LexEVS 6.1 Design Document - Detailed Design - CTS2 REST Services

Contents of this Page

- 1 Architecture Design
- 2 Model View Architecture (MVC)
  - 2.1 Model
  - 2.2 View
  - 2.3 Controller
- 3 Logical View
- 4 Software Architecture
  - 4.1 High Level Structure
  - 4.2 Basic Request Sequence
  - 4.3 Independent Module Structure
    - 4.3.1 CodeSystemVersionReadService
    - 4.3.2 CodeSystemVersionQueryService
    - 4.3.3 EntityDescriptionReadService
    - 4.3.4 EntityDescriptionQueryService
    - 4.3.5 AssociationQueryService
    - 4.3.6 MapReadService
    - 4.3.7 MapQueryService
    - 4.3.8 ValueSetReadService
    - 4.3.9 ValueSetQueryService
    - 4.3.10 ValueSetDefinitionReadService
    - 4.3.11 ValueSetDefinitionQueryService
- 5 URI Resolution
- 6 Glossary

Document Information

Author: Scott Bauer, Kevin Peterson
Email: bauer.scott@mayo.edu, peterson.kevin@mayo.edu
Team: LexEVS

Contract: ST12-1106
Client: NCI CBIIT
National Institutes of Heath
US Department of Health and Human Services

Revision History

Version	Date	Description of Changes	Author
1.0	2013/02/21	Initial Version	Peterson, Kevin Bauer, Scott

1 Architecture Design

The CTS2 Development Framework is a development kit for rapidly creating CTS2 compliant applications. The Development Framework allows for users to create plugins which may be loaded into the Development Framework to provide REST Web Services that use CTS2 compliant paths and model objects. Because the Development Framework is plugin based, users are required only to implement the functionality that is exclusive to their environment. For example, in any REST service, a large portion of code must be written to accept HTTP requests, handle errors, accept parameters, etc. The Development Framework provides all of this infrastructure, as well as utilities to help create plugins. The developer then is left to create the implementation plugin, which is the code specific to the individual environment. For example, this code may retrieve data from a database, read data from a file system, aggregate two more other services, etc.

CTS2 is modeled abstractly (the Platform Independent Model or 'PIM') and for specific platforms (the Platform Specific Model or 'PSM'). The CTS2 Development Framework attempts to transform these ideas into the MVC architecture style. Much like a traditional MVC implementation, multiple 'Views' expose the 'Model' with the help of the 'Controller'. If we think of a 'PSM' as a 'View', and the CTS2 Development Framework acting as the 'Controller', a developer needs only to produce the 'PIM' based 'Model'. Going further:

2 Model View Architecture (MVC)

2.1 Model

Transforms View (CTS2 PIM) structures into state (aka “backing store”)
Enforces post-conditions
May also enforce some invariants

2.2 View

Implements the static portion of the CTS2 model
(Indirectly) enforces some invariants

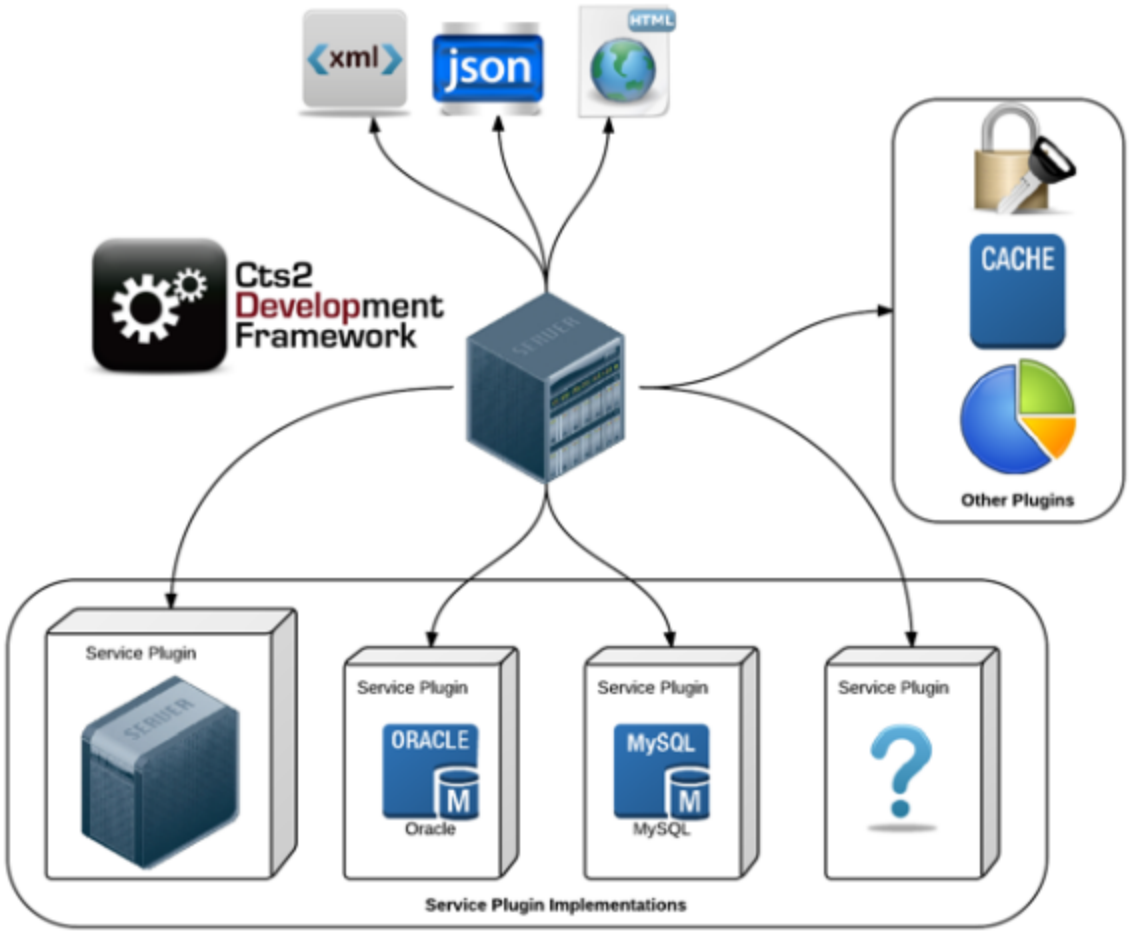
2.3 Controller

Implements the behavioral portion of the CTS2 model
Accepts events
Validates invariants
Enforces preconditions

3 Logical View

The CT2 Development Framework (CTS2DF) is a Plugin-based Java toolkint. It is built on the OSGi framework, which allows it to adopt a modular architecture. At the center, the CTS2DF is a web application, capable of being deployed to a web application server such as Tomcat , Jetty , WebSphere , etc. At the core, the CTS2DF starts an Apache Felix OSGi environment. This allows the OSGi framework to be self-contained, eliminating the need for special application servers to be used.

