General City Digital Twin  
User Manual

**Anderson Wong**

Urban Data Research Centre (urbandatacentre.ca)  
School of Cities (schoolofcities.ca)  
University of Toronto

4 March 2025

# Table of Contents

[Table of Contents 1](#_Toc1441721113)

[Introduction 1](#_Toc186884184)

[Accessing the Dashboard 1](#_Toc779132099)

[Representation of City Data 2](#_Toc1505257559)

[Administrative Areas 3](#_Toc1486957014)

[Address 4](#_Toc1629983695)

[Geospatial Location 4](#_Toc578045790)

[Census Profiles and Characteristics 5](#_Toc1538562036)

[Toronto Police Service Crime Data 6](#_Toc553618290)

[Complete Community Walking Distance Indicator 8](#_Toc738890493)

[Parks 9](#_Toc443853334)

[Schools 10](#_Toc503373422)

[Kindergartens 11](#_Toc578537294)

[Universities 12](#_Toc141659514)

[Colleges 13](#_Toc1342352192)

[Hospitals 14](#_Toc1439665753)

[Clinics 15](#_Toc1053846896)

[Doctor’s Office 16](#_Toc960110567)

[Pharmacy 17](#_Toc1486210960)

# Introduction

The City Digital Twin project at the University of Toronto’s Urban Data Research Centre aims to develop a digital twin of the City of Toronto. It builds on existing city data standards such as ISO/IEC 21972:2020, ISO/IEC 5087-1:2024, and ISO/IEC 5087-2:2024, and conforms to Linked Data requirements. The goal of this project is to demonstrate standards-based semantic interoperability of city data. These standard ontologies enable the integration of data from multiple sources, opening up new possibilities for the development of data analysis and visualization tools. Our City Digital Twin is represented as a knowledge graph containing a growing variety of city data (e.g. Canadian Census, Toronto Police Crime Data, City transportation infrastructure, parks, food stores, schools, …). Our City Digital Twin also supports a dashboard that can generate data visualizations using the city data from the graph database.

Our City Digital Twin is general in the sense the the ontologies and standards being used to represent city diata are city independent. Therefore the CDT can be applied to any city with modification.

# Accessing the Dashboard

The City Digital Twin Dashboard can be directly accessed using the following link: <http://ec2-3-97-59-180.ca-central-1.compute.amazonaws.com:3001/.>

Alternatively, it can be downloaded from GitHub (<https://github.com/csse-uoft/city-digital-twin/tree/develop>) and run locally. You can run it locally on your computer by completing the following steps:

1. Download the “develop” branch of the City Digital Twin from our GitHub: <https://github.com/csse-uoft/city-digital-twin/tree/develop.> If you downloaded the code as a .zip file, unzip the file before proceeding to the next step.
2. Open the Command Prompt. On Windows 10 or 11, you can do this by pressing the Windows key and typing “cmd”. For Mac OS users, you can open Terminal instead by clicking the Launchpad icon and typing Terminal in the search field.
3. In the Command Prompt, navigate to the unzipped folder that you downloaded from GitHub using the “cd” command (e.g., “cd folder\_name”).
4. After you have navigated to the folder, enter and run “npm install” in the Command Prompt to install the dependencies.
5. Enter and run “cd backend” to navigate to the “backend” folder
6. Enter and run “npx nodemon index.js” to start up the backend of the dashboard
7. Open a new Command Prompt window by repeating step 2
8. In this new Command Prompt window, navigate to the unzipped folder by repeating step 3
9. Enter and run “cd frontend” to navigate to the “frontend” folder
10. Enter and run “npm start” to start up the frontend of the dashboard
11. Once the above steps are completed, the dashboard should be up and running at a localhost address specified in the second Command Prompt window. You can enter this localhost address into a web browser to access the dashboard.

