Common Terminology Service 2 Specification
Version: Alpha 1

OMG document number: ad/2010-05-12

Primary contact: Jobst Landgrebe and Alan Honey,
{jobstLandgrebe, aphoneysys}@gmail.com

Proposed acronym: CTS2

May 19, 2010

This specification is a response to the RFP Common Terminology Services Release 2 (CTS2), OMG document number ad/2009-09-17.
The full reply to the RFP consists of this document, there are no additional documents.

Contents

Part I.

Evaluation specific information (non normative)

1. General information

1.1. List of OMG members submitting this specification

This specification is submitted by the International Institute for the Safety of Medicines (ii4sm), Basel, Switzerland and Visumpoint Enterprise Architecture, Atlanta, Georgia, USA.

1.1.0.1. List of supporters

This specification is supported by:

- Volz Innovation GmbH, Karlsruhe, Germany
- Jim Davies, Computing Laboratory, Oxford University, Oxford, UK
- Ocean Informatics, New South Wales, Australia
- The Clinical Information Consultancy, Reading, Berkshire, UK

1.2. Copyright waiver

1.3. Submitter contact points

ii4sm: Jobst Landgrebe, jobst.landgrebe@ii4sm.com, phone +49.151.23525359
Visumpoint: Robert Lario, robert.lario@viusmpoint.com, phone +1.404.713.0612

2. Specification contents

2.1. Overview

2.1.1. Purpose

Over the last four decades, knowledge representation for the use in computers has evolved from simple terminologies (lists of terms) to complex knowledge representation systems based on first order predicate logic. The latter type of systems also form the backbone of the semantic web vision.

In many domains, e.g. medicine and defense, more complex terminologies and ontologies are evolving and moving into practical usage, e.g. SNOMED-CT. This evolution is creating a market demand for solutions which support logical knowledge consistency checks for large ontologies, logic-based expressive query operations, logical entailment checking, concept semantics resolution and many other machine inference tasks. Existing knowledge representation standards and solutions have addressed this problem, and these solutions also support the various issues of terminology management, e.g. terminology versioning, concept identification versus representation, concept relationships and many more as the are needed to accomodate the needs arising in the usage of more conventional terminologies such as LOINC, ICD 10 and C-NPU.

Therefore, this specification proposes to leverage existing knowledge representation standards to fulfill the requirements of the CTS2 RFP: we propose to use W3C OWL2 as the core of the specification and extend OWL2 with several additional features absent in OWL and its de facto standard Java API, the OWL API:

- a flexible and expressive metadata structure superimposed on the ontology to capture provenance, versioning and other administrative data
- pre-defined subsets of ontologies with a metadata structure: value sets which fulfill terminology usage requirements in information systems using class models which instantiate class attributes using pre-defined terminology subsets, e.g. as found in medical informatics
- mappings, which are metadata-annotated sets of relations between elements of different ontologies; such mappings are frequently required to allow instance data interoperability, exchange and harmonisation

- a proper, platform independent service interface specifying interface types and behaviour

2.1.2. Structure of the specification

This submission is strucutre in the following fashion: part I contains the design rationale and the resolution of the RFP requirements and requests, part II contains the specification. The latter is structured according to the OMG template. The specification comprises a platform indepentend model and a platform specific model.

2.2. Design rationale

As stated above, our submission approach is to specify CTS2 as a service interface to a knowledge representation system component based on description logic (DL), more specifically, on W3C OWL2 ($\mathcal{SROIQ}(\mathcal{D})$) and SKOS. With the addenda on top of OWL which we introduce (metadata, value sets and mappings), we allow implementers to capture the benefits of contemporary knowledge representation with machine inference capabilitites while fully supporting traditional terminology requirements - each terminology can also be expressed as an ontology (i.e. any terminology can be seen as the Tbox of an ontology). In our opinion, a terminology service specification not using knowledge representation (e.g. description logic) would be stillborn as the market is increasingly asking for machine inference against computable representations of human knowledge, and so a conventional terminology service without inference capabilities would not be sufficient to meet the market demand. The requirements for OWL2 and machine inference capabilities for CTS2 are stated in section 5.4 of the HL7 SFM as well as in the problem statement of the OMG RFP (section 6.1, item no. 6).

2.3. Proof of concept statement

As a proof of concept underpinning this specification, we have implemented a prototype for CTS2 using semantic web technology (implemented using an RDF triple store with SPARQL and Sesame API with CTS2-compliant interface exposed as web service). This prototype will be demonstrated at the June 2010 OMG meeting to the AD and HD task forces.

2.4. Resolution of RFP requirements and requests

There are a number of areas identified in the RFP where we feel that the initial CTS2 Specification should postpone or omit in order to provide a meaningful and reasonable initial implementation. The following provides detailed discussion on our approach to each of the requirements and issues for discussion in the RFP.

2.4.1. Mandatory requirements (RFP section 6.5)

1. The submission contains a PIM and a PSM. The interface of the service is expressed using UML. However, the syntax and semantics of OWL2, the interface types and the behaviour of the operations is described using Z notation, an ISO standard (ISO/IEC 13568:2002) for a set-theory based software specification language. We use these additional means which go beyond ODM, MOF, and OCL in order to provide a more precise characterisation of data semantics and interface behaviour semantics (cf. part II section 3.2.2).

2. A WSDL PSM is included.

3. Scope of interfaces/operations

   - *Administration interface:* we only provide operations to import ontologies and value sets as well as their versions. We believe that an "application" that maintains terminologies/ontologies should in general provide capabilities such as export, notifications etc, but we do not believe that this capability needs to be or should be provided through a service interface. Primarily, these are fine grained, interactive functions better suited to an integrated client application rather than a separate service.
   - *Query interface:* With the exception of operations supporting HL7 concept domains and usage contexts, all operations are provided via the proposed CTS2 query or reasoning interfaces (mappings to the HL7 SFM are provided in the specification). Concept domains and usage contexts are not provided as they are used to support the HL7-specific terminology binding approach. In our view, this approach to terminology binding is beyond the scope of a terminology service – in a well structured Service Oriented Architecture, the appropriate modularity would be compromised by including this functionality which we believe belongs into a separate service or software module such as a rules engine. Another aspect of the HL7 SFM which is not directly supported by this specification are value sets containing elements from more than one code system. In our view, such value sets result from erroneous design of the information model class attributes to which such value sets bind, i.e. the attributes can always be designed in a way to require value sets from only one ontology. However, applications which need value sets from more than one ontology could retrieve more than one value set from CTS2 and construct the composite value set. Another way to support this HL7 requirement is to construct an ontology which contains the HL7 code systems from which a value set is drawn (which is not always possible as this may lead to inconsistent ontologies in some cases).

- *Authoring interface:* We are not intending to include an authoring interface at this time. The reason is that terminology authoring is a collaborative, distributed effort as used in software development. It is a fine-grained user intensive process with complex human-to-human and human-to-machine-to-human interactions. Such work heavily involves version, branch and state control, and is usually supported via local applications/tools. Collaboration support is provided via distributed authoring version control tools like subversion and communication via meetings, forums, email threads and other forms of telecommunication. We do not believe that it makes sense to expose the functionality for such processes via a SOA service, which are not optimal to support fine-grained, highly interactive work. Therefore, we do not provide an authoring interface.
  However, sophisticated tooling support for ontology authoring already exists and is evolving further. Many of these tools do or will allow export of ontologies into RDF/XML, which allows direct import into our proposed CTS2 solution. One example is the IHTSDO workbench, used to author SNOMED. More general purpose tools, such as TopBraid and Protégé offer authoring of Ontologies.

4. With the exception of the HL7 specific Concept Domain support (as noted above), the specification realises the CTS2 query functional profile and parts of the terminology administration functional profile for mature terminologies.

5. The HL7 profile and the HL7 vocabulary requirements are only partially supported. The main reasons for the lack of full support are i) HL7 requirements encompass terminology binding using the HL7 approach which is out of scope of a terminology service (more details cf. above) and ii) The HL7 terminology requirements as stated on the supplied HL7 URL comprise solution assertions which contradict fundamental linguistic results on knowledge representation and therefore are not supported. As a typical example, consider the following HL7 requirement concerning associations/relationships: "Relationship types need to be able to be linked to a concept within the code system". Relationship types are designated as role assertions in OWL2/SROIQ and as predicate characteristics in general first order (predicate) logic. However, they are not linked to concepts, but characterize DL roles. One could of course list all concepts of an ontology which are arguments of roles of a certain type. But from a linguistic perspective, such a list does not make sense (e.g. it is not sensible to generate a list of all concepts which are arguments of transitive roles).

6. The submission is based on the generic standards OWL2 and SKOS and therefore ready for usage in all industries.

7. All of the terminologies mentioned in item 7 can be supported. Details on post-coordination support are provided in the specification, section 7.3.4.4

8. The submission does not directly support the ISO11179.3 R2 notion of a conceptual domain, neither in its information model nor via operations. The reasons for this are the following.

   - An ISO11179.3 conceptual domain is intended to define the meaning of set of concepts while a value domain provides concept representations (in the sense of designations). This idea pre-supposes that computational representations of concepts can distinguish concept meaning and designation. However, all contents stored in a machine are still symbols, i.e. designations, and as such machines have no inherent sense of meaning. Therefore, from a computer linguistics perspective, the notion of a conceptual domain is problematic.
   - However, the CTS2 specification uses the notion of a value set to represent sets of concepts or individuals; these are sets of OWL2 class or individual identifiers (IRIs). For each of these ontology element identifiers, representations (designations) exist; these are symbols used to designate concepts or individuals in human language. At the machine level, both of these, the identifiers and the designations, are mere concept symbols (representations, designations).
   - Therefore, CTS2 provides full support for concepts and their designations. However, this support is not organised along the lines of ISO11179.3, but along the lines of OWL2/SKOS. So the requirement expressed in mandatory requirement no. 8 is indirectly supported.

2.4.2. Optional requirements (RFP section 6.6)

1. This submission covers machine inference for ontologies via its reasoning interface. The reasoning interface covers some of the CTS2 query operations (value set membership and subsumption, concept subsumption), but also additional operations required for reasoning against ontology T-boxes and A-boxes. The reasoning interface makes CTS2 a contemporary service able to meet market demand for machine inference against knowledge representation.

2. No PSM interface beyond WSDL supplied at this stage.

3. Check value set subsumption, determine transitive concept relationship are provided. Operations to create mappings or notification operations are not provided for the reasons state above.

4. An HL7 profile for CTS2 in the sense of the SFM is not provided. The primary reason is that we believe that the capabilities described belong in an separate service or other module which would be a client of the proposed CTS2 specification, and which would complement it to provide terminology binding capabilities. Using this combination, HL7

would be able to meet all requirements for terminology management and binding and be able to use a contemporary solution to ontology management.

5. Purchaser support language not provided in version alpha 1.

6. Nomenclature mappings between the HL7 CTS2 SFM, the OMG CTS2 submission and OWL2 are provided in section 4 of the submission.

2.4.3. Issues to be discussed (RFP section 6.7)

1. There are two major deviations from the HL7 SFM requirements: the lack of an authoring interface and the lack of support for terminology binding. Both are discussed in detail above.

2. Terminology meta model (syntax) structure heterogeneity occurs at level of syntactic complexity (which types of elements exist and how they can be combined to create complex expressions) as well at the metadata level. All terminologies use concepts, some also roles and equivalence / subsumption relationships. In addition, ontologies often use existential and universal quantification and support individuals and data properties as well as other syntax elements. This specification supports all these elements, i.e. terminologies/ ontologies which can be expressed using OWL2. Terminologies may also allow the storage of various metadata about the core ontology elements. Many of these are explicitly supported by this specification as it is based on the HL7 SFM and SKOS. Metadata which are not supported by this specification directly can be supported using the characteristic type/characteristic extension mechanism. At the OWL2 level, all metadata can be represented as annotations. Terminologies/Ontologies violating or transgressing the OWL2 syntax are not supported, e.g. terminologies using roles with higher -arity than binary. Ontologies allowing second order logic are not supported. To allow import into a CTS2 instance, ontologies and terminologies have to be transformed into a form that is consistent in the sense of Tbox inference. This leads to unified representations of the ontologies within the CTS2 instance with a harmonisation of the meta models against the OWL2 syntax and semantics. Overall, the typical variance in terminology meta models is easily covered by this specification. When technical implementation and quality of service are considered, it may be necessary to make a trade-off between ontology expressiveness and computational properties. The OWL2 semantic profiles offer a good starting point for such considerations.

3. The following characteristics make this specification usable in a cross-industry fashion:

   a) Genericity of the CTS2 model, which provides a layer of metadata and some terminology grouping classes (value set and mapping) on top of OWL2

394 b) Broad scope of the CTS2 operations within the clean boundary of a SOA service

395 c) Exclusion of domain specific (e.g. HL7) terminology bindings, i.e. recommendation
396 for inclusion into a separate service or module

397 4. Information about a CTS2 instance can be stored in a metadata repository or a UDDI
398 repository. The variance in the terminology meta-models is dealt with upon import
399 into an instance where a unified representation of the imported ontologies is generated,
400 so that the combinatorial explosion described in the SFM section 5.2 does not occur.
401 We believe that it is not feasible to manage terminologies/ontologies in any reasonable
402 fashion without the unified representation. Therefore, the main metadata about CTS2
403 which need to be exposed are the interface details (which can for instance be represented
404 as a WSDL file), the quality of service and the supported ontologies.

405 5. At the PIM specification level, the size and complexity of supported ontologies is not
406 critical: the full behaviour is specified for the full complexity. At the implementation
407 level, dealing with large ontologies of relatively low complexity (SNOMED-CT is huge
408 but only has EL-DL syntax) requires careful considerations. For SNOMED-CT, a ser-
409 vice instance only providing the OWL2 EL profile can be considered to optimise the
410 computational properties and the quality of service of the CTS2 instance.

411 6. The CTS2 designation class offers support for localisation and internationalisation.
412 Whenever a non-default concept designation is required, a CTS2 call to obtain al-
413 ternative concept designations has to be done. This however corresponds to a simple
414 SPARQL query at the implementation level and should not lead to user-relevant perfor-
415 mance penalties at the user interface level. Another aspect of localization is the ability
416 for organizations to maintain their own local or extended versions of standard ontolo-
417 gies. This specification does not directly support this requirement as it does not specify
418 authoring functionality. The requirement can be supported by using authoring tools.
419 These can create an extended ontology as an ontology of its own and providing support
420 for source ontology version upgrade management.

421 7. SNOMED-CT is a huge ontology with a low expressivity (EL-DL) which represents
422 a tiny subset of the OWL2 syntax. CTS2 is specified to fully support OWL2 and
423 thus SNOMED-CT. However, due to its size, careful considerations with regard to
424 the functional support for inference have to be made. A CTS2 instance providing
425 SNOMED-CT should probably make usage of the OWL2 EL profile and carefully choose
426 a reasoning engine able to deal with large Tboxes without support for Aboxes and
427 reasoning beyond the needs of an EL-DL. In order to use SNOMED with CTS2, a
428 representation of SNOMED in OWL-compliant RDF-XML must be generated from the
429 SNOMED distribution format.

The submitters have carefully considered interaction with the IHTSDO workbench. Essentially, the IHTSDO workbench is an important element of the set of solutions for authoring support. To integrate the work bench with CTS2, functionality for the workbench could be developed in order to export SNOMED-CT or RefSets thereof (equivalent to CTS2 value sets of concepts) in proper OWL-compliant RDF/XML ready for import into a CTS2 persistence store (RDF triple store). As an alternative, the native workbench export format could be transformed into OWL-compliant RDF/XML. With either of these, a seamless deployment of IHTSDO workbench content via CTS2 (including post-coordination support) via the import operations of the management interface will be possible.

8. Section 7.3.4.4 of the specification gives details about the full support for post-coordination in CTS2.

9. This specification provides version management by version identifier for ontologies, value sets and mappings, but no version identifiers for ontology elements. According to ontology authoring best practices , concepts, roles and individuals are always versioned with the ontology they belong to. In an authoring setting, the time stamp attribute for these elements can be used to generate change logs which need to be published along with the publication of new ontology versions (for more details, cf. section 7.3.3.1).

10. The PIM model provides slots for the information of all the classes of the SFM model with the exception of the classes CodeSystemSupplement, JurisdictionalDomain, UsageContext, ValueSetContextBinding and ConceptDomain. While the functionality of the first class can be obtained by creating local ontology versions comprising supplements, the other classes are required for the HL7-specific terminology binding solution which is not supported by this specification for the reasons described above. Note that there is an attribute JurisdictionalDomain on the ValueSet schema, though.

11. As this specification does not specify an authoring interface and favours a ontology element versioning model using versioning at the ontology level, this point is not addressed.

2.4.4. Evaluation Criteria (RFP section 6.8)

5. Federated ontologies: Our view on ontology federation is that local ontology clients will in general use different local instances of ontologies without a need for federation as it is usually very predictable which applications need which ontologies. In the rare case that there is a need for federation, e.g. a local application has to look up a concept held in a non-local ontology, the service registry should provide the location of the ontology (at initialization or run time depending on the actual implementation strategy). In other words, the ontology federation problem is not that common in practice and therefore

should be supported using broker functionality outside the service. This is also sensible from a SOA service functionality scope and partitioning perspective.

2.4.5. General Requirements (RFP section 5)

There are no deviations from general requirements with the exception of the metadata representation: at the PSM level, the metamodel for the representation of the CTS2 classes is rendered in RDF/XML, a format which currently cannot be expressed using XMI.

Part II.

Specification (normative)

USE OF SPECIfICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification. Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means–graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems–without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA®, and XMI®are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™, MOF™, OMG Interface Definition Language (IDL)™, and OMG SysML™are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue
(http://www.omg.org/technology/agreement.)

1. Scope

The Common Terminology Service 2 (CTS2) specification is a SOA service interface specification which aims at the standardisation service interfaces and behaviour applicable to ontologies. Ontologies are formalised representations of human knowledge and comprise an *ontology spectrum* of increasing knowledge complexity which ranges from simple lists of concepts and subject heading systems to more structured taxonomies, thesauri, classifications, terminologies and complex ontologies with a sophisticated, first-order logic based syntax, set-theory based semantics as well as a separate representation of knowledge axioms and individual assertions. CTS2 is specified to allow querying and import of the full ontology spectrum and machine inference (reasoning) against ontologies which allow this. Support for fine-grained authoring via a service interface is not specified.

1.1. Purpose

1.1.1. Business purpose

Over the last four decades, knowledge representation for the use in computers has evolved from simple terminologies (lists of terms) to complex knowledge representation systems based on first order predicate logic. The latter type of systems also form the backbone of the semantic web vision.

In many domains, e.g. medicine and defense, more complex terminologies and ontologies are evolving and moving into practical usage, e.g. SNOMED-CT. This evolution is creating a market demand for solutions which support logical knowledge consistency checks for large ontologies, logic-based expressive query operations, logical entailment checking, concept semantics resolution and many other machine inference tasks. Existing knowledge representation standards and solutions have addressed this problem, and these solutions also support the various issues of terminology management, e.g. terminology versioning, concept identification versus representation, concept relationships and many more.