After initialization, the CTS2DF will use Service Plugin Implementations (or OSGi bundles) to implement the CTS2 functionality. These Service Plugins conform to a standard CTS2 interface, and are intended to be implemented according to the requirements of the implementer or organization. Furthermore, additional (non-CTS2) plugins can be registered to the service to provided extra functionality, such as security or caching. An added benefit of these plugins is that they can be shared across CTS2 implementations. Ultimately, the CTS2DF process the results of calls to these Service Plugins and transforms the responses into CTS2 compliant XML, JSON, or HTML.



4 Software Architecture

4.1 High Level Structure

The main responsibility of the CTS2DF is to accept incoming HTTP requests and delegate to the appropriate Service Plugin implementations. At a high level, the logic of accepting an incoming request is as follows:

```
START

   WHILE TRUE:

      ACCEPT HTTP REQUEST

      SERVICE_INTERFACE = ServiceProvider.getService()  #Ask the ServiceProvider for an Implementation

      IF SERVICE_INTERFACE == NULL

         THEN: RETURN NOT_IMPLEMENTED_ERROR             #If not found, assume CTS2 Service is not Implemented

         ELSE: SERVICE_INTERFACE.execute()              #If matching implementation is found, execute it

      END IF

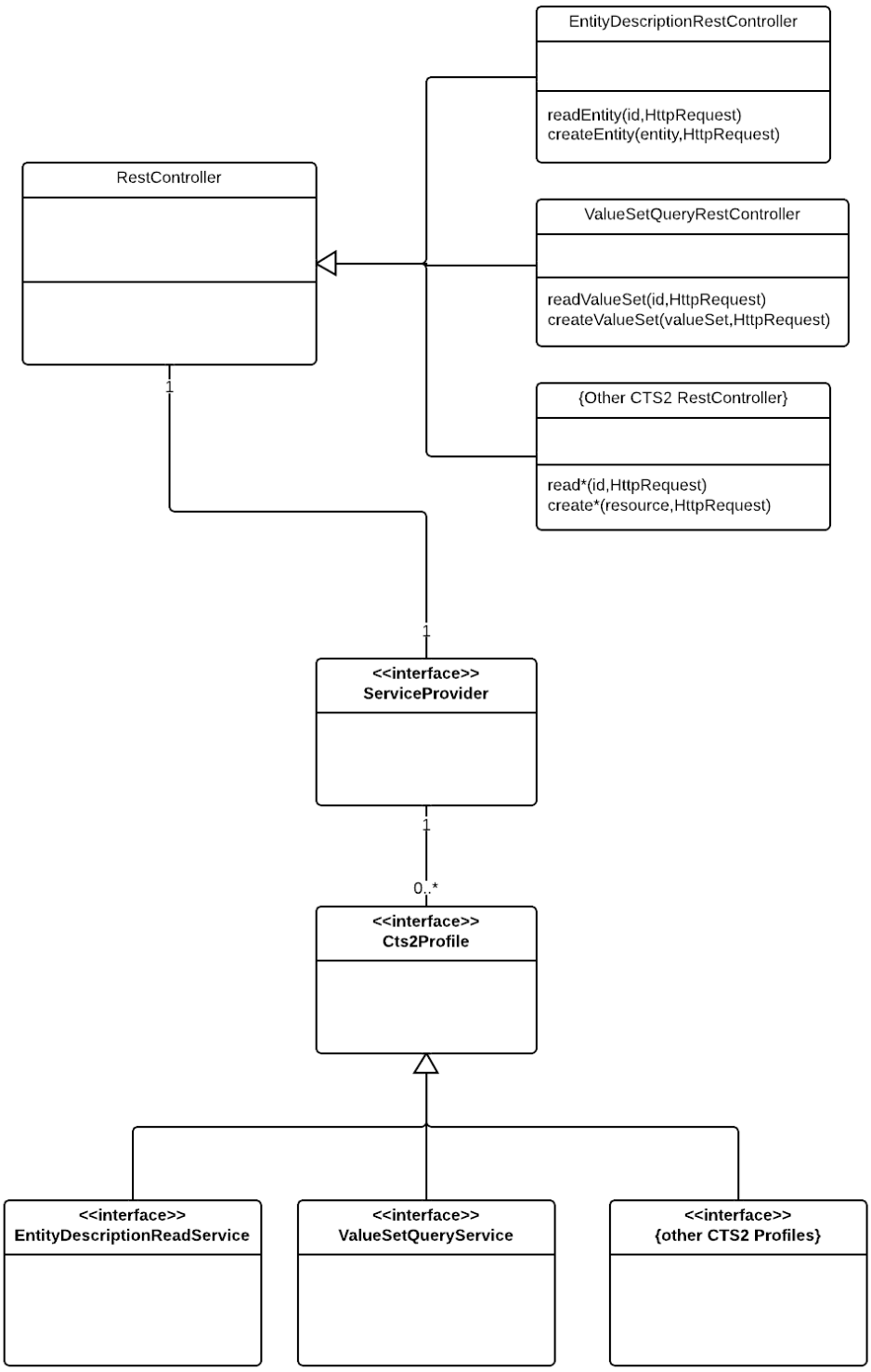
   END WHILE:

END
```

HTTP Requests are accepted and validated by a series of RestControllers. The responsibility of these RestControllers is to accept requests, parse parameters, and check all possible pre-conditions. They also connect the various HTTP requests to the appropriate CTS2 Profile Interfaces. Once the RestControllers have accepted the requests and identified the needed CTS2 Profile Interfaces to fulfill it, The process of discovering the available implementations of those needed CTS2 Profile Interfaces begins.

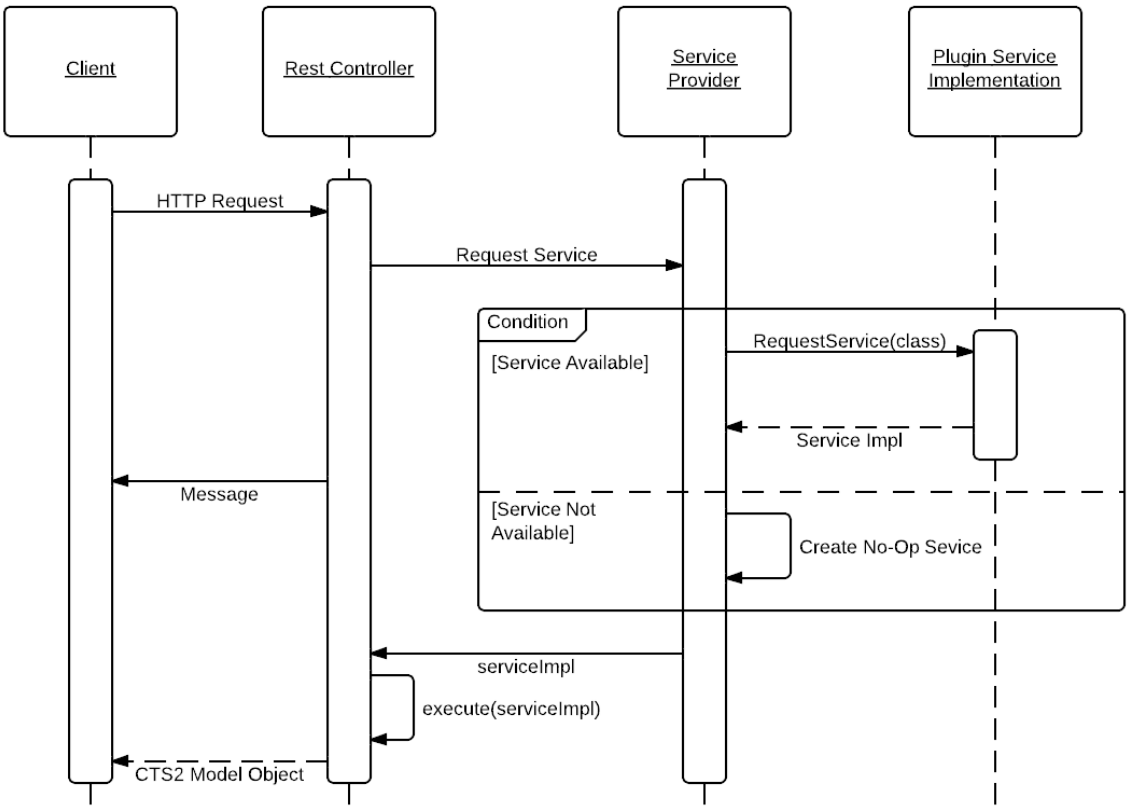
Service Discovery happens via the ServiceProvider interface. Each Service Plugin must register this Service as an OSGi Service. In this way, a Service Plugin advertises each CTS2 Profile that it implements.

