HL7 Version 3 Standard: Common Terminology Services

HL7 Draft Standard for Trial Use

DSTU Release 2

October 2009

Publication of this draft standard for trial use and comment has been approved by Health Level Seven, Inc. (HL7). Distribution of this draft standard for comment shall not continue beyond **twelve (12)** months from the date of publication. It is expected that following this 12 month period, this draft standard, revised as necessary, will be submitted to a normative ballot in preparation for approval by ANSI as an American National Standard. This draft standard is not an accredited American National Standard. Suggestions for revision should be submitted at http://www.hl7.org/dstucomments/index.cfm.

IMPORTANT NOTES:

A. If you are the individual that downloaded or ordered this HL7 Standard, specification or other work (in each and every instance "Material"), the following describes the permitted uses of the Material.

B. If you are NOT such individual, you are not authorized to make any use of the Material. To obtain an authorized copy of this Material, please visit http://www.hl7.org/implement/standards/index.cfm.

C. If you are not an HL7 Organizational Member, the following are your permitted uses of this Material:

1. Read and Copy License Only. HL7 hereby grants you the right, without charge, to download and copy (for personal use only) this Material for study purposes only. This license grant does not include the right to sublicense or modify the Material, or to implement the Material, either in whole in part, in any product or service.

Please see http://www.hl7.org/legal/ippolicy.cfm for the full license terms governing the Material.

D. If you are an HL7 Organizational Member, the following are your permitted uses of this Material.

1. Implementation License Terms.

1.1 Definitions. As used in this Agreement, the following terms shall have the following definitions:

"Compliant Product" is a product or service that implements Material that is an HL7 Specification in whole or in part.

"End User" is a company, entity or individual that is the ultimate purchaser or licensee from Licensee of a Compliant Product.

1.2 License. In consideration of becoming an Organizational member of HL7 and continuing to pay the appropriate HL7 Organizational membership fees in full, HL7 hereby grants to you without additional charge, on a perpetual (except as provided for in the full license terms governing the Material), non-exclusive and worldwide basis, the right to (a) download, copy (for internal purposes only) and share this Material with your employees and consultants for study purposes, and (b) utilize the Material for the purpose of developing, making, having made, using, marketing, importing, offering to sell or license, and selling or licensing, and to otherwise distribute, Compliant Products, in all cases subject to the conditions set forth in this Agreement and any relevant patent and other intellectual property rights of third parties (which may include members of HL7). No other license, sublicense, or other rights of any kind are granted under this Agreement.

Please see http://www.hl7.org/legal/ippolicy.cfm for the full license terms governing the Material.

Service Functional Model Specification
Common Terminology Services Release 2 (CTS 2)

Project Leads	Russell Hamm (Apelon, Inc.) Craig Stancl (Mayo Clinic/Foundation)
Principal Contributors	Kristi Eckerson (Emory University) Davera Gabriel (University of California, Davis) Russell Hamm (Apelon, Inc.) Alan Honey (APHoney Systems, II4SM) Dr. Jobst Landgrebe (II4SM) Lloyd McKenzie (Lloyd McKenzie and Associates) Grahame Grieve (Kestral Computing)
Contributors	John Carter (Apelon, Inc.) Senthil K. Nachimuthu (3M Health Information Systems, Inc.) Harold Solbrig (Mayo Clinic/Foundation)
HL7 Vocabulary Co-Chairs	Heather Grain (Latrobe University) Russell Hamm (Apelon, Inc.) Ted Klein (Klein Consulting) Beverly Knight (Canada Health Infoway) Jobst Landgrebe (II4SM)

Preface

Notes to Readers

This document is the Service Functional Model Draft Standard For Trial Use (DSTU) for the Common Terminology Services 2 specification, which is specified under the Service Development Framework process under the auspices of the Healthcare Services Specification Project (HSSP). Further context is given in the overview section below, but one key point to note is that the SFM provides a Service **Interface** specification, NOT the specification of a Service implementation. This is a critical distinction in terms of Service Oriented Architecture. There could be many different ways of implementing all or part of the functionality to support the behavior described in this specification.

Publication of this draft standard for trial use and comment has been approved by Health Level Seven, Inc. (HL7). This draft standard is not an accredited American National Standard. The comment period for use of this draft standard shall end 12 months from the date of publication. Suggestions for revision should be submitted at: http://www.hl7.org/dstucomments/index.cfm .

Following this 12 month evaluation period, this draft standard, revised as necessary, will be submitted to a normative ballot in preparation for approval by ANSI as an American National Standard.

Implementations of this draft standard shall be viable throughout the normative ballot process and for up to six months after publication of the relevant normative standard.

For the purposes of this specification, the terms *vocabulary, terminology,* and *code system* are used interchangeably.

Even in terminology circles, some terms such as Codeset are overloaded; and we have found subtle differences between very similar (and similarly named) concepts defined in existing standards. In the preparation of this specification, we found the need to specifically define the terms (such as Value Set, Coding System, etc.) used. Please see Appendix B for both definition of these terms and in some cases a discussion of some of the subtle differences encountered.

Changes from Previous Release

This version of the document includes several changes:

- updated conceptual model of terminology
- more generic Detailed Functional Models
- parameters for the Detailed Functional Model have been made consistent

Acknowledgments

This document is the result of the collaboration of many individuals and organizations. The terminology and standards community - all involved in the numerous meetings and teleconferences are to be thanked for their contributions and support. In addition to the listed authors, we would like to thank all the individuals and organizations who have dedicated effort and resource into furthering the development of this specification.

NOTE: Sections of this document in blue indicate text that is consistent across HSSP specifications.

Table of Contents

1 Overview

1.1 Introduction and Scope

The Service Specification Development Framework Methodology is the methodology followed to define HSSP specifications. The methodology sets out an overall process, and also defines the responsibilities of the Service Functional Model (SFM). Section 2 sets out the business context for this particular specification, but firstly it is important to understand the overall context within which this specification is written, i.e. its purpose from a methodology standpoint.

1.1.1 HL7-OMG Healthcare Services Specification Project (HSSP)

The Healthcare Services Specification Project (HSSP) [http://hssp.wikispaces.com] is a joint endeavor between Health Level Seven (HL7) [http://www.hl7.org] and the Object Management Group (OMG) [http://www.omg.org]. The HSSP was chartered at the January 2005 HL7 meeting under the Electronic Health Records Technical Committee, and the project was subsequently validated by the Board of Directors of both organizations.

The HSSP has several objectives. These objectives include the following:

- To stimulate the adoption and use of standardized “plug-and-play” services by healthcare software product vendors
- To facilitate the development of a set of implementable interface standards supporting agreed-upon services specifications to form the basis for provider purchasing and procurement decisions.
- To complement and not conflict with existing HL7 work products and activities, leveraging content and lessons learned from elsewhere within the organization.

Within the process, HL7 has primary responsibility for (1) identifying and prioritizing services as candidates for standardization; (2) specifying the functional requirements and conformance criteria for these services in the form of Service Functional Model (SFM) specifications such as this document; and (3) adopting these SFMs as balloted HL7 standards. These activities are coordinated by the HL7 Services Oriented Architecture SIG in collaboration with other HL7 committees, which currently include the Vocabulary TC and the Clinical Decision Support TC.

Based on the HL7 SFMs, OMG will develop “Requests for Proposals” (RFPs) that are the basis of the OMG standardization process. This process allows vendors and other submitters to propose solutions that satisfy the mandatory and optional requirements expressed in the RFP while leaving design flexibility to the submitters and implementation flexibility to the users of the standard. The result of this collaboration is an RFP Submission, which will be referred to in the HSSP process as a Service Technical Model (STM). HL7 members, content, and concerns are integral to this process, and will explicitly included in the RFP creation and evaluation process.

It is important to note that the HL7 SFMs specify the *functional* requirements of a service, the OMG RFPs specify the *technical* requirements of a service, and the STM represents the resulting technical model, except

as specified below. In many cases, SFMs describe an overall coherent set of functional capabilities and / or define a minimum set of behaviors necessary to guarantee a minimal level of service in a deployment scenario. These capabilities may be specialized or subdivided from both functional and informational (semantic) perspectives to provide conformance "profiles" that may be used as the basis for the OMG RFP process and/or implemented.

1.1.2 Service Definition Principles

The high level principles regarding service definition that have been adopted by the Services Specification Project are as follows:

- Service Specifications shall be well defined and clearly scoped and with well understood requirements and responsibilities.
- Services should have a unity of purpose (e.g., fulfilling one domain or area) but services themselves may be composable.
- Services will be specified sufficiently to address functional, semantic, and structural interoperability.
- It must be possible to replace one conformant service implementation with another meeting the same service specification while maintaining functionality of the system.

A Service at the SFM level is regarded as a system component; the meaning of the term "(system) component" in this context is consistent with UML usage[1].A component is a modular unit with well-defined interfaces that is replaceable within its environment. A component can always be considered an autonomous unit within a system or subsystem. It has one or more provided and/or required interfaces, and its internals are hidden and inaccessible other than as provided by its interfaces.

Each Service's Functional Model defines the interfaces that the service exposes to its environment, and the service's dependencies on services provided by other components in its environment. Dependencies in the Functional Model relate to services that have or may in future have a Functional Model at a similar level; detail dependencies on low-level utility services should not be included, as that level of design is not in scope for the Functional Model.

The manner in which services and interfaces are deployed, discovered, and so forth is outside the scope of the Functional Model. However, HSSP Functional Models may reference content from other areas of HSSP work that deals with architecture, deployment, naming and so forth. Except where explicitly specified, these references are to be considered informative only. All other interactions within the scope of the scenarios identified above are in the scope of the Functional Model.

Reference may be made to other specifications for interface descriptions, for example where an interface is governed by an existing standard.

1.2 Overall disclaimers

- Examples are illustrative and not normative unless otherwise specified

- The scope of information content of HSSP service specifications is not limited to HL7 content models. At a minimum, however, specifications should provide a semantic profile as part of its conformance profile to provide support for HL7 content models where applicable.

1.3 Context of this SFM within HSSP Roadmap

As described above, the purpose of an HL7 SFM is to identify and document the functional requirements of services deemed important to healthcare. Accordingly, the CTS 2 service provides a critical component within the larger context of service specifications in that it defines both the expected behaviors of a terminology service and a standardized method of accessing terminology content. This consistent approach to terminology interaction will benefit other business context services by providing a level of terminology interoperability that currently only exists in a limited form.

Once adopted as an HL7 standard, it is anticipated that the CTS 2 service will serve as the basis for one or more OMG technical specifications. It is expected that CTS 2 will effectively leverage other HSSP specifications to enhance overall functionality in integration environments. In particular, the CTS 2 service is expected to interact with one or more infrastructure services as outlined below.

At a minimum, it is expected that CTS 2 will make use of an Entity Identification Service, which in turn references a set of Security Services. CTS 2 itself will make use of the Security Services to implement its own functional profile restrictions. Additionally, services such as a Decision Support Service, Clinical Research Functional Query, and Resource Locate and Update Service may find the use of the CTS 2 service a key resource in improving content disambiguation.

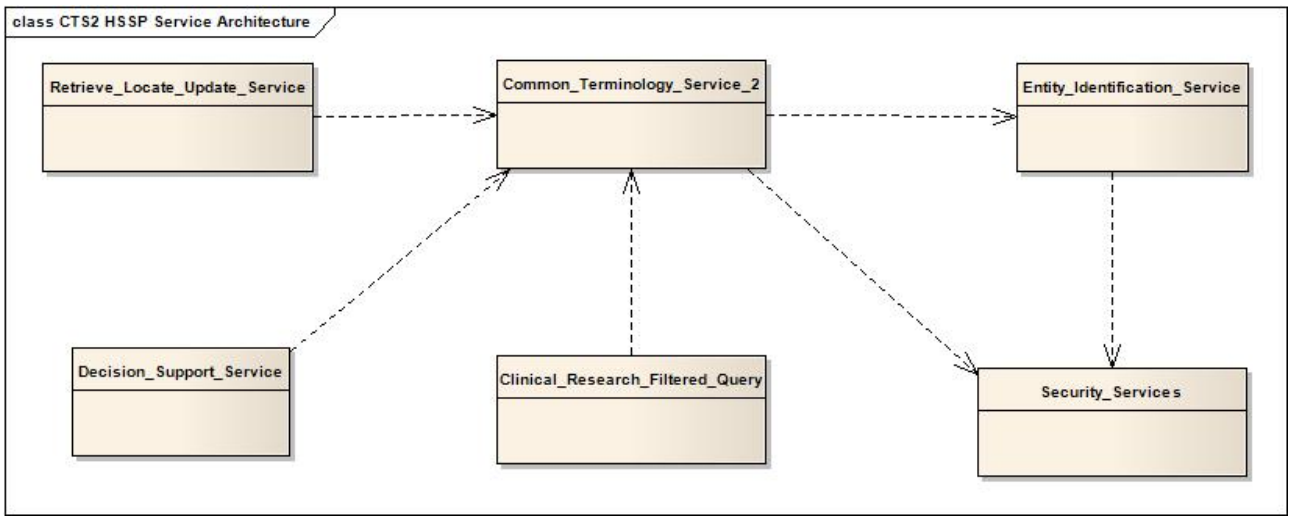

Figure 1 - HSSP Service Architecture

This specification will provide an important foundation component for many healthcare interoperability scenarios, both within and across organizations. Although in many business scenarios CTS 2 may be used in

conjunction with other services, it has been specified to provide stand alone capabilities when referenced solely for terminology access and management purposes.

2

Service Overview and Business case

2.1 Service Overview

2.1.1 CTS 2 Service Description and Purpose

The goal of the Common Terminology Services 2 (CTS 2) specification stack is to provide a standardized interface for the usage and management of terminologies. Terminologies provide the atomic building blocks of shared semantics, concepts. In a shared semantics environment, CTS2 provides a modular, common and universally deployable set of behaviors which can be used to deal with a set of terminologies chosen by the users of the service in their deployment environment. The service will contribute to interoperability by supporting an easy access to the foundational elements of shared semantics. It will also foster the authoring of high-quality terminologies via its authoring profile. This goal is realized via the expansion of the original functionality outlined in HL7's Common Terminology Service (CTS) Specification. CTS 2 defines the functional requirements of a set of service interfaces to allow the representation, access, and maintenance of terminology content either locally, or across a federation of terminology service nodes.

The CTS 2 specification strives to expand on the original functionality outlined in HL7's Common Terminology Service specification, specifically looking to:

1. Establish the minimal common structural model for terminology behavior independent from any specific terminology implementation or interchange model, and how it is related to meta-data (information about data) and data (the information itself)
2. Integrate into CTS 2 the functional coverage outlined in the existing CTS specification.
3. Specify both an information and functional model that addresses the relationships and use of terminology, e.g. how value sets are built and queried, and how terminological information is validated.
4. Specify the interactions between terminology providers and consumers – how terminology users can submit unambiguous requests for corrections and extensions and how revisions to content are identified, distributed and integrated into running systems.
5. Specify how mapping between compatible terminologies and data models is defined, exchanged and revised.
6. Specify how logic-based terminologies can be queried about subsumption and inferred relationships.
7. Engage broad community participation to describe the dimensions of use and purpose for vocabularies and value sets. This aim will attempt to harmonize these efforts in terms of models, use cases, and requirements for creating a functional model for CTS 2.

2.1.2 Why terminology as a service?

As stated above, the aim of this conceptual service specification is to describe the requirements for the representation, access, and maintenance of terminology content. A frequently asked question in this context is whether the problems around terminology should not be resolved using a common data repository. Historically, this approach has been tried, and with some success. However, experience shows that in order to share functionality and content, a hub and spokes model forcing terminology consumers to use a common hub is more intrusive for users and introduces adoption barriers, especially in distributed environments. This service specification has the aim to overcome such barriers by a separation of behavior (software functionality) from content (the deployed terminologies). This way, in an implementation, adopters can chose

the terminology contents they want and use the service functionality to provide the system behavior which is specific to their needs.

2.1.3 Scope

To address the above stated purpose of CTS2, the scope of functionality addresses several broad categories.

Terminology services represent functions necessary to manage, search, and access terminology content. Terminology services provide a consistent specification for accessing and managing terminology content, independent of the terminology content and underlying technology stack. Terminology content represents various resources including lists, value sets, taxonomies, and formal description logic based ontologies. The following thematic areas are considered in scope for CTS 2.

- **Administration**: This is a set of functionality that provides the ability to manage content as part of a terminology service. Administration functions include the ability to load terminologies, export terminologies, as well as the management of notifications. These functions are generally protected and accessible by service administrators with appropriate authorization.

- **Search / Query**: This is a set of functionality that provides the ability to find concepts based on some search criteria. This includes restrictions to specific associations or other attributes of the terminology, including navigation of associations for result sets. This represents the primary utility for using terminology content in a number of application contexts. The CTS 2 Query Profile includes capabilities for searching and querying terminology content as well as representing terminology content in the appropriate formats.

- **Authoring / Maintenance**: This is a set of functionality that provides the ability to create and maintain content. From a terminology service perspective, this would include the appropriate APIs to add, change, or delete concepts and associations. This would also include the processing of change events from various terminology providers.

- **Associations**: This is a set of functionality that provides the ability to map concepts and the concept's associated attributes from a source terminology to a concept in a target terminology, or create relationships between concepts within a single code system.

CTS 2 is intended to allow the look up and management of a wide variety of terminology components, including, but not limited to, Concepts, Associations, and Value Sets. This includes the ability to resolve content bound to a specific Concept Domain and/or *Jurisdictional Domain* (Binding Realm). At the functional level, the service interface will explicitly allow the query, definition, publication, and modification of the different terminology components that are required of terminologies and terminology services.

Conformance profiles are defined within this specification, and are intended to focus specific implementations of CTS 2 to address a specific class of functionality and pre-define minimum trait sets for each specified functional class. This will also allow for performance optimizations to be defined for terminology searches and queries (which are implementation considerations which will be considered in the technical specification

arising from the OMG RFP process). The scope of this functional specification covers support for multiple terminology sources and a federated terminology environment.

2.2 The reason why the service is necessary

The original HL7 CTS specification deliberately avoided issues related to terminology distribution, versioning and authoring; focused on static value sets and didn't fully address the definition or resolution of value sets that define post-coordinated expressions – issues that are now in scope.

Adopting organizations have recognized the existing HL7 CTS standard serves an important role in defining the common functional characteristics that a terminology service (either internal or external) must be able to provide. However, these organizations are also realizing that CTS fails to address many of the maintenance, versioning and representation requirements necessary for a truly interoperable terminology service.

In addition, CTS was deliberately silent on how the vocabulary content should be structured. However, vocabularies vary widely in their underlying metamodels and structures from simple lists, hierarchies, multi-axial systems to description logic. This silence resulted in CTS implementations that are inconsistent in their representations of vocabularies. CTS2 includes a conceptual information model that is conceived as a superset of the metamodels of the supported terminologies.

As a commonly accepted standard for terminology services, CTS 2 will enhance the capabilities of the original CTS specification for sub-setting and mapping, and extend the specification into domains such as terminology distribution, authoring, and versioning. Standardizing the functionality at this level will allow applications using terminology services to build on a common infrastructure, and improve interoperability at the terminology layer across applications.

CTS2 provides the terminology community with a defined set of standards interfaces that can be used to evaluate terminology source structure, terminology source content, and terminology tools.

2.3 Structure of the CTS 2 Service

In order to provide for the maximum implementation flexibility, this functional model defines several enumerated functional profiles for CTS 2. These profiles serve to subset and focus the functionality of a CTS 2 implementation to accomplish a targeted set of tasks. The functional profiles include:

- **CTS 2 Query Profile** - The CTS 2 Query profile outlines functional coverage necessary for a service to declare itself as being a conformant CTS 2 service. The CTS 2 Query Profile includes capabilities for searching and query terminology content, representing terminology content in the appropriate HL7 Datatypes, and structuring terminology content appropriately when HL7 Datatypes are not available for representing the necessary terminology content being queried (i.e. value sets.)

- **Terminology Administration Profile** - The functional operations necessary for terminology administrators to be able to access and make available terminology content obtained from a Terminology Provider. Terminology Administrators are required to interface with Terminology Provider systems in order to obtain the terminology content, and then load that terminology content on local Terminology Servers.

- **Terminology Authoring Profile** - The functional operations necessary for terminology authors to analyze the existing terminology content, as well as directly edit terminology content.

The above three functional profiles can be implemented against three distinct semantic profiles. The semantic profiles are intended to group together terminologies with similar designs or purposes. The semantic profiles include:

- **HL7 Profile** - The functional coverage and specificity necessary for a service to declare itself as being an HL7 conformant CTS 2 service. The HL7 Profile includes capabilities for searching terminology content, representing terminology content in the appropriate HL7 Datatypes, and being able to interchange terminology content in HL7's Model Interchange Format (MIF).

- **Mature Terminology Profile** - Terminologies in the Mature Terminology Profile make an attempt to conform to many of terminology best practices that are, for example outlined in “Desiderata for Controlled Medical Vocabularies in the Twenty-First Century”, James J. Cimino, MethInfoMed 1998, 37: 394-403 .

- **Developing Terminology Profile** - Terminologies in the Developing Terminology Profile are either developed using ad hoc techniques, or have degraded over time.

Additional details on profiles is outlined in [Section 7]

The degree to which an organization’s interoperability deployment supports a conformance profile is directly related to agreements implemented with a business partner. A single CTS 2 service may respond to different real-world business partners depending on the underlying agreements and needs. For example, an organization may implement a CTS 2 (Authoring) compliant service with a trusted partner (i.e., a Terminology Provider). A separate partner may only be allowed CTS 2 (Minimal) access to the content from that Terminology Provider as dictated by other factors.

Additionally, CTS 2 explicitly makes no distinctions at the functional level regarding semantics of the underlying systems. Instead, it provides for a semantic profile as part of CTS 2 conformance profiles. This allows definition, publication, and discovery of vital semantic artifacts between sharing partners through CTS 2 interfaces without requiring strict, tightly coupled integration. Thus, CTS 2 does not preclude a strategy for semantic interoperability to be realized, though it would likely depend on other factors (for example, a security service and / or an entity identification service). This improves CTS 2 as an interoperability mechanism by delegating the issue of semantic interoperability to any group that defines and maintains CTS 2 profiles, allowing semantic transformations to be performed at the least cost for the most derived value.

2.3.1 **Key concepts of this specification**

The CTS2 service deals with three major concepts, which are: concept, code system and value set. A concept is a unitary mental representation of a real or abstract thing; an atomic unit of thought. A concept is the representation-independent meaning which we associate with objects and their designations. A code system is a set of concepts pertaining to a semantic domain. A minimal code system provides a flat list of at least two concepts and may not differentiate concept and concept designation. Large code systems may contain hundred thousands of concepts, concept definitions, concept symbols (codes or other designations) and concept relationships. A value set is a set of concepts taken from one or more code systems. Value sets are defined to restrict the set of values which are allowed to instantiate a data element.CTS2 provides functionality to systems dealing with these concepts. This specification also comprises the HL7 specific notion of a concept domain. A concept domain is a semantic space which is bound to a class property, i.e. instances of the property have to be chosen from this semantic space (e.g. “diseases of the hematopoetic system”).

Further details on these concepts are to be found in section 2.4.2.

2.4 Implementation Considerations

2.4.1 Interface Interoperability Considerations

CTS 2 is an interface specification, not an implementation specification. As such, it is intended to facilitate the development of an implementable interoperability mechanism for terminology resources. There is nothing inherent in the CTS 2 specification that restricts its use to be within a single organization. To the contrary, CTS 2 is intended to expose a single or multiple terminology sources for use by various applications that may or may not be within the same organization, providing a standardized method for terminology access.