Therefore, this specification proposes to leverage existing knowledge representation standards to fulfill the requirements of the CTS2 RFP: we propose to use W3C OWL2 as the core of the specification and extend OWL2 with several additional features absent in OWL and its de facto standard Java API, the OWL API:

- a flexible and expressive metadata structure superimposed on the ontology to capture

598 provenance, versioning and other administrative data

599 • pre-defined subsets of ontologies with a metadata structure: value sets which fulfill ter-
600 minology usage requirements in information systems using class models which instan-
601 tiate class attributes using pre-defined terminology subsets, e.g. as found in medical
602 informatics

603 • mappings, which are metadata-annotated sets of relations between elements of different
604 ontologies; such mappings are frequently required to allow instance data interoperability,
605 exchange and harmonisation

606 • a proper, platform independent service interface specifying interface types and behaviour

607 1.1.2. Technical purpose

608 The standardisation of service interfaces allows to reduce the price for software component
609 re-use and recombination and integration by enabling service client developers and integrators
610 to rely on the service contract as expressed via the interface. Ontology querying and machine
611 inference against ontologies are behaviours types required by many contemporary system.
612 The purpose of this specification is to standardise access to such behaviour.

613 1.2. Relationship to HL7 CTS2 SFM

614 The CTS2 specification is the product of a collaborative effort between Health Level 7 (HL7)
615 and the Object Management Group (OMG). Within HL7, members of the Vocabulary Work-
616 ing Group produced a functional specification document, a Computation Independent Model
617 designated as Service Functional Model - SFM. This was used as the basis for the require-
618 ments expressed in the RFP against which this specification was developed.
619 The HL7 SFM is terminology-centric. Because evert terminology can be defined as the subset
620 of an ontology, the terminology requirements for CTS2 as stated in the HL7 SFM are covered
621 by this specification.

2. Conformance

CTS2 provides a canonical representation for input and output parameters. The next version of this specification will add a *representationFormat* input parameter which will allow clients to select an output format (e.g. Java object serialisation, RDF/XML etc.).

3. Normative References

3.1. Relationship to OWL2

This standard uses description logic (DL), a computable subset of first order (predicate) logic, as a foundation of CTS2 because it offers a viable, well established syntax and semantics for the formalisation of terminologies and more complex forms of knowledge representation and enables the usage of ontology-based reasoning against information schemata (Tbox) and instances (Abox). Following industry best practices, this specification uses an established description logic standard, the W3C's OWL2, as its foundation. In addition to OWL2, another W3C standard for knowledge representation, simple knowledge organisation, SKOS, is used to the maximum extent possible within CTS2 (cf. 3.2.1).
Source for this specification are $\mathcal{SROIQ}(\mathcal{D})$ (1) and OWL2: *http://www.w3.org/TR/owl2-syntax/*, *http://www.w3.org/TR/owl2-direct-semantics/*

3.1.1. OWL2 semantic profiles

Implementers of this specification can conform to the OWL2 semantic profiles defined by the W3C: *http://www.w3.org/TR/owl2-profileAs/*. The profiles EL, QL and RL are specified for OWL2, we cite the summaries of the profiles here:

- EL is a profile which "is particularly suitable for applications employing ontologies that define very large numbers of classes and/or properties, captures the expressive power used by many such ontologies, and for which ontology consistency, class expression subsumption, and instance checking can be decided in polynomial time."

- The QL "profile is designed so that sound and complete query answering is in LOGSPACE [...] with respect to the size of the data (assertions), while providing many of the main features necessary to express conceptual models such as UML class diagrams and ER diagrams."

- The RL profile "is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate both OWL 2 applications that can trade the full expressivity of the language for efficiency, and RDF(S) applications that need some added expressivity from OWL 2."

Implementers should decide which of the profiles to support based on the characteristics of the ontologies which are to be supported by the implementation as well as the quality of service requirements (e.g. computation time) for the implementation. Usually, a trade-off between expressivity and computation properties has to be made. As an example, to support medical terminologies, the EL profile is probably well suited.

3.2. Relationship to other standards than OWL2

3.2.1. W3C SKOS

W3C Simple Knowledge Organisation System[1] is a specification which defines types and a common syntax for the representation of knowledge collections such as *thesauri, taxonomies, classification schemes and subject heading systems*. Many of the types defined in this standard are closely related to CTS2, and are utilised throughout this specification with appropriate reference to SKOS and an explanation of eventual deviations.

However, for some types SKOS allows constructs which deviate from syntactic restrictions of OWL2 in order to allow a looser expressivity at the expense of computational decidability. Examples are skos:Concept, skos:ConceptScheme, or skos:SemanticRelations. For this specification, we propose to follow the rationale that every form of knowledge representation supported by CTS2 must be expressed as a subset or full set of OWL2 syntax[2] to ensure computational decidability of the ontologies. Therefore, knowledge organisation systems expressed in SKOS would have to be transformed to be fully OWL2 compliant upon import into a CTS2 instance. While such a transform may impose too high costs on web publication of such artefacts, it is generally feasible for the purpose of distribution of the contents of such systems via CTS2 without any loss of semantics.

SKOS does not specify behaviour. CTS2 uses types from SKOS and adds additional metadata types as well as behavioural specifications not present in SKOS.

3.2.2. OMG MDA, UML, MOF, OCL and ODM

OMG standards like UML, MOF, OCL, QVT and ODM are helpful to illustrate or render the syntax of some of the types of this specification, but are not optimal to specify them fully (2). Furthermore, we believe that the semantics of description logic are not optimally expressed in ODM, and much of the behaviour described in this specification to provide OWL2 entities and metadata pertaining to them to service clients requires a more expressive specification languages than OCL. However, this specification provides a UML model (class diagram) of the service's entities for illustrative purposes and normative UML models of the service interfaces.

[1] URL: http://www.w3.org/TR/skos-reference

[2] with the exception of cross-ontology value sets and mappings, cf. 7.3.4

Furthermore, this specification is fully MDA compatible; it provides a PIM and a PSM using an alternative, non-object technology specification paradigm.

3.2.3. Z notation

This Platform Independent Model specification uses a conceptual variant of the Z notation (3, 4). Fully valid Z notation will be used in a later version of this specification.

The Z notation is an ISO standard (Information Technology - Z Formal Specification Notation - Syntax, Type System and Semantics (ISO/IEC 13568:2002 ed.). 2002-07-01). Here we propose a slightly modified flavour of this standard in order to use it for the description of SOA interface types and behaviour. We believe that Z is a good choice Because a set-theory based notation is an adequate way to specify a service that deals with first-order logic, a formal language defined using set theory. The PIM consists of Z axioms and schemata which represent the types of the system and its state space (in the Z sense). For more details on usage of Z for SOA service specification, cf. section 7.4.1.

Note that we only provide one full sample operation description in Z notation (List Concepts), while merely specifying input and output parameters in Z notation and using English for behaviour description for the other operations.

4. Terms and Definitions

The following table provides a mapping between terms used in the HL7 SFM, this submission and the W3C OWL2 specification.

Nomenclature Mappings		
CTS2 HL7 SFM	This submission	W3C OWL2
Code system	Ontology	Ontology
Value set	Value set	NA
Concept	Concept	Class
Association	Role	Role
Association	Equivalence and subsumption relationsship	
NA	Individual	Individual
Designation	Designation	(Annotation)
Defined Entity Property	Characteristic type	(Annotation)
Entity property version	Characteristic	(Annotation)
Concept domain, usage context, supplement	NA	NA

NA - non applicable.

Note: the HL7 SFM conflates roles and equivalence/subsumption relationships using a single term, "association".

Mappings between HL7 SFM operations to the operations of this specification are provided in the operation definnitions.

5. Symbols

All symbols are defined in the main part of the specification.

6. Additional Information

6.1. Acknowledgememts

The following companies submitted this specification:

- International Institute for the Safety of Medicines, Basel, Switzerland
- Visumpoint Enterprise Architecture, Atlanta, Georgia

The following companies supported this specification:

- Volz Innovation GmbH, Karlsruhe, Germany; rv@volzinnovation.com
- Jim Davies, Computing Laboratory, Oxford University, Oxford, UK; jim.davies@comlab.ox.ac.uk
- Ocean Informatics, New South Wales, Australia; thomas.beale@oceaninformatics.com
- The Clinical Information Consultancy Ltd., Reading, Berkshire, UK; david@clininfo.co.uk

7. CTS2 Platform Independent Model

Because CTS2 is intended to support richly-axiomatized terminologies such as SNOMED-CT, we utilize description logics as the semantic core for representation of terminologies. We have identified the W3C standard OWL as appropriate representation formalism to represent terminologies and extend it with elements needed in medical settings. In particular, (a) the means to attach reach metadata to each element in the terminology and (b) to specify (parts of) terminologies as value sets for use in medical information systems.

As the syntactic and semantic core of CTS2 is based on OWL2, this specification is constructed in a layered fashion: we start by recapitulating OWL2 syntax and semantics in a condensed form and then add those entities from them by adding metadata and value sets.

Later sections define the interface operations, their I/O types and their behaviour as required to manipulate and access terminologies..

7.1. CTS2 foundation for terminologies: OWL ($\mathcal{SROIQ}$) axioms

OWL itself is a syntactic representation of a logical formalism called description logic. In particular a logic denoted in scientific settings as $\mathcal{SROIQ}$. The basic elements of a terminology (also called ontology) are

- atomic concepts (A), with two predefined concepts called bottom $\bot$ and top $\top$,
- named (o) or anonymous[1](a) individuals that instantiate concepts or stand for themselves,
- data types (V) utilizing XML Schema datatype definitions,
- data values (u) permissible for each datatype,
- predicates predicating over concepts and individuals: roles (OWL: object property) R including so-called simple roles S,
- predicates predicating over individuals and data values designated as datatype properties P,

[1] Called anonymous as they do not have an identifier and are represented in the OWL carrier syntax as RDF blank nodes.

- instantiations of these relationships that relate individuals with individuals ($r(o_1, o_2)$) or individuals with data values ($p(o_1, u)$).

Using logic-based constructors arbitrary (composite) concept descriptions C can be built. These constructors are detailed in Table 7.1. Furthermore, OWL has annotation properties (Q). Q is not used at the CTS2 interface level - all the metadata which CTS2 adds to OWL2 have their own types that specialize OWL annotation proprerties. Hence, at the realisation level, CTS2 metadata can be serialized using Q.

7.1.1. Axioms

The second part of OWL are so-called axioms, which make statements about the basic elements above and thereby give them meaning. By convention those axioms are separated in three distinct parts, called Rbox, Tbox and ABox that axiomatize roles and concepts and assert individuals and relationships between individuals, respectively. We define the set of axioms about concepts and roles as:

$$[\mathsf{axiom}]$$

7.1.1.1. Rbox $\mathcal{R}$

A $\mathcal{SROIQ}$ Rbox $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ consists of a role hierarchy $\mathcal{R}_h$ and a set of role assertions $\mathcal{R}_a$. $\mathcal{R}_h$ is a set of role inclusion axioms of the form $w \sqsubseteq R$ where w denotes a role composition, for details cf. Horrocks et al. (1).

The set of binary roles R predicating over concepts or individuals is declared as:

$$[\mathsf{role}]$$

We define an inverse operation on roles declared as a bijection: every role which has a (unique)inverse is itself the inverse of a single role.

$$\_^- : R \rightarrowtail\!\!\!\!\!\rightarrow R$$

Role axioms which establish subsumption or equivalence relationships between roles have the form:

$$\_ \sqsubseteq \_, \_ \equiv \_ : (\mathsf{role} \times \mathsf{role}) \rightarrow \mathsf{axiom}$$

The RBox also comprises role assertions $\mathcal{R}_a$:

1. R is symmetric

2. R is asymmetric
3. R is transitive
4. R is reflexive
5. R is irreflexive
6. R is functional
7. R, S are disjoint

This role assertions are defined via their sematics in the next section.

Thus the set of all possible RBoxes $\mathcal{R}$ is defined as:

$$\mathcal{R} : \mathbb{P}(\mathbb{F}\,\mathsf{axiom})$$
$$\mathcal{R} = \mathbb{F}(\mathrm{ran}(\_ \sqsubseteq \_) \cup \mathrm{ran}(\_ \equiv \_)) \cup \mathcal{R}_a$$

$S \in \mathcal{R}$ denotes simple roles as defined by Horrocks et al. (1). In the remainder of this specification, regular roles are denoted as R_r and simple roles as R_s, both elements of R.

OWL2 data properties P are roles which relate a value (range) to an individual (domain). Their range may only have the data types defined for this specification as types, e.g. role $P =$ *hasAge* has a range type of $\mathbb{N}$ and is thus allowed.

7.1.1.2. Tbox $\mathcal{T}$

The set of all DL concepts is defined as

$$[\Phi].$$

The sets of atomic and complex concepts and the basic concepts are defined as follows:

$$\Phi_A, \Phi_C : \mathbb{P}\,\Phi$$
$$\bot, \top : \Phi$$

By drawing elements from the set of atomic (primitive) concepts A, the set of complex concepts C is defined using recursive combinations of atomic concepts connected via constructors presented in Table 7.1 and the default concepts $\top$ and $\bot$.

In Z notation, the constructors are defined as follows:

$$
\begin{array}{|l}
\_ \sqcap \_, \_ \sqcup \_ : \Phi \times \Phi \to \Phi \\
\neg : \Phi \to \Phi \\
\forall \_.\_, \exists \_.\_ : (\mathsf{role} \times \Phi \leftrightarrow \mathsf{role} \times \Phi) \to \Phi \\
\leqslant, \geqslant : \mathbb{N} \to (\Phi \leftrightarrow \Phi) \to \Phi \to \Phi \\
\hline
\Phi_C = \mathrm{ran}(\_ \sqcap \_) \cup \mathrm{ran}(\_ \sqcup \_) \cup \mathrm{ran}(\neg) \cup \mathrm{ran}(\forall \_.\_) \cup \mathrm{ran}(\exists \_.\_) \cup \mathrm{ran}(\leqslant) \cup \mathrm{ran}(\geqslant)
\end{array}
$$

In DL, sentences which relate a non-empty set of concept to another non-empty set of concepts via equivalence or subsumption are designated as general inclusion axioms.

Thus sets of concepts can be related to each other using the following relation types to yield axioms:

$$
\begin{array}{|l}
\_ \sqsubseteq \_, \_ \equiv \_ : (\Phi \times \Phi) \to \mathsf{axiom}
\end{array}
$$

The set of all possible Tboxes $\mathcal{T}$ is defined like this:

$$
\begin{array}{|l}
\mathcal{T} : \mathbb{P}_1(\mathbb{F}\,\mathsf{axiom}) \\
\hline
\mathcal{T} = \mathbb{F}(\mathrm{ran}(\_ \sqsubseteq \_) \cup \mathrm{ran}(\_ \equiv \_))
\end{array}
$$

As an example, consider the following Tbox:

$$\mathsf{Male} \sqcap \mathsf{Female} \sqsubseteq \bot \tag{7.1}$$

$$\mathsf{Person} \equiv \mathsf{Male} \sqcup \mathsf{Female} \tag{7.2}$$

$$\mathsf{Sibling} \equiv \mathsf{Person} \sqcap (\exists\, \mathsf{isBrother}.\mathsf{Person} \sqcup \exists\, \mathsf{isSister}.\mathsf{Person}) \tag{7.3}$$

The symbols on the left hand side of the equivalence axioms are name symbols for composite concepts; the right hand side of the axiom is the definition of the composite concept which may consist of composite concepts itself (e.g. Person in the definition of Sibling).

7.1.2. Abox $\mathcal{A}$

In the Abox assertions about individuals instantiating classes and relations between individuals (and datatypes) are made. We define the set of all individuals as

$$[\Upsilon]$$

with the set of named individuals o and the set of anonymous individuals a:

$$
\begin{array}{|l}
a, o : \mathbb{P}\,\Upsilon
\end{array}
$$

A $\mathcal{SROIQ}$ ABox $\mathcal{A}$ is a finite set of individual assertions. The following types of individual assertions are defined in $\mathcal{SROIQ}$:

$$[\mathsf{assertion}]$$

$$\begin{array}{l} \_ \in_c \_ : \Upsilon \times \Phi \pfun \mathsf{assertion} \\ \_ \in_r \_, \_ \notin_r \_ : (\Upsilon \times \Upsilon) \times R \pfun \mathsf{assertion} \\ \_ = \_, \_ \neq \_ : \Upsilon \times \Upsilon \to \mathsf{assertion} \\ \_ \sqcup \_ \ldots \_ \sqcup \_ : \Upsilon \times \cdots \times \Upsilon \to \mathsf{assertion} \end{array}$$

These assertions declare individual of type, individuals as role arguments or as not being role arguments, same individuals, different individuals and individual choice (`oneOf` in OWL).

Thus the set of all possible Aboxes $\mathcal{A}$ is defined as:

$$\begin{array}{l} \mathcal{A} : \mathbb{P}(\mathbb{F}\, \mathsf{assertion}) \\ \hline \mathcal{A} = \mathbb{F}(\mathrm{ran}(\_ \in_c \_) \cup \mathrm{ran}(\_ \in_r \_) \cup \mathrm{ran}(\_ \notin_r \_) \cup \mathrm{ran}(\_ = \_) \\ \qquad \cup \mathrm{ran}(\_ \neq \_) \cup \mathrm{ran}(\_ \sqcup \_ \ldots \_ \sqcup \_)) \end{array}$$

7.2. Semantics

Note: this version of the specification does not use an expression of the OWL semantics in Z notation yet, this will be provided in the next version of the specification. A full reconciliation with the syntax represented in the perivous section is also evnisaged for the next version.

We give precise meaning to the elements of the terminology by interpreting them in terms of first-order logic: atomic concepts correspond to unary predicates, atomic roles to binary predicates, and individuals to constants. As we have learned in the previous section, we have a rich set of constructors at our disposal to complex concepts and roles from simpler ones. Table 7.1 shows the common constructors and the calligraphic letters used to identify them as well with the semantic interpretation of these constructors.