NOTE: Service Plugins are not required to implement all of the CTS2 Profile Interfaces. They may implement one, or all, or (most likely) some subset that matches the desired CTS2 functionality.



4.2 Basic Request Sequence

Represented graphically, an incoming request will follow the general sequence as outlined below. It is important to note the Condition case, as this allow Service Plugins to leave CTS2 Profile Interfaces unimplemented if the functionality is not needed. In this case, a "No-Op" Service is defaulted. The responsibility of this No-Op Service is to inform the requester, via the appropriate message, that the requested CTS2 Functionality is not available in this service implementation.



4.3 Independent Module Structure

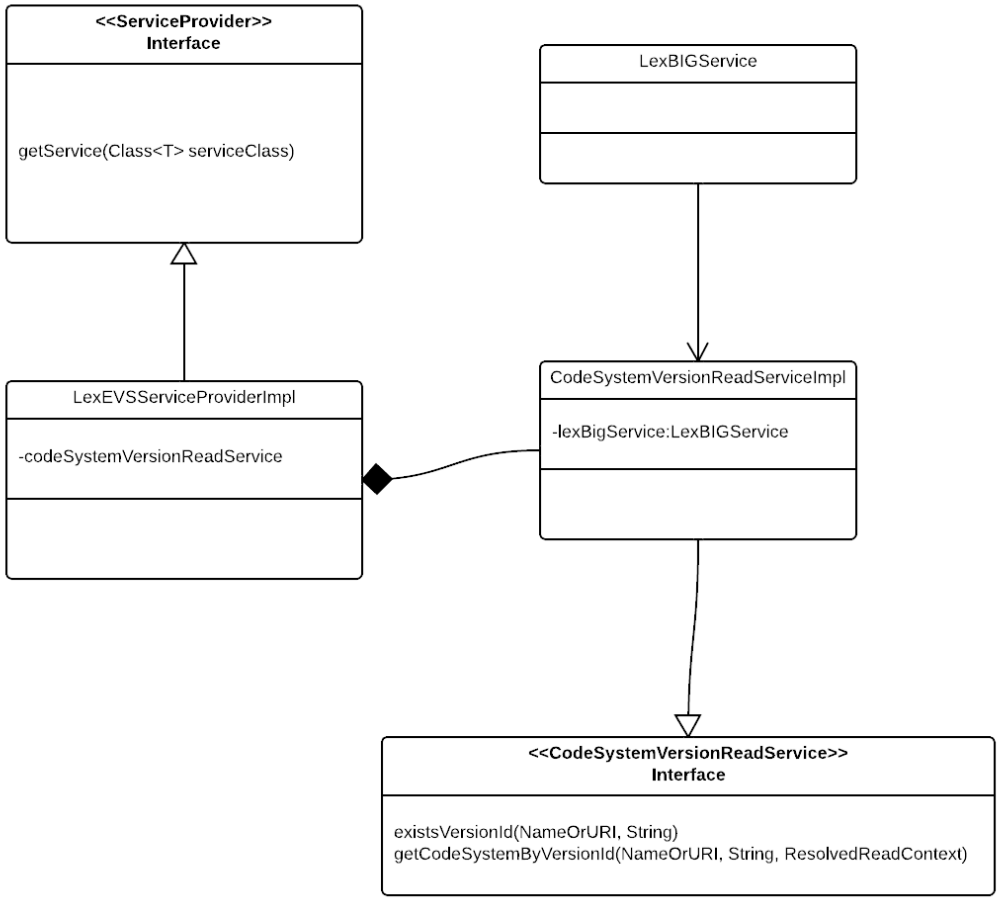
LexEVS, implemented as a CTS2 Service Plugin, will implement the following CTS2 Profile Interfaces. These Interfaces will delegate to LexEVS functionality as described below.

4.3.1 CodeSystemVersionReadService

The CodeSystemVersionReadService CTS2 Profile Interface will be implemented using the LexBIGService.

Specifically, the resolveCodingScheme method of the LexBIGService will be used.