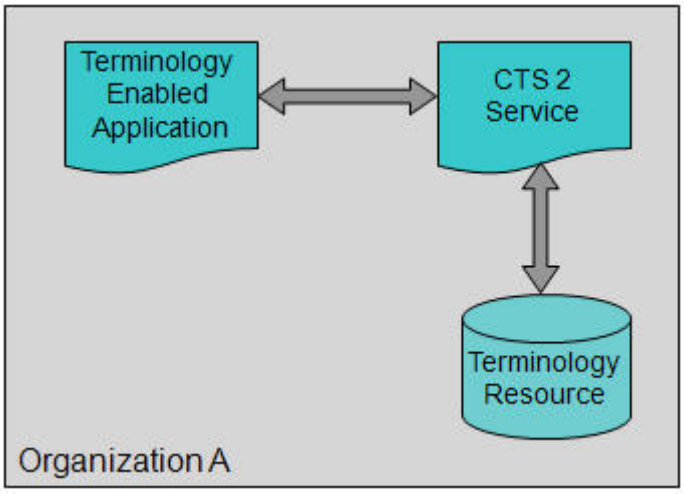

Figure 2 - CTS 2 Service Accessed by a Single Organization

CTS2 provides for terminology interoperability between organizations. While coded concepts from structured terminology can unambiguously identify the concept(s) being communicated, a standard way of structuring and communicating those coded entries is required.

CTS 2 can be used in an inter-organizational setting where each organization maintains its own security and application specific provisions. CTS 2 will enable consistent access to a high availability or international standard terminology resource, made available to subscribers via a CTS 2 interface.

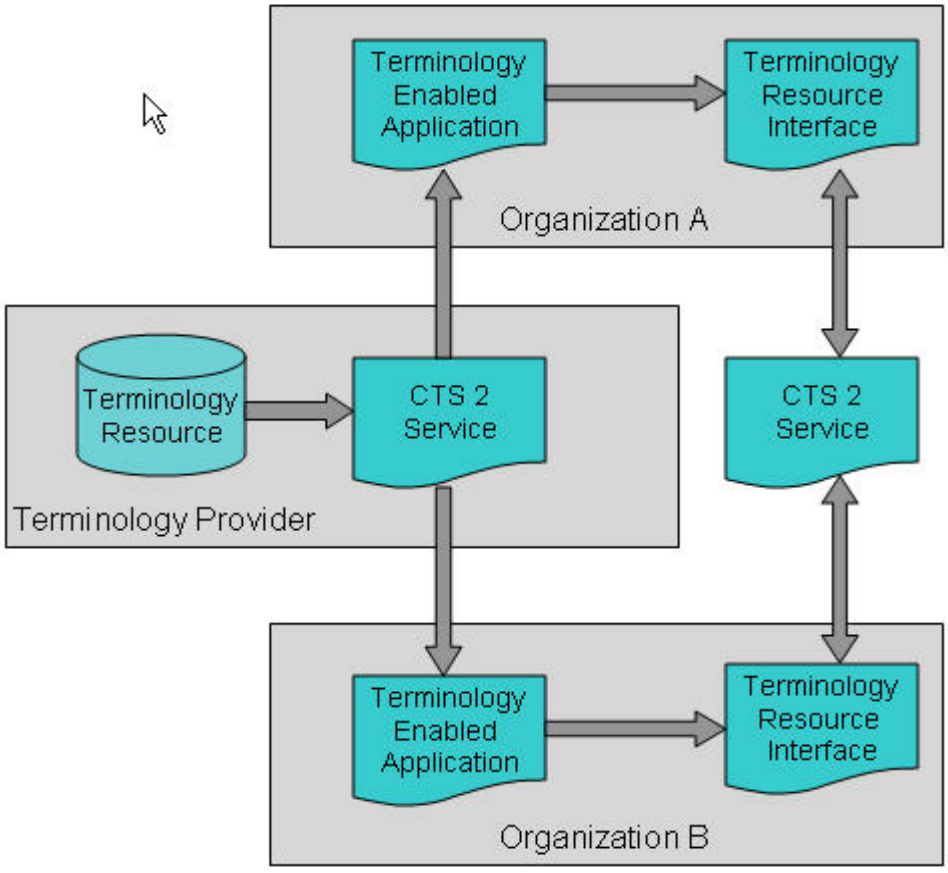

Figure 3 - Multi Organizational access to a CTS 2 Service

Since terminology content is not static, CTS 2 will also provide functionality to maintain and update terminology content. Updates and update requests to terminology sources need to be reviewable and traceable over time. Often, a terminology source provider will want to maintain the “gold standard” or master release of a code system, as to maintain a consistent standard terminology that can be used across multiple organizations and realms. Notwithstanding, users of any given source terminology may wish to extend that terminology for their own use, and may even wish to recommend the addition of those “local” extensions to the terminology provider to be included as part of the release.

CTS2 provides a mechanism to allow for terminology users to extend a given terminology, share those extensions with others, or feed those extensions back to the source provider in a structured format to be reviewed, modified as necessary, and fed into a CTS 2 server as input to update the source terminology with the content contained in the change request. As depicted in Figure 2.4-3, Organization A is applying its own local extensions to a terminology resource being served by a CTS 2 service. In addition to applying its own local extensions, Organization B is feeding some of those local extensions back to the terminology provider as suggestions to be included in the next release of the code system.

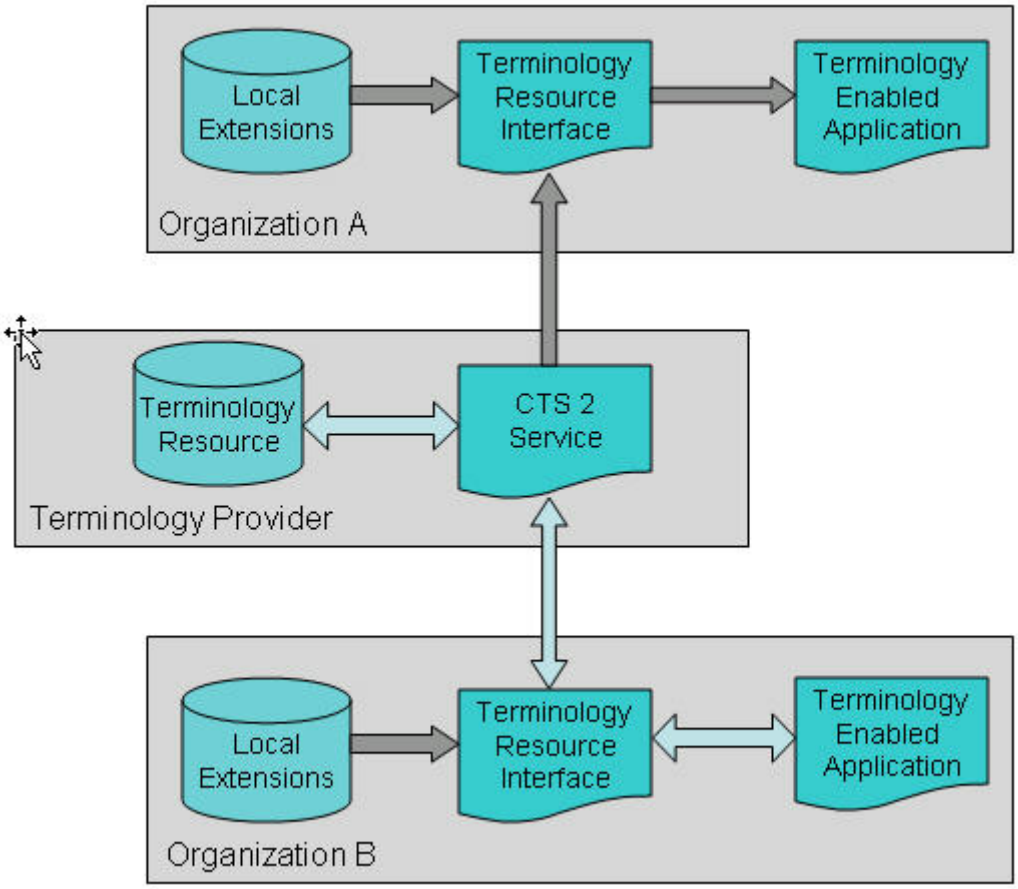

Figure 4 - Multi Organization Access with Write Permissions by One Organization

2.4.2 Terminology Structure Considerations

Controlled terminologies are developed with specific purposes and use cases in mind, and as such are structured very differently. The format of any given terminology may range from a simple flat list of concepts, to more complex poly-hierarchies. The variety of attributes of the entities of code systems vary as well. Even the formats of the identifiers are different, with some concept identifiers being meaningless identifiers, to others which have explicit or implied meaning.

The functional components of CTS 2 must be able to operate on this broad spectrum of terminology sources. At a minimum, CTS 2 must specify a concept based terminology model that is capable of representing most varieties of structured terminologies. The basic requirements of the CTS 2 terminology model are illustrated in the **CTS 2 Conceptual Model** below. This model outlines the various components of terminologies and the usual cardinality between them, but does not dictate particular levels of data normalization or other technical details of implementation. The **CTS 2 Conceptual Model** describes a logical (analysis level) model with the intent that most - if not all - key attributes that affect terminology behavior have been described. It is the intent of this model to support both *well behaved* and more arbitrarily defined code systems. Code Systems, Concepts and Value Sets have specific *version* classes. Additionally *curation* has been supported explicitly by inclusion of a number of classes that permit the versioning of key terminology entities. Specific concerns of audit trail / history have not been explicitly represented.

In a typical implementation, it is likely that more than one status value may be needed on certain classes, to cover different purposes (e.g. curation vs. actual *business* state of the class) This level of detail can be fully resolved in the detailed design specification (Platform Independent Model, PIM), as will the actual values of the various state related attributes. Also a note on the use of "identifiers". In general, identifiers have been included to ensure uniqueness and would be more for system generation, especially on some of the lower level classes.

The purpose of this model is to represent the semantics of the key requirements of CTS 2. It is intended to assist with understanding the expected behaviors and interactions of terminology components supporting a CTS 2 terminology service. It is a conceptual model derived from business requirements, and as such is rather meant to reflect these requirements and not intended to explicitly provide implementation level detail. For example, the presence of identifier or version identifier attributes in classes in the Conceptual Model is not intended to indicate that a unique identifier property will be assigned by the code system author or that identifiers are persisted across versions of a code system. The intent of this model is NOT to serve as an implementation blueprint. Implementation guidance will subsequently be provided in Platform Independent Models (PIMs) and Platform Specific Models (PSMs).

The model must support HL7 vocabulary requirements, however it should not constrain the CTS 2 specification to solely HL7 usage, but promote acceptance in the user and standards community beyond HL7 (e.g. openEHR, IHTSDO). Recommended constraints on the physical implementation based on the requirements are included in Section 10 of this SFM.

NOTE: As is being expressed in the current HL7 SAEAF, the cost for interoperability between implementations based on the SFM alone will be higher than if a PIM level specification is used and again higher than if they both build based on the same PSM.

Also, the Status code sets for the model classes are undefined in this specification. It is explicitly intended that these are left to the technical specification. The values need to be determined by the requirements for maintaining metadata. The primary reason is that this Service is designed for handling of many different kinds of code systems, and these may have different state models.

Technical specifications must define minimum explicit state models for the purposes of interoperability. The basic “default” minimum assumption is that the concept of “active” and “inactive” states is covered, whereby inactive instances would not show up in normal searches. However, these restrictions are seen as realm or organization specific configuration settings. Individual deployments may also use an additional separate curation state (e.g. pending, approved etc.). This is not explicitly covered in this specification since this would be organization specific and often handled by separate workflow tools.

Several classes in the model include an attribute for provenance information, named "provenanceDetails". This should be more concretely defined in Technical Specifications, but should include such information as: author, copyright, generation type, source and source type, approval information, whether it is modifiable, and so on. This may vary to some degree by the class to which it applies, but a consistent structure should be defined where possible.

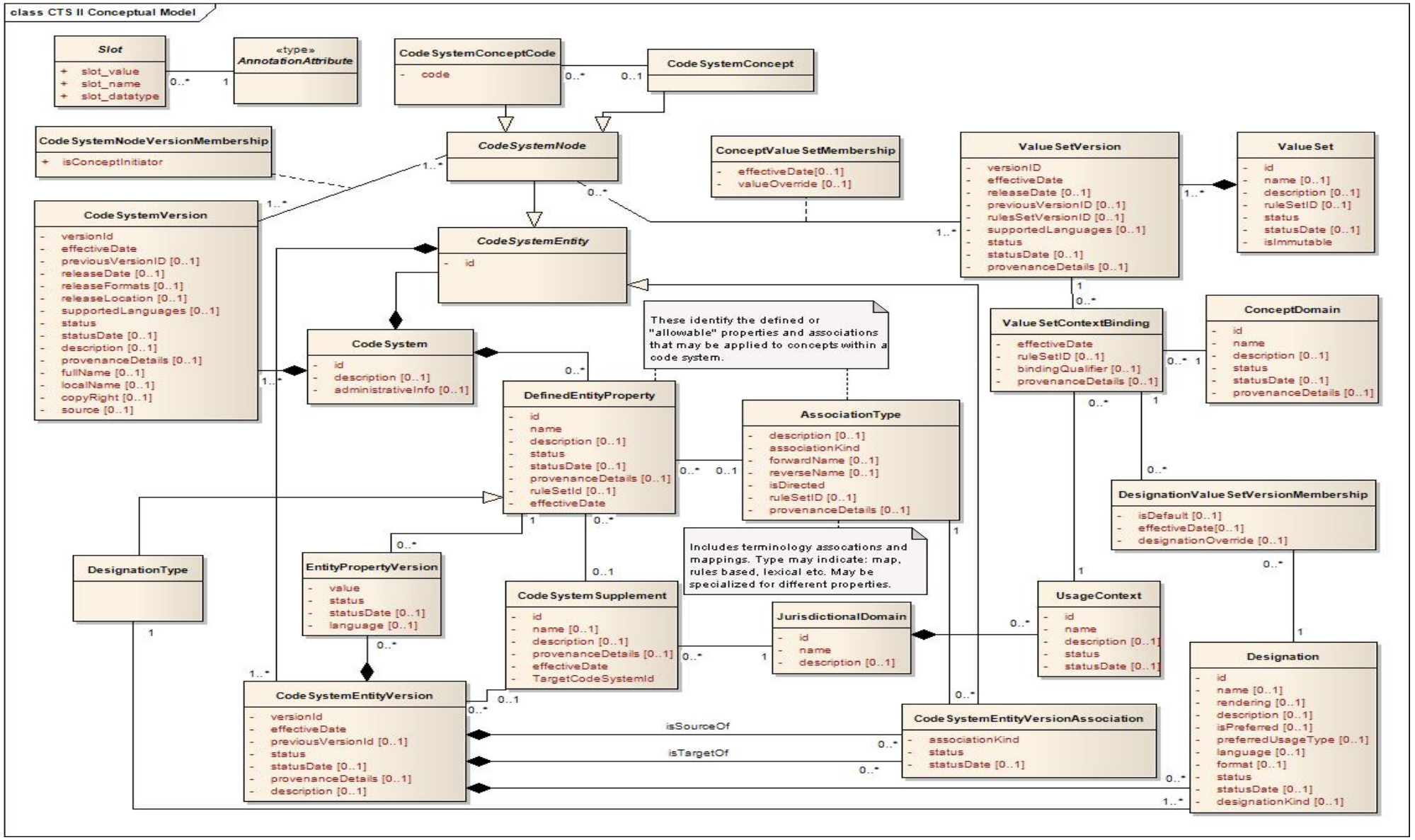

Figure 5 - CTS 2 Conceptual Model

2.4.2.1 CodeSystem

A code system is a resource that makes assertions about a collection of terminological entities. They are often described by a collection of uniquely identifiable concepts with associated, representations, designations, associations, and meanings. Examples of code systems include ICD-9 CM, SNOMED CT, LOINC, and CPT. To meet the requirements of a code system as defined by HL7, a given concept representation must resolve to one and only one meaning within the code system. In the terminology model, a code system is represented by the Code System class.

At a minimum, Code Systems have the following attributes:

- An identifier (id) that uniquely identified the Code System. In HL7, this ID is in the form of an ISO OID.
- A description (description) that describes the Code System. This may include the code system uses and intent.
- Administrative information (administrativeInfo) which is a placeholder to allow for additional information relating to the overall Code System, that is separate from any specific version of the Code System.

NOTE: Throughout this specification, code systems may also be referred to as terminologies, vocabularies or coding schemes.

2.4.2.2 CodeSystemVersion

Code systems are generally not static entities and change over time. A CodeSystemVersion is a static snapshot of a CodeSystem at a given point of time and in force for a period until the subsequent version supersedes it. It and enables identification of the versions of the code system in which any given concept can be found. A CodeSystemVersion is represented by attributes including:

- A version identifier (versionId) that uniquely identifies each version of a Code System
- The start date (effectiveDate) when the version is deemed to be valid for use. This is specific to the usage context in which the service is used (e.g. for a particular terminology server instance), so can be used for historical date based queries to establish when a version was in use.
- Identification of the previous version (previousVersionId) of the code system, which enables tracking of sequencing of versions, and identification of missing versions on a server instance.
- A set of dates (releaseDates) that represent the date when the version of the Code System became available within a particular domain (could be universal, realm specific or for an organization)
- The formats (releaseFormats) in which the version of the Code System is available in.
- The official location (releaseLocation) at which the version of the code system is made available
- The different languages (supportedLanguages) supported by the Code System in this version
- A status (status) to identify the state of the Code System Version (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)
- A description (description) that describes the Code System Version.
- Provenance information (provenanceDetails) relating to the release (version) of the Code System

- A name (fullName) that is the official name of the code system as assigned by the terminology provider at the time of the version being issued.
- A name (localName) to which the Code system is normally referred to for the life of this version.
- Copyright information (copyright) pertaining to the release (version) of the Code System
- An attribute (source) that identifies the authority or source of the code system in this version. i.e. IHTSDO

2.4.2.3 CodeSystemEntity

A Code System Entity is an abstract class that represents either a node (CodeSystemNode) or an association (CodeSystemEntityVersionAssociation) within an overall ontology for a Code System.

This allows for property and version information to be defined for any of the subclasses of code system entity using a consistent structure.

CodeSystemEntity is represented by attributes including:

- An optional unique identifier (id) for the code system entity

Note that most of the "interesting" information and relationships relating to the Code System Entity (and its subclasses) is versioned, and as such is identified at the level of the Code System Entity Version.

2.4.2.4 CodeSystemNode

A Code System Node represents an individual "node" within a Code System, which is either a Code System Concept or a Code System Concept Code. Some code systems apply relationships, properties and designations at the overall "concept" level, while others apply these at the level of each individual code, for which there may be more than one for a single Concept. The CodeSystemNode provides the flexibility to handle each approach. It also allows definition of membership of Value Sets at either level.

In most code systems, the concept and its associated code are 1 to 1, so during technical specification and implementation, ways to simplify this structure may be found, but some systems do use multiple codes for the same concept, so this structure provides for those cases.

NOTE: The functions identified in this SFM are stated in terms of dealing with "Concepts". Unless explicitly stated otherwise, this should be read as "Concept Node" (i.e. Concept and/or Concept Code, depending on how the containing code system is actually set up and populated.)

2.4.2.5 CodeSystemConcept

A Code System Concept defines a unitary mental representation of a real or abstract thing within the context of a specific Code System; an atomic unit of thought. Generic representations of concepts outside of the bounds of a code system are not included in the model. Concepts embody the meaning we attribute to a symbol (a concept code) of an entity. When concept code associations (maps) across code systems are set up, the equivalent codes refer to the same concept, i.e. the same meaning. Concepts should be unique within a given code system, but may have synonyms in terms of both the codes used and in textual representation (as Designations). An example for a medical concept is “femur”, an example for a concept from cell biology is “mitochondrium”. Concepts may be simple or compositional in nature. A compositional concept is one that contains more than one concept concatenated within it – for example “severe hypertension” – a combination of “hypertension” and a qualifier for severity. Each CodeSystem contains a set of at least one Concepts. Terminology best practices dictate that concepts are not deleted from code systems, but are instead deprecated or retired from use, although nothing in the model prevents this.

2.4.2.6 CodeSystemConceptCode

A Code System Concept Code defines a single "code value" that is used to represent (symbolize) a Code System Concept. A Code System Concept Code is the representation of a concept published by the author of a code system as part of the code system. It is intended to be used as the preferred unique identifier for that concept in that code system (though there are code systems which use homonyms). There may be more than one Code for a single concept in a single code system, e.g. for ISO Country Codes which uses two and three character codes for the same countries.

CodeSystemConceptCodes are represented by attributes including:

- A concept code (conceptCode)that according to terminology best practices is unique within the context of the Code System, although some code systems do allow reuse of codes over time.

Note on the difference between a "code" and a "designation". In many code systems the actual code value is also the identifier of the concept and is unique and definitional in nature within the context of a code system. Designations have no such restriction or intent and are merely alternate display formats

2.4.2.7 CodeSystemSupplement

A Code System Supplement can be appended to a code system to add additional concepts, their properties and designations to the code system. The supplement can then be referenced in value set definitions. A code system supplement is related to one JurisdictionalDomain. CodeSytemSupplement has the following attributes:

- An identifier (id) that uniquely identified the CodeSytemSupplement
- A name (name) that the CodeSytemSupplement is normally referred to

- A description (description) that describes the CodeSytemSupplement.
- Provenance information (provenanceDetails) relating to the release (version) of the CodeSytemSupplement

- effectiveDate: The start date (effectiveDate) when the version is deemed to be valid for use.
- TargetCodeSystemId: the identifier of the code system to which the supplement is appended

2.4.2.8 JurisdictionalDomain

A JurisdictionalDomain identifies a country, region, organization or other domain that may define and manage its own code systems or concepts, including localization of a broader code system. Its presence in this model is specifically to allow for localization of certain concept elements. HL7 rules prohibit new codes being added to a code system locally, but does allow for additional concept relationships, concept properties and designations as a supplementation of an existing code system. (However, any organization could use the same model, interfaces etc. to define its own code systems for internal use.) The JurisdictionalDomain class provides the link to classes required to enable the localization to be recorded. The Jurisdictional Domain has the following attributes:

- An identifier (id) that uniquely identified the Jurisdictional Domain
- A name (name) that the Jurisdictional Domain is normally referred to
- A description (description) that describes the Jurisdictional Domain.

NOTE: This encompasses the HL7 concept of "Binding Realm", but is broader (a hyperonym to realm), allowing finer grained control of code system and value set management as appropriate. When defined as HL7 Binding Realms, they would abide by any rules applied to HL7 Realms.

2.4.2.9 DefinedEntityProperty

A defined entity property is a named characteristic of a code system. An example might be “atomic” indicating that there are no composite entities in the code system. Such properties apply to all entities of the code system and can be assigned a value at the code system entity level via the entity property version class. This explicitly defines the allowable properties for any given CodeSystem and not individual changes or versions or what is supported by a specific version of a code set. Each Code System may have zero to many properties. Each Code System allows a set of the properties for the concepts it contains supported in a specific version using the EntityPropertyVersion class, where the actual values of the properties are defined. Defined Entity Properties are represented with attributes including:

- An identifier (id) that uniquely identifies the property
- A name (name) for the property
- A description (description) of the property
- A status (status) to identify the state (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)
- The language (language) in which the property is expressed

- Provenance information (provenanceDetails) relating to the entity property.