Table 7.1.: Common DL Constructs and Their Semantics

Letters	Syntax	Name	Semantics
		Concepts and Roles	
	$\top$	Top concept	$\top^I = \triangle^I$
	$\bot$	Bottom concept	$\bot^I = \emptyset$
	A	Atomic concept	$A^I \subseteq \triangle^I$
	R	Atomic role	$R^I \subseteq \triangle^I \times \triangle^I$
$\mathcal{ALC}$	$\neg C$	Concept negation	$\triangle^I \setminus C^I$
	$C \sqcap D$	Concept intersection	$C^I \cap D^I$
	$C \sqcup D$	Concept union	$C^I \cup D^I$
	$\exists\, R.C$	Existential quantifier	$\{s \mid \exists\, t \in \triangle^I : \langle s,t\rangle \in R^I \wedge t \in C^I\}$
	$\forall\, R.C$	Universal quantifier	$\{s \mid \forall\, t \in \triangle^I : \langle s,t\rangle \in R^I \rightarrow t \in C^I\}$
$\mathcal{I}$	R^-	Inverse role	$\{\langle t,s\rangle \mid \langle s,t\rangle \in R^I\}$
$\mathcal{O}$	$\{a\}$	Nominals	$\{a^I\}$
$\mathcal{N}$	$\geq n\, R$	Unqualified number	$\{s \mid \sharp\{t \mid \langle s,t\rangle \in R^I\} \geq n\}$
	$\leq n\, R$	restrictions	$\{s \mid \sharp\{t \mid \langle s,t\rangle \in R^I\} \leq n\}$
$\mathcal{Q}$	$\geq n\, R.C$	Qualified number	$\{s \mid \sharp\{t \mid \langle s,t\rangle \in R^I \wedge t \in C^I\} \geq n\}$
	$\leq n\, R.C$	restrictions	$\{s \mid \sharp\{t \mid \langle s,t\rangle \in R^I \wedge t \in C^I\} \leq n\}$
		TBox Axioms	
	$C \sqsubseteq D$	Concept inclusion	$C^I \subseteq D^I$
$\mathcal{H}$	$R \sqsubseteq S$	Role inclusion	$R^I \subseteq S^I$
$\mathcal{S}$	$\mathsf{Trans}(R)$	Transitivity	$\forall\, s,t,u \in \triangle^I : \langle s,t\rangle \in R^I \wedge \langle t,u\rangle \in R^I \rightarrow \langle s,u\rangle \in R^I$
$\mathcal{R}$	$R \circ S \sqsubseteq T$	Role composition	$\forall\, s,t,u \in \triangle^I : \langle s,t\rangle \in R^I \wedge \langle t,u\rangle \in S^I \rightarrow \langle s,u\rangle \in T^I$
		ABox Axioms	
	$C(a)$	Concept assertion	$a^I \in C^I$
	$R(a,b)$	Positive role assertion	$\langle a^I, b^I\rangle \in R^I$
	$\neg R(a,b)$	Negative role assertion	$\langle a^I, b^I\rangle \notin R^I$
	$a \approx b$	Equality assertion	$a^I = b^I$
	$a \not\approx b$	Inequality assertion	$a^I \neq b^I$

A DL knowledge base $\mathcal{K}$ is given semantics by interpreting it in a first-order structure—usually called an *interpretation*—where concepts correspond to unary predicates and roles correspond to binary predicates. More precisely, an interpretation I consists of a domain set $\triangle^I$ and an interpretation function $\cdot^I$ that interprets (*i*) each individual a as some element $a^I \in \triangle^I$, and (*ii*) concepts and roles as shown in Table 7.1. An interpretation I is a model of $\mathcal{K}$ if it satisfies all TBox and ABox axioms as shown in the bottom part of the table. Furthermore, $\mathcal{K}$ *entails* an axiom α, written $\mathcal{K} \models \alpha$, if α is true in all models of $\mathcal{K}$. The basic reasoning problem for $\mathcal{K}$ is checking its *satisfiability*—that is, checking whether $\mathcal{K}$ admits a model. Many DL-based applications also use the *entailment checking* problem, which is the problem of checking whether an axiom is entailed by a DL knowledge base. If the axiom is of

the form $C \sqsubseteq D$ with C and D concepts, the problem is called *subsumption checking*, and if the axiom is an assertion, the problem is called *assertion checking*.

7.2.1. Datatypes

The central notion in the datatype system of OWL, which is also leveraged for CTS2, is the *datatype map*. Datatype maps can additionally support *facets*—expressions that can be applied to a datatype to restrict its interpretation.

Definition 7.2.1 *A* datatype map *is a 4-tuple* $\mathcal{D} = (N_D, N_C, N_F, \cdot^{\mathcal{D}})$*, where*

- N_D *is a set of* datatypes*,*
- N_C *is a function assigning a set of* constants $N_C(d)$ *to each* $d \in N_D$*,*
- N_F *is a function assigning a set of* facets $N_F(d)$ *to each* $d \in N_D$*,*
- $\cdot^{\mathcal{D}}$ *is a function assigning a* datatype interpretation $d^{\mathcal{D}}$ *to each datatype* $d \in N_D$*, a* facet interpretation $f^{\mathcal{D}} \subseteq d^{\mathcal{D}}$ *to each facet* $f \in N_F(d)$*, and a* data value $v^{\mathcal{D}} \in d^{\mathcal{D}}$ *to each constant* $v \in N_C(d)$*.*

By a slight abuse of notation, let $N_C = \bigcup_{d \in N_D} N_C(d)$*; the intended usage of* N_C *should be clear from the context.*

A facet expression *for a datatype* $d \in N_D$ *is a formula* φ *built using propositional connectives over the elements from* $N_F(d) \cup \{\top_d, \bot_d\}$*. The function* $\cdot^{\mathcal{D}}$ *is extended to facet expressions for* d *by setting, for* $f_{(i)} \in N_F(d)$*,* $\top_d^{\mathcal{D}} = d^{\mathcal{D}}$*,* $\bot_d^{\mathcal{D}} = \emptyset$*,* $(\neg f)^{\mathcal{D}} = d^{\mathcal{D}} \setminus f^{\mathcal{D}}$*,* $(f_1 \wedge f_2)^{\mathcal{D}} = f_1^{\mathcal{D}} \cap f_2^{\mathcal{D}}$*, and* $(f_1 \vee f_2)^{\mathcal{D}} = f_1^{\mathcal{D}} \cup f_2^{\mathcal{D}}$*.*

For CTS2 we assume that datatypes in N_D are pairwise disjoint—that is, that $d_1, d_2 \in N_D$ and $d_1 \neq d_2$ imply $d_1^{\mathcal{D}} \cap d_2^{\mathcal{D}} = \emptyset$. This leads to no loss of generality, simplifies our reasoning algorithm, and allows for a modular treatment of different datatypes. Our datatypes are, in this respect, comparable to datatype groups (5).

For example, $\mathcal{D}$ might be a datatype map with $N_D = \{str, real\}$, where $str^{\mathcal{D}}$ and $real^{\mathcal{D}}$ are the set of all strings and real numbers, respectively. The sets $N_C(str)$ and $N_C(real)$ would then contain all string constants and all decimal representations of real numbers. Finally, the set $N_F(real)$ might contain the facet *int*, interpreted as the set of all integers, and facets of the form $<_w$, $>_w$, $\leq_w$, and $\geq_w$ for each decimal number w. Thus, the facet expression $int \wedge >_{12} \wedge <_{15}$ would represent the integers 13 and 14.

Based on the definition of *data ranges*—expressions over the predicates in $\mathcal{D}$ — we can integrate data ranges into CTS2/OWL concepts and axioms.

Table 7.2.: Model-Theoretic Semantics of $\mathcal{DL}$+$\mathcal{D}$

Semantics of Data Ranges	
$(\top_{\mathcal{D}})^{\mathcal{D}} = \triangle^{\mathcal{D}}$	$(d[\varphi])^{\mathcal{D}} = \varphi^{\mathcal{D}}$
$(\{v_1, \ldots, v_n\})^{\mathcal{D}} = \{v_1^{\mathcal{D}}, \ldots, v_n^{\mathcal{D}}\}$	$\overline{dr}^{\,\mathcal{D}} = \triangle^{\mathcal{D}} \setminus dr^{\mathcal{D}}$
Semantics of Concepts	**Semantics of Axioms**
$(\forall\, U.dr)^{\mathcal{I}} = \{x \mid \forall\, y : \langle x, y\rangle \in U^{\mathcal{I}} \rightarrow y \in dr^{\mathcal{D}}\}$	$\mathsf{Dis}(U_1, U_2) \Rightarrow U_1^{\mathcal{I}} \cap U_2^{\mathcal{I}} = \emptyset$
$(\exists\, U.dr)^{\mathcal{I}} = \{x \mid \exists\, y : \langle x, y\rangle \in U^{\mathcal{I}} \wedge y \in dr^{\mathcal{D}}\}$	$U_1 \sqsubseteq U_2 \Rightarrow U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
$(\leq n\, U.dr)^{\mathcal{I}} = \{x \mid \sharp\{y \mid \langle x, y\rangle \in U^{\mathcal{I}} \wedge y \in dr^{\mathcal{D}}\} \leq n\}$	$U(a, v) \Rightarrow \langle a^I, v^{\mathcal{D}}\rangle \in U^{\mathcal{I}}$
$(\geq n\, U.dr)^{\mathcal{I}} = \{x \mid \sharp\{y \mid \langle x, y\rangle \in U^{\mathcal{I}} \wedge y \in dr^{\mathcal{D}}\} \geq n\}$	
Note: $\sharp N$ is the number of elements in a set N.	

Definition 7.2.2 *Let $\mathcal{D} = (N_D, N_C, N_F, \cdot^{\mathcal{D}})$ be a datatype map. The set of* data ranges *for $\mathcal{D}$ is the smallest set that contains $\top_{\mathcal{D}}$, d, $d[\varphi]$, $\{v_1, \ldots v_n\}$, $\overline{dr}$, for $d \in N_D$, φ a facet expression for d, $v_i \in N_C$, and dr a data range.*

Let $\mathcal{DL}$ be a description logic, defined over a set of individuals N_I, and let N_{DP} be a set of data properties *disjoint from each of the sets of symbols used in $\mathcal{DL}$. The logic $\mathcal{DL}$+$\mathcal{D}$, obtained by extending $\mathcal{DL}$ with $\mathcal{D}$, is defined as follows. The set of concepts of $\mathcal{DL}$+$\mathcal{D}$ extends the set of concepts of $\mathcal{DL}$ with* datatype concepts *of the form $\exists\, U.dr$, $\forall\, U.dr$, $\geq n\, U.dr$, and $\leq n\, U.dr$, for $U \in N_{DP}$, n a nonnegative integer, and dr a data range for $\mathcal{D}$. The set of axioms of $\mathcal{DL}$+$\mathcal{D}$ extends the set of axioms of $\mathcal{DL}$ with* data property disjointness axioms $\mathsf{Dis}(U_1, U_2)$, data property inclusion axioms $U_1 \sqsubseteq U_2$, *and* data property assertions $U(a, v)$, *for $U_{(i)} \in N_{DP}$, $a \in N_i$, and $v \in N_C$.*

An interpretation *for $\mathcal{DL}$+$\mathcal{D}$ is a triple $\mathcal{I} = (\triangle^{\mathcal{I}}, \triangle^{\mathcal{D}}, \cdot^{\mathcal{I}})$, where $\triangle^{\mathcal{I}}$ and $\triangle^{\mathcal{D}}$ are nonempty disjoint sets such that $d^{\mathcal{D}} \subseteq \triangle^{\mathcal{D}}$ for each $d \in N_D$, and $\cdot^{\mathcal{I}}$ is a function assigning to each concept C, property R, and individual a of $\mathcal{DL}$, and to each data property $U \in N_{DP}$, interpretations $C^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$, $R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$, $a^{\mathcal{I}} \in \triangle^{\mathcal{I}}$, and $U^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{D}}$ respectively. The functions $\cdot^{\mathcal{D}}$ and $\cdot^{\mathcal{I}}$ are extended to data ranges and datatype concepts as shown in Table 7.2. For $\mathcal{DL}$+$\mathcal{D}$ knowledge bases $\mathcal{K}$ and $\mathcal{K}'$, an interpretation $\mathcal{I}$ is a* $\mathcal{D}$-model *of $\mathcal{K}$, written $I \models_{\mathcal{D}} \mathcal{K}$, if all axioms of $\mathcal{DL}$ are satisfied in $\mathcal{I}$ as specified by $\mathcal{DL}$, and the additional axioms are satisfied in $\mathcal{I}$ and $\mathcal{D}$ as specified in Table 7.2; furthermore, $\mathcal{K}$* $\mathcal{D}$-entails *$\mathcal{K}'$, written $\mathcal{K} \models_{\mathcal{D}} \mathcal{K}'$, if $I \models_{\mathcal{D}} \mathcal{K}'$ whenever $I \models_D \mathcal{K}$.*

7.3. CTS2 logical models and type schemata

For the purpose of CTS2 each element of the logical model is defined by a type specified in Z. Thereby we achieve a conceptual correspondence to UML logical models while obtaining a more precise defintion of each model element. Each terminology element can be annotated with metadata. We therefore start with introducing the types required to represent

this metadata structure. We then define the types required to create the elements of CTS2 terminologies and value sets.

7.3.1. Basic types

7.3.1.1. Cardinality schemata

We define the following generic type cardinality schemata to create Z notation equivalents for UML class attribute cardinalities:

$$[T]$$

$$T[0\,..\,1], T[0\,..\,*], T[1\,..\,*] : \mathbb{P}(\mathbb{P}\ T)$$

$$T[0\,..\,1] = \{s : \mathbb{P}\ T \mid \#s \leq 1\}$$
$$T[0\,..\,*] = \mathbb{P}\ T$$
$$T[1\,..\,*] = \mathbb{P}_1\ T$$

7.3.1.2. Types for basic ontology elements and data type schemata

Each basic $\mathcal{SROIQ}$/OWL2 type is axiomatically defined as follows:

$$A, D, R, S, P, o, u, C, Q : IRI$$
$$a : \texttt{emptyNode}$$

To achieve a syntactic representation in OWL each instantation of those basic types must have a resource identification schmema IRI following IETF RFC 3987 (like URI but using unicode (ISO 10646) instead of ASCII):

IRI

$$scheme, i\text{-}hier\text{-}part : CHAR$$
$$iquery, ifragment : CHAR[0\,..\,1]$$

To represent datatypes, the following basic Z types are used:

$$\mathbb{N}, \mathbb{N}_1, \mathbb{Z}$$

with $\mathbb{N}$ as the set of natural numbers, $\mathbb{N}_1 = \{\mathbb{N} \setminus 0\}$ and $\mathbb{Z}$ as the integers. They correspond to OWL datatypes.

These and some other basic datatypes are defined as OWL2 data types V for the purpose

of CTS2:

$$CHAR == xsd.string$$
$$\mathbb{R} == owl.real$$
$$BOOLEAN == xsd.boolean$$
$$Binary == xsd.hexBinary$$

The built-in types $\mathbb{N}, \mathbb{N}_1,$ and $\mathbb{Z}$ correspond to the XML schema types `nonNegativeInteger, positiveInteger` and `integer`, respectively.

7.3.2. CTS2 metadata and characteristic schemata

The following schema defines the metadata common to the schemata valid for CTS2. The attributes note, changeNote, definition, editorialNote, example, historyNote and scopeNote are compliant with SKOS documentation properties. The attributes contributor, creator, format, language, publisher, rights, source, subject, title, type are compliant with the Dublin Core Metadata.

7.3.2.1. Metadata

Metadata

$state : CHAR[0 \,..\, 1]$
$timeStamp : \mathbb{N}[0 \,..\, 1]$
$effectiveDate : \mathbb{N}[0 \,..\, 1]$
$releaseDate : \mathbb{N}[0 \,..\, 1]$
$stateDate : \mathbb{N}[0 \,..\, 1]$
$description, administrativeInfo, provenanceDetails, note,$
$changeNote, definition, editorialNote, example, historyNote,$
$scopeNote, contributor, creator, format, language, publisher,$
$rights, oid, source, subject, title, type : CHAR[0 \,..\, 1]$

Ontologies, ValueSets and Mappings are versioned using versioned identifiers which must be unique.

VersionedIdentifier

$id : IRI$
$versionId : IRI[0 \,..\, 1]$

7.3.2.2. Characteristic Type and Characteristic

A characteristic type is a kind of characteristic pertaining to an ontology which defines allowed characteristics of the entities which constitute the ontology. An example might be "confidence level" indicating the level of trust which the authors have in the maturity of an ontology or its elements. Such characteristics can apply to all entities of the ontology and *can be assigned a value at the ontology entity level* via the characteristic class, in the above example this could be level types like "low, medium and high".
The characteristic type explicitly defines the allowable characteristics for the ontology as a whole and is independent of individual changes or versions. Each ontology may have zero to many characteristic types. Each ontology allows to define a set of characteristic types for the entities it contains and to use the characteristic class to define the actual values of the characteristics of concepts, individuals and roles.
This class is an annotation (metadata) type and not used for DL reasoning.

CharacteristicType

$id : IRI$
$name : CHAR$
$commonMetadata : Metadata$

The characteristics class explicitly defines the values of supported characteristics for any given ontology entity. This class is an annotation (metadata) type and not used for DL reasoning.

Characteristic

$id : IRI$
$value : CHAR$
$language : CHAR[0 \,..\, 1]$
$state : CHAR[0 \,..\, 1]$
$stateDate : \mathbb{N}[0 \,..\, 1]$
$characteristicType : CharacteristicType$

7.3.2.3. Designation

A designation is a symbol used to designate a concept, an individual or a role. The preferdnessLevel is compliant with skos:preLabel, skos:altLabel and skos:hiddenLabel.

$Preferredness ::= isPreferred \mid isAlternative \mid isHidden$

Designation

$id : IRI$

$name : CHAR$

$relatedConcept : IRI[0 \,..\, 1]$

$relatedRole : IRI[0 \,..\, 1]$

$relatedIndividual : IRI[0 \,..\, 1]$

$commonMetadata : Metadata.\{state, administrativeInfo, effectiveDate, releaseDate, stateDate\}$

$rendering : Binary[0 \,..\, 1]$

$preferrednessLevel : Preferredness$

$preferredUsageType : CHAR[0 \,..\, 1]$

$language : CHAR[0 \,..\, 1]$

$format : CHAR[0 \,..\, 1]$

$designationType : CHAR[0 \,..\, 1]$

$\#(relatedConcept \cup relatedRole \cup relatedIndividual) = 1$

7.3.3. CTS2 core schemata

This subsection introduces the core schemata of CTS2 and defines their relationships to each-other. Figure 1 shows a conceptual, merely informative UML class diagram illustrating the relationships[2] between the schemata. The schema attributes are defined in the Z schemata following the figure.

[2]These are modeled as referenced types, Zermelo-Fraenkel relationships or functions in the Z representation of the schemata

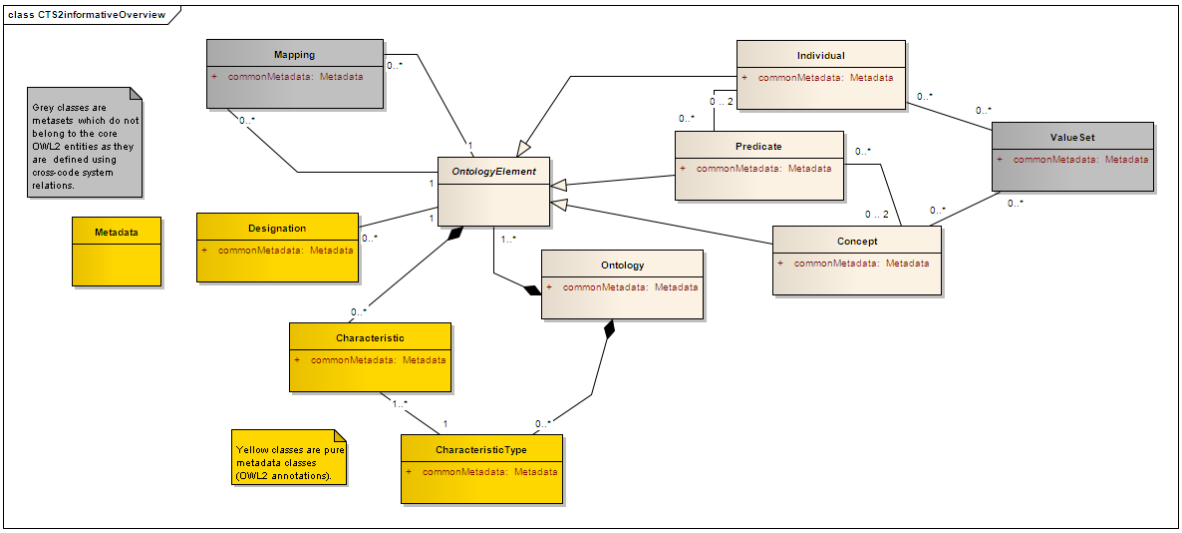

Figure 7.1.: Illustrative UML class diagram showing the relationships between the CTS2 schemata. Grey classes are meta sets which go beyond OWL2 as they are created using 1 or more ontologies. Yellow classes are pure metadata classes (OWL annotations). Syntax and semantics not fully correct (illustrative).