Code System Version Read Service

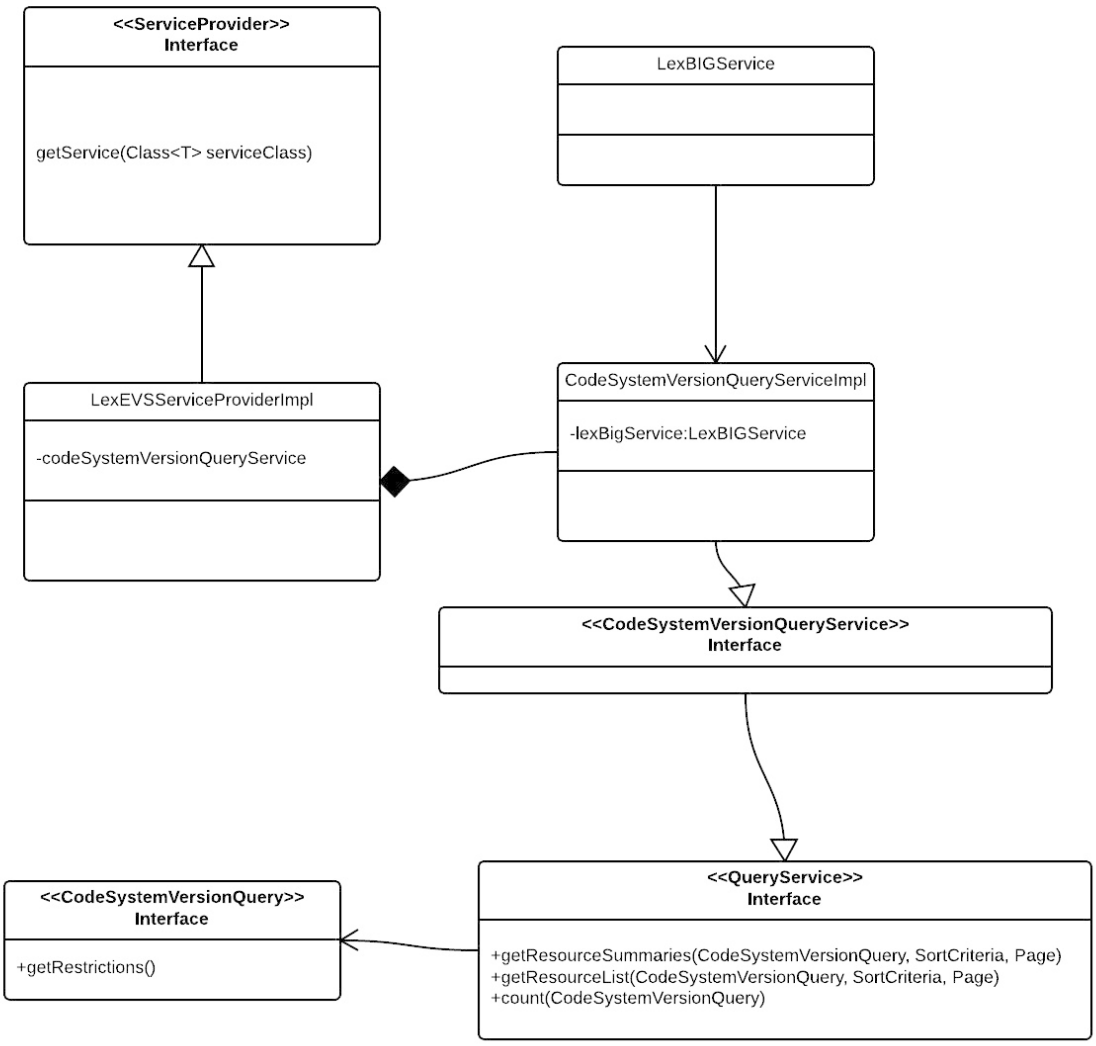


4.3.2 CodeSystemVersionQueryService

The CodeSystemVersionQueryService CTS2 Profile Interface will be implemented using the LexBIGService interface .

Specifically, the getSupportedCodingSchemes method of the LexBIGService interface will be used.