2.4.2.10 EntityPropertyVersion

This explicitly defines the supported properties for any given Version of a Code System Entity. Such properties may themselves be concepts or associations of the code system. For example, the concept of Hematocrit with a LOINC code of 11271-4 has a specimen property with the value of "blood" and a method property with the value of "automated count." The specimen and method properties are part of how LOINC assigns the code and, when these properties change, a different LOINC code will be assigned. This implies that properties do not change over time. However, exceptions may be possible when the addition of a new property or change in property value does not change the concept and code (however it would result in a new "CodeSystemEntityVersion" as represented in this model, where the effective date is held). For example, if LOINC decides to add "analyte chemical structure" as a new property via the DefinedEntityProperty class, there may not be a need to change the existing LOINC codes since the new information can apply to all of the LOINC concepts. As a result, each Code System may have its own unique set of properties associated with its Concepts for any specific Code System Version (note that the relationship to the CodeSystemVersion is not in the model explicitly but can be derived, and may be explicit in implementation models).

But the properties may also be "metaproperties" which are not themselves concepts or associations of the terminology, e.g. a product license number for a medicinal product in a terminology that does not have the former as a concept (but the latter).

Entity Property Versions are represented with attributes including:

- A value (value) of the property.
- A status (status) to identify the state (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)

2.4.2.11 CodeSystemEntityVersionAssociation

Associations define the relationships or linkages between concepts or associations. For example, in SNOMED CT, the concept of "pneumonia" has an "is-a" relationship to the concept of "lung consolidation," and "lung consolidation" has an "is-a" relationship to the concept of "disorder of lung." This represents the logical conclusion that "pneumonia" is a "disorder of lung." An example for an association between associations is 'is grandfather of' "is a subtype of" 'is ancestor of'.

Associations are defined as directed semantic relationship triples involving the association and the two concepts. Because the CTS2 model only allows binary associations, they can also be seen as binary predicate relations predicating over two arguments (concepts), however, this more linguistic view is not shown in the model.

It is not mandatory for concepts to have associations to other concepts. However, when associations exist, the cardinality and the explicit declaration of source and target would indicate the directionality that restricts the designation of the association. For example, from the concept relationship (an association between concepts within a single code system) in the above example, we can infer that "pneumonia" is a "disorder of lung," but the inverse concept relationship of "disorder of lung" is-a "pneumonia" cannot be inferred. If we want the inverse concept relationship, it must be explicitly stated, that is, there has to be a specific relation of "disorder

of lung" "is-a" "pneumonia. In the case of Concept Maps (where the source and target concepts are from different code systems) the direction and designation of the relationship have similar restrictions, except in the case where the Concept Map indicates semantic equivalence. The equal association in this case obviates the requirement for interpreting the association direction. An association links a source Concept to a target Concept. The CodeSystemEntityVersionAssociation class identifies the instances of AssociationTypes supported within a specific version of a Code System. The same considerations of variability apply as for the EntityPropertyVersion described above). Code System Entity Version Associations are minimally defined by attributes including:

- A type (associationKind) that describes the nature of the association (e.g. ConceptMap vs. Hierarchic relationship within a code system).
- A status (status) to identify the state (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)

2.4.2.12 AssociationType

In the terminology model, allowable relationship types are represented by the AssociationType class. The class provides information about the types of the association class related to it. It is not a part of a code system, but a separate entity which is used to further characterize associations. An example for an association type is "subsumption association". Association types are minimally defined by attributes including:

- A type (associationKind) that describes the nature of the association (e.g. ConceptMap vs. Hierarchic relationship within a code system) - see also note below.
- A name (forwardName) that represents how the association should be represented when reading from source concept to target concept.
- A name (reverseName) that represents how the association should be represented when reading from target concept to source concept.
- A flag (isDirected) indicating whether the association is one-way or can be interpreted as semantic equivalence.
- A rule set Identifier (ruleSetId) which provides further constraints on the nature of the relationship (see CreateConceptAssociation operation definition for details). The default assumption is that changes to the rules that define artifacts generally cause a version change on the artifact to which they apply, and if the maintenance is done by reference, appropriate curation responsibilities will need to be in place to ensure that the rules do not change without an appropriate change to the artifact version.

A further note on associationKind. This is intended to distinguish between lexical, rules based and explicitly enumerated relationships. Further relationship types can be explicitly defined using the operations that manipulate relationship types explicitly. Further investigation will determine whether we actually need two levels of this attribute to distinguish between these purposes. This will be investigated further during production of the detailed specification (PIM).

2.4.2.13 CodeSystemEntityVersion

Although there are no changes in a code value and its explicit identification of a specific concept (in well behaved code systems), associated information may, e.g. associations, designations and concept properties. The Code System Entity Version provides a means to organize and manage variations to these elements over time. Typically, (but not always) these variations will coincide with new versions of the owning code system. There may be some kinds of changes at the lower "entity" level that do not require release of a new overall Code System Version, so potentially there may be several versions at the CodeSystemEntity level associated with a single CodeSystemVersion. Note that the model does NOT include an explicit association between Code System Version and CodeSystemEntityVersion, however this could be derived using the appropriate effectiveDates. (Technical specification and implementation may choose to make this association explicit).

Code System Entity Versions are represented by attributes including:

- A version identifier (versionId) that uniquely identifies each version of a Code System Entity Version.
- The start date (effectiveDate) when the version is deemed to be valid for use.
- Identification of the previous version (previousVersionId) of the code system entity, which enables tracking of sequencing of versions, and identification of missing versions on a server instance.
- A status to identify the state (mainly for curation)
- A status date to identify the date the status was set to its current value (for curation)
- Provenance information (provenanceDetails) relating to the version of to the Code System Entity Version
- A textual description (description) of the code system entity
-

NOTE: This representation is intended to indicate that a new code system entity version would be created for any change to designations, properties or associations, so individual history is not depicted for each of these items. This can be considered further when the detailed technical design is carried out.

2.4.2.14 CodeSystemNodeVersionMembership

This class represents the use (or membership) of a specific Node (Concept or ConceptCode) within a specific Version of a Code System. This has the following attributes:

- A (derivable but useful) indicator (isConceptInitiator) that identifies which Code System Version initiated the definition of the Concept or Code.

(The absence of date attributes on this class deliberately imposes the restriction that a new concept being added would require a new code system version. In the case of code systems defined and maintained within an organization, they could lift that restriction by adding date information here)

2.4.2.15 Designation

Concept designations are representations of concepts. The designation identifier must uniquely map to a given text string, bitmap, etc. within the context of the containing concept. In some terminologies, every unique text string will have exactly one identifier, meaning that the identifier is unique to the designation. In other terminologies, there may be more than one identifier for a given designation, which means that the same identifier may occur under more than designation. Service software must not assume either model. For example, in SNOMED CT, the concept of “fever” has the fully specified name of “fever (finding),” a preferred name of “fever,” and synonyms of “febrile” and “pyrexia.” These are all designations for the concept of “fever.” In the terminology model, designations are represented by the Designation class. Each Designation is a representation of the Concept and is assigned a unique designation identifier. In most instances, concept designations are human readable forms, but machine readable forms may also be present.

The Designation class is minimally defined by the following attributes:

- A unique identifier (id) for the designation
- A name (name) for the designation
- An optional graphical representation (rendering) that allows for the use of an icon as the designation rather than text.
- A description (description) for the designation
- An optional (isPreferred) attribute that identifies whether an attribute has a type of usage preference.
- A repeatable preferences structure (preferredUsageType) which works in combination with isPreferred to outline they type of usage that a designation is preferred for. This allows identification of whether the designation is preferred for combinations of language, UI, e-prescribing, patient friendliness, etc.
- The language (language) in which the designation is expressed
- A format (format) for the designation
- A status (status) to identify the state (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)
- A type (designationKind) that describes the nature of the designation (e.g. human readable text vs. BLOB, etc.).

2.4.2.16 Designation Type

Concept designations are a specific type of property that may be defined for a CodeSystemEntity. This class allows for identification of the "type" of a designation, i.e. a designation category. The set of designation types contains designation categories like “term”, “picture”, “symbol”.

2.4.2.17 Value Set

A value set represents a uniquely identifiable set of concept codes grouped for a specific purpose. It consists of persisted information that, at a given point in time, may be resolved to a uniquely identifiable set of valid concept representations (codes) called the expansion of the value set which, in addition to the definition, may or may not be persisted depending on system deployment considerations and non functional requirements. Value set complexity may range from a simple flat list of concept codes drawn from a single code system, to an unbounded hierarchical set of possibly post-coordinated expressions drawn from multiple code systems. In

the terminology model, a value set is represented by the ValueSet class. Value Set has the following properties:

- An identifier (id) that uniquely identifies the value set.
- A name (name) for the value set
- A description (description) for the value set.
- An optional expression (ruleSetId) that defines (by value or reference) the algorithm to determine the members for intensionally defined value sets. The default assumption is that changes to the rules that define artifacts generally cause a version change on the artifact to which they apply, and if the maintenance is done by reference, appropriate curation responsibilities will need to be in place to ensure that the rules do not change without an appropriate change to the artifact version.
- A status (status) to identify the state (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)

Notes on use of Value Sets (and Value Set Versions):

1. Representation of "Intensional" Value Sets. These are defined by sets of rules which can be executed to generated a value set expansion. One approach may be to define a superordinate concept, either immediately above or at a higher level, followed by the characteristic(s) that distinguish the concept from other concepts, i.e. using an algorithm (set of rules) that describe how to derive the actual concepts that belong in the Value Set. Other approaches use concept properties or set theory operators on branches of a terminology to define value sets. Since these rules can be arbitrarily complex, they are represented in the model in the form of an identifier of a rules set. This identifier has been included at both the Value Set and Value Set Version levels to allow for change of rules to be applied at whichever level is required. It is intended that the "owner" of the value set would define rules for when changes to a rule would result in a new value set vs. a new version of a value set. By default, Intensional value sets contain the full set of designations defined by the intensional statement. **Examples** : a) codes symbolizing concept lymphozyte - a subtype of the cell type leukocyte with a function in the humoral or cytotoxic immune system, b) codes symbolizing medicinal products - any compound with a marketing authorization number granted by a national competent authority, c) all concept codes symbolizing concepts with properties "cell", "anaplastic", "carcinoma forming".

2. Hierarchic and nested Value Sets. The model includes a "recursive" aggregation relationship on Value Set Version. This allows (as a convenience mechanism) for definition of Value Sets that "include" other Value Sets. This is explicitly restricted to inclusion of complete Value Sets (and not partial ones). Anything more complex would need to be described using the rule sets mentioned above. However this would allow for a Value Set to be defined that consists of a number of other complete Value Sets together with additional concepts intensionally or extensionally defined concepts. In all cases, at any given point in time, the definition of the Value Set Version MUST be able to resolve to an exact set of individual Concepts (together with their Designations where appropriate). This set might be hierarchical or a flat list.

3. Implementers should consider the workflow processes which are necessary to maintain and curate content in a distributed environment. Implementers should discuss how they would implement the workflow management and the coordination across nodes as a part of the implementation process, including how to bind functional components to workflow.

4. Clarification on value set requirements[1]

With regard to value sets, maximally scoped implementations of this service should:

- Be able to specify the contents of a value set by referencing multiple different code systems in scope of a given service instance deployment with different semantics and internal structures (explicit in conceptual class model)
- Be able to build a value set by enumerating codes from code systems (explicit in conceptual class model)
- Be able to define a value set using rules by:
    - specifying a query on a code system
    - pattern matching on the code (for example regex)
    - queries on the properties or relationships defined by a code system if the code system provides appropriate properties (for example, is-a, leaf vs. container, defined attributes, SNOMED qualifiers)
    - by union, intersection or exclusion of two or more other value sets
    - specify rules with regard to how composite expressions can be constructed in the case that code systems define composite expressions

2.4.2.18 ValueSetVersion

A Value Set Version represents a point in time view of a Value Set definition. The Value Set Version identifies the set of concepts that are available in the value set for any specific version of the value set definition. The versioning of the value set reflects the fact that the value set definition may change over time. Note that Value Set Version can be created by aggregation of other value set versions. The expansion of two different versions of one value set definition may (but must not) yield different sets of concept codes.

Note that the "resolution" of a value set version is not explicitly represented in the model. For value sets with intesional definitions (i.e. those defined using rules rather than an explicit listing of concepts like in extensionally defined value sets) the resulting "resolved" value set would *depend on the point in time* in which the resolution takes place because the targets of the rules can change over time.

For "extensional" value sets, where membership is explicitly defined, the membership may be defined in terms of Nodes (Concepts or Codes) and/or Designations. The latter may also be defined with respect to specific Usage Contexts and Concept Domains (see Value Set Context Binding). Where the definition is made in terms of both Concepts/Codes and Designations, processing should ensure that they are compatible. In general, this would only occur where Designations are being defined on an exception basis, e.g. where a "non-preferred" designation is being identified for use in a Value Set for one of the Concepts.

[1] These are not minimum requirements of the implementations for implementations of the spec. (proposed minimum: enumerating codes)

Value set versions may be identified by several different means, such as a serially increasing version number, an effective date or a semantics-free UID.

A Value Set Version has the following attributes:

- An identifier (ValueSetVersion.id) that uniquely identifies the value set version.
- An effective date (ValueSetVersion.effectiveDate) that identifies when the value set version became effective, i.e. is able to be used (bound to from information models).
- A date (ValueSetVersion.releaseDate) when the version of the value set was released (which allows for value set versions to be released in advance of their becoming "effective" or usable.
- An optional order (ValueSetVersion.versionOrder) that identifies the order in which the version should be applied
- An optional expression (ValueSetVersion.ruleSetVersionId) that defines (by value or reference) the algorithm to determine the members for “intensionally defined” value sets
- The different languages (ValueSetVersion.supportedLanguages) supported by the Value Set Version
- A status (ValueSetVersion.status) to identify the state (mainly for curation)
- A status date (ValueSetVersion.statusDate) to identify the date the status was set to its current value (for curation)
- Provenance information (ValueSetVersion.provenanceDetails) relating to the version of to the Value Set

2.4.2.19 ConceptValueSetMembership

This provides the means to link a Value Set Version to the concepts and/or codes which it includes. In “extensionally” defined value sets, this would be created manually, in “intensionally” defined Value Sets, this could optionally be used to record expansion of the value set definition, i.e. the result of applying the rule set at a given point in time (i.e. the date of the Value Set Version where this was required). The absence of date attributes on this class deliberately imposes the restriction that a new concept being added would require a new value set version for extensionally defined value sets. This includes the following attribute:

- A date (effectiveDate) which is required for “intensionally” defined Value Sets where the resulting concepts from applying the algorithm at a specific point in time were being explicitly instantiated and recorded (maybe to meet non functional requirements or because of deployment considerations).
- An override value (valueOverride) which allows for a local code value (specific to the value set) to be used to replace the default one defined in the code system itself.

2.4.2.20 DesignationValueSetVersionMembership

This identifies which Designations for Concepts may be used within a specific Value Set Version for a specific combination of concept domain and usage context. This provides an additional level of flexibility for constraining the use of specific designations in specific circumstances. This includes the following attributes:

- An optional indicator (isDefault) which identifies whether the designation is the default designation representation for the concept in the value set.
- An optional date (effectiveDate) which would only be needed in “intensionally” defined Value Sets where the resulting concept designations from applying the algorithm at a specific point in time were being explicitly instantiated and recorded.
- An override value (designationOverride) which allows for the designation to be overridden with a local value for a specific value set version/context/concept domain.

2.4.2.21 Concept Domain

Concept domains describe a semantic space. They are a named category of like concepts (a semantic type) that will be bound to one or more attributes in a static model. *Concept Domains* exist to specify the intent of the coded element while deferring the association of the element to a specific coded terminology until later in the model development process. Concept domains are intended to allow “late binding” of vocabulary to a specific value set or value set version. At design time, rather than referencing a specific value set, a given attribute (or datatype property) may reference a concept domain to provide an abstract description of “this is the type of stuff that goes here”. The same concept domain may be referenced in multiple designs, effectively saying “the same set of codes – whatever it is – that is used in place A should also be used here”. The Concept Domain has the following attributes:

- An identifier (id) that uniquely identified the Concept Domain.
- A name (name) that the Concept Domain is normally referred to.
- A description (description) that describes the Concept Domain.
- A status (status) to identify the state (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)

2.4.2.22 Usage Context

A Usage Context identifies an environment or set of conditions in which a Value Set may be used. This can be at various levels of specificity, e.g. "Canadian pediatric psychiatry", "psychiatry", "disease codes" and so on. This is used in conjunction with Concept Domain to allow realms to bind to specific Value Sets in specific circumstances. These may be "universal" or owned by a specific Jurisdictional Domain (Realm). (From a modeling perspective, we have classed universal as the highest level Jurisdictional Domain). The Usage Context has the following attributes:

- An identifier (id) that uniquely identified the Usage Context.
- A name (name) that the Usage Context is normally referred to.
- A description (description) that describes the Usage Context.
- A status (status) to identify the state (mainly for curation)
- A status date (statusDate) to identify the date the status was set to its current value (for curation)

2.4.2.23 Value Set Context Binding

This provides a mechanism to bind a concept domain and a particular value set version in a defined usage context. Note that the broader the context, the higher the potential of the binding to support interoperability. At runtime, the operating context is identified and then used together with the concept domain referenced in the specification to resolve to the particular value set used. The Value Set Context Binding has the following attributes:

- A date for evaluating the value set expansion for the binding
- A rule set (ruleSetId) that defines (by value or reference) an algorithm used to constrain and/or determine the designations covered or the conditions in which the binding is applied. The default assumption is that changes to the rules that define artifacts generally cause a version change on the artifact to which they apply, and if the maintenance is done by reference, appropriate curation responsibilities will need to be in place to ensure that the rules do not change without an appropriate change to the artifact version.
- ProvenanceDetails to identify which of the potentially more than one bindings to evaluate.
- A qualifier (bindingQualifier) which indicates whether the binding is "overall", "minimum" or "maximum".

NOTE : The binding is explicitly modeled at the Value Set Version level, in order to enable flexibility. However, in HL7, a dynamic binding is done by relating to a value set (not a value set version). If a timestamp for a binding is provided, the binding is static. If a timestamp for the binding is not provided, the binding is dynamic, and the time point of the function may be used as the timestamp for the binding. At the implementation level, there would be several choices if the intent was to bind to the Value Set independent of Version, which should be addressed in the technical specification.

2.4.2.24 Annotation Attribute

This type class provides a type for an "annotation attribute" within the classes which needs additional annotations beyond basic annotation like the one provided with the attribute “provenanceDetails” present in many classes. This may be used for HL7 semantic profiles (requirements defined at http://wiki.hl7.org/index.php?title=Requirements-Annotations). This class has no attributes in the conceptual model, its attributes are provided using the slot class depending on the needs of the semantic profile a given service instance is supposed to support.

2.4.2.25 Slot

This class allows defining attributes for the annotation attribute class depending on the information modeling/semantics requirements of a given semantic profile using this class (e.g. the HL7 semantic profile). This flexibility is introduced on purpose to support various annotation approaches. Classes “Annotation Attribute” and “Slot” are not normative and require further elaboration for the HL7 profile.

3 Business Scenarios

3.1 Scenario Actors

Actors will use the CTS 2 service for different purposes. These different actors can be generalized into a basic *Terminology User* an Actor that is simply an individual, organization, or application that requires access to terminology content for some purpose. Specializations of the *Terminology User* actor participate in additional operational specific scenarios that are defined by this Service Functional Model to address the Scope that is outlined in section 2.1.2. Actors described in this section are not necessarily human actors, but also include organizations and systems Figure3.1-2 outlines the specializations and composition of the different actors used in this specification. These actors are described below. **NOTE**: The below list of actors is not intended to be exhaustive. Additional roles (such as Terminology Quality Reviewers) can be added to this enumeration; however, analysis should be performed to ensure that the addition of a new Actor results in a corresponding addition of terminology service functionality that is not already present. The translator actor is intended to be a human being or translation software, but automated translation is not deemed in scope of CTS2.

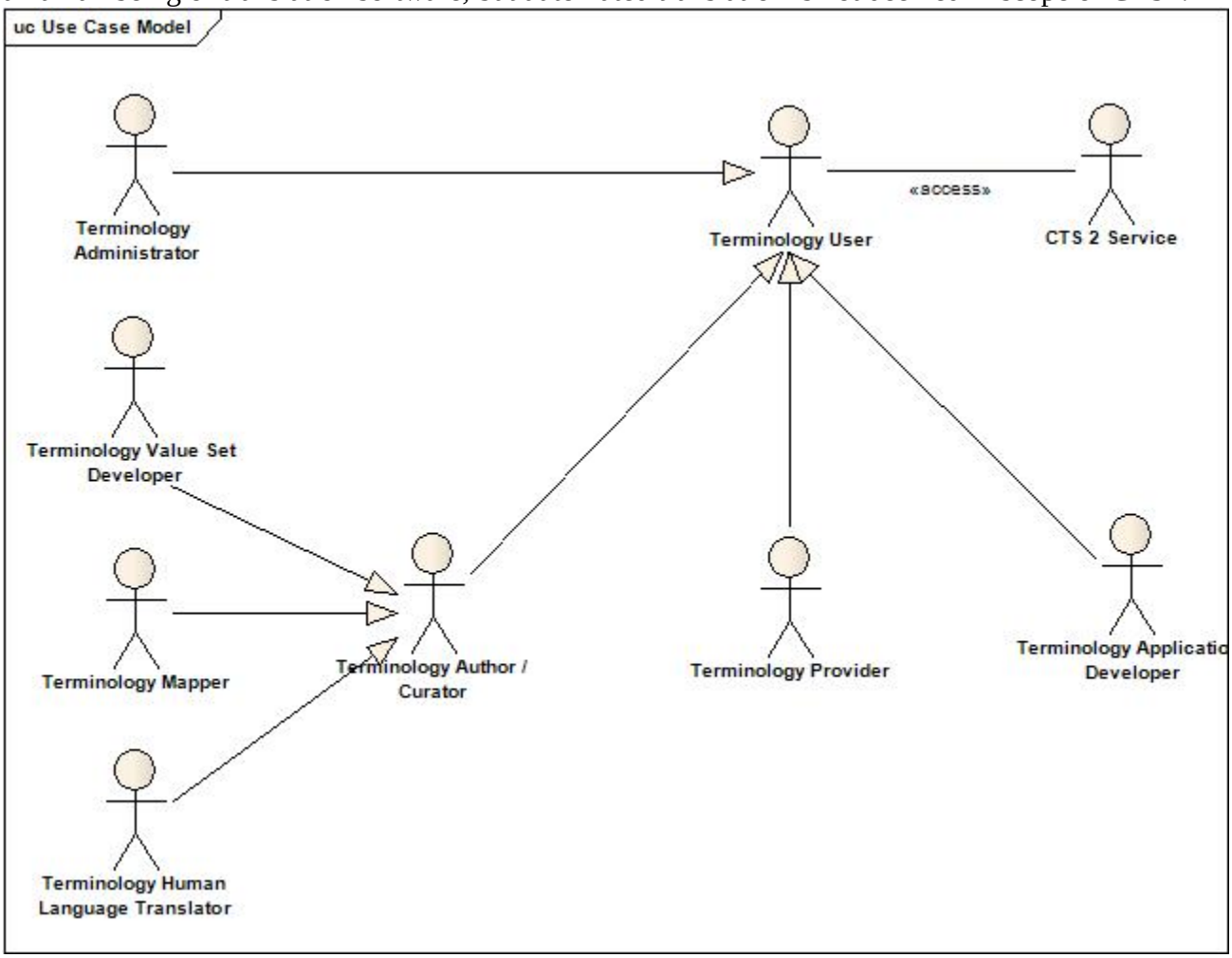

Figure 6 - Scenario Actors

The following actors take a role in the CTS 2 scenarios.