Every schema is related to exactly one element $Metadata_i \in Metadata$.

7.3.3.1. Ontology versioning

To manage the publication of ontology revisions, ontologies need to be versioned. Versioning is important to maintain usability of data persisted against older versions (data accessibility), to allow consistent inference, synchronisation (of locally changed ontologies with new versions of the source ontolgy) and data translation and to maintain interopearbility between software units using the ontologies. The scientific literature on ontology versioning (6, 7) concludes that ontologies are optimally versioned at the ontology level, i.e. for the purpose of ontology publication all elements of an ontology should be co-versioned with the ontology and that versioning for publication should not occur at the ontology element level.
However, users of ontologies need to be able to identify the changes which have occured upon version upgrade in order to achieve the goals summarised above and ontology authors need to be able to create a summary of these changes. One practical example for this is the SNOMED-CT RF2 specification of differential encoding of updates. Klein et al. (6) provides a framework to create such change logs[3]. To enable the generation of such change logs, time stamps at the level of the elements of an ontology are required and provided by this specification.

[3]This is not pursued in this specification any further as this is an authoring and deployment issue not in scope of this specification

937 This specification defines ontology-related sets (value sets and mappings) which are defined
938 as functions applied to or relations between ontology elements. As such, they are not part of
939 ontologies and require their own versioning which is described below.

940 7.3.3.2. Ontology

7.3.3.2.1. Ontology types

$$\mathcal{O}\,Type ::= DL - ontology \mid terminology \mid classification \mid thesaurus \mid taxonomy \mid subjectHeadingSystem \mid flatConceptList$$

941 **7.3.3.2.2. Ontology** The schema for an Ontology $\mathcal{O}$ (a code system, classification, termi-
942 nology or ontology) is defined as the union of set of roles (Rbox), general inclusion axioms of
943 the form $C \subseteq D$ and equivalence axioms of the form $C \equiv D$ (Tbox) and a set of individual
944 assertions (ABox) and a set of metadata (annotations).

$\mathcal{O}$

$\mathcal{T} : \mathbb{P}_1(\mathbb{F}\,\mathsf{axiom})$
$\mathcal{A} : \mathbb{P}(\mathbb{F}\,\mathsf{assertion})$
$\mathcal{R} : \mathcal{R}_h \cup \mathcal{R}_a$
$id : IRI$
$characteristicType : \mathbb{P}\ CharacteristicType$
$commonMetadata : Metadata \setminus language$
$fullName : CHAR$
$shortName : CHAR[0\,..\,1]$
$versionId : IRI$
$previousVersionId : IRI[0\,..\,1]$
$releaseFormats : \mathbb{P}\ CHAR$
$supportedLanguages : \mathbb{P}\ CHAR$
$releaseLocation : IRI[0\,..\,1]$
$localName : CHAR[0\,..\,1]$
$type : \mathcal{O}\,Type$

$versionId \neq previousVersionId$

945 7.3.3.3. Concept and Individual

946 Concept and individual are identified using IRIs in OWL.

The externalCode and externalId attributes of the schema are used to capture the identifiers created by external code system editors if they do not use IRIs as primary identifiers. In CTS2, they are represented as designations (each identifier in a computer system is a designation, but the main identifier has a special role and is not represented using the Designation schema). It is assumed that upon import into an CTS2 instance, for each concept an IRI is created as identifier.

Concept and individuals always have the version of the ontology they belong to (no own version identifier). However, to support ontology version management, they may have time-stamps which allow to computationally determine whether they were created and/or changed upon an ontology revision (which is the case when the timeStamp is newer than the timestamp of the previous version of the ontology).

The schema for concept comprises atomic concepts A, the $\top$ and the $\bot$ concept as well as complex concepts C. A concept always belongs to an ontology $\mathcal{O}_i$ (each concept belongs to an ontology and each ontology has at least one concept as a member), and may belong to several versions of this ontology. Each concept may be related to one or more designations. The preferred designation of the concept is used to derive the name attribute.

Concept

$\mathcal{O}membershipId : \mathcal{O}.id$
$\mathcal{O}membershipVersionId : \mathbb{P}_1\, \mathcal{O}.versionId$
$C, A, \top, \bot : IRI$
$prefName : Designation$
$conceptDefinition : \mathbb{P}\, \Phi_C$
$externalCode : Designation[0\,..\,1]$
$externalId : Designation[0\,..\,1]$
$timeStamp : \mathbb{N}[0\,..\,1]$
$commonMetadata : Metadata.\{state, description, definition, note, source, type\}$
$characteristic : \mathbb{P}\, Characteristic$

$\#(A \cup \top \cup \bot \cup C) = 1$
$prefName.preferrednessLevel = isPreferred$

An individual always belongs to an ontology $\mathcal{O}_i$ (each individual belongs to one ontology but an ontology must not comprise an individual). One individual may be related to 0 to many designations. An individual types 0 to many concepts. Zero to many roles and datatype properties may predicate over the individual.

Individual

$\mathcal{O}membershipId : \mathcal{O}.id$
$\mathcal{O}membershipVersionId : \mathbb{P}_1\, \mathcal{O}.versionId$
$types : \mathbb{P}\, Concept.\{\{C, A\}, prefName.name\}$
$predicatingRoles : \mathbb{P}\, Role.\{R, R_t, name.name\}$
$predicatingDataypeProperties : \mathbb{P}\, DataypeProperty.\{P, name.name\}$
$o : IRI$
a : `rdf.emptyNode`
$externalCode : Designation[0\,..\,1]$
$externalId : Designation[0\,..\,1]$
$prefName : Designation[0\,..\,1]$
$commonMetadata : Metadata.\{state, description, definition, note, source, type\}$
$timeStamp : \mathbb{N}[0\,..\,1]$
$characteristic : \mathbb{P}\, Characteristic$

$\#(o \cup a) = 1$
$prefName.preferrednessLevel = isPreferred$

7.3.3.4. Role

The role schema adds metadata to $\mathcal{SROIQ}$ roles (synonyms: association, relationship, property). The $\mathcal{SROIQ}$-allowed role characteristics (denoted as role assertions) are defined in $\mathcal{R}_a$.

A role always belongs to one ontology $\mathcal{O}_i$ as indicated by the ontologyMembership attribute (each role belongs to an ontology but an ontology does not necessarily contain roles). A role may have 0 to many argument pairs, either concepts or individuals. Each role may be related to one or more designations as indicated by the designationRelation. The preferred designation of the role is used to derive the name attribute.

Like concepts, roles do not have versions, only timestamps. The attribute R_t is the role type (regular or simple role).

Role

$\mathcal{O}membershipId : \mathcal{O}.id$
$\mathcal{O}membershipVersionId : \mathbb{P}_1\, \mathcal{O}.versionId$
$R : IRI$
$roleArgument_1 : \Phi, \Upsilon$
$roleArgument_2 : \Phi, \Upsilon$
$R_t : \{R_r, R_s\}$
$commonMetadata : Metadata.\{state, description, definition, note, source, type\}$
$name : Designation$
$timeStamp : \mathbb{N}[0\,..\,1]$
$externalCode : Designation[0\,..\,1]$
$externalId : Designation[0\,..\,1]$
$roleAssertion : \mathcal{R}_a$
$characteristic : \mathbb{P}\, Characteristic$

$name.preferrednessLevel = isPreferred$
$roleArgument_1, roleArgument_2 \in \Upsilon \vee roleArgument_1, roleArgument_2 \in \Phi$

The HL7 SFM has the notion of association types which describe "allowable associations". In this specification, this is replaced by the $\mathcal{SROIQ}$ notion of role assertions $\mathcal{R}_a$ in compliance with OWL2. OWL role assertions are not identical to HL7 association types, they rather characterise roles.

7.3.3.5. Datatype and datatype property

The basic types are used by the Datatype schema:

Datatype

$id : IRI$
$type : V$
$description : CHAR[0\,..\,1]$
$state : CHAR[0\,..\,1]$
$source : CHAR[0\,..\,1]$

$\#type = 1$

Predicates which relate individuals to data types (data type properties P) are defined in the following manner:

DatatypeProperty

$\mathcal{O}membershipId : \mathcal{O}.id$
$\mathcal{O}membershipVersionId : \mathbb{P}_1\, \mathcal{O}.versionId$
$P : IRI$
$argument_1 : \Upsilon$
$argument_2 : u$
$commonMetadata : Metadata.\{state, description, definition, note, source, type\}$
$name : Designation$
$timeStamp : \mathbb{N}[0\,..\,1]$
$externalCode : Designation[0\,..\,1]$
$externalId : Designation[0\,..\,1]$
$characteristic : \mathbb{P}\, Characteristic$

7.3.4. Ontology subsets and mappings

Ontology subsets can be derived from one ontology or from several ontologies. If a subset is created by using elements of more than one ontology, it breaks the syntax and semantics of OWL2. Mappings which are relations between ontologies (thus sets of cross-ontology maplets) also break OWL2. Such sets are not meant to be used for reasoning, but to organise knowledge in a cross-ontology fashion or to point out relations between ontologies. Both set types are described in this separate subsection.

Ontology applications often require different subgraphs of an $\mathcal{O}_i \in \mathcal{O}$.

7.3.4.1. Value Sets

A value set is the union of a sets of concepts or individuals with a set of metadata describing the set of concepts.

In HL7, value sets may contain concepts from more than one ontology and are therefore non-OWL2 entities, meta sets which can contain elements from more than one ontology. Value sets are often used to constrain the vocabulary which can be used to instantiate vocabulary-coded information model attributes, e.g. the attribute symptom of a class may be restricted to vocabulary from the SNOMED-CT neurological symptoms of the CNS. Given this role, the necessity for such value sets seems to be a model design issue (well designed attributes will not need several ontologies per value set as constraining mechanisms). To reduce design and implementation complexity, this implementation feature is not supported by this specification.

Note that because CTS2 is based on OWL2, this specification supports value sets of individ-

uals in addition to the HL7 value sets of concepts.

The CTS2 ValueSet corresponds to a skos:Collection or a skos:orderedCollection if its isOrdered attribute is $\top$.

The elements of a value set are defined by the value set definition:

ValueSetDefinition

$subsettingStatment : w3cSPARQL$

$sourceOntology : \mathcal{O}\, VersionedIdentifier$

Subsetting statements can be of any form that can be expressed as a SPARQL query[4]. They can define sets of concepts or individuals. It is assumed that like in the CTS2 prototype, value sets are created from informal (natural language) or semi-formal (using set-theory notation) specifications as persisted SPARQL queries (with metadata) which can retrieved and executed when the listValueSetElements operation is called.

ValueSet

$definition : ValueSetDefinition$

$id : IRI$

$versionId : IRI$

$jurisdictionalDomain : CHAR[0 \,..\, 1]$

$name : Designation[0 \,..\, 1]$

$commonMetadata : Metadata$

$previousVersionId : IRI[0 \,..\, 1]$

$supportedLanguages : \mathbb{P}\, CHAR$

$preferredLanguage : CHAR$

$isOrdered : BOOLEAN$

$isImmutable : BOOLEAN$

$\#(subsettingStatement) = 1$

7.3.4.2. Ontology mappings

Ontology mappings are binary relations between ontologies. The mapping schema represents a set of maplets $x \mapsto y$ which establish semantic relatedness between concepts, individuals or roles belonging to two different ontologies. As such, they are not OWL2 entities, but meta

[4] http://www.w3.org/TR/rdf-sparql-query/

sets relating elements of different ontologies to each-other.
When for given concepts $A \in \mathcal{O}_i$ and $\{B, C\} \in \mathcal{O}_j$ there are two maplets $A \mapsto B$ and $A \mapsto C$, in some cases a selection rule which selects one of those mappings as appropriate for a certain context may be provided. We accommodate for this requirement with a selectionRule function but do not specify the details of the function's behaviour at the PIM level.
CTS2 mappings are conformant with skos:mappingRelation. As such, they encompass an attribute expressing the strength of the relationship between the mapped concepts, which comply with SKOS: skos:closematch, skos:exactmatch, skos:broadmatch, skos:narrowmatch, skos:relatedmatch.

$$MappingStrength ::= closematch \mid exactmatch \mid broadmatch \mid narrowmatch \mid relatedmatch$$

Mapping

$id : IRI$
$versionId : IRI$
$name : Designation[0 \,..\, 1]$
$mappingRelation : \mathbb{P}_1\, \mathcal{O}_i \leftrightarrow \mathcal{O}_j$
$selectionRule : X \rightarrow Y$
$mappingStrength : MappingStrength[0 \,..\, 1]$
$commonMetadata : Metadata$
$characteristic : Characteristic[0 \,..\, 1]$

7.3.4.3. Schemata which represent system components

The operation descriptions given in the interface definition section of this specification require schemata which describe the components holding ontologies, value sets and mappings. These are not defined in detail in this version of the specification, but merely mentioned here:

$\mathcal{O}$repository

ValueSetRepository

MappingRepository

A definition will follow in the next version of this specification.

7.3.4.4. Post-coordination support

Pre-coordination and post-coordination are terms from the medical terminology domain. They describe what are called concept descriptions in description logic, i.e. concepts formed using atomic concepts, roles and DL-syntax elements. When a concept description D is related to a concept C via equivalence of subsumption,

$$C \sqsubseteq D, C \equiv D,$$

we call this an axiom in DL. From a DL perspective, pre-coordinated concepts are concept descriptions/axioms authored at design time, i.e. composed as an integral part of the ontology before publication. Post-coordinated concepts are DL concept descriptions/axioms authored after publication time, i.e. composed by applying DL syntax to concepts of a published ontology. This specification supports post-coordination via the reasoning interface in the following manner: a CTS2 client parsing a post-coordinated expressionn (e.g. created by an application or received via a gateway) can validate the satisfiability of post-coordinated concept by using the reasoning interface operation *determineConceptSatisfiability* or check whether the concept is entailed by the ontolgy using the operation *determineConceptEntailment*. This allows to validate a post-coordinated concept against a published ontology. The operation *returnFullAxiomExpansion* provides the full expansion of a concept description (the right side of an axiom), i.e. an expression using only atomic concepts. This is semantically equivalent to the IHTSDO SNOMED-CT *normal form*.

7.4. Interfaces

Note: in this version of the specification, some operations are not fully defined - for these only the I/O parameters are defined and the high level behaviour description is provided.
For each operation which corresponds to an operation defined in the HL7 SFM, a mapping is provided in the operation definition.

7.4.1. SOA interface specification conventions in Z notation used in this specification

As the Z literature (3, 4) does not explicitly describe the usage of Z for the specification of SOA service operations, a short account is give here. We are proposing a slightly extended version of the Z notation.
PIM level SOA service interface operations descriptions comprise input and output parameters, data types for these parameters, operation behaviour, pre-conditions, post-conditions, invariants, status (error) conditions as well as notes for service realisation. In Z notation, these are provided as Z schemata, which comprise a declaration part and a predicate part separated

by a horizontal line. The declaration contains the operation *invariant* denoted as 'Ξ', which describes the objects which are not altered by the operation, but only read-accessed. If the behaviour of an operation alters stored objects (create, update, delete), the declaration part of the schema contains a 'Δ' line listing these objects. The declaration describes *input parameters* and *outputs* denoted using the decorations '?' and '!', respectively. The *parameters are typed* using the Z declaration style set element notation, with a colon (and not an '$\in$') like in $x : A$ where x is the parameter (a set element) and A is the type (a set). The complex types are described in separate schemata.

The operation behaviour is described using the predicate part. The style of the behaviour specification is purely declarative in accordance with the nature of a PIM.

A Z behaviour description in this specification consists of a initialisation, an input checking and one or more output computation sections, which are delimited using uninterpreted comment lines (denoted by $\ddagger \ldots \ddagger$ - the sign $\ddagger$ is defined as uninterpreted comment in the Z variant used in this specification).

The initialisation part specifies the initialisation of status and status condition variables. The input checking part specifies the syntactic and semantic validation of all input parameters and specifies *status conditions* related to input errors. The output computation section defines the output and implicitly provides the *post-conditions*. Intermediary computational steps altering objects are specified using the decoration notation '$'$'. If an output set $out' = \varnothing$ after a valid computation, the postcondition is still fulfilled (*successful empty return*); this is not explicitly stated in the operations, but would have the form:

$$\textbf{if } out' = \varnothing \textbf{ then } out! = out'$$

Z does not have a special symbol for assignments, both assignments and evaluations are denoted with '='. However, the syntactic context can be used to disambiguates the meaning: a '=' sign means evaluation if an input parameter ('?') is on its left. It designates assignment if on its left side there is set element designated by an index i as in $\mathcal{S}_i(\in \mathcal{S})$ or an output decoration '!' or a '$'$' decoration which indicates that a previously introduced object is altered. In the rare cases where the context is not sufficient to disambiguate the meaning of '=', Z allows the usage of **if** ... **then** ... **else** constructs to specify evaluation as in the example given above.

The *pre-conditions* of a schema are rendered in the predicative part of the schema that deals with the input validation. The indentation used in the behaviour descriptions delimits the logical computation steps, i.e. a line without an indentation indicates a new step. The $\top$ and $\bot$ symbols are overloaded in this specification: in the context of concept types, they designate the top and bottom concepts of an ontology, in the context of behaviour, they designate the elements of the type boolean. The statement $status = \bot$ stops the execution of an operations with an exception.

Each Z operation is preceded by schemata specifing operation specific types and an informal behaviour description; after each schema, notes for service implementation can be found.

7.4.2. Common interfaces types and adjuvant functions

7.4.2.1. Interfaces types

Complex type schemata required for more than one operation as well as common adjuvant functions are given in the following subsections. Operation specific type schemata are given as a subsection of the resp. operation definitions.

The following status codes for status conditions are defined:

StatusCode

$x : CHAR$

$c : \langle x_1 \ldots x_n \rangle$

$c(1)$ = (Info) Operation success
$c(2)$ = (Error) $\mathcal{O}$Identifier not found
$c(3)$ = (Error) *filter?* syntactically incorrect
$c(4)$ = (Error) *queryControl?* syntactically incorrect
$c(5)$ = (Error) Maximum result set size exceeded
$c(7)$ = (Error) ConceptIdentifier not found
$c(13)$ = (Error) ValueSet name not typed correctly
$c(14)$ = (Error) ValueSet name already exists
$c(15)$ = (Error) ValueSet metadata input syntactically incorrect
$c(17)$ = (Error) Entity (concept, role or individual) identifier not found.
$c(20)$ = (Error) ValueSet definition syntactically incorrect.

DateRange

$initDate : \mathbb{N}$

$endDate : \mathbb{N}$

$dateType : \{effectiveDate, stateDate\}$

$initDate \leq endDate$

QueryControl

$maximumResultSetSize : o\mathbb{N}$

7.4.2.2. Adjuvant functions and operators

7.4.2.2.1. String matching operator $\sim_{type,r}$ The $\sim_{type,r}$ function is used to assign a matching strength value to the matching of two strings. The function is parametrised by *type*.