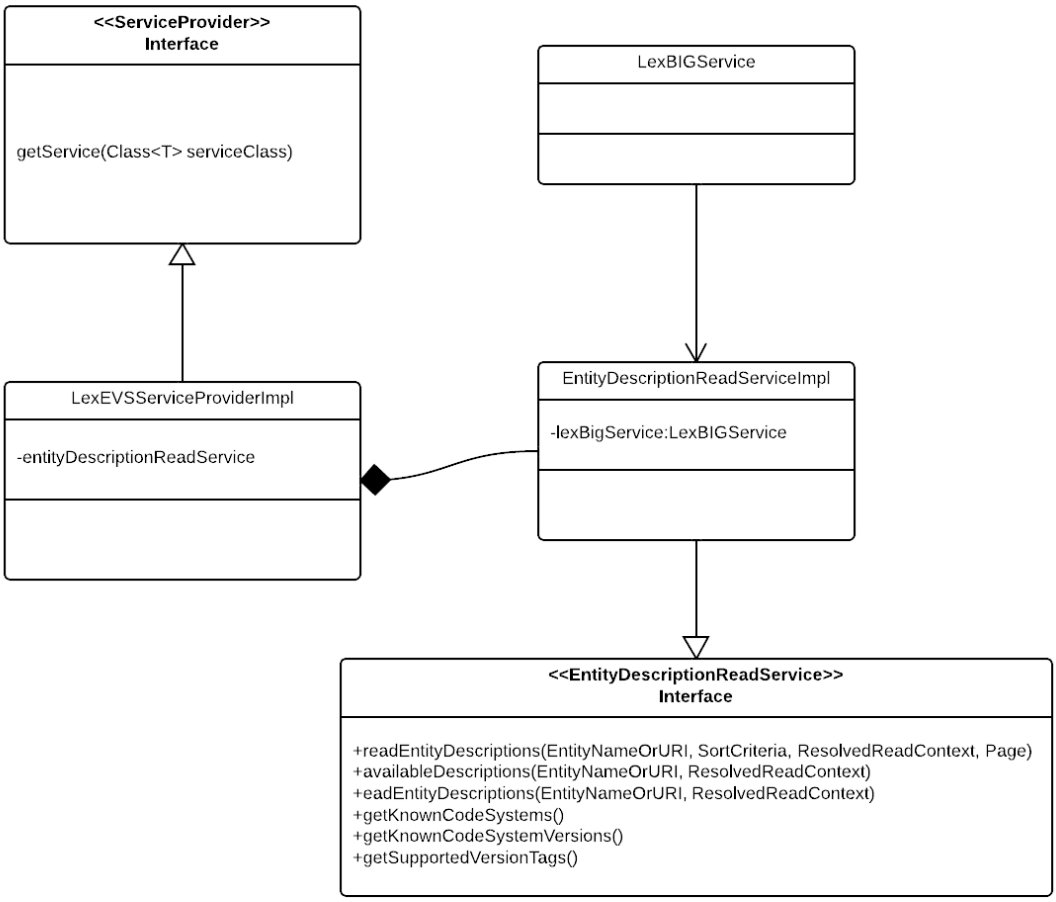
Code System Version Query Service



4.3.3 EntityDescriptionReadService

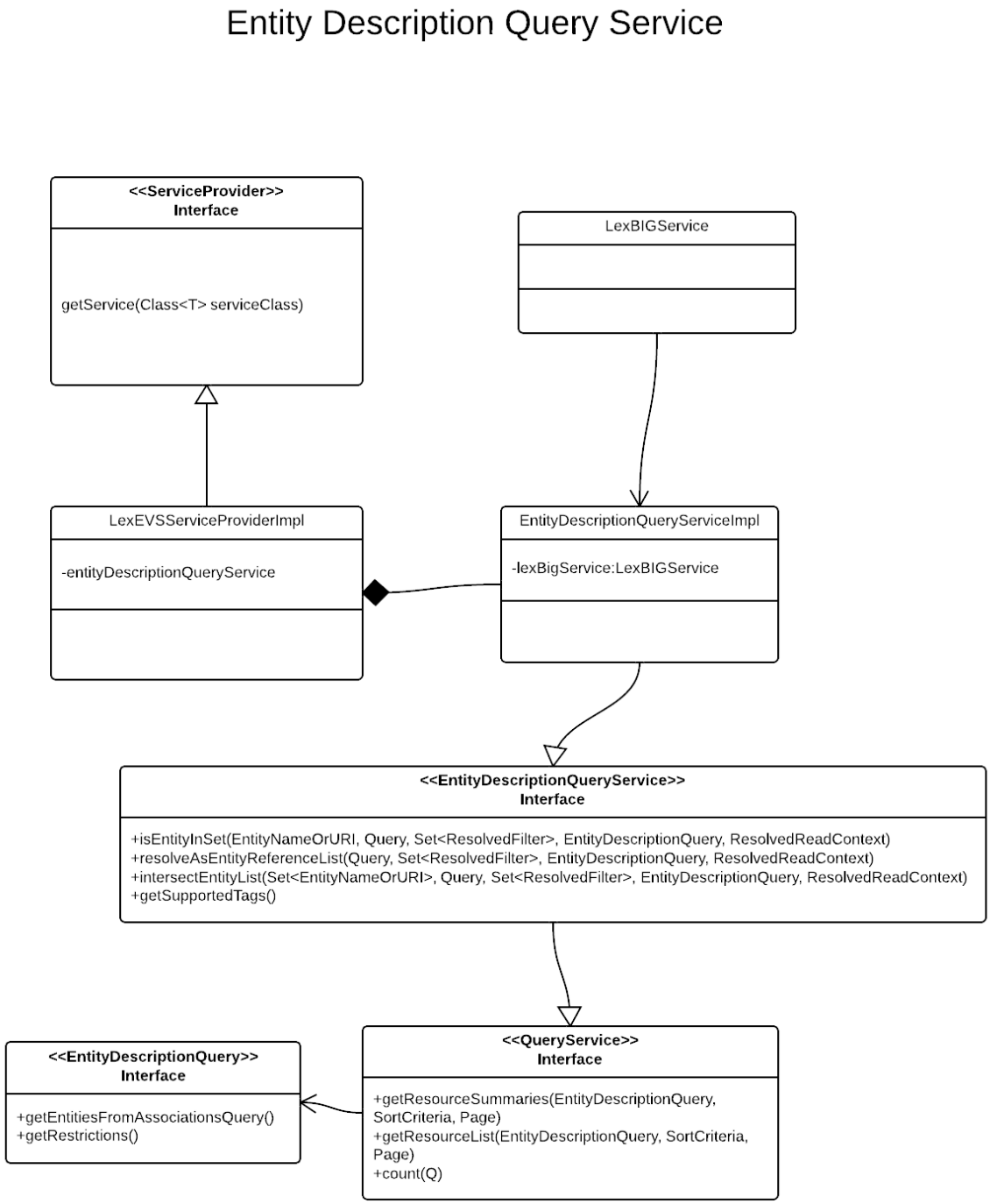
The EntityDescriptionReadService CTS2 Profile Interface will be implemented using the LexBIGService -> CodedNodeSet interface .

Entity Description Read Service



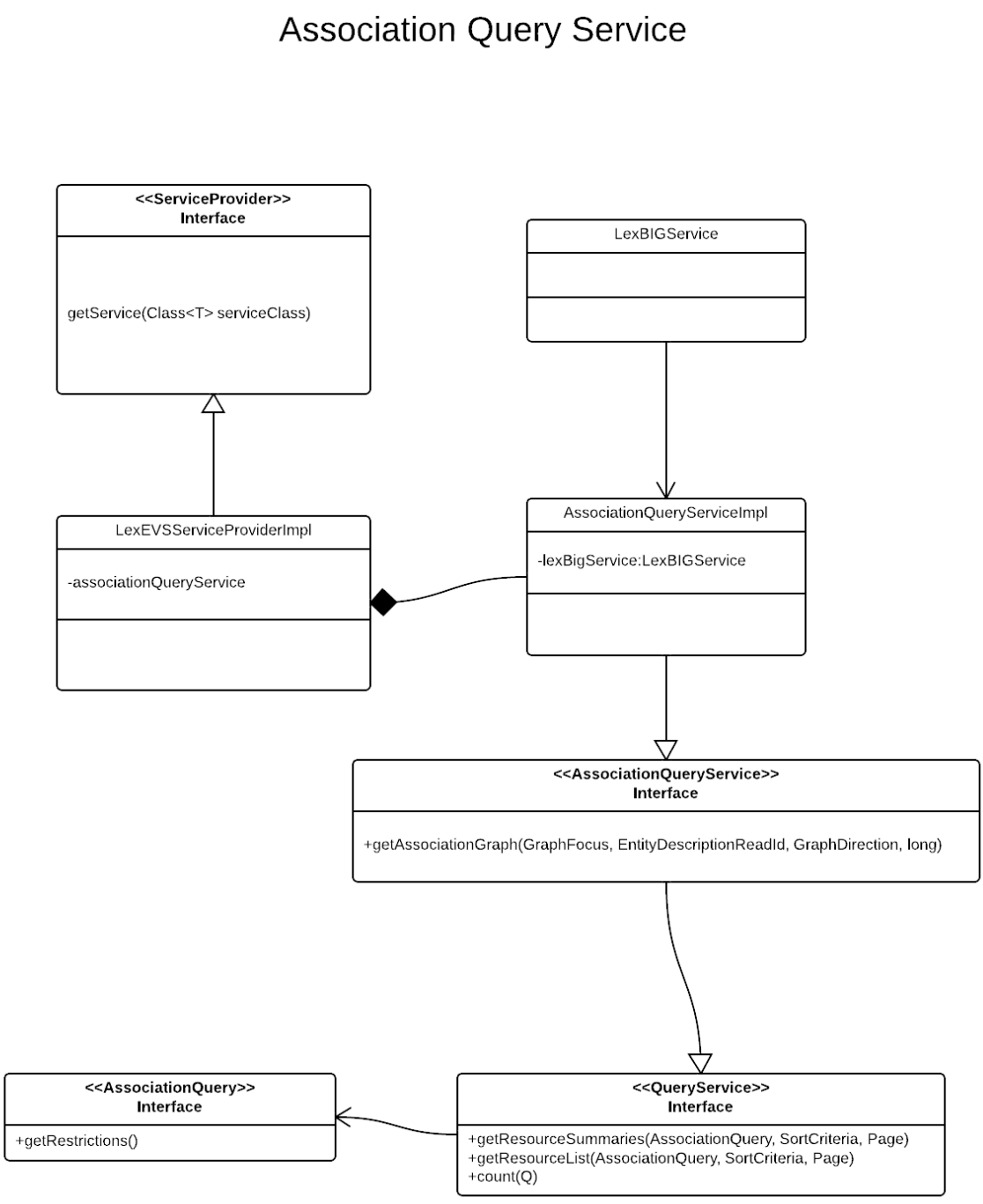
4.3.4 EntityDescriptionQueryService

The EntityDescriptionQueryService CTS2 Profile Interface will be implemented using the LexBIGService -> CodedNodeSet interface .