3.1.1 CTS 2 Service

The *CTS* 2 *Service* is a specific implementation of the CTS 2 Terminology Server.

3.1.2 Terminology User

A *Terminology User* is an actor such as a subject matter expert, terminologist or terminology enabled application. *Terminology User* activities include, but are not limited to, querying for specific concept codes and browsing or comparing value sets. Specializations of the *Terminology User* actor follow below.

3.1.3 Terminology Administrator

The *Terminology Administrator* is an actor responsible for ensuring the availability and overall maintenance of the terminology server. This includes, but is not limited to loading content into the terminology server, and making available the required functionality to address the specific conformance profiles implemented by the Terminology Server instance.

3.1.4 Terminology Enabled Application Developer

A *Terminology Enabled Application Developer* is an actor who is responsible for the development of software applications that make explicit use of controlled terminologies.

3.1.5 Terminology Author / Curator

A *Terminology Author / Curator* is an actor who is responsible maintaining terminology content, including but not limited to, the development of new concepts and associations that may be submitted to the *Terminology Provider* or the extension of an existing terminology with local concepts. This may also determine who can validate and perform quality control of terminology content. Terminology Authors / Curators may not necessarily belong to the *Terminology Provider's* organization.

3.1.6 Terminology Human Language Translator

A Terminology Human Language Translator is an actor with domain knowledge who is also familiar with the languages and dialects which they are responsible for translating.

3.1.7 Terminology Mapper

A *Terminology Mapper* is an actor (human or system) that is responsible for creating or maintaining specialized associations, or "mappings" between concepts from different code systems.

3.1.8 Terminology Provider

The *Terminology Provider* is the actor the individuals or organization that is responsible for the development of Terminology Content.

3.1.9 Terminology Value Set Developer

A *Terminology Value Set Developer* is an actor with specific domain knowledge, as well as expertise in controlled terminologies who develop s and maintains domain-or application-specific terminology value sets.

3.2 Primary Scenarios

Primary scenarios are tied to one or more conformance profiles. Note, that as an aid to reading this specification, Actors that are identified in the text are italicized. In addition, when a scenario references another scenario, that referenced scenario is in ***bold italics***.

3.2.1 Administrative Scenarios

The administration scenarios are intended to provide the functional operations necessary for terminology administrators to be able to access and make available terminology content obtained from a Terminology Provider. Terminology Administrators are required to interface with Terminology Provider systems in order to obtain the terminology content, then load that terminology content on local Terminology Servers.

3.2.1.1 Import Content

A *Terminology Administrator* is required to make available terminology content from a *Terminology Provider* available to *Terminology Users* through a *Terminology Server*. This may or may not include the removal of a previously loaded terminology content from the terminology server. To accomplish this, the *Terminology Administrator* may be required to convert the content from the format provided by the *Terminology Provider* to a format that the *Terminology Server* is capable of importing (the conversion being out of scope of this

service). Example of terminology content that may be available for loading into the *Terminology Server* include but is not limited to:

- Source terminologies (complete sources and deltas)
- Value Sets
- Mappings

These content sources may either be new sources, or updated versions of an previously existing content sources.

Associated Functional Models: Import Code System, Import Code System Revision, Import Association Version, Import Value Set Version

3.2.1.2 Export Content

A *Terminology Administrator* is required to export a terminology or terminology value set from the *Terminology Server*. This may require filtering of the exported content. In cases where standard or common terminology interchange formats are available (such as HL7 Vocabulary MIF or LexGrid), conversion of content on the CTS 2 server into said standard would be considered.

Associated Functional Models: Export Code System Content, Export Association

3.2.1.3 Decommission Terminology Content

A *Terminology Administrator* is required to decommission a terminology or terminology version from the terminology service, rendering it unavailable for subsequent access by other service functions.

Associated Functional Models: Change Code System Status

3.2.1.4 Change Content Status

A *Terminology Administrator* is required to activate or inactivate a given terminology, thus changing its availability for access by other terminology service functions.

Associated Functional Models: Change Code System Status

3.2.1.5 Update Notification

A *Terminology User* has a dependency on a specific terminology element that is available to a Terminology Server. The *Terminology User* is interested in knowing when this terminology element is modified in any way, and would like to receive an electronic notification in the event of that change to that terminology element

Associated Functional Models: Register for Notification

3.2.1.6 Update Notification Management

A *Terminology User* is required to update the notification information pertinent to their notification account.

Associated Functional Models: Update Notification Registration, Update Notification Registration Status

3.2.1.7 Content Dependency Notification

A *Terminology Administrator* is required to run a dependency check to compare updated content for a given code system, against the version of that code system currently used by the *Terminology Administrator's* organization. For example, to provide a list of all terminology elements which are somehow affected by upgrading to a newer version of a terminology.

Associated Functional Models: Register for Notification, Update Notification Registration, Update Notification Registration Status

3.2.2 Search / Query Scenarios

The scenarios in this section describe the ability to query code system and value set. These scenarios attempt to outline the information requirements for querying. The detailed function models in section 5.2 call out the distinct functional requirements.

A given CTS 2 implementation will be required to advertise the specific search algorithms that it supports.

In each scenario below, the *Terminology User* may need to specify additional information pertaining to the query. This information may include:

- The ability to determine the status of metadata or contents of a code system, value set as it existed in a specified *version*, where *version* represents a meta-data component used to filter the result set of the query.

NOTE: Details of the available meta-data requirements will be identified as part of the Binding Document and Model harmonization activity.

3.2.2.1 Code System Search / Query

This section outlines Search / Query operations pertaining to Code Systems.

3.2.2.1.1 Retrieve Available Code Systems

A *Terminology User* wants to determine what code systems are available through a specific instance of a Terminology Service. The *Terminology User* is interested in seeing a listing of the available code systems, as well as the details pertaining to each code systems available through a specific Terminology Service instance.

Associated Functional Models: List Code Systems, Return Code System Details

3.2.2.1.2 Retrieve Coded Concepts from Code System

A *Terminology User* wants to browse or query the content of a specific code system. The *Terminology User* is interested in seeing a listing of specific coded concepts, associated attributes, as well as the metadata pertaining to each coded concept that meets some search criteria. For example, after a retrieval of concepts has been performed, the result set could be fed to a terminology browsing GUI

Associated Functional Models: List Code System Concepts, Return Concept Details

3.2.2.1.3 Validate Concept in Code System

A *Terminology User* wants to validate that a given concept exists in a given code system.

Associated Functional Models: List Code System Concepts

3.2.2.1.4 Identify Concept Language Translations

A *Terminology User* wants to determine what (if any) alternate language representations exist for a given Concept.

Associated Functional Models: Return Concept Details

3.2.2.1.5 Retrieve Concept Representations

A *Terminology User* wants to determine what (if any) alternate representations (designations) exist for a given Coded Concept. Examples of alternate representations for a concept may include abbreviations, or synonyms.

Associated Functional Models: Return Concept Details, List Value Set Contents (Expand value set)

3.2.2.1.6 Compare Code System Versions

A *Terminology User* wants to determine what differences exist between different versions or instances of a code system.

Note that the Service calls just return the required Code systems and concept information. Carrying out the side by side comparison would be a client function.

Associated Functional Models: Return Code System Details, List Code System Concepts

3.2.2.2 Value Set and Concept Domain Search / Query

This section outlines Search / Query operations pertaining to Value Sets and Concept Domains.

3.2.2.2.1 Retrieve Available Value Sets

A *Terminology User* wants to determine what value sets are available through a specific instance of a Terminology Service. The *Terminology User* is interested in seeing a listing of the available value sets that match some search criteria, as well as the details pertaining to each value set available through a specific *CTS 2 Service* instance.

Associated Functional Models: List Value Sets, Return Value Set Details

3.2.2.2.2 Retrieve Coded Concepts from Value Set

A *Terminology User* wants to browse or query the content of one or more value sets. The *Terminology User* is interested in seeing a listing of specific coded concepts, as well as the details pertaining to each coded concept in any of the given value sets. For example, the *Terminology User* may want to search for some criteria over a set of value sets.

Associated Functional Models: List Value Set Contents (Expand value set), Return Concept Details

3.2.2.2.3 Retrieve Available Concept Domains

A *Terminology User* wants to determine what Concept Domains and Usage Contexts are available through a specific instance of a Terminology Service. The *Terminology User* is interested in seeing a listing of the available concept domains that match some search criteria.

Associated Functional Models: List Concept Domains, Return Concept Domain Details, List Usage Contexts, Return Usage Context Details

3.2.2.2.4 Retrieve Coded Concepts for Concept Domain via Value Set Membership

A *Terminology User* wants to browse or query the content of one or more value sets and contained concept codes that are related to a specific Concept Domain (and optionally Usage Context) via a value set membership. The *Terminology User* is interested in seeing a listing of specific coded concepts, as well as the details pertaining to each coded concept in any of the given Concept Domains. (Note that this involves finding the value set bindings and then querying the value sets, although the service should hide that from the user to some degree).

Associated Functional Models: List Concept Domain Bindings, Return Concept Details, Return Usage Context Details, List Value Set Contents (Expand value set)

3.2.2.2.5 Validate Coded Concept in Value Set

A *Terminology User* wants to validate that a given concept exists in a given value set. The value set may optionally be bound to a Concept Domain and Usage Context.

Associated Functional Models: Check Concept Value Set Membership, Check Concept to Concept Domain Association

3.2.2.2.6 Compare Value Set Versions

A *Terminology User* wants to determine what differences exist between different versions of a value set.

Value Sets can be defined as either enumerations of concepts (extensional value set), or by expression syntax that defines the content of the Value Set.

In the case of an Enumerated Value Set, the specific Value Set version identifier can be used as a compare point for the two value sets. For Intensionally defined Value Sets, the compare point is the set of concepts that are resolved at the point in time that is specified, or the current time the function is called. The time that may be specified is usually the time used in a specific STATIC binding of interest.

NOTE : Service calls just return the required Value Set information. Carrying out the side by side comparison would be a client function.

Associated Functional Models: Return Value Set Details, List Value Set Contents (Expand value set)

3.2.3 Authoring / Curation Scenarios

This section outlines the requirements of terminology systems that provide the capability of making changes to terminology elements such as code system or value sets. This includes both the direct modification of terminology content for use by individuals responsible for terminology authoring and curation. Such functionality includes:

- adding new concepts into a code system
- adding new relationships into a code system
- extending a code system with local terms
- creating or modifying value sets
- modifying other code system content and attributes
- Capability for validation of authoring requests without commit

In addition to direct modification of terminology content, this section also specifies functionality with the capability of creating structured change requests for consideration by terminology maintainers. This functionality is key in allowing Terminology Providers to solicit feedback pertaining to terminology structure and content from Terminology Users in a controlled and structured manner.

3.2.3.1 Code System Authoring / Curation

This section outlines the business scenarios specific to terminology systems that provide the capability of making changes to code system components which include coded concepts, representations (textual), Associations or Relationships, and value sets.

3.2.3.1.1 Create Code System

A *Terminology Author* is required to create a new Code System to contain a set of new coded concepts. The Code System is created by defining the set of meta-data properties that describe it.

Associated Functional Models: Create Code System

3.2.3.1.2 Maintain Code System

As part of ongoing terminology maintenance, a *Terminology Author* is required to perform maintenance to the defining characteristics of an existing code system.

Associated Functional Models: Maintain Code System Version, Update Code System Version Status, Maintain Code System Supplement

3.2.3.1.3 Create Concept

A *Terminology Author* is required to create a concept to be included in a Code System.

For example, as part of providing *Terminology Service* infrastructure to another department, a *Terminology Author* is required to add additional concept codes to a code system to represent the domain concepts that are important to the new department.

The new concept is defined by the set of meta-data properties that describe it, which may include its proper placement via association binding within the hierarchy of the Code System.

Associated Functional Models: Create Concept

3.2.3.1.4 Maintain Concept

A *Terminology Author* is required to maintain a concept. This includes but is not limited to functionality such as:

- making updates to the associated concept attributes,
- changing the presentation,
- changing preferred name,
- changing synonymy,
- technical corrections to the concept
- modifying the associations bound to concepts
- deprecating a concept
- deprecating a concept and superseding it with another concept

These types of changes may result in a new version of the code system being modified depending on the versioning policy of the terminology publisher.

Associated Functional Models: Maintain Concept, Update Concept Status

3.2.3.2 Value Set Authoring / Curation

This section outlines the business scenarios specific to terminology systems that provide the capability of creating and maintaining sub-sets of codes residing in code systems on CTS 2 servers, otherwise known as value sets.

3.2.3.2.1 Create Value Set

A *Terminology User* is required to create a value set by intension or extension. Both of them can be defined by a computable expression that can be resolved to an exact list of coded concepts at any given point in time. For

example, an intensional value set might be expressed as, SNOMED CT concepts that are children of the SNOMED CT concept “Diabetes Mellitus.” An extensionally defined value set is an enumerating rule.

NOTE : When creating a value set, the *Terminology User* may or may not bind the value set definition to a specific version of the Code System(s) from which the concepts are being drawn. If the value set expression is bound to a specific version of the Code System(s), the value set will always resolve the same set of concept codes for any given version of the value set (see static binding in Core Principles). If the value set expression is **not** bound to a specific version of the Code System(s), the value set will resolve a different set of concept codes as the version (and presumably contents) of the Code System changes (see dynamic binding in Core Principles).

Examples of expressions (or Examples of Rule Sets):

- All active unique identifiers from a given coding system.
- All unique identifiers that participate in a specified relationship with a given local code in a coding system, which may or may not include the specified code itself.
- The transitive closure of a specified transitive relationship with a given code, including or excluding the code itself.
- A reference to another value set
- Other mechanisms that have to have external human or computational resolution.
- Nested value set definition is a rulein which a value set entry references another value set (a child value set). There is no preset limit to the level of nesting allowed within value sets. Value sets cannot contain themselves, or any of their ancestors (i.e., they cannot be defined recursively).
- Unions, intersections and exclusions of any of the above
- A list of concept identifiers from one or more code systems
- Additional details and examples can be found at: http://wiki.hl7.org/index.php?title=Category:V3_Methodology_Requirements

Associated Functional Models: Create Value Set

3.2.3.2.2 Maintain Value Set (meta-data)

As part of ongoing value set maintenance, a Terminology Author is required to perform maintenance to the descriptive characteristics of an existing value set.

Associated Functional Models: Maintain Value Set

3.2.3.2.3 Maintain Value Set

A *Terminology User* is required to maintain (i.e. remove or update) a value set which will create a new value set version.

For example, a *Terminology User* identifies an error in how a value set is defined. A *Terminology Author* corrects the error in the value set to be accurate to the understanding of the *Terminology User*.

Associated Functional Models: Maintain Value Set, Update Value Set Status

3.2.3.3 Concept Domain and Usage Context Authoring / Curation

This section outlines the business scenarios specific to terminology systems that provide the capability of creating and maintaining Concept Domains and associated Usage Contexts, which provide a further level of reusability for Value Sets.

3.2.3.3.1 Create Concept Domain

A *Terminology Author* is required to create a Concept Domain.

Associated Functional Models: Create Concept Domain

3.2.3.3.2 Create Usage Context

A *Terminology User* is required to create a Usage Context within a Jurisdictional Domain. (It is assumed that Jurisdictional Domains themselves would be created as part of system configuration.

Associated Functional Models: Create Usage Context

3.2.3.3.3 Maintain Concept Domain

A *Terminology Author* is required to maintain a concept domain, including the ability for a *Terminology User* to identify the value set bindings for specific usage contexts.

Associated Functional Models: Maintain Concept Domain

3.2.3.3.4 Maintain Usage Context

A *Terminology User* is required to maintain a Usage Context.

Associated Functional Models: Maintain Usage Context

3.2.4 Association Scenarios

The scenarios in this section describe the ability to create, query and maintain associations between coded concepts. These coded concepts may or may not come from the same code system, and as such can describe intra-code system associations as well as inter-code system associations (mappings) across different systems.

These scenarios attempt to outline the information requirements for associations. Since the behavior is similar for both types of association, a single set of operations are defined. Where an information model distinction exists, specific semantic profiles could be defined. In each scenario below, the Terminology Mapper may need to specify additional information pertaining to the source / target association of interest. This information may include:

- The version of the source and target code systems being used to create the association, or,
- The code system of interest, whether that pertains to a single code system or more than one code system
- The version of the source and target code systems being used to create the association, or,
- The cardinality of the association, i.e.: if the association is one-to-one, one-to-many, many-to-one, or many-to-many.

Additionally, the type of associations may include, but are not limited to:

- if the source concept is an *exact match* to the target concept,
- if the source concept is *equivalent* to the target concept,
- if the source concept is *broader than* the target concept,
- if the source concept is *narrower than* the target concept
- Other examples are *generic-to-brand name*, *ingredient-variant-of*, etc.

3.2.4.1 Association Administrative Scenarios

3.2.4.1.1 Enumerate Association Types

A *Terminology User* wants to determine the set of association types (AssociationType) that are available on a terminology server.

Associated Functional Models: List Association Types, Return Association Type Details

3.2.4.1.2 Identify / Retrieve Associations for a Single Concept

A *Terminology User* wants to identify all the associations that exist for a given concept. This includes both direct and indirect relationships, and may be depth limited where appropriate. This includes concept relationships (associations for the concept that are within its native code system) or concept maps (associations between the specified concept code system and another code system) or both. Returns a set of triples: the source, the target and the association

Associated Functional Models: List Concept Associations, Return Association Details

3.2.4.1.3 Identify / Retrieve Associations between Two or More Coded Concepts

A *Terminology User* is required to provide a listing of the associations that exist between coded concepts. For example, these associations may be required as part of a government regulatory compliance review or audit.

This includes concept relationships (associations for the concept that are within its native code system) or concept maps (associations between the specified concept code system and another code system) or both. Returns a set of triples: the source, the target and the association.

Associated Functional Models: List Concept Associations, Return Association Details

3.2.4.1.4 Import Associations

A *Terminology User* is required to make new associations available through a Terminology Server. This may or may not include the removal of previously loaded associations from the terminology server. This may include importing a set of mappings between or within code systems.

Associated Functional Models: Import Association Version

3.2.4.1.5 Export Associations

A *Terminology User* wants to export association type instances from the Terminology Server. This may require filtering of the content and converting the format of the export.

Associated Functional Models: Export Association

3.2.4.1.6 Remove Associations

A *Terminology User* is required to remove (deprecate) associations or association versions from the terminology service, rendering them unavailable for subsequent access by other service functions.

Associated Functional Models: Update Association Status

3.2.4.2 Association Search / Query Scenarios

This section outlines Search / Query operations specific to associations and association content.

3.2.4.2.1 Retrieve Available Associations

A *Terminology User* wants to determine what associations are available on the terminology service by browsing a list of available associations on the CTS 2 instance. The service differentiates between coded concept relationships and coded concept maps available for any specified concept.

Associated Functional Models: List Concept Associations

3.2.4.2.2 Validate Associations

A *Terminology User* wants to validate that a given association or set of associations are available on the CTS 2 service instance based upon specific search criteria.

Associated Functional Models: List Concept Associations

3.2.4.2.3 Retrieve Association Metadata

A *Terminology User* wants to retrieve metadata on available associations in the CTS 2 service instance. This may include metadata regarding the code system(s) employed, versions, authoring / curation content or additional data hosted on the CTS server designated to be used by external systems (i.e.: XML encoded or OWL formatted mapping rule content).

Associated Functional Models: Return Association Details

3.2.4.2.4 Compare Association Versions

A *Terminology User* wants to compare two or more versions of an association on a CTS 2 service instance by viewing each association version's identifying information or metadata. Retrieval is provided by the service, the actual comparison will be the responsibility of the service client application.

Associated Functional Models: Return Association Details

3.2.4.2.5 Request / Retrieve Association Instance

A *Terminology User* would like to request or retrieve metadata about an association from a CTS2 instance.

Associated Functional Models: Return Association Details

3.2.4.2.6 Compute Transitive Closure

A Terminology User want to compute the transitive closure of a binary relation such that whenever **(a,b)** and **(b,c)** are in the extension, **(a,c)** is also in the extension.

Associated Functional Models: Determine Transitive Closure Relationship

3.2.4.2.7 Subsumption

A Terminology User wants to determine if a concept is incorporated under a more general category. Examples of subsumption relationships are: A is-a B, A is-part-of B, A is-specialization-of B.

Associated Functional Models: Compute subsumption relationship

3.2.4.3 Association Author / Curation Scenarios

3.2.4.3.1 Create / Maintain an Association between Coded Concepts

A *Terminology User* wants to create or maintain (i.e. remove or update) an association between coded concepts. For example, these associations may be required to map a local Code System to standard Code Systems in order to be compliant with regulatory reporting policies.

NOTE: Only "create" operations have been identified since it is viewed that any changes to a relationship definition (other than state changes, which are included in a separate operation) are in fact definitions of new relationships (i.e. the rules for the relationship define the relationship, and are not something separate or merely properties. It is possible that the technical specification may introduce update operations, but at this level would not add meaning).

Search criteria may be accompanied by a "match algorithm code" that determines how the search text will be applied. The table below (from CTS) provides an example set of match algorithms.

NOTE: Refer to section 10.10 for details on the matching algorithm.

Associated Functional Models: Create Concept Association

3.2.4.3.2 Create Association Type

A *Terminology Author* is required to create a new association type that may be used to link two concepts.

Associated Functional Models: Create Concept Association Type

3.2.4.3.3 Maintain Association Type

A *Terminology Author* is required to modify or deprecate an association type that may be used to link two concepts.

Associated Functional Models: Maintain Concept Association Type

3.2.4.3.4 Create Lexical Association

A *Terminology User* wants to instantiate an association between two sets of coded concepts using a set of lexical rules (matching algorithms) to generate the associations.

NOTE: This operation is likely not an automated process, as many terminologies do not possess the ontological structures necessary to instantiate additional associations automatically. This however should not preclude the automated creation of associations for terminologies with the ontological infrastructure necessary to enable reasoning engines to effectively instantiate new associations.

Associated Functional Models: Create Lexical Association Between Coded Concepts

3.2.4.3.5 Create Rules Based Association

A *Terminology User* wants to instantiate an association between two sets of coded concepts using a set of description logic or inference rules that either assert or infer mappings between two Code Systems.

NOTE: This operation is likely not an automated process, as many terminologies do not possess the ontological structures necessary to instantiate additional associations automatically. This however should not preclude the automated creation of associations for terminologies with the ontological infrastructure necessary to enable reasoning engines to effectively instantiate new associations. These associations may be subject to human review to verify validity.

Associated Functional Models: Create Rules Based Association Between Coded Concepts

4 Assumptions and Dependencies

4.1 Dependencies on other Service Frameworks

As a service specification, the original CTS specified service discovery APIs. We assume that for CTS 2 a service discovery framework (e.g. using the Universal Description, Discovery and Integration [UDDI] registry) is available to aid with discovery and query of the CTS 2 service, and that the service is queriable by the common service metadata attributes outlined in the Service Discovery framework, in addition to terminology service specific metadata outlined in section 7.

CTS 2 offers a robust set of API requirements, many of which should be restricted to specific user classes. This specification outlines a set of functional profiles that specify the types of operations users may perform as part of a profile. CTS 2 assumes that security, identity management, and auditing services are available that can implement the necessary user role based access requirements outlined in section 6.