Supported *type*s of string matching algorithms are:

$$MatchingTypes ::== Karp - Rabin \mid Knuth - Morris - Pratt \mid Boyer - Moore \mid Shift - Or$$

Note: examples, more can be added at the PSM/realisation level

$$[X, Y, V, W]$$

$$\sim_{type}: X \times Y, V \rightarrow Z$$

$$\forall\, x : CHAR, y : CHAR, Z : \mathbb{R} \bullet 0 \leq r \leq 1, \forall\, type : V = MatchingTypes \bullet$$
$$\sim_{type} (x, y) = type\text{-match } (x, y) \in Z$$

7.4.2.2.2. Id creation function This function, total bijection, creates a new identifier for a given type. X is the set of identifiers of the type, e.g. if type is $\mathcal{O}$, then X is the set of all $\mathcal{O}$ identifiers.

$$[X]$$

$$createId : X \rightarrowtail\!\!\!\!\!\rightarrow X$$

$$\forall\, x : X \bullet createId(x) = X \cup \exists_1\, Id \bullet Id \notin X$$

7.4.2.3. Ontology Subsetting

Ontology subsetting is an important capability for the usage of ontologies. ValueSets are subsets of an ontology Tbox or Abox. ValueSets contain a definition which defines the elements of a value set and metadata for the value set (cf. section 7.3.4.1). The following function resolves the value set definition to a list of the elements.

7.4.2.3.1. ValueSet resolution function *resolveSet*

$$
\begin{array}{|l}
\hline
[X, Y] \\
resolveSet : X \times Y \rightarrow Z \\
\hline
\forall\, X : ValueSetDefinition.subsettingStatement, \\
\quad \forall\, Y = \{x : \mathcal{O} \mid x.VersionedIdentifier = ValueSetDefinition.sourceOntology\}, \\
\quad \forall\, Z = (\{x : Y \bullet \Phi\} \lor \{x : Y \bullet \Upsilon\}) \\
\qquad \bullet\ (resolveSet(X, Y) = Z) \land \#\{Z\} \geq 1 \\
\ddagger \text{ The result Z is either a set of concept or individual IRIs } \ddagger \\
\hline
\end{array}
$$

7.4.3. Query interface

The following diagram illustrates how the CTS2 query interface will be represented in UML in the next version of this specification.

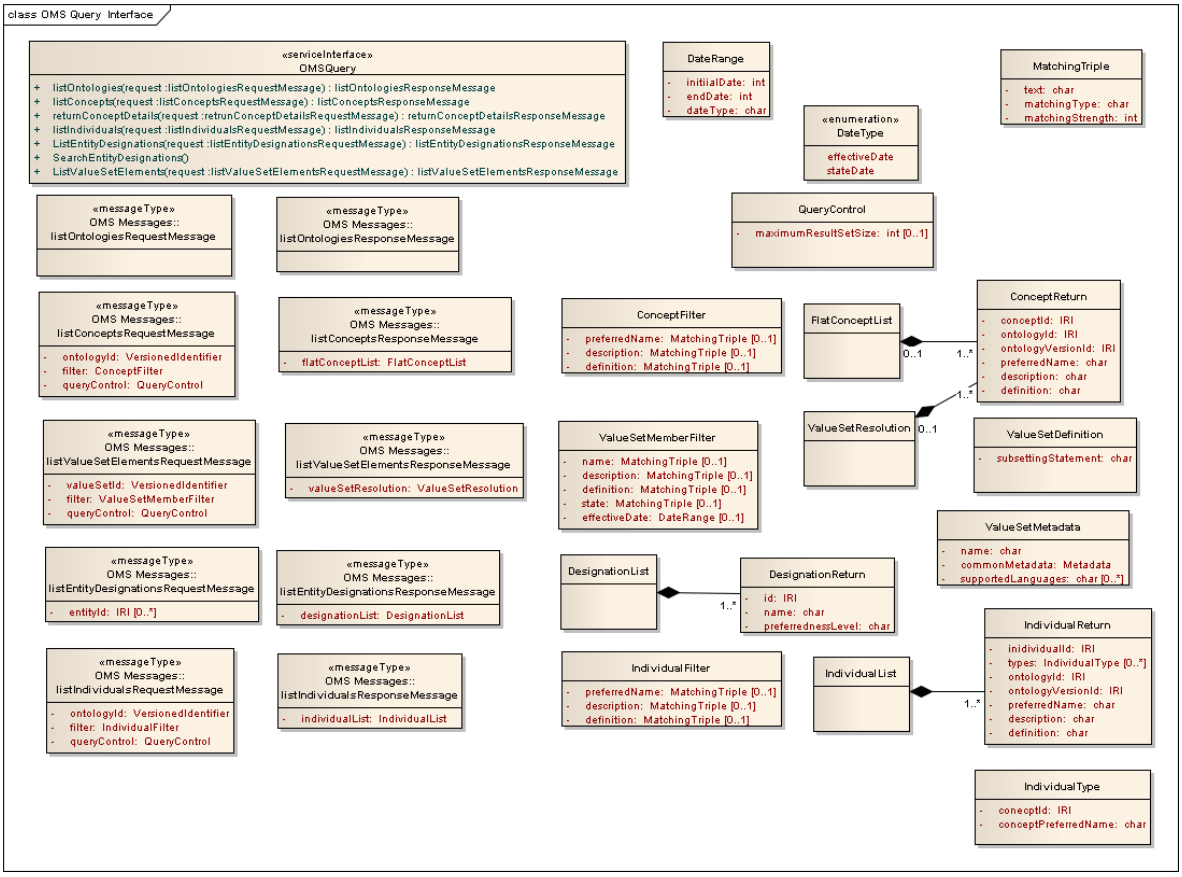

Figure 7.2.: CTS2 query interface represented as class diagram (only subset shown). Note that OMS is an ii4sm internal alias for CTS2.

7.4.3.1. List Ontologies

7.4.3.1.1. Operation specific interface types

$\mathcal{O}filter$

$description : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$fullName : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$releaseFormat : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$supportedLanguages : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$copyright : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$dateRange : DateRange [0 \,..\, 1]$

$matchingType \in MatchingType$
$matchingStrength \in \{x : \mathbb{R} \mid 0 \leq x \leq 1\}$

7.4.3.1.2. Operation definition This operation lists the ontologies available on an instance of the CTS2 service that match entered filter criteria.
It corresponds to the HL7 SFM operation “List Code Systems”.

listOntologies

$\Xi\, \mathcal{O}repository$

$state? : \mathbb{P}\, CHAR$

$filter? : \mathcal{O}filter$

$queryControl? : QueryControl$

$\mathcal{O}list! : \mathbb{P}\,\mathrm{seq}(\mathcal{O}.\{id, versionId, fullName, description\}, matchingStrengthValue)$

$status! : BOOLEAN$

$statusCondition! : StatusCode$

---

‡Initialisation‡

$status! = \top \land statusCondition! = StatusCode.c(1)$

‡Input checking‡

$\theta filter? \in \mathcal{O}Filter$ ‡θ evaluates to the binding of its argument‡

$\lor\ (status! = \bot \land statusCondition! = StatusCode.c(3) \land \mathcal{O}list! = \varnothing)$

$\theta queryControl? \in QueryControl$

$\lor\ (status! = \bot \land statusCondition! = StatusCode.c(4) \land \mathcal{O}list! = \varnothing)$

e‡Compute interim variable output ‡

$(state? = \varnothing \Rightarrow \mathcal{O}list' = \{x : \mathcal{O}\}) \lor state' = state? \land$

$\mathcal{O}list' = \bigcup\{\forall\, y \in state' \bullet \{x : \mathcal{O} \mid x.state = y.state\}\ \}$

$(filter?.dateRange = \varnothing \Rightarrow initDate' = 0 \land endDate' = \infty)$

$\lor\ initDate' = filter?.dateRange.initDate \land endDate' = filter?.dateRange.endDate$

$\mathcal{O}list'' = \{x : \mathcal{O}list' \mid initDate' \leq x.timeStamp \leq endDate'\}$

$(filter?.\{fullName, description, releaseFormat, supportedLanguages, copyright\} = \varnothing \Rightarrow$

$\mathcal{O}list''' = \{x : \mathcal{O}list'' \bullet x.\{id, versionId, fullName, description\}\ \} \lor$

$(\forall\, x \in \{\text{“}prefName''\text{”}, \text{“}description''\text{”}, \text{“}releaseFormat''\text{”}, \text{“}supportedLanguages''\text{”}, \text{“}copyright''\text{”}\}$

$\bullet\ filter?.x \neq \varnothing \land \sim_x = \sim_{filter?.x(2)} \land$

$\mathcal{O}list''' = \bigcap\{\forall\, y \in \{\text{“}prefName''\text{”}, \text{“}description''\text{”}, \text{“}releaseFormat''\text{”},$

$\text{“}supportedLanguages''\text{”}, \text{“}copyright''\text{”}\} \bullet \{x : \mathcal{O}list'' \mid match' = \sim_y (x.y, filter?.y(1)) \land$

$filter?.y \neq \varnothing \land match' \geq filter?.y(3) \bullet x.\{id, versionId, fullName, description\}\}\})$

‡Apply query control to generate output‡

$((queryControl? = \varnothing \lor queryControl?.maximumResultSetSize \geq \#\mathcal{O}list''')$

$\Rightarrow \mathcal{O}list! = \mathcal{O}list''') \lor$

$(queryControl?.maximumResultSetSize < \#\mathcal{O}list'''$

$\Rightarrow status! = \bot \land \mathcal{O}list! = \varnothing \land statusCondition! = StatusCode.c(5))$

7.4.3.2. Return Ontology Details

7.4.3.2.1. Operation definition This operation returns the details (metadata) for a given ontology available on the terminology service. It corresponds to the HL7 SFM operation "Return Code System Details".

returnOntologyDetails

$\Xi\, \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$\mathcal{O}details! : \mathcal{O} \setminus \{\mathcal{T}, \mathcal{A}, \mathcal{R}\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies an $\mathcal{O}identifier?$. If the repository does not contain an ontology with the supplied identifier, the operation stops and returns the status code C2 (ontology identifier not found). If no *versionId* was provided with the input, the operation automatically selects the highest available version of an ontology in the repository for requesting the ontology details. Based on this two parameters identifier and version id, the operation queries the details of an ontology and returns them with the output type $\mathcal{O}details$.

7.4.3.3. List Concepts

This operations returns a list of concepts from the Tbox of the specified ontology, optionally filtered by input criteria, as a flat list.
It corresponds to teh HL7 SFm operation "List Code System Concepts".

7.4.3.3.1. Operation specific interface types The ConcpetFilter input type allows the filtering by prefName, description, definition and dateRange.

ConceptFilter

$prefName : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$description : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$definition : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$dateRange : DateRange[0 \,..\, 1]$

7.4.3.3.2. Operation definition This operation returns a flat list of concept IRIs with some metadata.

listConcepts

$\Xi\ \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$filter? : ConceptFilter$
$state? : \mathbb{P}\ CHAR$
$queryControl? : QueryControl$
$flatConceptList! : \mathbb{P}\,\text{seq}(Concept.\{\{C \cup A \cup \bot \cup \top\}, prefName.name, description, definition\},$
$\quad matchingStrengthValue)$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies four input parameters. The $\mathcal{O}identifier?$ specifies the ontology to use in the repository for the operation. The *state?* parameter filters the concepts by their state. The *filter?* specifies the additional filters for concepts. In addition, the *queryControl?* allows to suppress a return if the number of result elements exceeds a supplied limit.

The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the provided $\mathcal{O}identifier$, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. The concept filter input is validated against its values *prefName*, *description* and *definition*. If these parameters are not of the type character, the method stops and returns error C3 (filter syntactically incorrect). The value prefName is set to $\top$ if all three values of the concept filter are empty - in other words this allows to query for all concepts of an ontology. In case no query control was provided, the method stops and returns error C4 (query control syntactically incorrect).

The second stage of the operation is to query the repository against the input parameters, starting with state. If no state is supplied, all concepts of the ontology are used as input for the next step, which filters the concept's time stamps against a date range. The last filtering step filters the string values of the concept filter. The query selects all concepts that match against the values of prefName, description and definition. Depending of the used string matching algorithm, it is also possible to calculate the matching strength against the provided input. The result of the query is a list of concepts, where in each element contains the matching strenght and the IRI of a concept together with the metadata values *prefName*, *name*, *description*, *definition*.

Finally, the last stage of the operation is to compare the number of results of the query against the maximum allowed elements of the result as provided in the query control. If the result set size of the query is bigger than the given *maximumResultSetSize* in the query control, the error C5 (Maximum result set size exceeded) is return. Otherwise, the operation

returns the flat concept list and sets the value of *status* to true to indicate that the query was succesfully operated.

7.4.3.4. Return Concept Information

7.4.3.4.1. Operation definition This operation returns the details for the known metadata of a concept inlcuding the concept definitions (the right side of the DL axiom) and, if requested, the concept expansion.
It corresponds to the HL7 SFM operation "Return Code System Details".

returnConceptInformation

$\Xi\, \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$conceptIdentifier? : ConceptIdentifier$
$returnExpansion? : BOOLEAN$
$conceptMetadata! : Concept \setminus \{C \cup A \cup \top \cup \bot\}$
$conceptExpansion! : \mathbb{P}\, \Phi_C[0 \,..\, 1]$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies the input parameters $\mathcal{O}Identifier?$ and a $conceptIdentifier?$ as well as $returnExpansion?$. The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the supplied $\mathcal{O}Identifier?$, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. The input of the ontology identifier is used to validate, if such a concept exists in the selected ontology. If the concept does not exists, the operation stops and returns the error code C7 (concept identifer not found).

The second stage of the operation is to perform the query against the repository using the ontology identifier. The result of the query is the type *conceptMetadata* which contains the meta data of the concept and its expansion (if requested).

7.4.3.5. Return individuals for concept

The operation returns the individuals instantiating a given concept.

returnIndividualsForConcept

$conceptIdentifier? : IRI$

$individualList! : \mathbb{P}\, Individual.\{\{o, a\}, types, prefName.name, description, definition\}$

$status! : BOOLEAN$

$statusCondition! : StatusCode$

7.4.3.6. List Individuals

The operation returns a flat list of individuals from the DL ABox of an ontology $\mathcal{O}$.

7.4.3.6.1. Operation specific interface types The filter is identical to the concept filter.

IndividualFilter

$ConceptFilter$

7.4.3.6.2. Operation definition

listIndividuals

$\Xi \mathcal{O} repository$

$\mathcal{O}Identifier? : VersionedIdentifier$

$state? : \mathbb{P}\, CHAR$

$filter? : IndividualFilter$

$queryControl? : QueryControl$

$individualList! : \mathbb{P}\, \mathrm{seq}(Individual.\{\{o, a\}, types, prefName.name, description, definition\},$
$matchingStrengthValue)$

$status! : BOOLEAN$

$statusCondition! : StatusCode$

The client supplies four parameters as input. $\mathcal{O}identifier?$ specifies the ontology to use in the repository for the operation. The parameter *state*? provides a list of zero to many strings. The *individualFilter?* contains parameters to filter the list of individuals. In addition, the *query control* allows to limit the number of results, the operation returns.

The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the provided *version identifier*, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. The individual filter input is validated against its values *prefName*, *description* and *definition*. If

these parameters are not of the type character or all three values are empty, the method stops and returns error C3 (filter syntactically incorrect). In the case that no query control was provided, the method stops and returns error C4 (query control syntactically incorrect).

The second stage of the operation is to query the repository against the state, the data and the string values of the individual filter. The query selects the union of all individuals that match one of the states. It then excludes those which do no match the supplied date range and uses the output of this step to generates the intersect of the set of individuals that match against the values of prefName, description and definition. Depending of the used string matching algorithm, it is also possible to calculate the matching strength against the provided input. The result of the query is a list of individuals, where in each element contains the matching strength and the IRI of a concept together with the metadata values *types*, *name*, *description* and *definition*.

Finally, the last stage of the operation is to compare the number of results of the query against the maximum allowed elements of the result as provided in the query control. If the result set size of the query is bigger than the given *maximumResultSetSize* in the query control, the error C5 (Maximum result set size exceeded) is return. Otherwise, the operation returns the *individual list* and sets the value of *status* to true to indicate that the query was succesfully operated.

7.4.3.7. Return Individual Details

7.4.3.7.1. Operation definition This operation returns the details for the known metadata of an individual including its type and the roles and datatype properties which predicate over it.

returnIndividualDetails

$\Xi\, \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$individualIdentifier? : IRI$
$individualMetadata! : Individual \setminus \{a, o\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The clients supplies the input parameters $\mathcal{O}Identifier?$ and $individualIdentifier?$ The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the provided *version identifier*, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but not *version id* was provided, the highest available version of an ontology in the repository is used. The input of the ontology identifier is used to validate, if such a individual exists in the selected ontology.

If the individual does not exists, the operation stops and returns the error code C7 (individual identifer not found).

The second stage of the operation is to perform the query against the repository using the ontology identifier. The result of the query is the type *individualMetadata* which contains the meta data of the individual. including its type and the roles and datatype properties which predicate over it.

7.4.3.8. Return individual assertions

The operation provides a set of individual assertions (same individual, different individuals and alternative individuals) which are explicitly asserted for the supplied individual in the Abox of the supplied ontology.

returnSameIndividuals

$\Xi\ \mathcal{O}repository$
$\mathcal{O}identifier? : VersionedIdentifier$
$individualIdentifier? : IRI$
$sameIndividuals! : \mathbb{P}\ Individual.\{\{a \cup o\}, types, prefName.name, description, definition\}$
$differentIndividuals! : \mathbb{P}\ Individual.\{\{a \cup o\}, types, prefName.name, description, definition\}$
$alternativeIndividuals! : \mathbb{P}\ Individual.\{\{a \cup o\}, types, prefName.name, description, definition\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.3.9. List Entity Designations

This operation lists a subset of the metadata available for the designations of a set of concepts, roles or individuals

7.4.3.9.1. Operation definition

listEntityDesignations

$\Xi\mathcal{O}repository$
$entityIdentifier? : \mathbb{P}_1\ IRI$
$designationList! : \mathbb{P}\,\mathrm{seq}(Designation.\{name, id, preferrednessLevel\},$
$\quad matchingStrengthValue)$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies one or more *entityIdentifier?*, IRIs for querying the entity designations of either concepts, roles or individuals.

In the first stage, the operation checks whether all IRIs in the entityIdentifier list are available in an ontology of the repository. If not, the operation stops and returns the error code C17 (Entity (concept, role or individual) identifier not found.)

The query in the second stage of the operation takes the entityIdentifier list to compute the output. Therefore, all related concepts, individuals or roles for this list are computed and their designation values are loaded. For the resulting output *designationList*, all elements get the values *name*, *id* and *preferrednessLevel* out of each computed designation element.

7.4.3.10. Return Designation Details

This operation returns the metadata of a supplied designation.

returnDesignationDetails

$\Xi\, \mathcal{O}repository$
$designationId? : IRI$
$designationDetails! : Designation \setminus id$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.3.11. Find Entity Designations

This operations finds entity designations based on supplied state and filter input.