4.3.5 AssociationQueryService

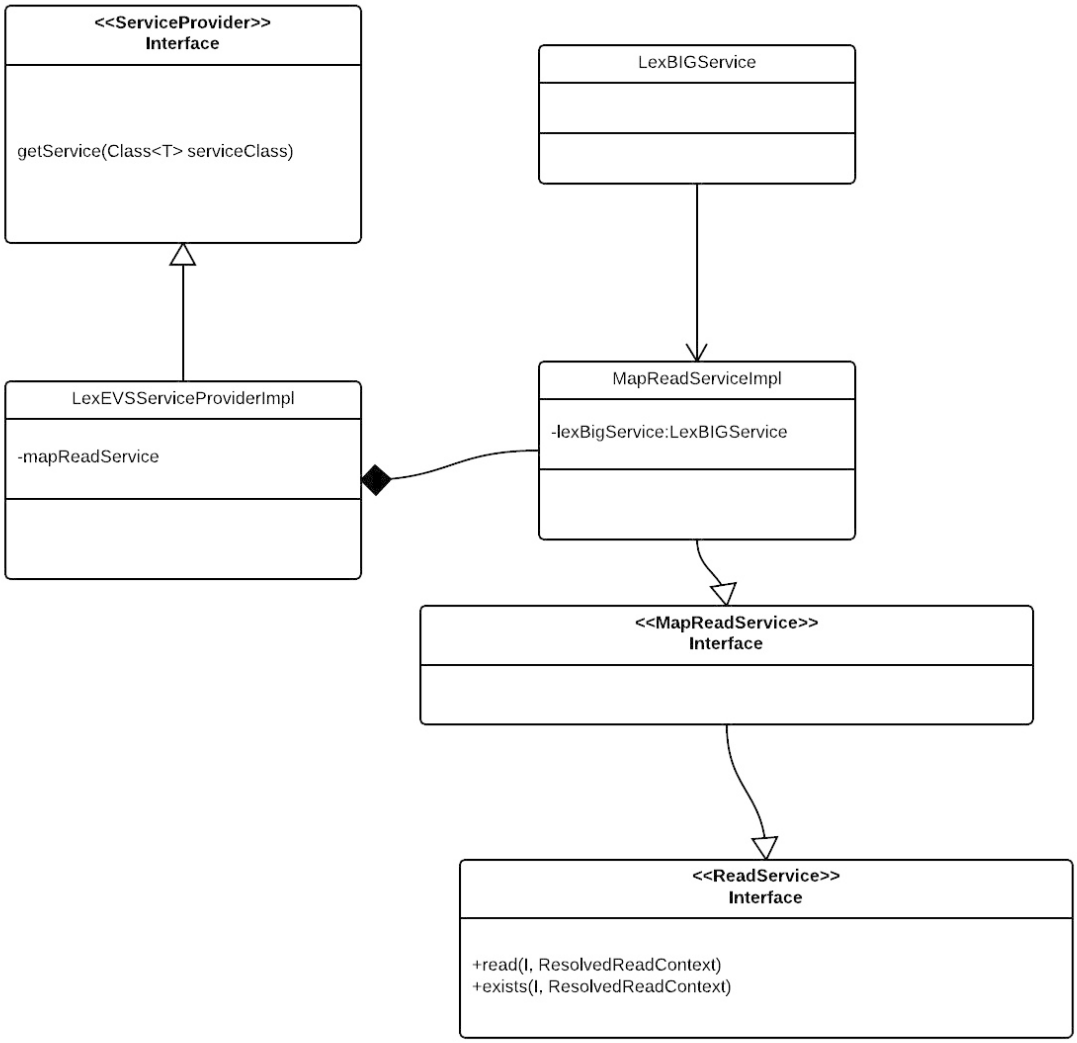
The AssociationQueryService CTS2 Profile Interface will be implemented using the LexBIGService -> CodedNodeGraph interface .



4.3.6 MapReadService

The MapReadService CTS2 Profile Interface will be implemented using a combination of the the MappingExtension interface and the LexBIGService. Specifically, the resolveCodingScheme method of the LexBIGService will be used, and the MappingExtension can be used to verify the CodingScheme is a Mapping or non-Mapping CodingScheme.