4.2 CTS Backwards Compatibility

4.2.1 Message API Support (MAPI)

In the original CTS, the Message API component is specific to HL7. Its primary purpose is to allow a wide variety of message processing applications to create, validate and translate CD (concept descriptor)-derived data types in a consistent and reproducible fashion. It is assumed that this level of functionality will remain specific to HL7, and as such will be managed through the development of a profile specific to HL7 outlined in [Section 7].

4.2.2 General CTS API Support

Unless otherwise indicated, it is assumed that CTS 2 provides the functional coverage required for backwards compatibility to CTS. It is assumed that areas where CTS 2 compatibility with CTS will vary include areas such as:

- HL7 Datatypes - Where the version of the datatypes has been updated since CTS was developed.
- Service Discovery - The CTS service discovery APIs are no longer needed assuming the existence of Service Discovery infrastructure.
- Separating MAPI APIs into a HL7 specific terminology profile as outlined in [Section 7].

4.2.3 Operations on ISO21090 CD datatype

The ISO21090 datatype Concept Descriptor is a reference to a concept defined in an external code system, terminology, or ontology. From an HL7 perspective, at the tooling/application level, high level operations dealing with the relationships between CDs are needed. Such operations include subsumption relationship of two CDs, CD translation equivalence as well as CD clusters related operations. Because CTS2 is a utility service, it will not provide these operations. However, these operations can be obtained be combining lower level CTS2 operations.

5 Considerations for Technical Specification

5.1 Functional Overloading

CTS 2 services are required to support access to terminologies with very different designs and purposes. Some terminologies support concept uniqueness, concept permanence, unique identifiers, formal definitions, and track history to support 'graceful evolution'. Other terminologies lack one or more of these design practices, and may require additional modifications in the functional definitions of CTS 2 functions to be supported by a CTS 2 service. For example, terminologies that reuse concept identifiers among different domains (e.g. 'M' may mean male, million or meter) need the domain identifier in addition to the concept identifier to uniquely identify a concept. Additionally, the separation of codes, concept references and representations as different entities is not reflected in all terminology sources. For example, it is not possible to deprecate 'F' as a code without deprecating 'F' as a display value.

The objective of CTS 2 is to support these various terminologies within a single terminology service, and not to directly influence terminology design. To support terminologies with varying designs, different semantic profiles may need to be used to cope with these variations. In some cases, the CTS2 functions have to include additional optional input parameters (for the terminologies with non-standard designs) to return the *expected* results. Thus, some input parameters for any given CTS 2 function may vary based on the design of the terminology that is being queried. The CTS 2 function will contain both required and optional input parameters, but the optional parameters apply only to terminologies with designs where those parameters are necessary.

However, specifying input parameters as just required or optional can lead to confusion in implementation, which may lead to difficulty for the user to understand which combinations of optional parameters are to be used for a specific terminology. As the number of optional parameters for a function increases linearly, the number of combinations of input parameters can increase exponentially.

For a function with **n** optional input parameters, up to $\mathbf{2^n}$ combinations of input parameters are possible. It then becomes hard for the implementer or user to decipher which combinations are valid and which combinations should be used for different terminologies. In reality, most of these combinations are invalid and cannot be used. In addition, such a function is hard to automate, and requires human intervention to specify the parameters that are applicable to each terminology. Thus, specifying required and optional parameters alone can lead to unnecessary and confusing combinatorial explosion and may lend poorly to automation.

There are several solutions that are possible to this problem. One is the use of different semantic profiles for different terminologies (or classes of terminologies - see further discussion below), as long as the behavior of the operations is not affected (other than validation of mandatory and optional inputs).

Another is 'function overloading'. This is similar to overloading a method in Object Oriented Programming (OOP). A given method in OOP may have many overloaded variants - each with different inputs, but all performing the same basic function and returning the same output. When the technical specification is produced, the resolution to this issue must be defined explicitly. Functional overloading requires more effort to create the functional variants, but this reduces the combinatorial explosion of optional input parameters, avoids creation of invalid combinations of input parameters, and provides a solution to tie a specific

terminology to a specific variant for each function. This also lends better to automation than specifying the input parameters as just optional or required. Functional overloading still cannot be automated at runtime, because we need a way to tie a specific variant of a function to a specific terminology. This is achieved through a metadata discovery service.

5.2 Metadata Discovery

The metadata discovery service helps the calling program to discover the available variants of a CTS 2 function, the input parameters required for each function, and the terminologies that each variant applies to. For example, a function that returns the descriptions of a given concept will require the concept identifier as a coded datatype (ConceptCode, CodeSystemId) as the input. However, if a given terminology reuses the same concept code to represent what are effectively different concepts, a way to disambiguate them is needed. The conceptual model provides a separate unique identifier, however an additional user friendly mechanism is needed. At the level of the code system itself, as well as the identifier, the code name/description will allow a human user to differentiate them, e.g. if "m" is used for "male" and "meter" in the same code system. Over and above that, when using value sets and binding information model instances to them, the "concept domain" may be used to enable clear distinction in these cases. The metadata discovery service will cover all of these variants.

However, creating a relationship between a given terminology and a functional variant can be daunting given the number of CTS 2 functions and the number of terminologies supported. This can be overcome by grouping the terminologies together based on common design and structural characteristics. This is achieved by using semantic profiles. A given semantic profile groups together terminologies with similar designs. Many such semantic profiles are thus possible. The metadata discovery function will list the variants applicable to different semantic profiles, rather than different terminologies.

The CTS 2 authors define the semantic profiles as a set of design characteristics, and assign the better known terminologies into these profiles. At the technical specification/implementation level, the CTS 2 functions will then have overloaded variants based on common design characteristics, rather than those based on individual terminologies. The metadata discovery service will provide the available variants of each CTS 2 function and the different semantic profiles they apply to (rather than the different terminologies).

Functions that do not have overloaded variants will just have the single (default) variant returned by the metadata discovery service. Terminologies that are not yet classified need to be classified into a semantic profile. This is discussed in detail under the Semantic Profiles section.

By using functional overloading, metadata discovery and semantic profiles together, the combinatorial explosion and invalid combinations are avoided, the ambiguity and the amount of effort required are reduced, and automation is made easier.

5.3 Artifact Versioning

CTS 2 exists to allow authoring and editing functionality for many different terminologies with very different designs and purposes. These different terminologies will each have their own approach to version management. For instance, if the definition of a concept changes, does the version of the concept change, or even the version of the code system? Terminologies may define their own rules that must be followed.

The CTS specification needs to describe how artifact versions are maintained with regard to the functioning of the service, and how these changes are implemented in different terminologies. At a minimum, the specification should describe or consider

• The impact of the operations described here on artifact versions as represented on the service interface

• The effect of transactional scope on this behavior

• The implications of this for different terminologies (including particularly those terminologies identified in the mandatory requirements section of the OMG RFP)

• That artifacts may have multiple cycles on various levels (existence, availability, curation related) and may transit between different states repeatedly

• How the versioning approach relates to the various date attributes expressed on the service interface

• How the service describes – perhaps through metadata discovery – the versioning policy of the artifacts the service expresses

Note that the default assumption is that changes to the rules that define artifacts (referred to through the ruleSetId property on several artifacts in the conceptual model) generally cause a version change on the artifact to which they apply, and if the maintenance is done by reference, appropriate curation responsibilities will need to be in place to ensure that the rules do not change without an appropriate change to the artifact version.

This specification assumes that versioning and state management are duties of the authors (often SDOs) and managers of specific terminologies. CTS 2 only seeks to outline functional capabilities (operations and semantics) to support versioning and state updates, but does not seek to define the policy by which any given terminology handles versioning and state updates.

5.4 Properties

Code systems use named properties to define, describe, and identify the entities (classes, properties and individuals) within the code system. Some of these properties will have semantic significance to the services described in this document, including "description", "designation", "association", and possibly others (e.g. comment", "editorial note", etc.) that may be identified as significant by respondents to this document. There may also be properties that have no direct significance to the services themselves, beyond being used to sort or

search code system entities.

In the remainder of this discussion, we will explicitly differentiate between a property **type** and a property **instance**. Every property instance is associated with a property type, where the instance describes a code system entity and the type describes the meaning of the instance. As an example, the property type dc:title might be used to identify a particular type of designation. Instances of dc:title would then be associated with entities in the code system,

(e.g. <rdf:Descriptionrdf:ID="C0002940"><dc:title>Aneurysm</dc:title></rdf:Description>)

Property instances may, themselves, have associated properties that describe or refine how the property is used. As an example, a property instance might identify the language of the property value, the mime type of the value, information about when the property applies, etc.

- Respondents to the RFP shall identify which property types are semantically significant to the service itself and will describe how they are identified.

- Respondents to the RFP shall address both property types and property instances. They should address how properties types are assigned meaning, how they can be identified as representing designations, associations, descriptions, etc. Respondents should describe how property types might be used to validate property instances through the association of data types, mandatory characteristics such as language, etc.

- Respondents to the RFP shall discuss how to support terminologies that are expressed in and make full usage of OWL-DL 2.0 like syntax, semantics and features (OWL classes, properties, restrictions and individuals)

5.5 Code System Partitions

Some larger code systems include the notion of "partitions" and/or "extensions", whereby distinct subsets of the code system may be separately maintained and published independently.

For example, SNOMED CT has 19 "partitions" (or "pipes"), which are the 19 top level concepts under the one primary concept of "SNOMED CT concept". Although it has not yet happened, it is believed that these individual pipes may be published in future at different frequencies. National extensions are concepts that are attached to the core, and are in core format, but are only applicable to a particular realm or realms. They are maintained separately and may be released separately. These include "branded" medicinal products, administrative concepts and other realm only concepts. These may also be versioned separately from the main core. They may be distributed separately or with the core. For example, in the UK extension of SNOMED CT, the "pharmaceutical and biological hierarchy concepts" are about to be published separately on a different release cycle (monthly rather than six monthly), in the TRUD (the Terminology Release, Update and Distribution service).

RFP responders must indicate how they will support such partitions and/or extensions.

5.6 Code System Supplement

Often implementer groups will find it necessary to "supplement" the metadata maintained about a code system beyond that provided by the code system author. This may include the creation of localized display names, translated display names, concept relationships to some local code system or creation of additional concept properties. These supplements are not code systems in and of themselves because they do not define any new codes or concepts. The codes and code system ids used when identifying a concept in an instance would remain the code and code system id of the base terminology same whether a supplement was being used or not. They are also not a "part" of the base code system as they are developed and maintained by a completely different group, potentially without the knowledge of the author of the base code system.

While these supplements do not change the codes that can be referenced, they do provide additional information that can be used in the construction of value sets and may impact how codes from those value sets are exposed to the user. Respondents must indicate how they will handle the creation and maintenance of "Code System Supplements" (including managing dependencies between code system supplement versions and versions of the underlying code system). They must also show how these supplements will be supported in value set definition.

6 Detailed Functional Model for each Interface[2]

6.1 Administration Operations

6.1.1 Import Code System

Description	Installs a code system (aka terminology) into the terminology service for subsequent access by other service functions. This operation is used for the initial install of the overall terminology structure itself. This may include the full set of concepts, relationships and so on, or some of these elements may be loaded using the Import Code System Revision operation. The actual contents may be supplied by value or reference, i.e. as a complete set of explicit content or as a reference to a location where the content can be separately obtained for loading.
Inputs	1. Code System ID 2. Code System Description (optional) 3. Code System Administrative Info (optional) 4. Code System Version (see model for class description) 5. Code System Contents (i.e. either: Source Location (URI) OR Code System Contents (Concepts, Concept Properties, ConceptAssociations, Designations etc.)
Outputs	1. An acknowledgment indicating whether the terminology has been successfully loaded or not.
Precondition	1. 2. Code System source is available in a format directly consumable by CTS 2 import tools.
Postconditions	1. The code system is available for access via the CTS 2 service functions.
Exception Conditions	1. Code system not found at source location 2. Supplied contents cannot be processed (may be a number of specific errors) (Information pertaining to all failures is logged and reported for analysis and serviceability.)
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous	

[2] All version input parameters are optional, if not provided, the latest version is used by the operation.

Notes	
Other Relevant Content	
Associated Scenario	Import Content, Import Associations

1

6.1.2 Import Code System Revision

Description	Installs either an entire new version or the necessary revision updates for an already loaded code system (terminology) into the terminology server repository (content included by value or by reference to a location). Includes indicator as to whether intent is to replace whole code system or just replace some elements (codes, associations etc), cf. Maintain Concept
Inputs	1. Code System Revision Source ID 2. Code System ID 3. Current Code System Version ID (optional) 4. New Code System Version (class) 5. Scope of Replacement (encoded and description) 6. Code System Location (URI) .. OR .. 7. Updated Contents (Concepts, Concept Properties, ConceptAssociations, Designations etc.)
Outputs	1. An acknowledgment indicating whether the terminology revision has been successfully loaded or not.
Precondition	1. 2. Code System revision source is available in a format directly consumable by CTS 2 import tools
Postconditions	1. The code system revision is available for access via the CTS 2 service functions.
Exception Conditions	1. Code System does not exist. 2. Code System source is not consumable by CTS 2 import tools. 3. Code System not found at source location 4. Supplied contents cannot be processed (may be many different error conditions) (Information pertaining to the failures is logged and reported for analysis and serviceability.)
Aspects left to Technical Specification	Detailed specification of the mechanism to recognize the specific information structures that may be imported.
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	Code System revisions may be available as either:

	1. complete code systems 2. set of deltas to be applied sequentially to the previous version. In either case, all previous versions/iterations should be available until specifically removed.
Other Relevant Content	Depending on the scope of the actual import, many different information structures may be included. One particular area (Associations) is expanded in more detail below. This could include elements such as: 1. Association Identifier (if known) 2. Association Description 3. Association Source Code System Id and Concept Code 4. Association Target Code System Id and Concept Code 5. Association Type 6. Association Restrictions 7. Association Cardinality 8. Association Group 9. Association Order 10. Association is Reciprocal 11. Association is Refinable 12. Association is Transitive 13. Association is Cyclic 14. Association is Inheritable 15. Association Curation / Authoring Information 16. External systems association data hosted on the CTS server (e.g. XML encoded or OWL formatted mapping rule content). **NOTE**: This level of detail is not reflected in the conceptual model at this stage.
Associated Scenario	Import Content , Import Associations

1

6.1.3 Import Value Set Version

Description	Installs a value set version into the terminology service for subsequent access by other service functions. This operation may also be used for the initial installation of the value set. The value set contents may be explicitly provided, or may be resolved at run time in the case of intensional value sets (where the value set is defined as a computable expression).
Inputs	1. Value Set ID 2. Value Set Description (optional) 3. Value Set Administrative Info (optional) 4. Value Set Version (see model for class description) 5. Value Set Contents (ruleSetID) OR Value Set Contents)
Outputs	1. An acknowledgment indicating whether the value set has been successfully loaded or not.
Precondition	1. 2. Value set source is available in a format directly consumable by CTS 2 import tools.
Postconditions	1. The value set is available for access via the CTS 2 service functions.
Exception Conditions	1. Value set not found at source location 2. Supplied contents cannot be processed (may be a number of specific errors) (Information pertaining to all failures is logged and reported for analysis and serviceability.)
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Import Content

6.1.4 Import Association Version

Description	Installs a set of code system associations (mappings) into the terminology service for subsequent access by other service functions. This operation may be used for the initial installation of a set of associations. This may include the full set of associations between concepts from different code systems. The actual associations may be supplied by value or reference, i.e. as a complete set of explicit association or as a reference to a location where the associations can be separately obtained for loading.
Inputs	1. Association (set) Identifier 2. Association Description (optional) 3. Association Administrative Info (optional) 4. Association Version (optional) 5. Association Location (URI) OR 1. Association Contents (Concepts, ConceptAssociationType, ConceptAssociations (maps) etc.)
Outputs	1. An acknowledgment indicating whether the associations have been successfully loaded or not.
Precondition	1. Associations are available in a format directly consumable by CTS 2 import tools.
Postconditions	1. Association are available for access via the CTS 2 service functions.
Exception Conditions	1. Association not found at source location 2. Supplied contents cannot be processed (may be a number of specific errors) (Information pertaining to all failures is logged and reported for analysis and serviceability.)
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	
Other Relevant Content	

Associated Scenario	Import Content, Import Associations

1

6.1.5 Export Association

Description	Exports association type instances from a code system applying filter criteria
Inputs	1. Code System ID 2. Code System Version ID 3. Association Type(s) 4. Filter criteria (optional) 5. Export Format
Outputs	1. Requested association type instances of the code system
Precondition	1. CTS 2 Service installed and running 2. Code System source is available for querying
Postconditions	1. The association instances are exported
Exception Conditions	1. Invalid criteria/format input 2. Unrecognized location input 3. Association source is not exportable by CTS 2 export tools.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Export Content , Export Associations

6.1.6 Export Code System Content

Description	Exports a code system (terminology), subset criteria (specific set of concepts and/or associations/maps from the Terminology Server by filtering the content and converting to the requested format for export (according to the semantic profile of the deployment jurisdiction).
Inputs	1. Code System ID 2. Code System Version ID 3. Code System filter criteria 4. Export Format 5. Target Location (URI)
Outputs	1. Code System Location 2. Code System Content (Depending on whether the return will be the actual content or a location in which the associations (maps) are placed for subsequent retrieval.)
Precondition	1. CTS 2 Service installed and running 2. Code System source is available for export
Postconditions	The code system is exported to the required location or is returned to the requestor
Exception Conditions	1. Invalid criteria/format input 2. Unrecognized location input 3. Association source is not exportable by CTS 2 export tools.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Export Content

6.1.7 Change Code System Status

Description	Changes the state of a code system. (Includes inactivation, activation etc.) This allows a *Terminology Administrator* to manage the state of a given code system, thus managing the level of availability for access by other terminology service functions.
Inputs	1. Code system identifier 2. Code system version 3. Code system status
Outputs	1. An acknowledgment indicating whether the state of the code system has been successfully changed.
Precondition	1. 2. Code system available on the CTS 2 service
Postconditions	1. The code system source state is changed to the requested value.
Exception Conditions	1. Code System state transition not allowed 2. Input parameter does not match
Aspects left to Technical Specification	1. Definition of actual state values / model
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Change Content Status, Decommission Terminology Content

6.1.8 Register for Notification

Description	Register to be notified whenever a vocabulary element (code system, value set, concept or relationship) is modified in any way. The Notification Directions parameter is a placeholder for allowing specific directions to be given for the notification (e.g. whether it is required immediately or may be delayed)
Inputs	1. URL or other electronic address which to send the terminology element modification notification to. 2. Notification Target Type (Code System, Value Set, Concept, Concept Relationship) 3. Notification Target Identifier 4. Notification Target Version 5. Notification Directions
Outputs	1. An acknowledgment indicating whether the terminology element notification request was successfully created. 2. Notification Identifier
Precondition	1. 2. User or appropriate proxy (system administrator, etc) are authorized to access registry
Postconditions	1. Notification records are updated appropriately
Exception Conditions	1. Identified element / version does not exist 2. Notification Directions not recognized
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	Subsequent notifications do not require a confirmation. Where appropriate, however, negative feedback on the channel (unable to deliver message, unable to connect), should result in attempts to retransmit and/or the placement of a temporary hold on notifications until connection problem is corrected.
Other relevant content	
Associated Scenario	Update Notification, Content Dependency Notification

1

2

6.1.9 Update Notification Registration

Description	Revises a notification entry for a particular vocabulary element.
Inputs	1. Notification Entry Identifier 2. URL or other electronic address which to send the terminology element modification notification to. 3. Notification Target Type (Code System, Value Set, Concept, Concept Relationship) 4. Notification Target Identifier 5. Notification Target Version 6. Notification Directions
Outputs	1. An acknowledgment indicating whether the terminology element notification revision request was successfully processed.
Precondition	1. 2. User or appropriate proxy (system administrator, etc) are authorized to access registry
Postconditions	1. Notification records are updated appropriately.
Exception Conditions	1. Notification Entry Identifier does not exist. 2. Identified element / version does not exist 3. Notification Directions not recognized
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	Subsequent notifications do not require a confirmation. Where appropriate, however, negative feedback on the channel (unable to deliver message, unable to connect), should result in attempts to retransmit and/or the placement of a temporary hold on notifications until connection problem is corrected.
Other Relevant Content	
Associated Scenario	Update Notification Management, Content Dependency Notification

6.1.10 Update Notification Registration Status

Description	Changes the status of a notification entry for a particular vocabulary element (suspend, reinstate, cancel, remove).
Inputs	1. Notification Entry Identifier 2. Notification Status
Outputs	1. An acknowledgment indicating whether the notification status revision request was successfully processed.
Precondition	1. 2. User or appropriate proxy (system administrator, etc) are authorized to access registry
Postconditions	1. Notification status is updated appropriately.
Exception Conditions	1. Notification Entry Identifier does not exist. 2. Invalid state change
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Administration
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Update Notification Management, Content Dependency Notification

6.2 Search / Access Operations

The following detailed functional models outline functionality supporting the business scenarios in Section 3.2.2 of this document. Input parameters specified here are intended to correlate with the attributes outlined in the Terminology Structure Considerations. There are however, notable exceptions:

Filter Criteria - Filter Criteria is intended to specify the types of filtering mechanisms specific to the Code System Entities being queried. It is understood that the identification attributes for code system elements such as concepts and value sets are different, and as such different filter criteria may be applied to appropriately identify and filter code system entity information being queried. As part of the technical specification, Filter Criteria will be specified explicitly in accordance with the developed Platform Independent Model.

Query Control - Query Control defines a placeholder for parameters that have the capacity of constraining the query results returned from a query for code system elements. This allows a user to restrict the amount of data returned, or the order in which the data is sorted. This is different from Filter Criteria, which specifies restriction on the business data being returned.