7.4.3.11.1. Operation specific interface types

DesignationFilter

$name : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$designationType : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$language : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$dateRange : DateRange [0 \,..\, 1]$

7.4.3.11.2. Operation definition

findEntityDesignations

$\Xi\ \mathcal{O}repository$
$state? : \mathbb{P}\ CHAR$
$filter? : DesignationFilter$
$designationList! : \mathbb{P}\ Designation.\{id, name, relatedConcept, relatedIndividual, relatedRole, designationType\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.3.12. Return Role Assertions

This operation returns the role assertions available in a supplied ontology.

It corresponds to the HL7 SFM operation "List Association Types". There is no equivalent to the HL7 operation "Return Association Type Details" as this specification only uses the OWL role assertions for which there are no custom metadata.

returnRoleAssertions

$\Xi\ \mathcal{O}repository$
$\mathcal{O}identifier? : VersionedIdentifier$
$roleAssertions! : \mathbb{P}\ \mathcal{R}_a$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.3.13. List datatypes properties

The operation lists minimal metadata for the data types available in the supplied ontology based on filter criteria supplied by the client, including the arguments (individual and data value) if requested.

7.4.3.13.1. Operation specific interface types

datatypePropertyFilter

$name : \langle CHAR, matchingType, matchingStrength\rangle[0\,..\,1]$
$description : \langle CHAR, matchingType, matchingStrength\rangle[0\,..\,1]$
$definition : \langle CHAR, matchingType, matchingStrength\rangle[0\,..\,1]$
$dateRange : DateRange[0\,..\,1]$

7.4.3.13.2. Operation definition

listDatatypeProperties

$\Xi \mathcal{O}repository$
$\mathcal{O}identifier? : VersionedIdentifier$
$state? : \mathbb{P}\, CHAR$
$filter? : datatypePropertyFilter$
$listArguments? : BOOLEAN$
$datatypePropertyList! : \mathbb{P}\,\mathrm{seq}(DatatypeProperty.\{P, name, description, definition\},$
$\quad \mathbb{P}\, Individual.\{\{a \cup o\}, prefName.name\}, \mathbb{P}\, u, matchingStrengthValue)$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

1268 #### 7.4.3.14. Return DataType Property Details

1269 This operation returns the metadata for a supplied data type property including its arguments.

returnDataTypePropertyDetails

$\Xi\ \mathcal{O}repository$
$\mathcal{O}identifier? : VersionedIdentifier$
$dataTypePropertyIdentifier? : IRI$
$dataTypePropertyDetails! : DatatypeProperty \setminus P$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

1270 #### 7.4.3.15. List Roles

1271 This operations returns the roles of an ontology, optionally filtered.
1272 It corresponds to the HL7 SFM operation “List Associations”.

7.4.3.15.1. Operation specific interface types

RoleFilter

$roleAssertion : \mathcal{R}_a[0 \,..\, 1]$
$R_{type} : R_t$
$name : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$definition : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$description : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$

7.4.3.15.2. Operation definition

listRoles

$\Xi\, \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$state? : \mathbb{P}\, CHAR$
$filter? : RoleFilter$
$queryControl? : QueryControl$
$listArguments? : BOOLEAN$
$roleList! : \mathbb{P}\,\mathrm{seq}(Role.\{R, R_t, name, description, definition\},$
$\quad \mathbb{P}\, Entity_1.\{id, prefName.name\}, \mathbb{P}\, Entity_2.\{id, prefName.name\}, matchingStrengthValue)$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies four parameters as input. The $\mathcal{O}identifier?$ specifies the ontology to use in the repository for the operation. The *state*? parameter allows filtering of roles by state. The *roleFilter?* specifies the query input to generate the list of roles. In addition, the *query control* allows to limit the number of results the operation returns. The boolean parameter *listArguments*? allows to indicate whether the arguments of the role should be returned.

The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the provided *version identifier*, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. The role filter input is validated against its values. The input for the *role type* is mandatory. Additionally, the values *role assertion*, *forward name*, *reverse name*, *definition* and *description* of a role and the types *characteristic value* and *characteristic type* allow for further filtering the result. If these parameters are not valid, the method stops and returns error C3 (filter syntactically incorrect). In the case that no query control was provided, the method stops and returns error C4 (query control syntactically incorrect).

In the second stage of the operation, the *role list* is generated by querying for the union

of all roles that match against the *state*? parameter. This set is further matched against the role type and the optional filter values. To generate the output list, for all resulting roles, the metadata values described under *roleList*! are returned, including the role's arguments if requested.

Finally, the last stage of the operation is to compare the number of results of the query against the maximum allowed elements of the result as provided in the query control. If the result set size of the query is bigger than the given *maximumResultSetSize* in the query control, the error C5 (Maximum result set size exceeded) is return. Otherwise, the operation returns the role list and sets the value of *status* to true to indicate that the query was succesfully operated.

7.4.3.16. Return Role Details

This operation returns the metadata of the supplied role.

It corresponds to the HL7 SFM operation "Return Association Details".

returnRoleDetails

$\Xi\, \mathcal{O}repository$
$roleIdentifier? : IRI$
$roleMetadata! : Role \setminus R$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies a role identifier. The operation first checks whether the role exists an raises an exception if this is not the case. In the second step, the operation retrieves the metadata of the supplied role including the role arguments. These values are returned.

7.4.3.17. List Value Sets

This operation lists the value sets that are available to a CTS 2 service instance.
It corresponds to the HL7 SFM operation of the same name.

7.4.3.17.1. Operation specific interface types

ValueSetFilter

$$
\begin{aligned}
&name : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1] \\
&description : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1] \\
&definition : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1] \\
&dateRange : DateRange [0 \,..\, 1]
\end{aligned}
$$

7.4.3.17.2. Operation definition

listValueSets

$$
\begin{aligned}
&\Xi\ ValueSetRepository \\
&state? : \mathbb{P}\, CHAR \\
&filter? : ValueSetFilter \\
&queryControl? : QueryControl \\
&valueSetList! : \mathbb{P}\, \mathrm{seq}(ValueSet.\{id, versionId, name\}, matchingStrengthValue) \\
&status! : BOOLEAN \\
&statusCondition! : StatusCode
\end{aligned}
$$

The client supplies three input parameters: *state*?, *filter*? and *queryControl*?.

The first stage of the operation is to validate the input parameters. If the value set filter contains incorrect values, the operation stops and returns the error code C3 (filter syntactically incorrect). In the case that no query control was provided, the method stops and returns error C4 (query control syntactically incorrect).

In the second stage of the operation, the list of value sets is computed. Therefore, all value sets that match against the *name* value in the value set filter are queried. If the values *description*, *state* and *state date* are provided in the filter, the list of value set is filtered against them. If the parameter *span ontology* is set to true, the query result may also contain concepts from more than one ontology. For each of the resulting concepts, the *minimal metadata* values name, description, id and version are loaded and added to the return type of the operation *value set list*.

Finally, the last stage of the operation is to compare the number of results of the query against the maximum allowed elements of the result as provided in the query control. If the result set size of the query is bigger than the given *maximumResultSetSize* in the query control, the error C5 (Maximum result set size exceeded) is return. Otherwise, the operation returns the flat concept list and sets the value of *status* to true to indicate that the query was succesfully operated.

7.4.3.18. Return Value Set Details

7.4.3.18.1. Operation definition This operation returns the details (metadata) for a supplied value set. It corresponds to the HL7 SFM operation of the same name.

returnValueSetDetails

Ξ *ValueSetRepository*
valueSetIdentifier? : *VersionedIdentifier*
valueSetDetails! : *ValueSet* $\setminus$ {*id*, *versionId*}
status! : *BOOLEAN*
statusCondition! : *StatusCode*

The client supplies a *valueSetIdentifier*?.

The operation first checks whether a version id was provided for the value set. If not, the highest version id of the value set in a repository is used for the query. In the second step, the query against the repository takes the version identifier to load the value set and all its meta data. These values are mapped to the output type *value set details* and returned.

7.4.3.19. List Value Set Elements

This operations lists the elements of a given value set, filtered based on input criteria. It corresponds to the HL7 SFM operation "List Value Set Contents (Expand Value Set)".

7.4.3.19.1. Operation specific interface types

ValueSetMemberFilter

name : $\langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
description : $\langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
definition : $\langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
state : $\mathbb{P}\, CHAR$
effectiveDate : $DateRange[0 \,..\, 1]$

7.4.3.19.2. Operation definition

listValueSetElements

Ξ *ValueSetRepository*

valueSetIdentifier? : *VersionedIdentifier*

filter? : *ValueSetMemberFilter*

queryControl? : *QueryControl*

valueSetResolution! : $\mathbb{P}\,\mathrm{seq}(\mathcal{O}\,VersionedIdentifier,$
$\mathrm{seq}(\{Concept.\{\{C \cup A \cup \bot \cup \top\}, prefName.name, description, definition\}$
$\cup Individual.\{\{o \cup a\}, prefName.name, description, definition\}\},$
$matchingStrengthValue))$

status! : *BOOLEAN*

statusCondition! : *StatusCode*

The client supplies the three parameters *valueSetIdentifier?*, a *filter?* and a *queryControl?* parameter.

The first stage of the operation is to validate the input parameters. If the repository contains no value set matching to the *value set identifier*, the operation stops and returns a empty list. If the value set exists, but not *version id* was provided for the value set identifier, the highest available version of the value set in the repository is identified and used. If the *value set member filter* contains incorrect values for its character based values, the operation stops and returns the error code C3 (filter syntactically incorrect). In the case that no query control was provided, the method stops and returns error C4 (query control syntactically incorrect).

In the second stage of the operation, the *value set resolution* list is computed. The first step is to load all concepts and individuals that belong to the value. If the value set member filter contains values, they are used to filter the result against the values *name*, *description*, *definition*, *state* and *effective date*. To generate the result list *valueSetResolution*, for each of the concepts and individuals in the current result, the versioned identifier of the ontology it belongs is computed. In the valueSetResolution, each element finally contains the versioned identifier of the ontology it belongs, the metadata of the concept or individual and the matching strength against the string matching algorithm used in the query.

The last stage of the operation is to compare the number of results of the query against the maximum allowed elements of the result as provided in the query control. If the result set size of the query is bigger than the given *maximumResultSetSize* in the query control, the error C5 (Maximum result set size exceeded) is return. Otherwise, the operation returns the valueSetResolution and sets the value of *status* to true to indicate that the query was succesfully operated.

7.4.3.20. List Mappings

This operation lists the mappings that are available to a CTS 2 service instance.

7.4.3.20.1. Operation specific interface types

MappingFilter

$name : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$description : \langle CHAR, matchingType, matchingStrength \rangle [0 \,..\, 1]$
$mappingStrength : MappingStrength [0 \,..\, 1]$
$date : DateRange [0 \,..\, 1]$

7.4.3.20.2. Operation definition

listMappings

$\Xi\, \mathcal{O}repository$
$state? : oCHAR$
$filter? : MappingFilter$
$queryControl? : QueryControl$
$mappingList! : \mathbb{P}\, \mathrm{seq}(Mapping.\{id, versionId, name, description\}, matchingStrengthValue)$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies a *mapping filter* and a *query control* parameter.

The first stage of the operation is to validate the input parameters. If the values in the mapping filter contains invalid values, the operation stops and returns error code C3 (filter syntactically incorrect). In the case that no query control was provided, the method stops and returns error C4 (query control syntactically incorrect).

In the second stage of the operation, all mapping that match against the mapping filter are queried. For each of the mappings, the minimal metadata values name, description, id and version are loaded added to the output list of the operation mappingList.

Finally, the last stage of the operation is to compare the number of results of the query against the maximum allowed elements of the result as provided in the query control. If the result set size of the query is bigger than the given *maximumResultSetSize* in the query control, the error C5 (Maximum result set size exceeded) is return. Otherwise, the operation returns the mapping list and sets the value of *status* to true to indicate that the query was succesfully operated.

7.4.3.21. Return Mapping Details

7.4.3.21.1. Operation definition This operation returns the details (metadata) for a given mapping available on the terminology service.

returnMappingDetails

Ξ *MappingRepsitory*
mappingIdentifier? : *VersionedIdentifier*
mappingDetails! : *Mapping* $\setminus$ {*id*, *versionId*, *mappingRelation*, *selectionRule*}
status! : *BOOLEAN*
statusCondition! : *StatusCode*

The operation returnMappingDetails gets as input parameter the *versiond identifier* of a mapping.

The operation first checks whether a version id was provided for the mapping. If not, the highest version id of the mapping in a repository is used for the query. In the second step, the query against the repository takes the version identifier to load the mapping and all its meta data. These values are mapped to the output type *mapping details* and returned.

7.4.3.22. returns Mapping Tuples

This operation returns the tuples of one $Mapping_i \in Mapping$.

7.4.3.22.1. Operation definition

returnMappingTuples

Ξ *MappingRepository*
mappingIdentifier? : *VersionedIdentifier*
queryControl? : *QueryControl*
mappingTuples! : $\text{seq}(\mathbb{P}\{(\{A_1 \cup C_1\} \mapsto \{A_2 \cup C_2\}) \cup (\{a_1 \cup o_1\} \mapsto \{a_2 \cup o_2\}) \cup (R_1 \mapsto R_2)\}, \langle \mathcal{O}_1.\mathit{VersionedIdentifier}, \mathcal{O}_2.\mathit{VersionedIdentifier} \rangle)$
status! : *BOOLEAN*
statusCondition! : *StatusCode*

The client supplies a *mappingIdentifier*? and a *queryControl*? parameter. The operation ensures that the supplied mapping exists and raises an exception if this is not the case. If no versionID is supplied, the operation uses the latest available version of the supplied mapping. The operation retrieves the mapping tuples of the mapping's mapping relation and if the result side does not exceed the supplied *queryControl*? parameter, the operation returns the

tuples along with the versioned identifier of the a the source and the target ontologies.

7.4.4. Reasoning interface

The following diagram illustrates how the CTS2 reasoning interface will be represented in UML in the next version of this specification.

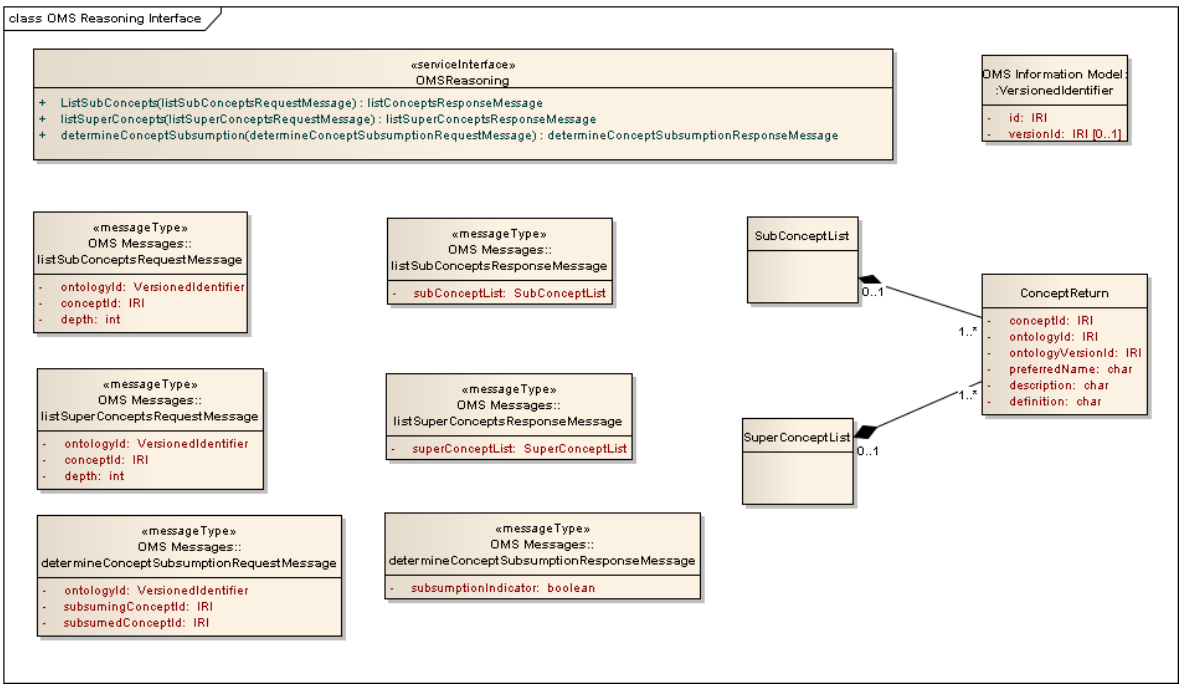

Figure 7.3.: CTS2 reasoning interface represented as class diagram (only subset of operations shown). Note that OMS is an ii4sm internal alias for CTS2.

CTS2 requires a reasoning interface in order to provide access to machine inference enabled by description logic. The reasoning interface allows to compute inference patterns such as concept satisfiablity and concept entailment which can be of great value in real world systems as described in section 1.

Note for implementers: The operations described in this interface could be realised using the OWL API. This Java API was created in 2001 and has continuously evolved since. It is a reference implementation for creating, manipulating and serializing OWL ontologies. In addition, its reasoner interface is implemented by commonly used reasoner such as Pellet, Racer and HermiT. The OWL API is currently available in version 3. Its interface is stable, well accepted by the Semantic Web community and can therefore be seen as a quasi standard for the communication with reasonining components.

Therefore, after each operation definition, we provide some directions how the operations can be implemented using the OWL API. The complete interface description of the OWL API is

available at owlapi.sourceforge.net[5].

7.4.4.1. List Concept Subconcepts

This operation returns the concept sequence(s) subsumed by a concept at the supplied depth. The output of the subsumption function $\dot{\subseteq}$ comprises explicit and implicit subconcepts (therefore this is an operation of the reasoning interface).
This operation provides a part of the functionality of the HL7 SFM operation "Compute Subsumption Relationship".

7.4.4.1.1. Operation specific interface types

RootConcept

$\mathcal{O}Identifier : VersionedIdentifier$
$conceptIdentifier : IRI$
$depth : \mathbb{N}[0\,..\,1]$

$depth = 1$

listConceptSubConcepts

$\Xi\ \mathcal{O}repository$
$rootConcept? : RootConcept$
$subConcepts! : \mathbb{P}\{Concept.\{\{C \cup A \cup \bot \cup \top\}, prefName.name, description, definition\}\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies a *rootConcept*? which comprises an ontology identifer, the root concept identifier and the depth of the subsumption tree that is to be built.
The first stage of the operation is to validate the input *rootConcept*? type. If the repository contains no ontology that matches to the provided *version identifier*, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. If the concept identifier does not point to an existing concept in the ontology to use for the query, the operation stops and returns with error code c7 (concept identifier not found).
In the second stage of the operation, the list of sub concepts starting from the concept to start from is computed. Therefore, based on the parameter depth, the query is recursively iterated to build the complete list. For all queried concepts, the concept itself together with

[5]OWL API interface description http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/reasoner/OWLReasoner.html

its meta data values *prefName*, *description* and definition is loaded and returned as result of the operation

An implementation using the OWL API would use the method *getSubClasses*.

7.4.4.2. List Concept Superconcepts

This operation is the inverse of the operation *List Concept Subconcepts*. Instead of recursively querying for sub concepts, this method recursively selects the super concepts of a given concept in an ontology.

listConceptSuperConcepts

$\Xi\, \mathcal{O}repository$
$rootConcept? : RootConcept$
$superConcepts! : \mathbb{P}\{Concept.\{\{C \cup A \cup \bot \cup \top\}, prefName.name, description, definition\}\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

An implementation using the OWL API would use the method *getSuperClasses*.