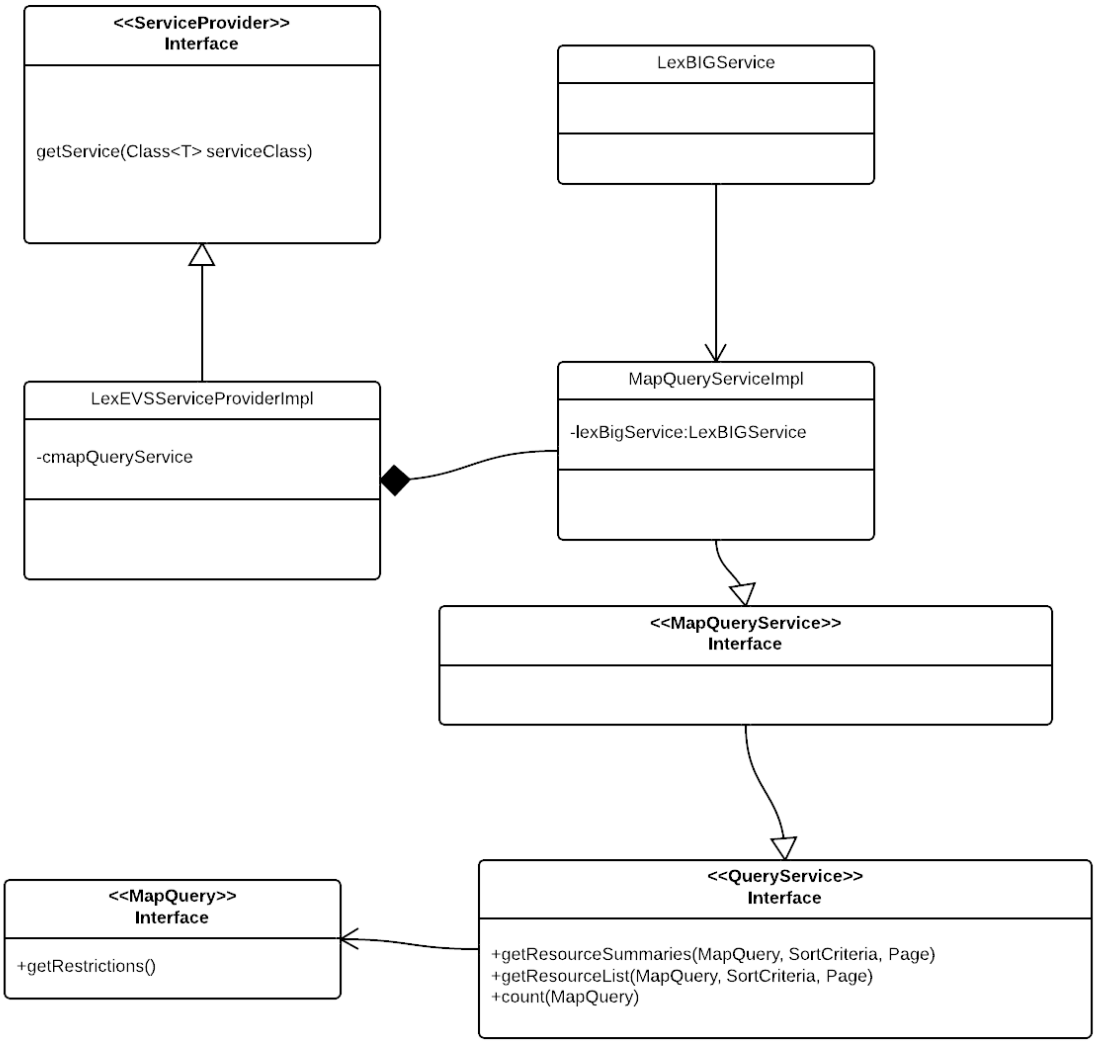
Map Read Service



4.3.7 MapQueryService

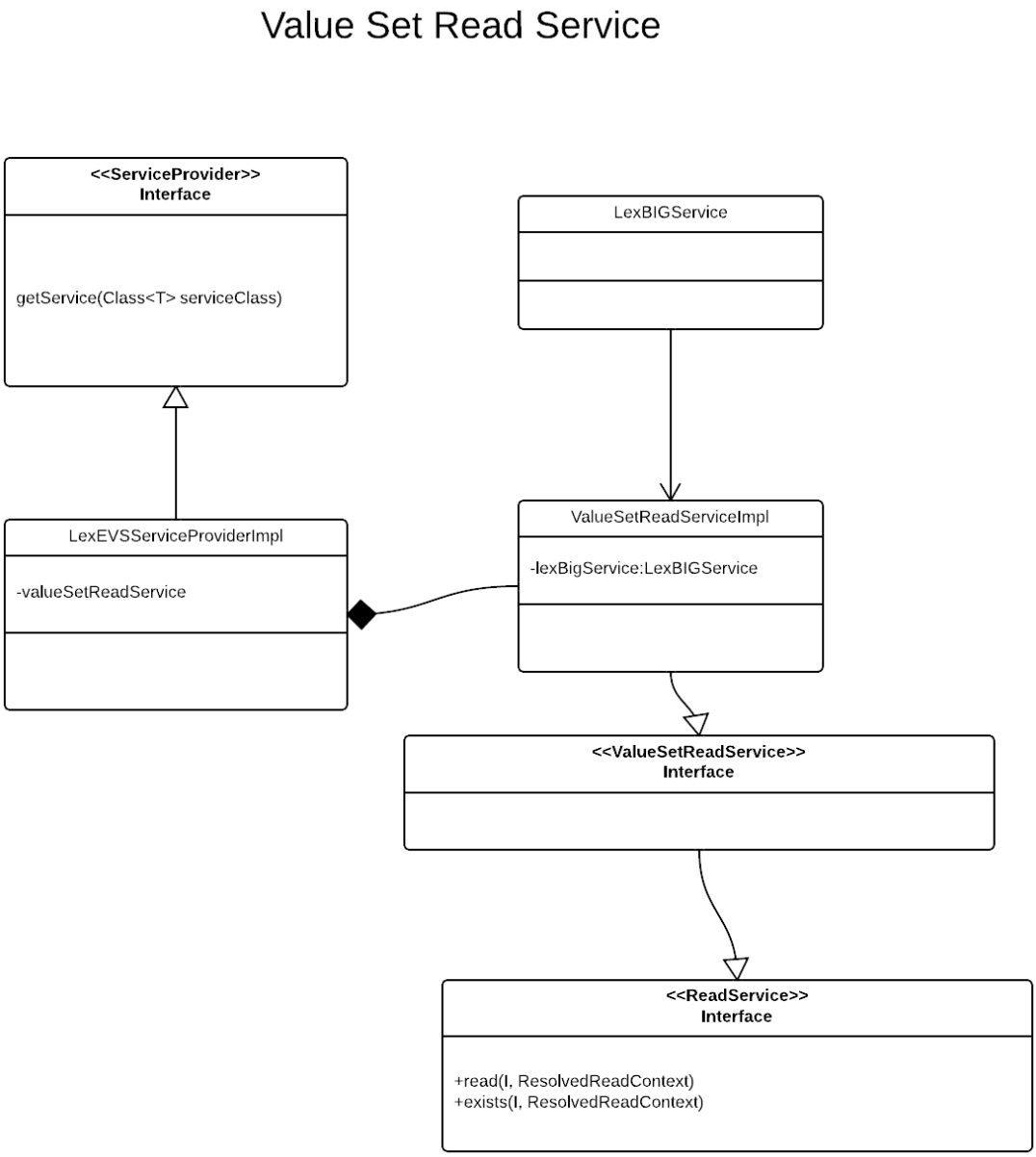
The MapQueryService CTS2 Profile Interface will be implemented using a combination of the the MappingExtension interface and the LexBIGService . Specifically, the getSupportedCodingSchemes method of the LexBIGService will be used, and the MappingExtension can be used to verify the CodingScheme is a Mapping or non-Mapping CodingScheme.

Map Query Service



4.3.8 ValueSetReadService

The ValueSetReadService CTS2 Profile Interface will be implemented using the LexEVS LexEVSValueSetDefinitionServices Interface.