6.2.1 Code System Search / Access

6.2.1.1 List Code Systems

Description	List the code systems available on this instance of the CTS 2 Service that match entered filter criteria.
Inputs	1. Filter Criteria 2. Query Control
Outputs	1. A listing of zero or more code systems, together with metadata properties available on the instance of the terminology service.
Precondition	1.
Postconditions	None.
Exception Conditions	1. Invalid filter criteria 2. Invalid query control
Aspects left to Technical Specification	Definition of filter criteria. Should include name for a simple means of retrieving ID where not known.
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Retrieve Available Code Systems

1

2 **6.2.1.2 Return Code System Details**

Description	Returns the details (metadata) for a given code system available on the terminology service
Inputs	1. Code System Identifier 2. Code System Version
Outputs	1. Details pertaining to a specific code system (metadata or attributes for the code system, e.g. description)
Precondition	1.
Postconditions	None.
Exception Conditions	1. Code System Identifier not found. 2. Code System version not found.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Retrieve Available Code Systems, Compare Code System Versions

3

6.2.1.3 List Code System Concepts

Description	Returns the set of all concepts (and concept codes) in the specified code system, optionally filtered by input criteria, such as state or specific concept property, e.g. values.
Inputs	1. Code System Identifier 2. Code System Version 3. Filter Criteria 4. Query Control
Outputs	1. The set of zero or more concepts and concept codes in the specified code system matching the filter criteria.
Precondition	
Postconditions	None.
Exception Conditions	1. Code System Identifier not found. 2. Code System version not found. 3. Invalid filter criteria 4. Invalid query control
Aspects left to Technical Specification	Definition of filter criteria and the state model.
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	Returning all concepts and concept codes in a code system is generally impractical for large code sets. Indexing, query optimization is necessary. As noted earlier in the conceptual model description, all functions dealing with "Concepts" (this being the first)identified in this SFM are stated in terms of dealing with "Concepts". Unless explicitly stated otherwise, this should be read as "Concept Node" (i.e. Concept and/or its associated Concept Code(s), depending upon how the actual containing code system is actually set up and populated.)
Other Relevant Content	
Associated Scenario	Retrieve Coded Concepts from Code System, Compare Code System Versions

6.2.1.4 Return Concept Details

Description	Returns the details for the known attributes (metadata) of a coded concept
Inputs	1. Code System Identifier 2. Code System Version Id 3. Concept Identifier 4. Concept Version Id
Outputs	1. The details of the attributes (metadata) of the coded concept
Precondition	1.
Postconditions	None.
Exception Conditions	1. Code System Identifier not found. 2. Code System Version not found. 3. Concept Identifier not found. 4. Concept Version not found.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	For versions, an acceptable input should be "current" or "latest"
Other Relevant Content	
Associated Scenario	Retrieve Coded Concepts from Code System, Retrieve Coded Concepts from Value Set, Retrieve Concept Representations, Retrieve Coded Concepts from Code System, Validate Concept in Code System, Identify Concept Language Translations, Retrieve Coded Concepts for Concept Domain via Value Set Membership

6.2.1.5 List Association Types

Description	List association types that are available on a terminology server that match entered filter criteria (e.g. code system, code system version).
Inputs	1. Filter Criteria 2. Query Control
Outputs	1. Returns a list of zero or more association types, together with associated metadata
Precondition	
Postconditions	None.
Exception Conditions	1. Invalid filter criteria 2. Invalid query control
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Enumerate Association Types

6.2.1.6 Return Association Type Details

Description	Returns the details for the known attributes (metadata) of a relationship type.
Inputs	1. Association Type Id
Outputs	1. Full metadata for the Relationship type
Precondition	1.
Postconditions	None.
Exception Conditions	1. Association Type Id not found
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Enumerate Association Types

6.2.2 Value Set Search / Access

6.2.2.1 List Value Sets

Description	Lists the value sets that are available to the CTS 2 service.
Inputs	1. Value Set Filter Criteria 2. Query Control
Outputs	1. Listing of zero or more value sets on this instance of the terminology server that match the input filter criteria, together with associated metadata.
Precondition	1.
Postconditions	None.
Exception Conditions	1. Invalid filter criteria 2. Invalid query control
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	When search attributes are applied, the result set is restricted to the value sets that match the search filter criteria. Examples include: 1. restricting to matching properties such as: 1. Value Set Name 2. Value Set version 3. Code Systems that comprise the values of the value set 4. Metadata attributes/properties of the value set
Other relevant content	
Associated Scenario	Retrieve Available Value Sets

1

2 #### 6.2.2.2 Return Value Set Details

Description	Look up detailed information (metadata) for a given value set.
Inputs	1. Value Set Identifier 2. Value Set version
Outputs	1. Details (metadata) pertaining to the specified value set (resolved meta data or attributes for the value set.)
Precondition	1.
Postconditions	None.
Exception Conditions	1. Value Set Identifier not found. 2. Value Set version not found.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	For version, an acceptable input should be "current" or "latest".
Other relevant content	
Associated Scenario	Retrieve Available Value Sets, Compare Value Set Versions

3

6.2.2.3 List Value Set Contents (Expand value set)

Description	Lists out the contents (entries) of a given value set, filtering based on input criteria. This function is to be used to create the value set expansion.
Inputs	1. Value Set Identifier 2. Value Set version (Optional) 3. Value Set time stamp (optional) 4. Filter criteria 5. Query Control
Outputs	1. A list of zero or more code system nodes for the given value set. Output truncated if value set not finite.
Precondition	1.
Postconditions	None.
Exception Conditions	1. Value Set Identifier not found. 2. Value Set version not found. 3. Invalid filter criteria 4. Invalid query control
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	Value sets may not be finite (e.g. the set of all real numbers between 1 and 10) Obviously we don't want to list them all.
Other relevant content	
Associated Scenario	Retrieve Coded Concepts from Value Set, Compare Value Set Versions, Retrieve Concept Representations, Retrieve Coded Concepts for Concept Domain via Value Set Membership

1

2 **6.2.2.4 Check Value Set Subsumption**

Description	Determine whether one of the two supplied value sets subsumes the other
Inputs	1. Pair of two <Value Set Identifier, Value Set Version > conjunctions
Outputs	1. Return True if value set A subsumes Value set B 2. Return False if not
Precondition	1.
Postconditions	Boolean returned.
Exception Conditions	1. At least one conjunction of <Value Set Identifier , Value Set version> not found., indicate which
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	

3

1

2 **6.2.2.5 Check Concept Value Set Membership**

Description	Determine whether the supplied coded concept exists in the supplied value set
Inputs	2. Code System Identifier 3. Code System Version (optional) 4. Concept Identifier or Concept Code 5. Value Set Identifier 6. Value Set Version
Outputs	3. Return True if coded concept exists in value set 4. Return False if coded concept does not exist in value set
Precondition	
Postconditions	None.
Exception Conditions	2. Code System Identifier not found. 3. Concept code not found. 4. Value Set Identifier not found. 5. Value Set version not found.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Validate Coded Concept in Value Set

3

6.2.3 Concept Domain and Usage Context Search / Access

6.2.3.1 List Concept Domains

Description	Lists the concept domains that are available to the CTS 2 service.
Inputs	1. Concept Domain Filter Criteria 2. Query Control
Outputs	1. Listing of zero or more concept domains on this instance of the terminology server that match the input filter criteria, together with associated metadata.
Precondition	
Postconditions	None.
Exception Conditions	1. Invalid filter criteria 2. Invalid query criteria
Aspects Left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	When search attributes are applied, the result set is restricted to the concept domains that match the search filter criteria. Examples include: 1. restricting to matching properties such as: 1. Concept Domain Name 2. Value Sets that are bound to the Concept Domain 3. Usage Contexts in which the Concept Domain is bound to value sets 4. Metadata attributes/properties of the Concept Domain
Other relevant content	
Associated Scenario	Retrieve Available Concept Domains

6.2.3.2 Return Concept Domain Details

Description	Look up detailed information (metadata) for a given concept domain.
Inputs	1. Concept Domain Identifier
Outputs	1. Concept Domain details (resolved meta data or attributes for the concept domain)
Precondition	
Postconditions	None.
Exception Conditions	1. Concept Domain Identifier not found.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Retrieve Available Concept Domains

6.2.3.3 List Usage Contexts

Description	Lists the usage contexts that are available to the CTS 2 service.
Inputs	1. Usage Context Filter Criteria 2. Query Control
Outputs	1. Listing of zero or more usage contexts on this instance of the terminology server that match the input filter criteria, together with associated metadata.
Precondition	
Postconditions	None.
Exception Conditions	1. Invalid filter criteria 2. Invalid query control
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS2 Query
Miscellaneous Notes	When search attributes are applied, the result set is restricted to the usage contexts that match the search filter criteria. Examples include: 1. restricting to matching properties such as: 1. Usage Context Name 2. Concept Domains that are bound to value sets in the Usage Context 3. Metadata attributes/properties of the Concept Domain
Other relevant content	
Associated Scenario	Retrieve Available Concept Domains

1

2 #### 6.2.3.4 Return Usage Context Details

Description	Look up detailed information (metadata) for a given usage context.
Inputs	1. Usage Context Identifier
Outputs	1. Detailed Usage Context description (resolved meta data or attributes for the usage context)
Precondition	
Postconditions	None.
Exception Conditions	1. Usage Context Identifier not found.
Aspects Left to Technical Specification	
Relationship to Levels of Conformance	CTS2 Query
Miscellaneous Notes	
Other Relevant Content	
Associated Scenario	Retrieve Available Concept Domains, Retrieve Coded Concepts for Concept Domain via Value Set Membership

3

6.2.3.5 List Concept Domain Bindings

Description	List the value set identifier and metadata for the specified domain bindings. This does not include returning the value set expansion.
Inputs	1. Concept Domain Identifier 2. Filter criteria 3. Query Control
Outputs	1. A list of zero or more value set bindings.
Precondition	
Postconditions	None.
Exception Conditions	1. Concept Domain Identifier not found. 2. Invalid filter criteria 3. Invalid query criteria
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS2 Query
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Retrieve Coded Concepts for Concept Domain via Value Set Membership

6.2.3.5.1 Check Concept to Concept Domain Association

Description	Determine whether the supplied coded concept exists in a code system in use for the specified concept domain, optionally within specific usage contexts. Returns true if a coded concept is an element of a value set expansion bound to the provided concept domain, or bound to both concept domain and usage context.
Inputs	1. Code System Identifier 2. Concept Identifier or Concept Code 3. Concept Domain Identifier 4. List of Usage Context Identifiers (optional)
Outputs	1. Return True if coded concept exists in specified concept domain and usage contexts 2. Return False if coded concept does not exist in specified concept domain and usage contexts
Precondition	
Postconditions	None.
Exception Conditions	1. Code System Identifier not found. 2. Concept code not found. 3. Concept Domain Identifier not found. 4. Usage Context Identifier not found.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS2 Query
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Validate Coded Concept in Concept Domain via Value Set Membership , Validate Coded Concept in Value Set

6.2.4 Association related queries

6.2.4.1 List Associations

Description	List the Association available on an instance of the CTS 2 Service that match a set of input filter criteria
Inputs	1. Filter Criteria 2. Query Control
Outputs	A listing of zero or more associations that match the input filter criteria
Precondition	
Postconditions	none
Exception Conditions	1. Invalid filter criteria 2. Invalid query control
Aspects left to Technical Specification	Definition of filter criteria. This is expected to include: Code System Ids and/or Code System Version Ids, Concept Codes / Ids, Association Types, whether only direct associations are considered or whether the transitive closure of the relation are used. There are potentially many different types of Associations that can be defined. In some cases, the relationships are explicitly defined, persisted and managed, in other cases, queries can be made based on algorithms (such as those defined in the descriptions of the Create Lexical and Rules Based Associations). To some degree, these are also both more targeted "query" operations in that they return a list of associations based on the input rules, as well as allowing for persisting of the Associations if so required. For the purposes of the SFM, this operation will allow retrieval of any type of Association, including lexical and rules based, but the technical specification may define a number of more explicit operations.
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Identify / Retrieve Associations for a Single Concept, Validate Associations, Identify / Retrieve Associations Between Two or More Coded Concepts

6.2.4.2 Determine Transitive Concept Relationship

Description	Determine whether there exists a transitive relationship between two concepts, if it exists, provide the association path. Determine Transitive Concept Relationship determines if it is possible to get from a given ParentCodeSystemNode into a given ChildCodeSystemNode in one or more association transitions.
Inputs	1. ParentCodeSystemNodeID (support for compositional expression optional) 2. ChildCodeSystemNodeID (support for compositional expressions optional) 3. ConceptAssociationType
Outputs	1. Determine Transitive Concept Relationship will return the edge graph necessary to traverse from ParentCodeSystemNode to ChildCodeSystemNode using the given ConceptAssociationType, or null if the transitive closure does not exist.
Precondition	1. At least one code system is available in the CTS 2 service.
Postconditions	None.
Exception Conditions	1. Invalid ParentCodeSystemNode 2. Invalid ChildCodeSystemNode 3. Invalid ConceptAssociationType 4. Compositional concepts not supported (if input is compositional and not supported)
Aspects left to Technical Specification	
Relationship to Levels of Conformance	CTS 2 Query
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Compute Transitive Closure

1

2 **6.2.4.3 Compute Subsumption Relationship**

Description	Subsumes tests whether the parent coded attribute subsumes (is implied by) the child. If ParentCodeSystemNode nor ChildCodeSystemNode are non compositional and are both drawn from the same code system, subsumes returns true if and only if ChildCodeSystemNode can be determined to belong to the transitive closure of the hasSubtype relationship graph headed by ParentCodeSystemNode. No further assertions of the semantics of subsumption beyond this one case. If the service supports subsumption involving qualifiers (compositional expressions) and/or subsumption tests across multiple code systems, it must define the appropriate translation semantics. If the service doesn't support subsumption on concepts defined using compositional expression, it should raise the QualifiersNotSupported exception if presented with a parentCode or childCode containing qualifiers. Similarly, if the service doesn't support cross-code system subsumption testing, it should raise SubsumptionNotSupported when supplied with codes with different code systems.
Inputs	1. ParentCodeSystemNodeID, compositional expressions optional (depending on service realization, no need to support for full compliance) 2. ChildCodeSystemNodeID, compositional expressions optional (depending on service realization, no need to support for full compliance)
Outputs	1. Subsumes will return **true** if the child code can be determined to be a subtype of the parent. Subsumes will also return **true** if the child and parent codes are equivalent.
Precondition	1. 2. At least one code system is available in the CTS 2 service.
Postconditions	None.
Exception Conditions	1. Invalid ParentCodeSystemNode 2. Invalid ChildCodeSystemNode 3. Compositional expression non supported
Aspects left to Technical Specification	
Relationship to Levels of Conformance	[CTS 2 Query]
Miscellaneous Notes	
Other relevant content	

Associated Scenario	Subsumption

1

6.2.4.4 Return Association Details

Description	Look up detailed information (metadata) for a given association
Inputs	1. Association identifier
Outputs	All available Association information (resolved meta data or attributes for the association.) Including: 1. Code system identifier(s) 2. Code system version(s) 3. Code system name(s) and description(s) 4. Concept codes / identifiers / names 5. Authoring / curation information 6. External systems association data hosted on the CTS server (i.e.: XML encoded or OWL formatted Association rule content).
Precondition	1.
Postconditions	None.
Exception Conditions	1. Association does not exist.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Associations
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Retrieve Association Metadata, Identify / Identify / Retrieve Associations for a Single Concept, Identify / Retrieve Associations Between Two or More Coded Concepts , Compare Association Versions, Request / Retrieve Association Instance

6.3 Authoring/Curation Operations

6.3.1 Code System Authoring/Curation

6.3.1.1 Create Code System

Description	Create a new Code System to contain a set of new coded concepts. The Code System is created by defining the set of meta-data properties that describe it.
Inputs	1. Code System Name 2. Code System Version 3. Code System properties
Outputs	1. An acknowledgment indicating whether the code system was created or not. 2. Code System Id, if the code system was successfully created.
Precondition	1.
Postconditions	1. The code system is available for access via the CTS 2 service functions.
Exception Conditions	1. Code System already exists.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	1. If the user wants to specify the Code System Id manually, it can be specified as a "Code System Property". 2. The service will denote that the code system exists when applicable. We will let the individual service administrator treat this exception as an error (refuse to create the code system) or a warning (create code system anyway). 3. Since there is a separate "import" operation for loading code sets, the semantics of the "create" should be to just create the code system and then create the Concepts. This implies the need for a "release" or "publish" capability, which will be handled by an "Update Code System Version Status" operation.
Other relevant content	
Associated Scenario	Create Code System

6.3.1.2 Maintain Code System Version

Description	Update Code System meta-data properties.
Inputs	1. Code System Id 2. Code System Name 3. Code System Version 4. Code System properties 5. Set of zero or more Concepts (Ids, Properties, Designations) 6. Set of zero or more Concept Associations
Outputs	1. An acknowledgment indicating whether the code system was updated or not.
Precondition	1.
Postconditions	1. The updated code system is available for access via the CTS 2 service functions.
Exception Conditions	1. Code System does not exist. 2. Code System version does not exist. 3. Concept not found 4. Invalid Concept Properties 5. Invalid Concept Details 6. Invalid Associations
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	This allows for maintenance of a Code Systems properties, including creation and updating of Code System Versions, and assignment of Concepts to a Version. Note that the set of concept IDs could either be a complete new full set or deltas only, which needs to be identified explicitly. Entering as new Version Id causes creation of a new version (assuming the state model allows it at the time) Concepts can be assigned and/or unassigned from Code System Versions, as can Concept Properties, Designations and Concept Relationships.
Other Relevant Content	
Associated Scenario	Maintain Code System

6.3.1.3 Update Code System Version Status

Description	Changes the status of a code system version (suspended, reinstated, canceled, removed).
Inputs	1. Code System Identifier 2. Code System Version 3. Status
Outputs	1. An acknowledgment indicating whether the status revision request was successfully processed.
Precondition	
Postconditions	1. Code System Version status is updated appropriately.
Exception Conditions	1. Code System Version Identifier does not exist. 2. Invalid state change
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Maintain Code System

6.3.1.4 Create Code System Supplement

Description	Create a new Code System Supplement as a container of a set of concepts and concept properties to be appended to a target code system. Does not add the concepts and properties.
Inputs	1. Target Code System Id 2. Code System Supplement Name 3. Code System Supplement description 4. Code System Supplement provenance details 5. Code System Supplement effective date
Outputs	1. An acknowledgment indicating whether the code system supplement was created or not. 2. Code System Supplement Id, if the code system was successfully created.
Precondition	
Postconditions	Supplement created.
Exception Conditions	2. Code System Supplement already exists.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	

6.3.1.5 Maintain Code System Supplement

Description	Update Code System Supplement meta-data properties and add concepts and properties to code system.
Inputs	1. Code System Property Id 2. Code System Supplement Name 3. Code System Supplement description 4. Code System Supplement provenance details 5. Code System Supplement effective date 6. Set of zero or more Concepts 7. Set of zero or more concept properties
Outputs	1. An acknowledgment indicating whether the code system supplement was updated or not.
Precondition	
Postconditions	Supplement updated.
Exception Conditions	1. Code System Supplement not found. 2. Code System version does not exist. 3. Concept not found 4. Invalid Concept Properties
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	This allows for maintenance of a Code Systems Supplement properties, including creation and updating of Code System Versions, and assignment of Concepts and their properties to a Supplement .
Other Relevant Content	
Associated Scenario	

1

2 **6.3.1.6 Create Concept**

Description	Create concept to be included in a Code System. The new concept is defined by the set of meta-data properties that describe it, which may include its proper placement via association binding within the hierarchy of the Code System.
Inputs	1. Code System ID 2. Code System Version 3. Concept Code 4. Concept Identifier or Concept Code (optional) 5. Concept Properties 6. (Allowable) Concept Relationships 7. Concept Designations
Outputs	1. An acknowledgment indicating whether the concept was created or not. 2. Concept Identifier or Concept Code
Precondition	1. A sequencer that provides Concept Identifiers must be available if the Concept Identifier is not provided as an input.
Postconditions	1. The concept is available in the code system and is available for access via the CTS 2 service functions.
Exception Conditions	1. Code System does not exist. 2. Code System version does not exist. 3. Concept already exists.
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	1. The service will denote that the concept exists when applicable. We will let the individual service administrator treat this exception as an error (refuse to create the concept) or a warning (create concept anyway). Generated ID can be used for disambiguation in code sets where codes are reused. The input Code System Version is the version of the code system in which the concept code is first created Entered Concept Properties and Relationships become the "allowable" or "Defined Concept Properties / Relationships" in the model and also associated with the

	Code System Version in which they were created.
Other relevant content	
Associated Scenario	Create Concept

1

1

2 **6.3.1.7 Maintain Concept**

Description	Update Concept meta-data properties.
Inputs	1. Code System Id 2. Concept Code 3. Concept Id 4. Concept VersionId 5. Jurisdictional Domain Id (optional) 6. Concept Properties 7. (allowable) associations 8. Concept Designations
Outputs	1. An acknowledgment indicating whether the concept was updated or not.
Precondition	
Postconditions	1. The concept is updated appropriately.
Exception Conditions	1. Code System does not exist. 2. Concept does not exist. 3. Unrecognized Jurisdictional Domain 4. Invalid Concept Properties 5. Invalid Concept Designation
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	Updates include but is not limited to functionality such as: 1. making updates to the associated concept attributes, 2. changing the presentation 3. changing preferred name 4. changing synonymy 5. technical corrections to the concept 6. modifying the associations bound to concepts 7. in keeping with good vocabulary practice, codes or identifiers for concepts cannot be reused. Additionally, in hierarchical Code Systems, it may be necessary to re-associate any concepts related to the concept being deprecated to prevent a part of the code system hierarchy from being orphaned

	Code System Version not input here. A set of Concept changes can be made then assigned as a new Code System Version before it is published (using Maintain Code System Version and then Update Code System Version Status) Note that Concept Version is NOT explicitly separately created and maintained. This would be automatic by the system. JurisdictionalDomain allows for localization. Entered Concept Properties and Relationships become the "Defined Concept Properties / Relationships" in the model and also associated with the Code System Version in which they were created. They can also be deprecated through use of status codes.
Other relevant content	
Associated Scenario	Maintain Concept

1

2

1

2 **6.3.1.8 Update Concept Status**

Description	Changes the status of a code system concept (suspend, reinstate, cancel, remove).
Inputs	1. Code System Identifier 2. Concept Code 3. 3. Concept VersionId 4. Status
Outputs	1. An acknowledgment indicating whether the status revision request was successfully processed.
Precondition	
Postconditions	1. Code System Concept status is updated appropriately.
Exception Conditions	1. Code System Identifier does not exist. 2. Concept Code does not exist 3. Invalid state change
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Maintain Concept

3

6.3.1.9 Create Association Type

Description	Create a new relationship type (as intended by the association type class of the conceptual model), an instance of which may be used to link two concepts. A list of code system IDs can be supplied if the intent is to restrict use to specific code systems. The default is availability to all code systems present on the server.
Inputs	1. Association Type ID (optional) 2. Association Type Name 3. Association Type Properties 4. List of zero or more Code System IDs
Outputs	1. An acknowledgment indicating whether the relationship type was created or not. The ID will be generated if not supplied.
Precondition	
Postconditions	1. The relationship type is available for use via the CTS 2 service functions.
Exception Conditions	1. Association Type already exists. 2. Invalid Association Type properties 3. Unrecognized Code System ID
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	1. Creating Associations types which can span across two different code systems is permitted by the interface, but should be used with care. It is not certain if semantic Associations can be reliably created/maintained at present even when a reference terminology (to which the queried terminologies are mapped) is available. Note that given that there will often be complex business rules associated at this level, this may also be handled by configuration rather than explicit interface operation.
Other relevant content	
Associated Scenario	Create Association Type

6.3.1.10 Maintain Association Type

Description	Update or deprecate an Association type that may be used to link two concepts.
Inputs	1. Association Type ID 2. Association Type Name 3. Association Type Properties 4. List of zero or more Code System IDs 5. Association Type Status
Outputs	1. An acknowledgment indicating whether the Association type was modified or not.
Precondition	
Postconditions	1. The updated Association type is available for access via the CTS 2 service functions.
Exception Conditions	1. Association Type does not exist. 2. Invalid Association Type properties 3. Invalid Status 4. Unrecognized Code System ID
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	Status change could be a separate operation, but seems unnecessary at this "meta" level.
Other Relevant Content	
Associated Scenario	Maintain Association Type

6.3.2 Value Set Authoring/Curation

6.3.2.1 Create Value Set

Description	Create a Value Set (extensional or intensional) that is defined by a computable expression that can be resolved to an exact list of coded concepts at any given point in time.
Inputs	1. Value Set Name (optional) 2. Value Set Id (mandatory) 3. Value Set Version 4. Value Set Properties 5. Value Set Rule Set (which may be an enumeration, i.e. an extensional value set) 6. Value Set Date Lock (boolean) 7. valueOverride (optional, for extensional definitions)
Outputs	1. An acknowledgment indicating whether the value set was created or not. 2. Generated Value Set Id, if not input
Precondition	1. A sequencer to provide the value set id, if not provided in the input.
Postconditions	1. The value set is available for access via the CTS 2 service functions.
Exception Conditions	1. Value Set already exists. 2. Invalid Value Set properties 3. Unrecognized/invalid rule set
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	1. Example rule set expression: an intensional value set might be expressed as, “SNOMED CT concepts that are children of the SNOMED CT concept “Diabetes Mellitus”. Note that identification of the target code set(s) is part of the rule set. 2. When creating an intensionally defined value set, the Terminology User may or may not lock the date of the evaluation (effectively binding the value set definition to a specific version of the Code System(s) from which the concepts are being drawn). This would ensure that the value set would always resolve to

	the same set of concept codes for any given version of the value set. If the value set expression is not locked in time, then it will resolve to a different set of concept codes as the version of the Code System changes. 3. Note that the model allows for the rule set to be defined at the overall Value Set level or at the Version level, to allow for flexibility. The Service consumer needs to be able to indicate which one is appropriate. Entering a new Version Id causes creation of a new version (assuming the state model allows it at the time) The extent to which the rule set allows selection of specific designations would need to be defined (but is an implementation issue), e.g. could specify the language.
Other Relevant Content	
Associated Scenario	Create Value Set

