,

And simply owl:import it (or include it directly) at the top level of the ontology.

1. The content-type reported by the web server for the TTL and JSON-LD file is text/plain, which causes problems for some tools
	1. Currently only the static bitbucket server is used. The plan is to use ddialliance.org for this purpose. We need to figure out with the DDI Alliance the IRIs for the DDI-CDI ontology and the content types.
	2. RESOLUTION see 17
	Where the context would be served at a similar but distinct IRI
	E.g. <http://ddialliance.org/Specification/DDI-CDI/1.0/RDF>/context
	Which must serve content-type: application/ld+json (NOT text/plain)
2. The JSON-LD contexts are invalid: the "generatedBy" and "basedOn" keys are not allowed here
	1. RESOLVED, see 7.
3. The IRIs of the terms in the ontology (cdi: namespace) should dereference to the ontology; I used a make-up IRI for the moment (http://ddialliance.org/Specification/DDI-CDI/1.0/RDF#Agent) which results in 404. If we want to keep this IRI, is it possible to setup a redirection to the actually ontology there?
	1. <http://ddialliance.org/Specification/DDI-CDI/1.0/RDF> should support Content Negotiation, see [Best Practice Recipes for Publishing RDF Vocabularies](https://www.w3.org/TR/swbp-vocab-pub/), [Content Negotiation by Profile](https://www.w3.org/TR/dx-prof-conneg/#motivation)
		1. If turtle, RDF/XML or JSON-LD is required, serve or redirect to a corresponding rendering of the *whole* ontology
		2. If HTML is required, serve a small HTML page (could be generated by Acceleo) containing Javascript code that knows where to redirect for every possible #fragment (e.g. #Agent redirects [there](https://ddi-alliance.bitbucket.io/DDI-CDI/DDI-CDI_2022-07-20b/doc/_build/DDICDILibrary/Classes/Agents/Agent.html))
		Hints for solution: [l1](https://stackoverflow.com/questions/298503/how-can-you-check-for-a-hash-in-a-url-using-javascript), [l2](https://developer.mozilla.org/en-US/docs/Web/API/Location/hash), [l3](https://stackoverflow.com/questions/503093/how-do-i-redirect-to-another-webpage)
4. UML makes a distinction between regular associations, aggregations, and compositions. DDI-CDI uses this but basically only for documentation purposes. Most syntax representations don’t have an equivalent. What to do with this?
	1. RESOLUTION: these distinctions have no equivalent in the RDF world, so we just ignore them in the mapping
5. Multiplicities/cardinalities are used for class and data type attributes as well as for association ends. What to do with this? Are owl:minCardinality/owl:maxCardinality useful? Should this be better enforced in ShEx/SHACL representations?
	1. OWL cardinality constraints are about the real world described by the data
	(e.g. every Person has exactly 2 parents – even if the database contains some persons without describing any of their parents).
	SHACL / ShEx cardinality constraints are about the data (e.g. every Pupil in a school database may have exactly 1 parent recorded)
	2. RESOLUTION: cardinality constraints in the UML model are mostly syntactic, so **they must NOT be reflected as OWL cardinality constraints**
	3. **FUTURE WORK** use Acceleo to create structural schemas:
	ShEx and/or SHACL and/or JSON-Schema
6. Class and data type attributes are ordered in DDI-CDI if more than one instance is allowed. The main purpose of the order is addressing a specific attribute. What would be the approach in RDF? The use case is the roundtrip of a metadata instance between multiple syntax representations.
	1. RESOLUTION: **keep the RDF modelling as it is**, i.e. it may lose the order of multiple values; noticing that
		1. JSON-LD encodes multiple values in arrays, and keeps the order as long as the data stays in the JSON-LD world (as opposed to triples)
		2. **RDF-star could be use to add order metadata on the edges without altering the current modelling**

		<#book> dct:creator <#alice> {| ex:order 1 |};
		 dct:creator <#bob> {| ex:order 2 |}.

		{ “context”: {
		 “creator”: {
		 “@id”: “<http://purl.org/dc/terms/creator>”,
		 “@container”: [“@ordered”]
		 }
		 },
		 “@id”: “#book”; “creator”: [ “#alice”, “#bob” ]
		}
7. Enumerations in JSON-LD
	1. RESOLUTION: they are embedded in the property declaration with the possibility of duplication at each use.