7.4.4.3. List role sub-roles

This operation recursively selects the sub-roles of a supplied role in an ontology.

7.4.4.3.1. Operation specific interface types

RootRole

$\mathcal{O}Identifier : VersionedIdentifier$
$roleIdentifier : IRI$
$depth : \mathbb{N}[0\,..\,1]$

$depth = 1$

listRoleSubRoles

$\Xi\, \mathcal{O}repository$
$rootRole? : RootRole$
$roleList! : \mathbb{P}\, Role.\{R, R_t, name, description, definition\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

An implementation using the OWL API would use the method *getSubDataProperties*.

7.4.4.4. List role super-roles

This operation is the same operation as the method *List Concept Subconcepts* for roles. In consequence, instead of recursively querying for super concepts, this method recursively selects the super-roles of a given role in an ontology.

listRoleSuperRoles

$\Xi\, \mathcal{O}repository$
$roleparameters? : RoleParameters$
$roleList! : \mathbb{P}\, Role.\{R, R_t, name, description, definition\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

An implementation using the OWL API would use the method *getSuperDataProperties*.

7.4.4.5. Return Equivalent Concepts

This operation returns minimal metadata for the set of concepts that are equivalent ($\equiv$) to the supplied concept.

returnEquivalentConcepts

$\Xi\, \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$conceptIdentifier? : IRI$
$equivalentConcepts! : \mathbb{P}\, Concept.\{\{A \cup C \cup \top \cup \bot\}, prefName.name, description, definition\}$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

Note for implementers: An implementation using the OWL API would use the method *getEquivalentClasses*.

7.4.4.6. Determine Concept Subsumption

This operations determines whether a concept is subsumed by another and gives out the chain(s) connecting the concepts. The concepts may be atomic or complex, and the subsumption may be explicit or implicit.

This operation provides a part of the functionality of the HL7 SFM operation “Compute Subsumption Relationship”.

determineConceptSubsumption

$\Xi\, \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$subsumedConcept? : IRI$
$subsumingConcept? : IRI$
$subsumptionIndicator! : BOOLEAN$
$subsumptionChains! : \mathbb{P}\, \Phi$
$status! BOOLEAN$
$statusCondition! : StatusCode$

The client supplies an $\mathcal{O}Identifier?$, and IRIs identifying the *subsumedConcept*? and the *subsumingConcept*?.

The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the provided *version identifier*, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. If the concepts as specified in subsumedConcept and subsumingConcept are not part of the ontology to use, the operation stops and returns error code c7 (concept identifier not found).

The second stage of the operation is to calculate whether there is a connection chain between the two given concepts. The simplest possibility therefore is to recursively follow all sub concepts starting from the concept subsumedConcept and to stop if the subsumingConcept is reached or no more recursions are possible. If the subsumingConcept was reached, the method returns the subsumingChain - the concepts between the two given concepts and sets the parameter subsumingIndicator to true.

7.4.4.7. Determine Entity Relationship

The operation determines whether there exists a relationship of a supplied type (role assertion) between two supplied entites (concepts or individuals). If the supplied role assertion is (ir)reflexive, the same entity must be supplied twice.
This operation corresponds to the HL7 SFM operation “Determine Transitive Concept Relationship”.

7.4.4.7.1. Operation definition

determineEntityRelationship

$\Xi\ \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$entityTuple? : (IRI_1, IRI_2)$
$roleAssertion? : \mathcal{R}_a \setminus Dis(R, S)$
$associationIndicator! : BOOLEAN$
$associationChains! : \mathbb{P}\,\Phi$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

The client supplies an $\mathcal{O}Identfier$, a entity tuple and a role assertion.

The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the provided $\mathcal{O}Identifier?$, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. If the individual or concepts as specified in entity tupel are not part of the ontology to use, the operation stops and returns error code c7 (concept identifier not found). If the given roleAssertion is not a valid role in the ontology, the operation stops and returns erroc code c9 (?). If the given role is either reflexiv or irreflexiv, the operation checks if the role was supplied twice. If not, the operation stops and returns error code c22 (Same entity must be supplied to test reflexivity).

The second stage of the operation is to compute whether the given role exists between the two given concepts or instance and concept in the entity tupel. If so, the method sets the boolean parameter associationIndicator to true and returns it. In that case, the operation also returns the association path which connects the entities.

7.4.4.8. Determine Entity Value Set Membership

This operation determine whether a concept or an individual is an element of a value set. It corresponds to the HL7 SFM operation “Check Concept Value Set Membership”.

7.4.4.8.1. Operation specific interface types

7.4.4.8.2. Operation definition

determineConceptValueSetMembership

Ξ *ValueSetRepository*
$\mathcal{O}$*Identifier*? : *VersionedIdentifier*
entityIdentifier? : *entityIdentifier*
valueSetIdentifier? : *VersionedIdentifier*
membershipIndicator! : *BOOLEAN*
status! : *BOOLEAN*
statusCondition! : *StatusCode*

The client supplies an $\mathcal{O}$*Identifier*?, an entity (concept or individual) identifier and a value set identifier.

The first stage of the operation is to validate the input parameters. If the repository contains no ontology that matches to the provided $\mathcal{O}$*Identifier*, the method stops and returns the error code C2 (ontology identifier not found). If the ontology exists, but no *version id* was provided, the highest available version of an ontology in the repository is used. If the entity specified in does not exists in the ontology, the operation stops and returns the error code C7 (identifer not found). If the value set identified with the version identifier does not exists, the operation stops and returns error code C11. If no version id for the value set was provided, the highest avaialable version of the value set in the repository is used.

The second stage of the operation is to check, whether the given entity exists in the specified value set. If so, the output parameter membershipIndicator is set to true and returned.

7.4.4.9. Determine concept entailment

The operation determines whether a supplied definition for a complex concept is entailed by an ontology of the CTS2 repository.

determineConceptEntailment

Ξ $\mathcal{O}$*repository*
$\mathcal{O}$*Identifier*? : *VersionedIdentifier*
conceptDefinition? : Φ_C
entailmentIndicator! : *BOOLEAN*
status! : *BOOLEAN*
statusCondition! : *StatusCode*

7.4.4.10. Determine concept satisfiability

The operation determines whether a supplied complex concept definition is satisfiable against a given ontology.

determineConceptSatisfiability

$\Xi\ \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$conceptDefinition? : \Phi_C$
$satisfiabilityIndicator! : BOOLEAN$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.4.11. Determine concept equivalence

The operation determines whether two supplied concepts are equivalent (implicitly or explicitly).

determineConceptEquivalence

$\Xi\ \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$conceptIdentifier_1? : IRI$
$conceptIdentifier_2? : IRI$
$equivalenceIndicator! : BOOLEAN$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.4.12. Determine Ontology consistency

The operation determines whether a given ontology is consistent.

determineOntologyConsistency

$\Xi\ \mathcal{O}repository$
$\mathcal{O}Identifier? : VersionedIdentifier$
$conceptIdentifier? : VersionedIdentifier$
$satisfiabilityIndicator! : BOOLEAN$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.5. Management interface

The management interface allows the import of ontologies and mappings and their versions as well as the creation of value sets and their versions. Notification support described in the HL7 SFM is not regarded as a functionality to be provided by a terminology service. Export capabilities are not seen as a part of the interface definition either as export is usually a design/maintenance activity and not a run-time (high-frequency) activity. Fine grained authoring support is not provided as a service interface.

The following diagram illustrates how the CTS2 management interface will be represented in UML in the next version of this specification.

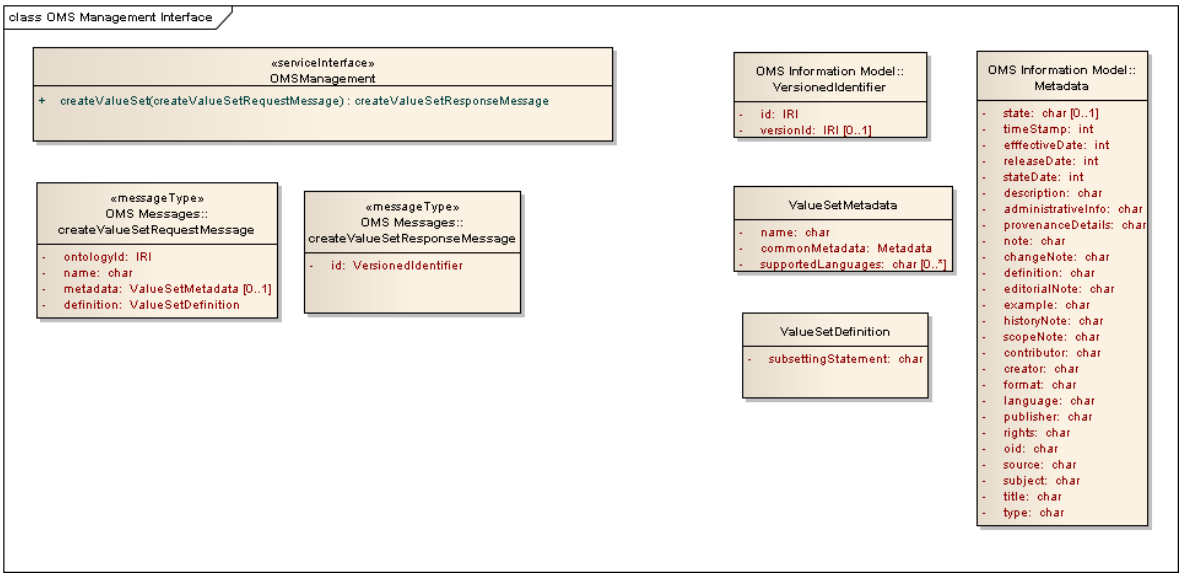

Figure 7.4.: CTS2 management interface represented as class diagram (only subset of operations shown). Note that OMS is an ii4sm internal alias for CTS2.

7.4.5.1. Import Ontology

This operation allows the import of an ontology persisted in OWL2 RDF XML. The client provides an ontology file URI or a physical ontology file as well as ontology metadata. The operation imports the RDF/XML into its persistence, validates the ontology using the reasoning interface operation *determineOntologyConsistency* and creates the ontology with its metadata if this is the case; otherwise, the operation returns an exception.
This operation corresponds to the HL7 operation “Import Code System”.

importOntology

$\Delta\ \mathcal{O}repository$
$ontologyMetadata? : \mathcal{O} \setminus \{\mathcal{T}, \mathcal{A}, \mathcal{R}, id\}[0\,..\,1]$
$ontologyFile? : IRI \cup Binary$
$\mathcal{O}Identifier! : VersionedIdentifier$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.5.2. Import Ontology Version

This operation behaves like the previous one, but creates a new version of an existing ontology. This operation corresponds to the HL7 operation "Import Code System Revision".

importOntologyVersion

$\Delta\ \mathcal{O}repository$
$\mathcal{O}Identifier? : IRI$
$ontologyMetadata? : \mathcal{O} \setminus \{\mathcal{T}, \mathcal{A}, \mathcal{R}, id\}[0\,..\,1]$
$ontologyFile? : IRI \cup Binary$
$\mathcal{O}Version! : VersionedIdentifier$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

7.4.5.3. Create Value Set

This operations allows to create a value set that is defined by an expression which defines a set of elements (concepts or individuals). This is an oepration used by developers when committing an intensional (SPARQL) value set definition for persistence. The operation validates that the SPARQL query is syntactically correct before creating the value set. This operation corresponds to the HL7 operation of the same name.

7.4.5.3.1. Operation specific types

ValueSetMetadata

$name : CHAR$
$commonMetadata : Metadata$
$supportedLanguages : oCHAR$

7.4.5.3.2. Operation definition

createValueSet

Δ *ValueSetRepository*
valueSetMetadata? : *oValueSetMetadata*
definition? : *ValueSetDefinition*
valueSetId! : *VersionedIdentifier*
status! : *BOOLEAN*
statusCondition! : *StatusCode*

7.4.5.4. Create value set version

This operation allows the creation of a value set version. The client supplies a value set identifier and optional updates of the value set definition and the value set metadata. The operation creates the new version by incrementing the version id of the pervious latest version and returns this versioned identifier to the client.
This operation corresponds to the HL7 operation of the same name.

createValueSetVersion

Δ *ValueSetRepository*
valueSetIdentifier? : *IRI*
valueSetMetadata? : *ValueSetMetadata*[0 . . 1]
definition? : *ValueSetDefinition*
valueSetVersion! : *VersionedIdentifier*
status! : *BOOLEAN*
statusCondition! : *StatusCode*

7.4.5.5. Import Mapping

This operation allows the import of a mapping. The client provides a mapping file URI or a physical mapping file as well as mapping metadata. The operation uses the file to create the mappingRelation set of mapping tuples.
This operation provides some of the functionality of the HL7 SFM operations “Create Lexical Association” and “Create Rules Based association”

importMapping

$\Delta\ \mathcal{O}repository$
$mappingMetadata? : Mapping.\{mappingStrength, commonMetadata\}[0\,..\,1]$
$mappingFile? : IRI \cup Binary$
$mappingIdentifier! : VersionedIdentifier$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

Note for implementation: The supported format of the mapping file is an implementation choice. The mapping file must contain the set of mapping relation tuples which constitute the mapping and may optionally contain selectionRules as well.

7.4.5.6. Import Mapping Version

This operation allows the import of a mapping version. It behaves like the previous one, but creates a new version of an existing mapping.

importMapping

$\Delta\ \mathcal{O}repository$
$mappingIdentifier? : IRI$
$mappingMetadata? : Mapping.\{mappingStrength, commonMetadata\}[0\,..\,1]$
$mappingFile? : IRI \cup Binary$
$mappingVersion! : VersionedIdentifier$
$status! : BOOLEAN$
$statusCondition! : StatusCode$

Note for implementation: The supported format of the mapping file is an implementation choice. The mapping file must contain the set of mapping relation tuples which constitute the mapping and may optionally contain selectionRules as well.

7.5. CTS2 implementation conformance functional profiles

This chapter describes the functional implementation conformance profiles for CTS2. This specification splits the HL7 query profile into the query and reasoning profiles. More use-case group oriented profiles will be provided in the next version of this specification.

With the exception of the HL7 SFM ioerations listing concept domains and usage contexts or returning details for them, all operations of the HL7 query profile are covered.

7.5.1. Query profile

All operations of the query interface.

7.5.2. Reasoning profile

All operations of the reasoning interface.

8. CTS2 PSM

As a platform specific model for the specification, we provide a web service interface expressed as a W3C WSDL. The interface shown is the CTS2 prototype interface based on an older version of the CTS2 PIM and therefore deviating from the PIM given in this specification for some of the classes. A fully PIM congruent version will be published in the next round of CTS2 standardisation.