1

6.3.2.2 Maintain Value Set

Description	Update properties or expression of a value set definition (extensional and intensional value sets).
Inputs	1. Value Set Id (mandatory) 2. Value Set Name (optional) 3. Value Set Version 4. Value Set Properties 5. Value Set Rule Set (which may be an enumeration, i.e. an extensional value set) 6. Value Set Date Lock (boolean)
Outputs	1. An acknowledgment indicating whether the value set was updated or not.
Precondition	
Postconditions	1. The updated value set is available for access via the CTS 2 service functions. 2. A new value set version is created.
Exception Conditions	1. Value Set does not exists. 1. Value Set does not exist. 2. Concept not found. Invalid Value Set properties 3. Unrecognized/invalid rule set
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	See notes on Create operation
Other relevant content	
Associated Scenario	Maintain Value Set, Maintain Value Set (meta-data)

1

2 **6.3.2.3 Update Value Set Status**

Description	Changes the status of a value set version (suspend, reinstate, cancel, remove).
Inputs	1. Value Set Identifier 2. Value Set Version 3. Status
Outputs	1. An acknowledgment indicating whether the status revision request was successfully processed.
Precondition	
Postconditions	1. Value Set Version status is updated appropriately.
Exception Conditions	1. Value Set Version Identifier does not exist. 2. Invalid state change
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Maintain Value Set

3

6.3.3 Concept Domain and Usage Context Authoring/Curation

6.3.3.1 Create Concept Domain

Description	Create a Concept Domain.
Inputs	1. Concept Domain Name 2. Concept Domain Id (optional) 3. Concept Domain Properties
Outputs	1. An acknowledgment indicating whether the concept domain was created or not. 2. Generated Concept Domain Id, if not input
Precondition	1. A sequencer to provide the concept domain id, if not provided in the input.
Postconditions	1. The concept domain is available for access via the CTS 2 service functions.
Exception Conditions	1. Concept Domain already exists. 2. Invalid Concept Domain properties
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Create Concept Domain

6.3.3.2 Maintain Concept Domain

Description	Update properties of a Concept Domain, including bindings to value sets within usage contexts
Inputs	1. Concept Domain Id 2. Concept Domain Name 3. Concept Domain Properties 4. Concept Domain Status 5. Set of zero or more pairs of Value Set Id and Usage Context Id
Outputs	1. An acknowledgment indicating whether the concept domain was updated or not.
Precondition	
Postconditions	1. The updated concept domain is available for access via the CTS 2 service functions.
Exception Conditions	1. Concept Domain does not exists. 2. Invalid Concept Domain properties 3. Invalid state transition 4. Unrecognized/invalid value set 5. Unrecognized/invalid usage context
Aspects left to Technical Specification	Support for action code required for conformance with future HL7 behavioral profile, more specification in normative edition of this spec.
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	A binding of a value set to a concept domain or usage context is a function relating a Value Set to ConceptDomains and/or Usage Contexts.
Other Relevant Content	
Associated Scenario	Maintain Concept Domain

6.3.3.3 Create Usage Context

Description	Create a Usage Context.
Inputs	1. Usage Context Name 2. Usage Context Id (optional) 3. Usage Context Properties
Outputs	1. An acknowledgment indicating whether the usage context was created or not. 2. Generated Usage Context Id, if not input
Precondition	1. A sequencer to provide the usage context id, if not provided in the input.
Postconditions	1. The usage context is available for access via the CTS 2 service functions.
Exception Conditions	1. Usage Context already exists. 2. Invalid Usage Context properties
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Create Usage Context

6.3.3.4 Maintain Usage Context

Description	Update properties of a Usage Context
Inputs	1. Usage Context Id 2. Usage Context Name 3. Usage Context Properties 4. Usage Context Status
Outputs	1. An acknowledgment indicating whether the usage context was updated or not.
Precondition	
Postconditions	1. The updated usage context is available for access via the CTS 2 service functions.
Exception Conditions	1. Usage Context does not exists. 2. Invalid Usage Context properties 3. Invalid State transition
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Maintain Usage Context

6.3.4 Association Authoring Operations

6.3.4.1 Update Association Status

Description	Update the status of a association (active, inactive, cancelled etc). This allows a Terminology User to activate or inactivate a given ssociation, thus changing its availability for access by other terminology service functions
Inputs	1. Association identifier. 2. Association version Id 3. Status
Outputs	An acknowledgment indicating whether the association status has been successfully updated.
Precondition	
Postconditions	
Exception Conditions	1. Association ID invalid or non existent 2. Status invalid or non existent
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Terminology Authoring
Miscellaneous Notes	
Other relevant content	
Associated Scenario	Create / Maintain an Association between Coded Concepts, Remove Associations

1

2 **6.3.4.2 Create Association**

Description	Relates a single specific coded concept within a specified code system (source) to a corresponding single specific coded concept (target) within the same or another code system, including identification of a specified Association type.
Inputs	1. Source code system identifier. 2. Target Code system identifier. 3. Source Concept code/identifier. 4. Target Concept code/identifier. 5. Association Type 6. Association Properties
Outputs	Association Id
Precondition	
Postconditions	A concept relationship of the specified type is created between two coded concepts
Exception Conditions	1. Source Code System does not exist. 2. Target Code System does not exist. 3. Source Coded concept not found. 4. Target Coded concept not found. 5. Unrecognized Association Type 6. Invalid Association properties 7. Info: the following association(s) already exist on the supplied concepts (list associations)
Aspects left to Technical Specification	
Relationship to Levels of Conformance	Terminology Authoring
Miscellaneous Notes	See notes under "Import Terminology Revision" for a list of sample properties for Association
Other relevant content	
Associated Scenario	Create/Maintain an Association between Coded Concepts

3

1

2 **6.3.4.3 Create Lexical Association between Coded Concepts**

Description	Relates a set of one or more coded concepts within a specified code system (source)to a corresponding set of one or more coded concepts (target) within that system or another code system using a set of lexical rules (matching algorithms) to generate the Association. The "Source Search Criteria" allows for identification of a subset of the Source Code System to apply the matching algorithm to, if required (this may include limiting the version of the code system).
Inputs	1. Source Code System Identifier 2. Target Code System Identifier 3. Source Search Criteria (optional) 4. Match Algorithm Code
Outputs	List of zero or more Association Ids (together with identification of each pair of Concepts that were linked as a result of the matching.
Precondition	
Postconditions	A set of zero or more Associations are created, each between a coded concept from the source code system and a coded concept in the target code system.
Exception Conditions	1. Source Code System does not exist. 2. Target Code System does not exist. 3. Unrecognized / Invalid algorithm. 4. Unrecognized / Invalid search criteria. 5. (Warning) No Associations matched the input algorithm.
Aspects left to Technical Specification	This may be a very time consuming operation across large code sets, and may potentially produce large result sets. Management of result set size is envisaged to require further consideration and probably additional input parameters.
Relationship to Levels of Conformance	Terminology Authoring
Miscellaneous Notes	Even though this is expressed as a "create" operation, it is essentially also a "return" operation in that it identifies which concepts are related by the input rules.
Other relevant content	

Match Algorithm Code	Description
IdenticalIgnoreCase	The lower case representation of the target text must match the lower case representation matchText exactly.
Identical	The target text must match the matchText exactly.
StartsWithIgnoreCase	The lower case representation of target text must begin with the lower case representation of matchText.
StartsWith	The target text must begin with the matchText.
EndsWithIgnoreCase	The lower case representation of the target text must end

		with the lower case representation of matchText.
	EndsWith	The target text must end with the matchText.
	ContainsPhraseIgnoreCase	The lower case representation of the target text must contain the lower case representation of the matchText.
	ContainsPhrase	The target text must contain the matchText.
	WordsAnyOrderIgnoreCase	The target text must contain all of the words in the match text, but in any order.
	WildCardsIgnoreCase	The match text may contain zero or more 'wild cards', designated by an asterisk (*). Wild cards match 0 of more characters in the target string. The escape character is a backslash('\') meaning that the matchText "a*b*' would match any string that begins with the string "a*b".
	RegularExpression	The match text may contain regular expressions, as defined in XML Schema Part 2: Datatypes.
	NYSIIS	New York State Identification and Intelligence System phonetic encoding
Associated Scenario	Create Lexical Association	

1

6.3.4.4 Create Rules Based Association between Coded Concepts

Description	Relates a set of zero or more coded concepts within a specified code system (source)to a corresponding set of zero or more coded concepts (target) within that system or another code system using a set of description logic or inference rules that either assert or infer Associations. The "Source Search Criteria" allows for identification of a subset of the Source Code System to apply the matching algorithm too, if required (this may include limiting the version of the code system).
Inputs	1. Source Code System Identifier 2. Target Code System Identifier 3. Source Search Criteria (optional) 4. Description Logic 5. Inference Rules
Outputs	List of zero or more Association Ids (together with identification of each pair of Concepts that were linked as a result of the matching.
Precondition	
Postconditions	A set of zero or more Associations are created, each between a coded concept from the source code system and a coded concept in the target code system.
Exception Conditions	1. Source Code System does not exist. 2. Target Code System does not exist. 3. Unrecognized / Invalid description logic. 4. Unrecognized / invalid inference rules. 5. Unrecognized / Invalid search criteria. 6. (Warning) No coded concepts satisfy the description logic or inference rules.
Aspects left to Technical Specification	This may be a very time consuming operation across large code sets, and may potentially produce large result sets. Management of result set size is envisaged to require further consideration and probably additional input parameters.
Relationship to Levels of Conformance	Terminology Authoring
Miscellaneous Notes	Even though this is expressed as a "create" operation, it is essentially also a "return" operation in that it identifies which concepts are related by the input rules.
Other relevant content	These Associations are subject to human review to verify validity.
Associated Scenario	Create Rules Based Association

7 Profiles

7.1 Introduction

A profile is a named set of cohesive capabilities. A profile enables a service to be used at different levels and allows implementers to provide different levels of capabilities in differing contexts. Service-to-service interoperability will be judged at the profile level and not the service level. Note that through the use of profiles, there are no “optional” interfaces. Conditions that might otherwise merit this optionality should be addressed via a dedicated profile.

NOTE : These profiles provide the "units of conformance", (see conformance profile below), and as such implementers should claim conformance to specific profiles in addition to, or rather than, the overall specification.

A set of profiles may be defined that cover specific functions, semantic information and overall conformance. The SSF explains in detail the meaning of each of these types of profile. In brief, they are as follows:

- **Functional Profile:** a named list of a subset of the operations defined within this specification which must be supported in order to claim conformance to the profile.
- **Semantic Profile:** identification of a named set of information descriptions (e.g. metamodels) that are supported by one or more operations.
- **Conformance Profile:** this is a combination of a set of functional and semantic profiles taken together to give a complete coherent set of capabilities against which conformance can be claimed. This may optionally include additional constraints where relevant.

7.2 CTS 2 Functional Profiles

7.2.1 CTS 2 Query Profile

The CTS 2 Query Profile specifies basic query only functional coverage necessary for a service to declare itself as being a minimally conformant CTS 2 service. The CTS 2 Query Profile includes capabilities for searching and query terminology content. If only this profile were supported, other means would be necessary for defining and structuring code systems and concepts, metadata, importing and exporting code systems and so on.

Profile	Member Operations	Operation Profile	Notes
CTS 2 Query Profile	List Code Systems	The ability to provide a listing of the available code systems that meet input search criteria.	The CTS 2 Query Profile specifies the minimal functional coverage necessary for a service to declare itself as being a

	Return Code System Details	The ability to retrieve a specific code system attributes (synonyms, associations) and other metadata.	conformant CTS 2 service. The CTS 2 Query Profile includes the ability to search the contents of code systems, value sets and value sets bound to Concept Domains
	List Code System Concepts	The ability to retrieve a list of all of the concepts, with associated attributes (synonyms, associations) and other metadata that meet input criteria.	
	Return Concept Details	The ability to retrieve a specific concept, with associated attributes (synonyms, associations) and other metadata.	
	List Value Sets	The ability to determine what value sets are available to a Terminology Service. This includes seeing a listing of the available value sets that match some search criteria, as well as the details pertaining to each value set available to the terminology service.	
	Return Value Set Details	The ability to retrieve a specific value set, with associated attributes and other metadata.	
	List Value Set Contents	The ability to see a listing of specific concepts, as well as the details pertaining to each concept in any of the given value sets available to a terminology service.	
	Check Concept Value Set Membership	The ability to validate that a given concept exists in a given value set.	
	List Concept Domains	The ability to determine what concept domains are available to a Terminology Service.	
	Return Concept Domain Details	The ability to retrieve a specific concept domain, with associated attributes and other metadata.	
	List Concept Domain Bindings	The ability to see a listing of specific value sets that are bound to a concept domain in specified usage contexts.	
	Check Concept	The ability to validate that a given	

	Domain Membership	concept code is bound to a given concept domain.	
	List Usage Contexts	The ability to determine what usage contexts are available to a Terminology Service.	
	Return Usage Context Details	The ability to retrieve a specific usage context, with associated attributes and other metadata.	
	List Associations	The ability to determine what associations are available on the terminology service by browsing a list of available associations on the CTS 2 instance that meet specified search criteria.	
	Return Association Details	The ability to retrieve metadata on available associations in the CTS 2 service instance.	
	List Association Types	Returns the details for the known attributes (metadata) of a coded concept	
	Return Association Type Details	The ability to return all information for a Association type.	
	Check Value Set Subsumption	Determine whether one of the two supplied value sets subsumes the other	
	Check Concept to Concept Domain Association	Determine whether the supplied coded concept exists in a code system in use for the specified concept domain, optionally within specific usage contexts.	
	Determine Transitive Concept Relationship	Determine whether there exists a transitive relationship between two concepts, if it exists	
	Compute Subsumption Relationship	Determine Whether One Concept Subsumes a Second	

1

7.2.2 Terminology Administration Profile

The Terminology Administration profile builds on the minimal profile and is intended to provide the functional operations necessary for terminology administrators to be able to access and make available terminology content obtained from a Terminology Provider. Terminology Administrators are required to interface with Terminology Provider systems in order to obtain the terminology content, then load that terminology content on local Terminology Servers.

Profile	Member Operations	Operation Profile	Notes
Terminology Administration Profile	Import Code System	Terminology content would be loaded into the terminology server as an entire terminology load or skeleton load (i.e. load of structure without loading the nodes).	The Terminology Administration profile utilizes the all of the operations defined in the Administrative Scenario section, as well as the functionality outlined in the CTS 2 Query Profile.
	Import Code System Revision	Terminology content would be loaded into the terminology server as a delta or set of changes from the previous version of the terminology.	
	Import Value Set Version	Ability to import values sets	
	Import Association version	Ability to import Associations	
	Export Association	Ability to export Association Type instances	
	Export Code System Content	Terminology content would be exported either in whole or in part based on filtering against terminology properties. The export format may also be specified.	
	Change Code System Status	Terminology content status would be changed, thus changing its availability for access by other terminology service functions.	
	Register for Notification	A client registers for notification so that an electronic notification would be sent to subscribed users in the event of a change to the specified	

	terminology element.	
Update Notification Registration	Subscription notification information can be updated for a subscriber's notification account.	
Update Notification Registration Status	Updates the status of a notification registration.	
CTS 2 Query Profile	The CTS 2 Query Profile specifies the minimal functional coverage necessary for a service to declare itself as being a conformant CTS 2 service. The CTS 2 Query Profile includes the ability to search the contents of code systems and value sets.	

7.2.3 Terminology Authoring Profile

Terminology authors require the capability to robustly query and access terminology content, as well as directly modify the terminology content. The Terminology Authoring profile is intended to provide the functional operations necessary for terminology authors to analyze the existing terminology content, as well as directly edit terminology content.

Profile	Member Operations	Operation Profile	Notes
Terminology Authoring Profile	Create Code System	The ability to create a new Code System to contain a set of new coded concepts. The Code System is created by defining the set of meta-data properties that describe it.	The Terminology Authoring Profile is intended to provide the capability to robustly query and access terminology content, as well as directly modify the terminology content. This includes the ability to modify code system content, value set content, as well as the metadata pertaining to each. This profile includes the functions necessary to administer and search terminology content as outlined in the CTS 2 Query Profile as well as the
	Maintain Code System Version	The ability to maintain the content and metadata of a version for a code system.	
	Update Code System Version Status	The ability to modify the status of a code system.	

	Create Concept	The ability to define and add a new concept to a code system.	Terminology Administration Profile
	Maintain Concept	The ability to modify a concept that exists in a code system.	
	Update Concept Status	The ability to modify the status of a concept that exists in a code system.	
	Create Value Set	The ability to create a dynamic value set that is defined by a computable expression that can be resolved to an exact list of coded concepts at any given point in time.	
	Maintain Value Set	Update properties or expression of a value set definition (extensional and intensional value sets).	
	Update Value Set Status	The ability to modify the status of a value set.	
	Create Concept Domain	The ability to define and add a new concept domain.	
	Maintain Concept Domain	The ability to modify a concept domain, including bindings to value sets within usage contexts.	
	Create Usage Context	The ability to define and add a new usage context.	
	Maintain Usage Context	The ability to modify a usage context.	
	Terminology Administration Profile	The Terminology Administration profile is intended to provide the functional operations necessary for terminology administrators to be able to access and make available terminology content obtained from a Terminology Provider.	

	Create Association	The ability to create an association between concepts.	
	Update Association Status	The ability to update the status of an association between concepts.	
	Create Association Type	The ability to create a new Association type that may be used to link two concepts.	
	Maintain Association Type	The ability to modify or deprecate an existing Association type that may be used to link two concepts.	
	Create Lexical Association Between Coded Concepts (optional for this profile)	The ability to instantiate an association between two sets of coded concepts using a set of lexical rules (matching algorithms) to generate the associations .	
	Create Rules Based Association Between Coded Concepts (optional for this profile)	The ability to instantiate an association between two sets of coded concepts using a set of description logic or inference rules that either assert or infer mappings between two Code Systems.	
	Create Code System Supplement	Create a new Code System Supplement as a container of a set of concepts and concept properties to be appended to a target code system	
	Maintain Code System Supplement	Update Code System Supplement meta-data properties and add concepts and properties to code system	

7.3 CTS 2 Semantic Profiles

Semantic profiles are created to group together vocabularies with similar designs. Vocabularies grouped under a single semantic profile can be queried using the same functional variants of CTS2 functions. A semantic profile can be seen as a superset of the terminology metamodels of the supported terminologies with information model classes as the elements of the sets. This approach provides the following advantages:

- It allows the CTS2 author to focus on a set of design attributes of terminologies and support them using functional variants, rather than having to focus on individual terminologies while authoring the standard.

- It allows the implementer to implement functional variants of CTS2 functions based on the semantic profiles they want to support rather than to create or implement functional variants for the terminologies that are to be supported by their implementation.

- It allows terminology authoring organizations to classify their terminology under a semantic profile and insulates them from the complexities of functional variants of CTS2 functions.

When conformance against a semantic profile is claimed, the implementer has to list typical terminologies which are supported by the CTS2 implementation.

These intrinsic qualities of terminologies allow the functional profiles to be implemented in accordance with the properties of the classes of these terminologies. The following Semantic Profiles for terminologies are defined currently:

7.3.1 HL7 Terminology Profile

Profile	Sample Terminology Criteria	Sample Terminologies Classified Under Profile	Notes
HL7 Terminology Profile	• The HL7 V3 Terminology	• HL7	The HL7 Terminology Profile specifies an implementation specific to the interoperability requirements outlined by HL7 Version 3. It outlines the specificity necessary for a service to declare itself as being an HL7 conformant CTS 2 service. The HL7 Profile includes capabilities for searching terminology content, representing terminology content in a form supporting the structures required by and specific HL7v3.

7.3.2 Mature Terminology Profile

Profile	Sample Terminology Criteria	Sample Terminologies Classified Under Profile	Notes

Mature Terminology Profile	• Unique identifiers for all concepts • Unique identifiers for all designations • Unique identifiers for all relationships • Identifiers are never reused.	• SNOMED CT, all versions • ICD 10 CM • LOINC • RxNorm • MEDCIN • NDF / NDF-RT • CPT	Terminologies in the Mature Terminology Profile make an attempt to conform to many of terminology best practices that are, for example outlined in *Desiderata for Controlled Medical Vocabularies in the Twenty-First Century, James J. Cimino*.

7.3.3 Developing Terminology Profile

Profile	Sample Terminology Criteria	Sample Terminologies Classified Under Profile	Notes
Developing Terminology Profile	• Codes that are not globally unique within the terminology but need further qualification, e.g. an additional identifier. • The concepts can be uniquely identified by combining the concept code and an additional identifier.	• Some HL7 Vocabulary tables (where the additional identifier may refer to a concept domain) • Some of the locally developed terminology sources or code sets	Terminologies in the Developing Terminology Profile are either developed using ad hoc techniques, or have degraded over time.

7.4 CTS 2 Conformance Profiles

7.4.1 Conformance Interoperability

The capabilities defined within the CTS 2 service functional model have been attributed to different functional profiles. The purpose of functional profiles is to group together functions to form cohesive levels of operational capability against which implementations can be tested for conformance. Thus, interoperability between CTS 2 implementations is assured within a specified conformance profile. In other words, two CTS 2 implementations that conform to the conformance profile defined by the Terminology Authoring and the

Mature terminology profiles will be able to interoperate using the functions described in the resulting cross-product of those profiles because they will support the same operations applied to the same superset of terminology metamodels (which is the semantic profile). The more profiles an implementation complies with, the more powerful it will be, i.e. it will be able to interoperate with all operations of implementations which are conformant against less profiles; however, the latter will not be able to interoperate with all operations of the former.

These profiles serve to educate the purchasing and implementation communities, allowing for implementation variation while still promoting interoperability. Service Level Agreements made between organizations are then testable because they are informed by these profiles. Governance of these agreements is less ambiguous and more enforceable due to precise functional levels of interoperability that may be expected.

Implementation of this functional specification should explicitly deal with the different interoperability roles that CTS 2 may fill using these conformance profiles. The business rules enforced by an organization's purchasing, implementation, and governance arms should be discussed, and the ways in which CTS 2 facilitates that enforcement should be made clear.

7.4.2 Conformance Assertion

Implementations of CTS 2 conform to a specified conformance profile, which is a combination of a functional and semantic profile. That is, conformance to a specific profile is asserted to against the quality metric of a specified semantic profile in association with the specified functional profile.