# Updating the Graph Database

The graph database for the City Digital Twin project can be accessed here: <http://ec2-3-97-59-180.ca-central-1.compute.amazonaws.com:7200/>. To update the contents of the graph database, please refer to the steps below as a guide and more detailed information about how to use SPARQL to update graphs along with examples can be found here: <https://docs.progress.com/bundle/marklogic-server-develop-with-semantic-graphs-11/page/topics/sparql-update.html.> Official SPARQL Update documentation can also be found here: <https://www.w3.org/TR/sparql12-update/>.

1. If triples need to be removed from the graph database:
   1. If an entire graph needs to be deleted, you can run a “drop graph” query in SPARQL to remove the specified graph: DROP GRAPH <insert\_graph\_IRI>
   2. If specific triples need to be deleted from the graph database:
      1. You can run a “delete data” query if you are deleting individual triples explicitly or:
      2. A “delete where” query if you are deleting patterns of triples
2. If triples need to be added to the graph database:
   1. If an entire RDF file (e.g. a TTL or OWL file) needs to be uploaded to the graph database, you can upload and import the file in GraphDB Workbench using its user interface:
      1. Click on the “Import” button on the left sidebar in GraphDB Workbench
      2. Click on “Upload RDF files” button and upload your RDF file(s)
      3. After the file(s) have been uploaded, click on the “Import” button next to the uploaded file and a popup window will appear
      4. It is highly recommended that you specify the name of the target graph by clicking on “Named graph” under the “Target graphs” section and entering a name for the graph in the textbox
      5. Click on the “Import” button in the bottom right corner of the popup window when you are ready to import the data
      6. An “Imported successfully” message should show if the update was successful
   2. If specific triples need to be added to the graph database:
      1. You can run a “insert data” query if you are adding explicitly defined triples or:
      2. A “insert where” query if you are adding patterns of triples

# Representation of City Data

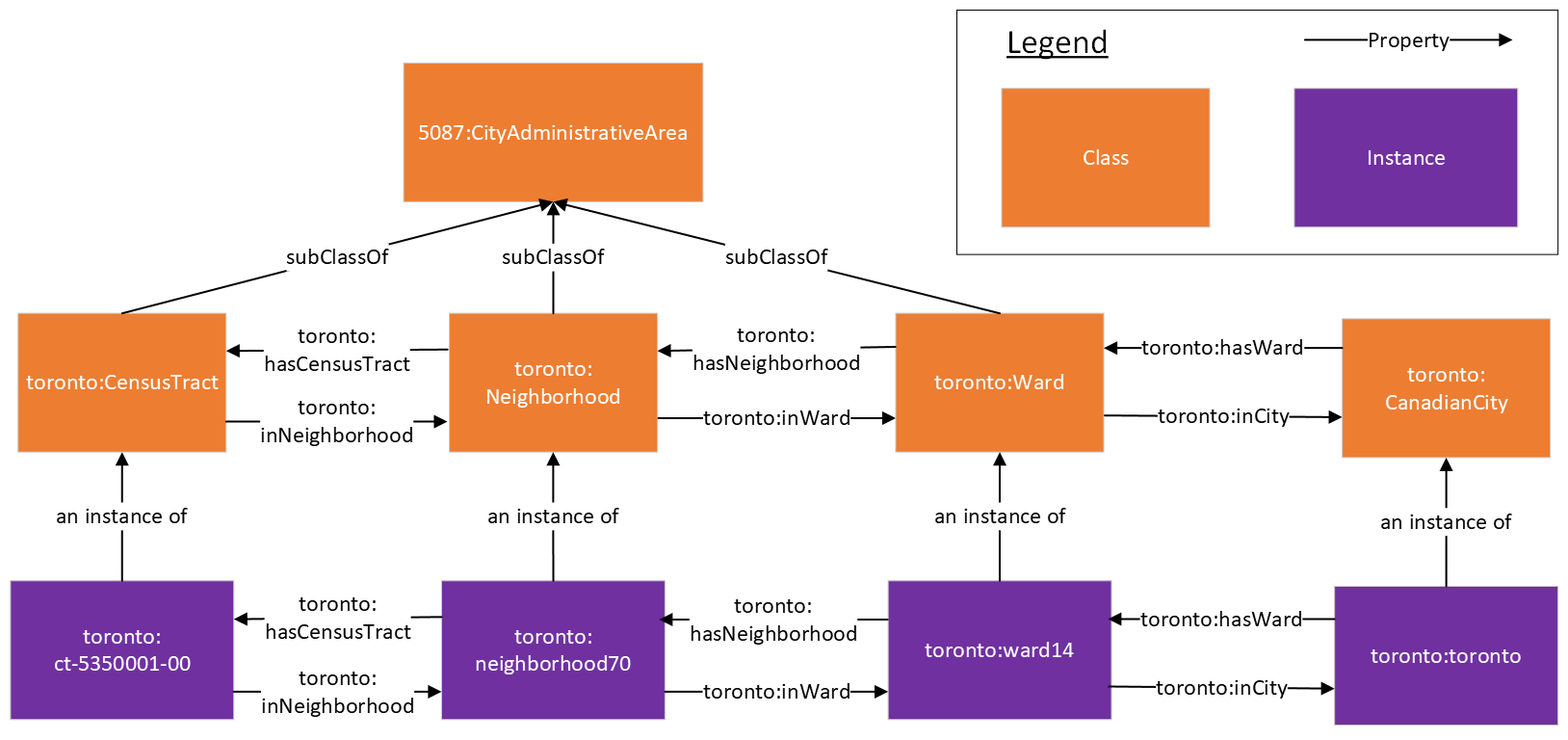
The following sections will provide an overview of how the city data in the graph database is ontologically represented.