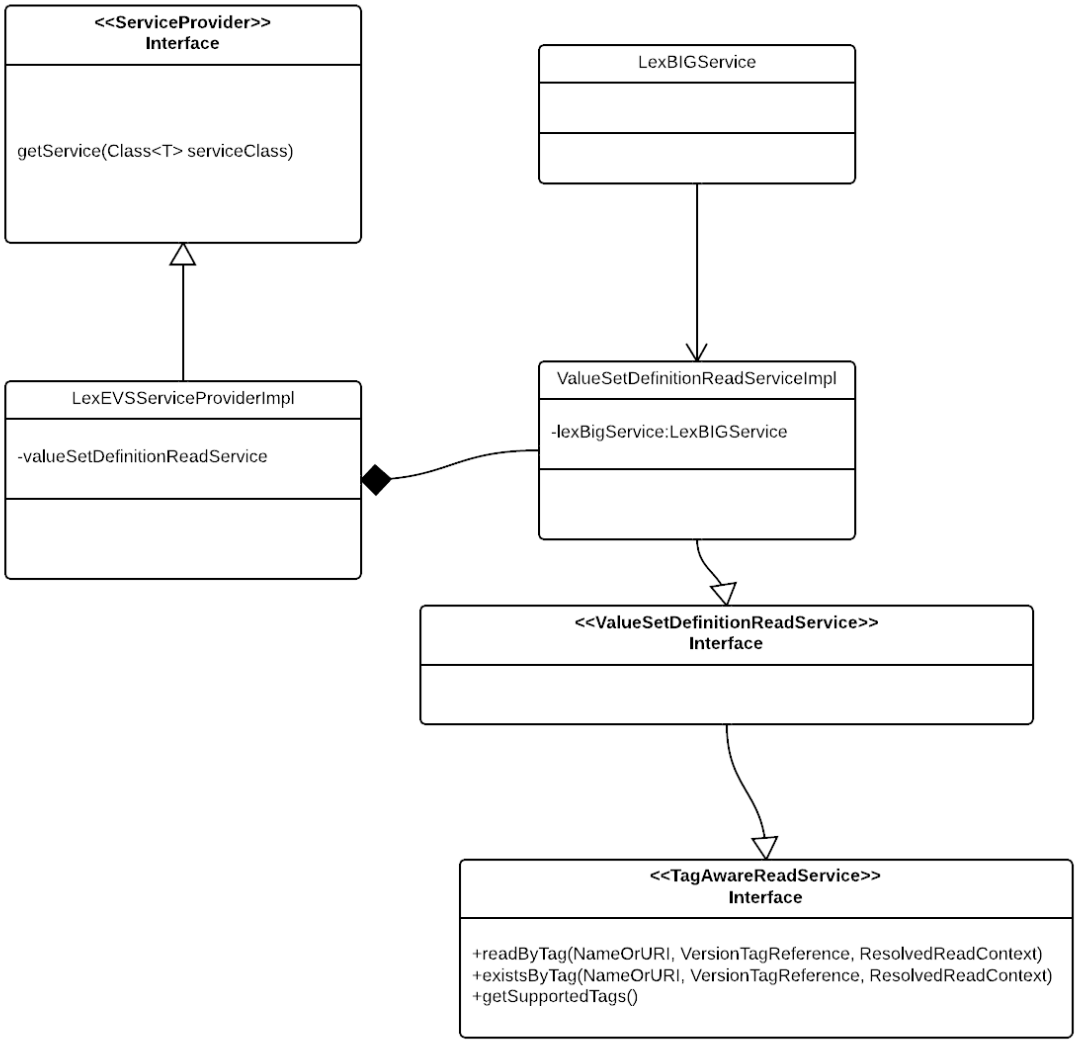
4.3.9 ValueSetQueryService

The ValueSetQueryService CTS2 Profile Interface will be implemented using the LexEVS LexEVSValueSetDefinitionServices Interface.

4.3.10 ValueSetDefinitionReadService

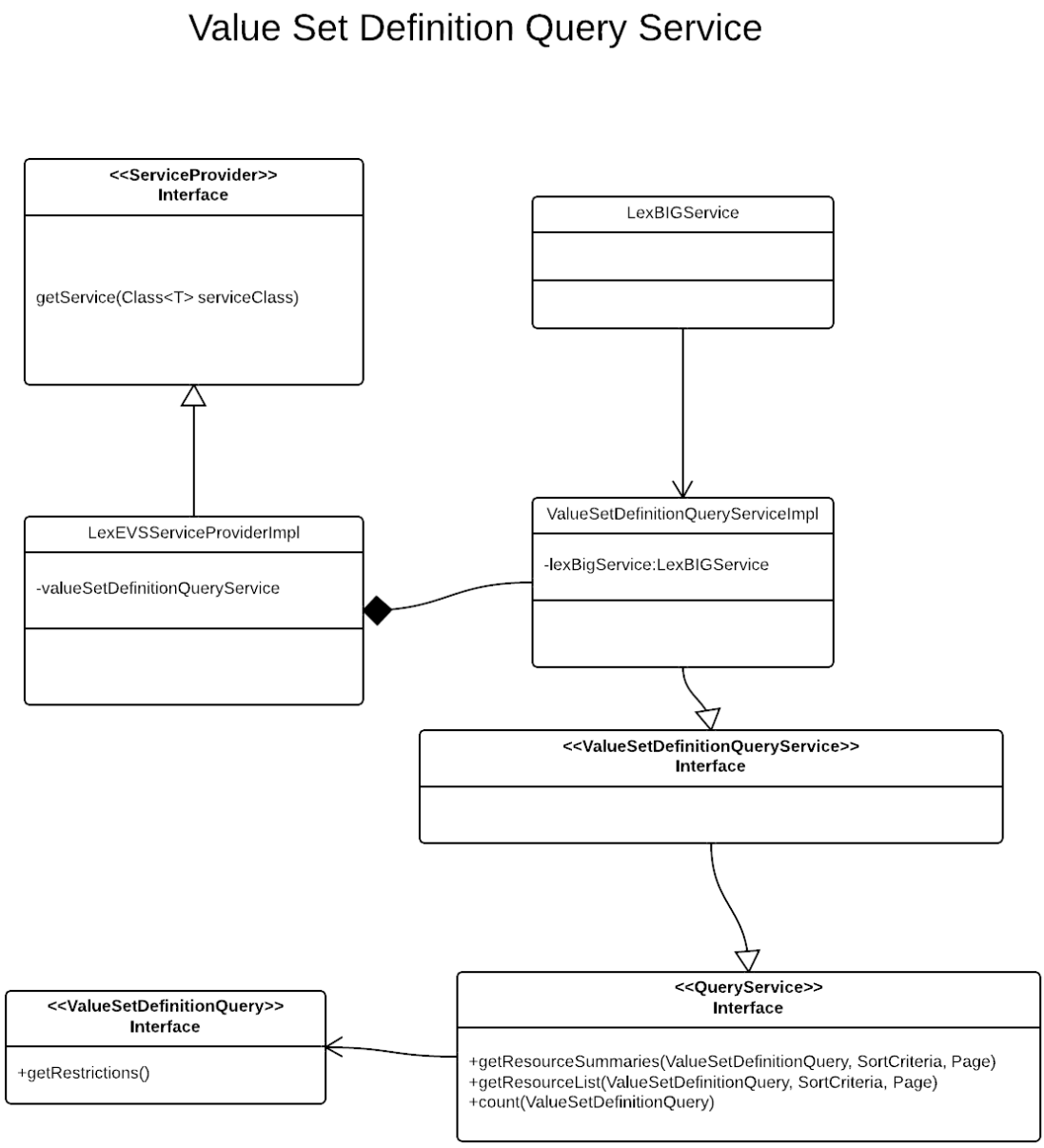
The ValueSetDefinitionReadService CTS2 Profile Interface will be implemented using the LexEVS LexEVSValueSetDefinitionServices Interface.

Value Set Definition Read Service



4.3.11 ValueSetDefinitionQueryService

The ValueSetDefinitionQueryService CTS2 Profile Interface will be implemented using the LexEVS LexEVSValueSetDefinitionServices Interface.



5 URI Resolution

One of the requirements of Resource Oriented Architecture is that every resource must have an identifier. As the Universal Resource Identifier (URI) is the primary identifier used by the Semantic Web, the CTS2 specification calls for all of its resources to be identified by URIs. This results in the need to distinguish the identity of the resource being described from the identity of the description itself, and the basic requirement that all servers in a CTS2 'ecosystem' need to use the same identifiers if information is to be aggregated and shared in a meaningful fashion. The EVS solution for this problem is a URI resolution service, which provides a canonical URI for any local URI and does so as a service. For the EVS implementation of this service, a URI Resolver Service has been implemented and runs in conjunction with the EVS CTS2 implementation. While CTS2 users may install their own resolving service, based on the installation instructions here, a service is also running at this location. A thorough description of this service is to be found here.

6 Glossary

Service Plugin: An OSGi bundle capable of being loaded into the CTS2 Development Framework. In order to be recognized as a Service Plugin, the bundle must export one (and only one) Service of type ServiceProvider . Note, however, that the bundle is no excluded from exporting other services – but it must at least export one and only one ServiceProvider Service.

Platform Independent Model (PIM): A implementation agnostic description of program functionality.

Platform Specific Model (PSM): An implementation model targeted towards a specific technical platform.