There are currently three different functional profiles defined, where each functional profile can be implemented according to ay of three semantic profiles. The available semantic profiles are identified as the **Mature Terminology**, **Developing Terminology** or **HL7 Terminology** semantic profiles, providing up to nine different conformance profiles to CTS 2. Implementers wishing to conform with the HL7 terminology semantic profile should

	Mature Terminology Semantic Profile	**Developing Terminology Semantic Profile**	**HL7 Terminology Semantic Profile**
CTS 2 Query Functional Profile	CTS 2 Query - Mature Terminology Conformance Profile	CTS 2 Query - Developing Terminology Conformance Profile	CTS 2 Query - HL7 Terminology Conformance Profile
Terminology Administration Functional Profile	Terminology Administration - Mature Terminology Conformance Profile	Terminology Administration - Developing Terminology Conformance Profile	Terminology Administration - HL7 Terminology Conformance Profile
Terminology Authoring Functional Profile	Terminology Authoring - Mature Terminology Conformance Profile	Terminology Authoring - Developing Terminology Conformance Profile	Terminology Authoring - HL7 Terminology Conformance Profile

8 The Services Framework Functional Model

The Services Framework Functional Model identifies common underlying enterprise infrastructure such as naming, directory, security, etc. that may be assumed and referenced by this Functional Model.

Note that the Services Framework Functional Model is being developed in parallel with other service Functional Models; candidate functionality for the Framework should be submitted to the HL7 SOA WG or the ArB for evaluation.

CTS 2 compliant service instances are intended to be middleware services, and operate within the context of supporting infrastructure services that may exist within an enterprise. As a result, a number of underpinning capabilities have been intentionally omitted from the scope of this specification. These include (but are not limited to) capabilities such as meta data provision and management, identity management, security and record location services.

The CTS 2 specification, by design, can be used as a means to integrate a new capability into a service-oriented architecture, or can be used to provide a service interface to access content in legacy applications. It is not intended as a replacement of any single system, but instead to act as a companion component that facilitates interoperability with data sharing partners through a standardized set of APIs.

CTS 2 serves as a simplifying resource for the organization, as it can provide a single point of access for all terminology resources.

9 Relationship to Information Content

The following principles shall be followed for specifying the information model to be used by the services being specified in this Service Functional Model:

1. PIMs shall provide a conformance profile supporting HL7 content where relevant
2. We shall not preclude the use of non-HL7 content
3. PIMs will reuse to the maximum extent possible the content models as defined in other standards (for example, HL7 RMIMs)
4. Information content representations shall be represented in platform-agnostic formalisms (e.g., UML)
5. PIMs may identify content at varying levels of granularity, depending upon the functions being specified. (For example, the Common Terminology Service will deal with different granularity of information than the Resource Location and Update Service).
6. Conformance Profiles may be balloted or adopted after the release of the initial SFM to address specialized business needs. (realm-specific profiles, domain-specific profiles, etc.)
7. Details about semantics specific to this SFM appear in other sections of this document

10 Recommendations for Technical RFP Issuance

This section includes Identification of topics requiring elaboration in candidate solutions provided through the OMG RFP process (http://www.omg.org/gettingstarted/processintro.htm). These may be service-specific, deployment related, or non-functional. In this section, responders to an OMG RFP for this SFM will be referred to generically as the *responder*.

10.1 HL7 Support

As an HL7 specification, CTS 2 implementation claiming conformance against HL7 profiles are required to support the use of terminology in HL7 environments, including access of terminology content in HL7 models, HL7 Messages, and access of externally developed terminology content. In the absence of a fully specified HL7 semantic profile in this DSTU, PIM developers wishing to claim conformance against the HL7 semantic profile are requested to study the MIF vocabulary model as a surrogate for this profile and derive an HL7 semantic profile.

10.2 Metamodels: Disparate Terminologies

While defining the semantics of payloads sent through CTS 2 is beyond the scope of this publication, the ability of CTS 2 to notify a service partner about the nature of the capabilities of that implementation of CTS 2 is essential to fulfilling terminology service interoperability.

CTS 2 could conceivably be used to access and maintain a great variety of terminology sources, including SNOMED, ICD, and RxNorm (to name a few). To create true terminological interoperability between organizations it is essential to provide a scalable and extensible terminology model that can be included in the description of and access to the terminology resources available on any given terminology service.

Though a limited number of metamodels have been included in this document as a mechanism of defining the necessary behaviors of a terminology, it is expected that HL7, HL7 member organizations, terminology providers, and terminology users will be producing representations that will be supported within a given CTS 2 implementation.

Responders will discuss how Metamodels for Disparate terminologies are structured in their implementation of CTS 2.

10.2.1 Metamodels: HL7 Terminologies

Where terminology content exposed through CTS 2 is from an HL7 domain it is necessary to include support for Concept Domains, Binding Realms and context bindings.

RFP submitters should take the requirement for Domain and Binding Realm descriptions as a starting point to discuss the additional physical information descriptions. The usage of the two should be described and modeled so as to paint a complete picture of the issue of semantic description and discovery through the CTS 2 interface.

Additionally, metamodels should allow for the use of logical operators in describing their hierarchy or aggregation. For example, Boolean Operators (AND, OR, NOT) should be available in creating query parameters.

Responders will discuss how metamodels for HL7 terminologies are structured in their implementation of CTS 2.

10.3 Conformance Profiles and Service Level Agreements

The capabilities defined within the CTS 2 SFM have been attributed to specific conformance profiles. The purpose of a conformance profile is to group together functions to form cohesive levels of operational capability against which implementations can be tested for conformance. Thus, interoperability between CTS 2 implementations is assured within a conformance profile. In other words, two CTS 2 implementations that conform to the Authoring profile will be able to interoperate using the functions outlined in that profile.

These profiles serve to educate the purchasing and implementation communities, allowing for implementation variation while still promoting interoperability. Service Level Agreements made between organizations are then testable because they are informed by these profiles. Governance of these agreements is less ambiguous and more enforceable due to precise functional levels of interoperability that may be expected. Implementation of this functional specification should explicitly deal with the different interoperability roles that CTS 2 may fill using these conformance profiles.

The responder should discuss the business rules enforced by an organization's purchasing, implementation, and governance arms, as well as the ways in which CTS 2 facilitates that enforcement.

10.4 Operationalizing CTS 2: Considerations in Implementation

10.4.1 Optimization

Structured terminologies can be quite large in nature in both the number of concepts, designations, associations, and other attributes that further describe terminology content. As such, efficiently accessing and querying terminology content is critical.

Responders to the RFP should discuss optimization strategies for accessing and updating specific terminologies.

10.4.2 Versioning

10.4.2.1 Versioning Mechanisms

To ensure consistent representation of vocabulary versions, the CTS 2 committee recognizes the following approaches to versioning vocabulary entities:

- A vocabulary specification may use a numeric or decimal representation of the version number, that is, the version identifier may be that string containing only numeric digits and decimal points.
- A vocabulary specification may use a release date to represent the version.

Responders to the RFP should contrast these two versioning mechanisms, and discuss how each are supported by the proposed solution (possibly as collaborative mechanisms), including the recommended string and date formats respectively.

10.4.2.2 Versionable Elements

Many different terminology elements are subject to change and as such need to be versioned. Examples of terminology versioning use cases include:

- Get the current state of a terminology element (the default behavior today).
- Get the state of a terminology element on date **D**.
- Get the state of a terminology element in Version **V**.
- Get a trace (date/versions) of terminology element changes.
- Get a trace of terminology element attribute changes, e.g. a specific Property Type.

Responders to the RFP will outline the specific versioning mechanisms they intend on supporting, as well as what types of changes constitute a new version of a terminology element.

10.4.3 Internationalization

Responders to the RFP will discuss what effect, if any, localization and internationalization of terminologies will have on technical implementations of CTS 2?

10.5 Service Description and Discovery

Because CTS 2 exists as a service between organizations, CTS 2 should be considered a perfect candidate to benefit from service description and discovery, such as what terminologies are available on any given CTS 2 implementation, ad the specific profiles implemented by that service implementation.

Responders to the RFP should explicitly discuss this deployment case, how to better describe CTS 2 to improve service discovery.

10.6 Federated Terminology Services

As implementers strive to organize CTS 2 within and between institutions, it is likely that a federation of terminology sources and terminology servers will develop. These service interfaces will be a set of CTS2 service instances, which may be responsible of serving different terminology resources. They are likely to occupy various information and domain levels within and between organizations. Common federation patterns are likely to emerge, such as a mesh or a hierarchical structure. However, other deployment scenarios are desirable as well. Special attention should be paid to implementation in a non-homogeneous environments.

Responders to the RFP discuss how the implementation would support federated terminologies, and how it would allow for a hierarchical service topology to satisfy most deployment requirements.

10.7 Terminology Structure Considerations

Section 2.4.2 of this document describes a logical (analysis level) model with the intent that most - if not all - key attributes that affect terminology behavior have been described. This model represents the minimal classes, attributes and associations necessary to represent conceptual terminologies.

Responders to the RFP should provide a detailed Platform Independent Model (PIM) model that can represent terminology sources that adhere to terminology best practices. Such a model can be seen as a superset of the metamodels of the supported terminologies. The PIM must be rich enough to support different terminologies like:

1. SNOMED CT.The International Health Terminology Standards Development Organization is an international not-for-profit organization based in Denmark. IHTSDO acquires, owns and administers the rights to SNOMED CT and other health terminologies and related standards. The purpose of IHTSDO is to develop, maintain, promote and enable the uptake and correct use of its terminology products in health systems, services and products around the world, and undertake any or all activities incidental and conducive to achieving the purpose of the Association for the benefits of the members. The responder will discuss how their implementation will specifically manage SNOMED CT, and discuss potential or actual integration points with the IHTSDO Workbench.
2. LOINC - Logical Observation Identifiers Names and Codes (LOINC) is a database and universal standard for identifying medical laboratory observations. It was developed and is maintained by the Regenstrief Institute, Inc., a US non-profit medical research organization, in 1994. LOINC was created in response to the demand for an electronic database for clinical care and management and is publicly available at no cost. It is endorsed by the American Clinical Laboratory Association and the College of American Pathologist. Since its inception, the database has expanded to include not just medical and laboratory code names, but also: nursing diagnosis, nursing interventions, outcomes classification, and patient care data set.
3. MedDRA - MedDRA or Medical Dictionary for Regulatory Activities is a clinically validated international medical terminology used by regulatory authorities and the regulated

biopharmaceutical industry throughout the entire regulatory process, from pre-marketing to post-marketing activities, and for data entry, retrieval, evaluation, and presentation. In addition, it is the adverse event classification dictionary endorsed by the International Conference on Harmonization of Technical Requirements for Registration of Pharmaceuticals for Human Use (ICH). MedDRA is used in the US, European Union, and Japan. Its use is currently mandated in Europe and Japan for safety reporting. MedDRA is managed by the MSSO (Maintenance and Support Services Organization), an organization that reports to the International Federation of Pharmaceutical Manufacturers and Associations (IFPMA). MedDRA is free for regulators and priced according to company revenue for industry. The FDA has committed to keeping current on MedDRA, and it has become the standard for adverse event reporting in the USA.

The PIM should also discuss a strategy for representing less mature terminologies in a format that allows them to be consistently accessed by the appropriate CTS 2 functions in accordance with the required semantic profiles.

10.8 Post coordination

Many applications dealing with complex terminologies are confronted with the problem of post-coordinated terms. Essentially, these are composite concepts created by combining atomic concepts of a code system according to certain semantic and syntactic rules. Example: expression of the pre-coordinated term hypophysectomy as {pituitray|excision}. Responders to the OMG RFP claiming conformance against a semantic profile comprising terminologies which use post coordination are obliged to comment on how they want to design operations affected by this problem (e.g. List Concepts with a post-coordinated term as input search parameter).

10.9 Terminology Maintenance

CTS 2 specifies a set of service functions specific to the maintenance of terminology sources. These service functions in and of themselves are expected to influence a broader set of application level functions that comprise a terminology maintenance solution. Such terminology maintenance solutions would include workflow and process that would not only allow terminology providers to efficiently and accurately maintain the terminology content, but may also include the ability for terminology users to submit, review and modification of terminology content *change requests*. These change requests would be considered by the terminology providers for inclusion into the next version of the terminology.

Responders to the RFP should discuss how CTS 2 query and maintenance functionality would be included in a broader terminology maintenance solution that includes workflow and change request processing.

10.10 Matching Algorithms

The match algorithm list shown below (from the original CTS specification) is not exhaustive. It is permissible for service implementations to extend the list below with additional, custom match algorithms as appropriate, although implementers are strongly encouraged to register the algorithm code to help ensure interoperability. However, all of the listed matching algorithms with the exception NYIIS are normative and have to be implemented to claim conformance to this specification.

Table 1

Match Algorithm Code	Description
IdenticalIgnoreCase	The lower case representation of the target text must match the lower case representation matchText exactly.
Identical	The target text must match the matchText exactly.
StartsWithIgnoreCase	The lower case representation of target text must begin with the lower case representation of matchText.
StartsWith	The target text must begin with the matchText.
EndsWithIgnoreCase	The lower case representation of the target text must end with the lower case representation of matchText.
EndsWith	The target text must end with the matchText.
ContainsPhraseIgnoreCase	The lower case representation of the target text must contain the lower case representation of the matchText.
ContainsPhrase	The target text must contain the matchText.
WordsAnyOrderIgnoreCase	The target text must contain all of the words in the match text, but in any order.
WildCardsIgnoreCase	The match text may contain zero or more 'wild cards', designated by an asterisk (*). Wild cards match 0 or more characters in the target string. The escape character is a backslash('\') meaning that the matchText "a*b*' would match any string that begins with the string "a*b".
RegularExpression	The match text may contain regular expressions, as defined in XML Schema Part 2: Datatypes.
NYSIIS	New York State Identification and Intelligence System phonetic encoding

11 Appendix A - Relevant Standards

11.1 HL7 Common Terminology Services

The Common Terminology Services (CTS) specification was developed as an alternative to a common data structure. The HL7 Common Terminology Services (HL7 CTS) is an Application Programming Interface (API) specification that is intended to describe the basic functionality that will be needed by HL7 Version 3 software implementations to query and access terminological content. It is specified as an API rather than a set of data structures to enable a wide variety of terminological content to be integrated within the HL7 Version 3 messaging framework without the need for significant migration or rewrite. Instead of specifying what an external terminology must look like, HL7 has chosen to identify the common functional characteristics that an external terminology must be able to provide. As an example, an HL7 compliant terminology service will need to be able to determine whether a given concept code is valid within the particular resource. Instead of describing a table keyed by the resource identifier and concept code, the CTS specification describes an Application Programming Interface (API) call that takes a resource identifier and concept code as input and returns a true/false value. Each terminology developer is free to implement this API call in whatever way is most appropriate for them. There are two layers between HL7 Version 3 message processing applications and the target vocabularies. The upper layer, the Message API communicates with in terms of vocabulary domains, realms, coded attributes and other artifacts of the RIM and HL7 messaging model. The lower layer, the Vocabulary API communicates in terms of coding system, concept codes, designations, and other vocabulary related entities.

The CTS 2 specification is an extension of the original HL7 Common Terminology Services approved standard, and as such played a key role in the development and design of CTS 2.

12 Appendix B – Glossary

12.1 Terminology Principles

The following Glossary provides definitions of key vocabulary terms used in this service specification. These definitions are compatible with definitions used in HL7, but can be seen as hyperonyms **1)** of these as they cover a broader spectrum of vocabulary user requirements than those of the HL7 users in order to meet the requirements of the entire CTS2 service user community. Is the intent that these terms align in the broadest sense with those outlined in the HL7 Core Principles document.

Footnote 1) e.g. the designation "jurisdictional domain" can be seen as a hyperonym of the HL7 term "realm" because the former covers a broader semantic spectrum than the latter, and "realm" is clearly its hyponym.

12.1.1 Code System

A *Code System* is a managed collection of concept identifiers, usually codes, but sometimes more complex sets of rules and references. They are often described as collections of uniquely identifiable concepts with associated representations, designations, associations, and meanings. Examples of *Code System*s include ICD-9 CM, SNOMED CT, LOINC, and CPT. To meet the requirements of a *Code System* as defined by HL7, a given concept representation must resolve to one and only one meaning within the *Code System*. In the terminology model, a *Code System* is represented by the *Code System* class.

NOTE: For the purposes of this document, the terms code system; terminology and vocabulary are treated as synonyms.

12.1.2 Code System Concept

A Code System Concept defines a unitary mental representation of a real or abstract thing within the context of a specific Code System; an atomic unit of thought. Generic representations of concepts outside of the bounds of a code system are not included in the model, although their effect can be approximated by setting up Associations (maps) across code systems. Concepts should be unique within a given code system, but may have synonyms in terms of both the codes used (e.g. “l” and “L” in UCUM for Liter) and in textual representation (as Designations). Concepts may be simple or compositional in nature. A compositional concept is one that contains more than one concept concatenated within it – for example “severe hypertension” – a combination of “hypertension” and a qualifier for severity. Each CodeSystem will have a set of Concepts associated with it. Terminology best practices dictate that concepts are not deleted from code systems, but are instead deprecated or retired from use, although nothing in the model prevents this.

12.1.3 Concept

A *Concept* defines a unitary mental representation of a real or abstract thing; an atomic unit of thought. It should be unique in a given *Code System*. A concept may have synonyms in terms of representation and it may be a primitive or compositional term

Concepts as abstract, designation-independent representations of meaning are important for the design and interpretation of HL7v3 models. They constitute the smallest semantic entities on which HL7v3 models are built. The authors and the readers of a model use concepts and their relationships to build and understand the models; these are what matters to the human user of HL7v3 based models. The rest of the vocabulary machinery exists to permit software manipulation of these units of thought.

12.1.4 Concept Domain

An HL7 Concept Domain is a named category of like concepts (a semantic type) that will be bound to one or more attributes in a static model whose datatypes are coded. Concept Domains exist to constrain the intent of the coded element while deferring the association of the element to a specific coded terminology until later in the model development process. Thus, Concept Domains are independent of any specific vocabulary or code system.

Concept Domains represents an abstract conceptual space such as "countries of the world", "the gender of a person used for administrative purposes", “languages of the world”, etc.

12.1.5 Association

Associations define binary relationships or linkages between concepts. The endpoints of an association are source and target concepts, often implying a direction of the association from a source to a target, which has bearing on the meaning of the association and the concepts it connects. An association is definitional, in that the association gives meaning to the concepts associated. For example, an association between a parent and child concept indicates that the child concept is a refinement or an example of the parent concept in a concept hierarchy. An association can also define other characteristics of a concept, as in associations between concepts in different parent-child hierarchies where the child may have a different set of associations than that one or more of its hierarchical parents.

For example, in SNOMED CT, the concept of “pneumonia” has an “is-a” association to the concept of “lung consolidation,” and “lung consolidation” has an “is-a” association to the concept of “disorder of lung.” This represents the logical conclusion that “pneumonia” is a “disorder of lung.”

In the case of Concept Maps (where the source and target concepts are from different code systems) the direction and designation of the association have similar restrictions, except in the case where the Concept Map indicates semantic equivalence. The equal association in this case obviates the requirement for interpreting the association direction. An association links a source Concept to a target Concept. As with DefinedConceptProperty, there is a separate Concept Version level representation (CodeSystemVersionConceptAssociation) to identify the Associations supported within a specific version of a Code System.

12.1.6 Designations

Designations are representations of concepts. The designation identifier must uniquely map to a given text string, bitmap, etc. within the context of the containing concept. In some terminologies, every unique text string will have exactly one identifier, which means that the same identifier may occur under more than one concept. In other terminologies, there may be more than one identifier for a given text string, meaning that the identifier is unique to the concept. Service software must not assume either model. For example, in SNOMED CT, the concept of "fever" has the fully specified name of "fever (finding)," a preferred name of "fever," and synonyms of "febrile" and "pyrexia." These are all designations for the concept of "fever."

12.1.7 Binding Realm

In HL7, the broadest binding context is the *Binding Realm*[3]. All model instances must declare a particular *Binding Realm* (or sub-*Binding Realm*) based on the jurisdiction from which they originate, for which they are destined, or for some third jurisdiction by site-specific agreement. The declared *Binding Realm* applies to the entire model or specification artifact: it is not specific to individual elements of that model or artifact.

A *Binding Realm* refers to a named interoperability conformance space, meaning that all static models within a particular *Binding Realm* share the same conformance bindings. In nontechnical terms, it can be considered a dialect where speakers use the semantics of the language but agree to use certain terms that are specific to their community. A *Binding Realm* has a unique code: the *Binding Realm* Germany has a code of DE, and the steward is HL7 Germany. In order to enable conformance, the name of the *Binding Realm* is carried in the model instance.

In the interest of maximizing interoperability, interoperability spaces should be as large as possible: *Binding Realm*s are preferred to be large-grained. A *Binding Realm* is used to provide and manage the bindings of *Value Set*s to reflect rules within a conformance space—e.g., a country.

12.1.8 Jurisdictional Domain

A Jurisdictional Domain identifies any body that may define and manage its own code systems or concepts, including localization of a broader code system. It is specifically to allow for localization of certain concept elements. A Jurisdictional Domain could be a country, group of countries, a territory (e.g. state), an SDO, an individual organization or even department within an organization.

Jurisdictional Domain is intended to encompass the HL7 concept of **Realm**, however is broader in scope than an HL7 Realm. HL7 rules prohibit new codes being added to a code system locally, but do allow for additional concept relationships, concept properties and designations. (However, any organization could use the same model, interfaces etc. to define its own code systems for internal use.) This class provides the link to those classes to enable the localization to be recorded and managed.

[3] A Binding Realm is represented as a JurisdictionalDomain in the CTS2 model

12.1.9 Value Set

A *Value Set* represents a uniquely identifiable set of valid concept representations, where any concept representation can be tested to determine whether or not it is a member of the value set.

Value set complexity may range from a simple flat list of concept codes drawn from a single code system, to an unbounded hierarchical set of possibly post-coordinated expressions drawn from multiple code systems.

A value set has a definition, which describes a set of codes referencing a collection of unique concept identifiers, and can be resolved to an expansion, which is a set of concept designations defined by the concept identifier. The collection of unique identifiers referenced by a value set are drawn from one or more code systems. Each of these identifiers is represented by a **code**.

Value sets can be specified in two ways, either by enumeration (*extension*), or definition (*intention*).
Extensional Value Set Representation (Enumeration)

An extensionally defined, enumerated set consists of a list of unique identifiers and the corresponding globally unique id. Whenever possible, it is strongly recommended that the value set identifier match the code used in the code system itself.

Note, that while all of the entries in a value set may be valid at the time that the value set was defined, it is quite possible that subsequent versions of a code system to retire some of the codes in the value set. For this reason, it is important that both explicit and implicit undergo the “value set binding” set described in a subsequent section.

Value sets defined by extension are comprised of an explicitly enumerated set of codes. The simplest case is when the value set consists of only one code.

An intensional value set definition describes the contents of a value set in a way that the definition can be resolved (ideally computationally) to a set of permissible values and unique identifiers. While the construction rules can potentially be quite elaborate and possible specific to individual coding schemes, HL7 has identified a core set of rules that appear to be useful in most circumstances. For the sake of value set definition interoperability, HL7 strongly recommends that these algorithms be used whenever possible.

For example, an intensional value set definition might be defined as, “ SNOMED CT concepts that are children of the SNOMED CT concept “Diabetes Mellitus.”

Implicit definitions can specify:

- All active unique identifiers from a given coding system.
- All unique identifiers that participate in a specified relationship with a given local code in a coding system, which may or may not include the specified code itself.
- The transitive closure of a specified transitive relationship with a given code, including or excluding the code itself.
- A reference to another value set

- Other mechanisms that have to have external human or computational resolution.
- Unions, intersections and exclusions of any of the above
- Nested value set definition in which a value set entry references another value set (a child value set). There is no preset limit to the level of nesting allowed within value sets. Value sets cannot contain themselves, or any of their ancestors (i.e., they cannot be defined recursively). Nested value sets are always intensionally defined in HL7.