The prototype implementation realising the interface was developed in Java, deployed as an EJB3 session bean and exposed as a web-service using the Java API for XML Web Services (JAX-WS).

```
<definitions name="CTS2Service" targetNamespace="http://webservice.cts2.ii4sm.com/">
-
<types>
-
<xs:schema targetNamespace="http://webservice.cts2.ii4sm.com/" version="1.0">
<xs:element name="createValueSet" type="tns:createValueSet"/>
<xs:element name="createValueSetResponse" type="tns:createValueSetResponse"/>
<xs:element name="getConceptDetails" type="tns:getConceptDetails"/>
<xs:element name="getConceptDetailsResponse" type="tns:getConceptDetailsResponse"/>
<xs:element name="getFlatListOntologyConcepts" type="tns:getFlatListOntologyConcepts"/>
<xs:element name="getFlatListOntologyConceptsResponse" type="tns:getFlatListOntologyConceptsResponse"/>
<xs:element name="getOntologyIndividuals" type="tns:getOntologyIndividuals"/>
<xs:element name="getOntologyIndividualsResponse" type="tns:getOntologyIndividualsResponse"/>
<xs:element name="getSemanticListOntologyConcepts" type="tns:getSemanticListOntologyConcepts"/>
<xs:element name="getSemanticListOntologyConceptsResponse" type="tns:getSemanticListOntologyConceptsResponse"/>
<xs:element name="getSubSuperConcepts" type="tns:getSubSuperConcepts"/>
<xs:element name="getSubSuperConceptsResponse" type="tns:getSubSuperConceptsResponse"/>
<xs:element name="listPredicatesForArgument" type="tns:listPredicatesForArgument"/>
<xs:element name="listPredicatesForArgumentResponse" type="tns:listPredicatesForArgumentResponse"/>
<xs:element name="listValueSetElements" type="tns:listValueSetElements"/>
<xs:element name="listValueSetElementsResponse" type="tns:listValueSetElementsResponse"/>
<xs:element name="listValueSets" type="tns:listValueSets"/>
<xs:element name="listValueSetsResponse" type="tns:listValueSetsResponse"/>
<xs:element name="testService" type="tns:testService"/>
<xs:element name="testServiceResponse" type="tns:testServiceResponse"/>
-
<xs:complexType name="getFlatListOntologyConcepts">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputWithConceptFilter"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputWithConceptFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="filter" type="tns:conceptFilter"/>
<xs:element minOccurs="0" name="ontologyIdentifier" type="tns:versionedIdentifier"/>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
</xs:sequence>
</xs:complexType>
-
```

```
<xs:complexType name="conceptFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="definition" type="xs:string"/>
<xs:element minOccurs="0" name="description" type="xs:string"/>
<xs:element name="exactMatch" type="xs:boolean"/>
<xs:element minOccurs="0" name="name" type="xs:string"/>
<xs:element minOccurs="0" name="rootSubsumptionConcept" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="subsettingStatement" type="tns:subsettingStatement"/>
<xs:element name="transitive" type="xs:boolean"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="ontologyElementIdentifier">
-
<xs:sequence>
<xs:element minOccurs="0" name="code" type="xs:string"/>
<xs:element minOccurs="0" name="id" type="xs:string"/>
<xs:element minOccurs="0" name="name" type="xs:string"/>
<xs:element minOccurs="0" name="versionId" type="tns:versionId"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="versionId">
-
<xs:sequence>
<xs:element minOccurs="0" name="currentVersion" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="subsettingStatement">
-
<xs:sequence>
<xs:element minOccurs="0" name="queryType" type="xs:string"/>
<xs:element maxOccurs="unbounded" minOccurs="0" name="subsettingElementList" nillable="true" type="tns:subsettingElement"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="subsettingElement">
-
<xs:sequence>
<xs:element minOccurs="0" name="booleanQueryType" type="xs:string"/>
<xs:element minOccurs="0" name="conceptLeftSide" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="conceptRightSide" type="tns:ontologyElementIdentifier"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="versionedIdentifier">
-
<xs:sequence>
<xs:element minOccurs="0" name="id" type="xs:string"/>
<xs:element minOccurs="0" name="versionId" type="tns:versionId"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="queryControl">
-
<xs:sequence>
<xs:element name="maximumResultSetSize" type="xs:int"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getFlatListOntologyConceptsResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputFlatOntologyConcepts"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="outputFlatOntologyConcepts">
-
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="flatConceptList" nillable="true" type="tns:minimalMetadata"/>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
```

```
<xs:element name="status" type="xs:boolean"/>
<xs:element minOccurs="0" name="statusCondition" type="tns:statusCode"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="minimalMetadata">
-
<xs:sequence>
<xs:element minOccurs="0" name="description" type="xs:string"/>
<xs:element minOccurs="0" name="id" type="xs:string"/>
<xs:element minOccurs="0" name="name" type="xs:string"/>
<xs:element minOccurs="0" name="versionId" type="tns:versionId"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="statusCode">
-
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="error" nillable="true" type="xs:string"/>
<xs:element maxOccurs="unbounded" minOccurs="0" name="info" nillable="true" type="xs:string"/>
<xs:element maxOccurs="unbounded" minOccurs="0" name="statusCodes" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="testService">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="testServiceResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getSemanticListOntologyConcepts">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputWithConceptFilter"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getSemanticListOntologyConceptsResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputSemanticListOntologyConcepts"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="outputSemanticListOntologyConcepts">
-
<xs:sequence>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
<xs:element minOccurs="0" name="rdfXMLGraph" type="xs:string"/>
<xs:element name="status" type="xs:boolean"/>
<xs:element minOccurs="0" name="statusCondition" type="tns:statusCode"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getConceptDetails">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputWithConceptIdentifier"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputWithConceptIdentifier">
-
<xs:sequence>
<xs:element minOccurs="0" name="conceptIdentifier" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="ontologyIdentifier" type="tns:versionedIdentifier"/>
```

```
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getConceptDetailsResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputWithConcept"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="outputWithConcept">
-
<xs:sequence>
<xs:element minOccurs="0" name="conceptMetadata" type="tns:concept"/>
<xs:element name="status" type="xs:boolean"/>
<xs:element minOccurs="0" name="statusCondition" type="tns:statusCode"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="concept">
-
<xs:sequence>
<xs:element minOccurs="0" name="characteristic" type="tns:characteristic"/>
<xs:element minOccurs="0" name="conceptIdentifier" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="conceptMetadata" type="tns:conceptMetadata"/>
<xs:element maxOccurs="unbounded" minOccurs="0" name="designation" nillable="true" type="tns:designation"/>
<xs:element minOccurs="0" name="ontologyMembership" type="tns:versionedIdentifier"/>
<xs:element minOccurs="0" name="previousVersionId" type="tns:versionedIdentifier"/>
<xs:element minOccurs="0" name="provenanceDetails" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="characteristic">
-
<xs:sequence>
<xs:element minOccurs="0" name="characteristicType" type="tns:characteristicType"/>
<xs:element minOccurs="0" name="language" type="xs:string"/>
<xs:element minOccurs="0" name="state" type="xs:string"/>
<xs:element name="stateDate" type="xs:int"/>
<xs:element minOccurs="0" name="value" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="characteristicType">
-
<xs:sequence>
<xs:element minOccurs="0" name="commonMetadata" type="tns:metadata"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="metadata">
-
<xs:sequence>
<xs:element minOccurs="0" name="administrativeInfo" type="xs:string"/>
<xs:element minOccurs="0" name="changeNote" type="xs:string"/>
<xs:element name="date" type="xs:int"/>
<xs:element minOccurs="0" name="definition" type="xs:string"/>
<xs:element minOccurs="0" name="description" type="xs:string"/>
<xs:element minOccurs="0" name="editorialNote" type="xs:string"/>
<xs:element name="effectiveDate" type="xs:int"/>
<xs:element minOccurs="0" name="example" type="xs:string"/>
<xs:element minOccurs="0" name="historyNote" type="xs:string"/>
<xs:element minOccurs="0" name="id" type="xs:string"/>
<xs:element minOccurs="0" name="name" type="xs:string"/>
<xs:element minOccurs="0" name="note" type="xs:string"/>
<xs:element minOccurs="0" name="oid" type="xs:string"/>
<xs:element minOccurs="0" name="scopeNote" type="xs:string"/>
<xs:element minOccurs="0" name="state" type="xs:string"/>
<xs:element name="stateDate" type="xs:int"/>
<xs:element minOccurs="0" name="versionId" type="tns:versionId"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="conceptMetadata">
```

```
-
<xs:sequence>
<xs:element minOccurs="0" name="administrativeInfo" type="xs:string"/>
<xs:element minOccurs="0" name="changeNote" type="xs:string"/>
<xs:element name="date" type="xs:int"/>
<xs:element minOccurs="0" name="definition" type="xs:string"/>
<xs:element minOccurs="0" name="editorialNote" type="xs:string"/>
<xs:element name="effectiveDate" type="xs:int"/>
<xs:element minOccurs="0" name="example" type="xs:string"/>
<xs:element minOccurs="0" name="historyNote" type="xs:string"/>
<xs:element minOccurs="0" name="note" type="xs:string"/>
<xs:element minOccurs="0" name="oid" type="xs:string"/>
<xs:element minOccurs="0" name="scopeNote" type="xs:string"/>
<xs:element minOccurs="0" name="state" type="xs:string"/>
<xs:element name="stateDate" type="xs:int"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="designation">
-
<xs:sequence>
<xs:element minOccurs="0" name="commonMetadata" type="tns:commonMetadataDesignation"/>
<xs:element minOccurs="0" name="designationType" type="xs:string"/>
<xs:element minOccurs="0" name="format" type="xs:string"/>
<xs:element minOccurs="0" name="language" type="xs:string"/>
<xs:element minOccurs="0" name="preferredUsageType" type="xs:string"/>
<xs:element minOccurs="0" name="preferrednessLevel" type="xs:string"/>
<xs:element name="rendering" type="xs:byte"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="commonMetadataDesignation">
-
<xs:sequence>
<xs:element minOccurs="0" name="administrativeInfo" type="xs:string"/>
<xs:element name="effectiveDate" type="xs:int"/>
<xs:element minOccurs="0" name="id" type="xs:string"/>
<xs:element minOccurs="0" name="name" type="xs:string"/>
<xs:element name="releaseDate" type="xs:int"/>
<xs:element name="stateDate" type="xs:int"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getOntologyIndividuals">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputWithIndividualFilter"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputWithIndividualFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="filter" type="tns:individualFilter"/>
<xs:element minOccurs="0" name="ontologyIdentifier" type="tns:versionedIdentifier"/>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="individualFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="role" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="type" type="tns:ontologyElementIdentifier"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getOntologyIndividualsResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputFlatOntologyIndividuals"/>
</xs:sequence>
</xs:complexType>
-
```

```
<xs:complexType name="outputFlatOntologyIndividuals">
-
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="individualList" nillable="true" type="tns:minimalMetadata"/>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
<xs:element name="status" type="xs:boolean"/>
<xs:element minOccurs="0" name="statusCondition" type="tns:statusCode"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getSubSuperConcepts">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputListConceptSubSuperConcepts"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputListConceptSubSuperConcepts">
-
<xs:sequence>
<xs:element minOccurs="0" name="conceptIdentifier" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="ontologyIdentifier" type="tns:versionedIdentifier"/>
<xs:element name="subsumer" type="xs:boolean"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="getSubSuperConceptsResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputListConceptSubSuperConcepts"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="outputListConceptSubSuperConcepts">
-
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="flatConceptList" nillable="true" type="tns:minimalMetadata"/>
<xs:element name="status" type="xs:boolean"/>
<xs:element minOccurs="0" name="statusCondition" type="tns:statusCode"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="listPredicatesForArgument">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputWithArgumentFilter"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputWithArgumentFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="filter" type="tns:argumentFilter"/>
<xs:element minOccurs="0" name="ontologyIdentifier" type="tns:versionedIdentifier"/>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="argumentFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="sourceConcept" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="sourceIndividual" type="tns:ontologyElementIdentifier"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="listPredicatesForArgumentResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputFlatOntologyConcepts"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="createValueSet">
```

```
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputCreateValueSet"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputCreateValueSet">
-
<xs:sequence>
<xs:element minOccurs="0" name="ontologyIdentifier" type="tns:versionedIdentifier"/>
<xs:element minOccurs="0" name="valueSetDefinition" type="tns:valueSetDefinition"/>
<xs:element minOccurs="0" name="valueSetMetadata" type="tns:valueSetMetadata"/>
<xs:element minOccurs="0" name="valueSetName" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="valueSetDefinition">
-
<xs:sequence>
<xs:element minOccurs="0" name="rootSubsumptionConcept" type="tns:ontologyElementIdentifier"/>
<xs:element minOccurs="0" name="subsettingStatement" type="tns:subsettingStatement"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="valueSetMetadata">
-
<xs:sequence>
<xs:element minOccurs="0" name="commonMetadata" type="tns:valueSetCommonMetadata"/>
<xs:element minOccurs="0" name="provenanceDetails" type="xs:string"/>
<xs:element minOccurs="0" name="supportedLanguages" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="valueSetCommonMetadata">
-
<xs:sequence>
<xs:element minOccurs="0" name="administrativeInfo" type="xs:string"/>
<xs:element minOccurs="0" name="changeNote" type="xs:string"/>
<xs:element name="date" type="xs:int"/>
<xs:element minOccurs="0" name="definition" type="xs:string"/>
<xs:element minOccurs="0" name="description" type="xs:string"/>
<xs:element minOccurs="0" name="editorialNote" type="xs:string"/>
<xs:element name="effectiveDate" type="xs:int"/>
<xs:element minOccurs="0" name="example" type="xs:string"/>
<xs:element minOccurs="0" name="historyNote" type="xs:string"/>
<xs:element minOccurs="0" name="id" type="xs:string"/>
<xs:element minOccurs="0" name="note" type="xs:string"/>
<xs:element minOccurs="0" name="oid" type="xs:string"/>
<xs:element name="releaseDate" type="xs:int"/>
<xs:element minOccurs="0" name="scopeNote" type="xs:string"/>
<xs:element minOccurs="0" name="state" type="xs:string"/>
<xs:element name="stateDate" type="xs:int"/>
<xs:element minOccurs="0" name="versionId" type="tns:versionId"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="createValueSetResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputCreateValueSet"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="outputCreateValueSet">
-
<xs:sequence>
<xs:element name="status" type="xs:boolean"/>
<xs:element minOccurs="0" name="statusCondition" type="tns:statusCode"/>
<xs:element minOccurs="0" name="valueSetId" type="tns:versionedIdentifier"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="listValueSets">
-
```

```
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputWithValueSetFilter"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputWithValueSetFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="ontologyIdentifier" type="tns:versionedIdentifier"/>
<xs:element minOccurs="0" name="valueSetFilter" type="tns:valueSetFilter"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="valueSetFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="dateRange" type="tns:dateRange"/>
<xs:element minOccurs="0" name="description" type="xs:string"/>
<xs:element minOccurs="0" name="name" type="xs:string"/>
<xs:element name="spanOntology" type="xs:boolean"/>
<xs:element minOccurs="0" name="state" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="dateRange">
-
<xs:sequence>
<xs:element name="endDate" type="xs:long"/>
<xs:element name="initDate" type="xs:long"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="listValueSetsResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputFlatOntologyConcepts"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="listValueSetElements">
-
<xs:sequence>
<xs:element minOccurs="0" name="arg0" type="tns:inputWithValueSetIdentifier"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="inputWithValueSetIdentifier">
-
<xs:sequence>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
<xs:element minOccurs="0" name="valueSetConceptFilter" type="tns:valueSetConceptFilter"/>
<xs:element minOccurs="0" name="valueSetIdentifier" type="tns:versionedIdentifier"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="valueSetConceptFilter">
-
<xs:sequence>
<xs:element minOccurs="0" name="conceptDefinition" type="xs:string"/>
<xs:element minOccurs="0" name="conceptDescription" type="xs:string"/>
<xs:element minOccurs="0" name="conceptName" type="xs:string"/>
<xs:element minOccurs="0" name="effectiveDate" type="tns:dateRange"/>
<xs:element minOccurs="0" name="state" type="xs:string"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="listValueSetElementsResponse">
-
<xs:sequence>
<xs:element minOccurs="0" name="return" type="tns:outputListValueSetElements"/>
</xs:sequence>
</xs:complexType>
-
<xs:complexType name="outputListValueSetElements">
```

```
-
<xs:sequence>
<xs:element minOccurs="0" name="queryControl" type="tns:queryControl"/>
<xs:element name="status" type="xs:boolean"/>
<xs:element minOccurs="0" name="statusCondition" type="tns:statusCode"/>
<xs:element maxOccurs="unbounded" minOccurs="0" name="valueSetResolution" nillable="true" type="tns:minimalMetadata"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
</types>
-
<message name="CTS2Service_getOntologyIndividuals">
<part element="tns:getOntologyIndividuals" name="getOntologyIndividuals"/>
</message>
-
<message name="CTS2Service_testService">
<part element="tns:testService" name="testService"/>
</message>
-
<message name="CTS2Service_getOntologyIndividualsResponse">
<part element="tns:getOntologyIndividualsResponse" name="getOntologyIndividualsResponse"/>
</message>
-
<message name="CTS2Service_testServiceResponse">
<part element="tns:testServiceResponse" name="testServiceResponse"/>
</message>
-
<message name="CTS2Service_createValueSetResponse">
<part element="tns:createValueSetResponse" name="createValueSetResponse"/>
</message>
-
<message name="CTS2Service_getFlatListOntologyConcepts">
<part element="tns:getFlatListOntologyConcepts" name="getFlatListOntologyConcepts"/>
</message>
-
<message name="CTS2Service_createValueSet">
<part element="tns:createValueSet" name="createValueSet"/>
</message>
-
<message name="CTS2Service_getConceptDetails">
<part element="tns:getConceptDetails" name="getConceptDetails"/>
</message>
-
<message name="CTS2Service_getSubSuperConceptsResponse">
<part element="tns:getSubSuperConceptsResponse" name="getSubSuperConceptsResponse"/>
</message>
-
<message name="CTS2Service_getConceptDetailsResponse">
<part element="tns:getConceptDetailsResponse" name="getConceptDetailsResponse"/>
</message>
-
<message name="CTS2Service_listPredicatesForArgument">
<part element="tns:listPredicatesForArgument" name="listPredicatesForArgument"/>
</message>
-
<message name="CTS2Service_listValueSetElementsResponse">
<part element="tns:listValueSetElementsResponse" name="listValueSetElementsResponse"/>
</message>
-
<message name="CTS2Service_listValueSetsResponse">
<part element="tns:listValueSetsResponse" name="listValueSetsResponse"/>
</message>
-
<message name="CTS2Service_listValueSets">
<part element="tns:listValueSets" name="listValueSets"/>
</message>
-
<message name="CTS2Service_getSubSuperConcepts">
<part element="tns:getSubSuperConcepts" name="getSubSuperConcepts"/>
</message>
-
<message name="CTS2Service_getFlatListOntologyConceptsResponse">
<part element="tns:getFlatListOntologyConceptsResponse" name="getFlatListOntologyConceptsResponse"/>
</message>
```

```
-
<message name="CTS2Service_getSemanticListOntologyConceptsResponse">
<part element="tns:getSemanticListOntologyConceptsResponse" name="getSemanticListOntologyConceptsResponse"/>
</message>
-
<message name="CTS2Service_listValueSetElements">
<part element="tns:listValueSetElements" name="listValueSetElements"/>
</message>
-
<message name="CTS2Service_listPredicatesForArgumentResponse">
<part element="tns:listPredicatesForArgumentResponse" name="listPredicatesForArgumentResponse"/>
</message>
-
<message name="CTS2Service_getSemanticListOntologyConcepts">
<part element="tns:getSemanticListOntologyConcepts" name="getSemanticListOntologyConcepts"/>
</message>
-
<portType name="CTS2Service">
-
<operation name="createValueSet" parameterOrder="createValueSet">
<input message="tns:CTS2Service_createValueSet"/>
<output message="tns:CTS2Service_createValueSetResponse"/>
</operation>
-
<operation name="getConceptDetails" parameterOrder="getConceptDetails">
<input message="tns:CTS2Service_getConceptDetails"/>
<output message="tns:CTS2Service_getConceptDetailsResponse"/>
</operation>
-
<operation name="getFlatListOntologyConcepts" parameterOrder="getFlatListOntologyConcepts">
<input message="tns:CTS2Service_getFlatListOntologyConcepts"/>
<output message="tns:CTS2Service_getFlatListOntologyConceptsResponse"/>
</operation>
-
<operation name="getOntologyIndividuals" parameterOrder="getOntologyIndividuals">
<input message="tns:CTS2Service_getOntologyIndividuals"/>
<output message="tns:CTS2Service_getOntologyIndividualsResponse"/>
</operation>
-
<operation name="getSemanticListOntologyConcepts" parameterOrder="getSemanticListOntologyConcepts">
<input message="tns:CTS2Service_getSemanticListOntologyConcepts"/>
<output message="tns:CTS2Service_getSemanticListOntologyConceptsResponse"/>
</operation>
-
<operation name="getSubSuperConcepts" parameterOrder="getSubSuperConcepts">
<input message="tns:CTS2Service_getSubSuperConcepts"/>
<output message="tns:CTS2Service_getSubSuperConceptsResponse"/>
</operation>
-
<operation name="listPredicatesForArgument" parameterOrder="listPredicatesForArgument">
<input message="tns:CTS2Service_listPredicatesForArgument"/>
<output message="tns:CTS2Service_listPredicatesForArgumentResponse"/>
</operation>
-
<operation name="listValueSetElements" parameterOrder="listValueSetElements">
<input message="tns:CTS2Service_listValueSetElements"/>
<output message="tns:CTS2Service_listValueSetElementsResponse"/>
</operation>
-
<operation name="listValueSets" parameterOrder="listValueSets">
<input message="tns:CTS2Service_listValueSets"/>
<output message="tns:CTS2Service_listValueSetsResponse"/>
</operation>
-
<operation name="testService" parameterOrder="testService">
<input message="tns:CTS2Service_testService"/>
<output message="tns:CTS2Service_testServiceResponse"/>
</operation>
</portType>
-
<binding name="CTS2ServiceBinding" type="tns:CTS2Service">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
-
<operation name="createValueSet">
```

```
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="getConceptDetails">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="getFlatListOntologyConcepts">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="getOntologyIndividuals">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="getSemanticListOntologyConcepts">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="getSubSuperConcepts">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="listPredicatesForArgument">
<soap:operation soapAction=""/>
-
```

```
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="listValueSetElements">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="listValueSets">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
-
<operation name="testService">
<soap:operation soapAction=""/>
-
<input>
<soap:body use="literal"/>
</input>
-
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
-
<service name="CTS2Service">
-
<port binding="tns:CTS2ServiceBinding" name="CTS2ServiceImplPort">
<soap:address location="http://127.0.0.1:8080/service/CTS2ServiceImpl"/>
</port>
</service>
</definitions>
```

Bibliography

1. Ian Horrocks, Oliver Kutz, and Uli Sattler. The Even More Irresistible SROIQ. In *In Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*. AAAI Press, 2006.

2. J. Landgrebe, A. Honey, and J. Davies. Specifying software engineering artefacts for knowledge representation using the model driven architecture. *under review*, 2010.

3. J. Spivey. *The Z Notation. A Reference Manual*. Prentice-Hall, 2 edition, 1992.

4. Jim Woodcock and Davies Jim. *Using Z*. Prentice-Hall, 1 edition, 1996.

5. Jeff Z. Pan and Ian Horrocks. Web ontology reasoning with datatype groups. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*, pages 47–63. Springer, 2003. ISBN 3-540-20362-1.

6. Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology versioning and change detection on the Web. *Lecture Notes in Computer Science*, 2473:197–??, 2002. ISSN 0302-9743.

7. Natalya Fridman Noy and Mark A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004. URL `http://doi.ieeecomputersociety.org/10.1109/MIS.2004.33`.