The following is a list of namespace prefixes:

* cdt: http://ontology.eil.utoronto.ca/CDT#
* code: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Code/
* contact: https://standards.iso.org/iso-iec/5087/-2/ed-1/en/ontology/Contact/
* crime: http://ontology.eil.utoronto.ca/CKGN/Crime#
* foaf: http://xmlns.com/foaf/0.1/
* gcie: http://ontology.eil.utoronto.ca/GCI/Education/GCI-Education.owl#
* gcir: http://ontology.eil.utoronto.ca/GCI/Recreation/GCIRecreation.owl#
* genprop: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/GenericProperties/
* geo: http://www.opengis.net/ont/geosparql#
* geof: http://www.opengis.net/def/function/geosparql/
* iso21972: http://ontology.eil.utoronto.ca/ISO21972/iso21972#
* iso50871: http://ontology.eil.utoronto.ca/5087/1/SpatialLoc/
* iso5087m: http://ontology.eil.utoronto.ca/5087/1/Mereology/
* loc: https://standards.iso.org/iso-iec/5087/-1/ed-1/en/ontology/SpatialLoc/
* rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
* rdfs: http://www.w3.org/2000/01/rdf-schema#
* time: http://www.w3.org/2006/time#
* toronto: http://ontology.eil.utoronto.ca/Toronto/Toronto#
* uoft: http://ontology.eil.utoronto.ca/tove/cacensus#
* xsd: http://www.w3.org/2001/XMLSchema#

## Administrative Areas

Administrative areas are geographic regions that are defined for governance and administrative purposes. Examples include areas like neighborhoods or city wards as seen in the below diagram and are expressed as a subclass of the CityAdministrativeArea class from ISO 5087-2. These administrative areas also have properties such as ”hasNeighborhood” or ”inWard” to describe the relationships between each other. These properties are useful for quickly finding, for example, a list of neighborhoods that can be found inside a given ward.

  
An example SPARQL query for finding a list of neighborhoods in ward 1 (Etobicoke North) can be found below:

PREFIX toronto: <http://ontology.eil.utoronto.ca/Toronto/Toronto#>

SELECT ?neighborhood

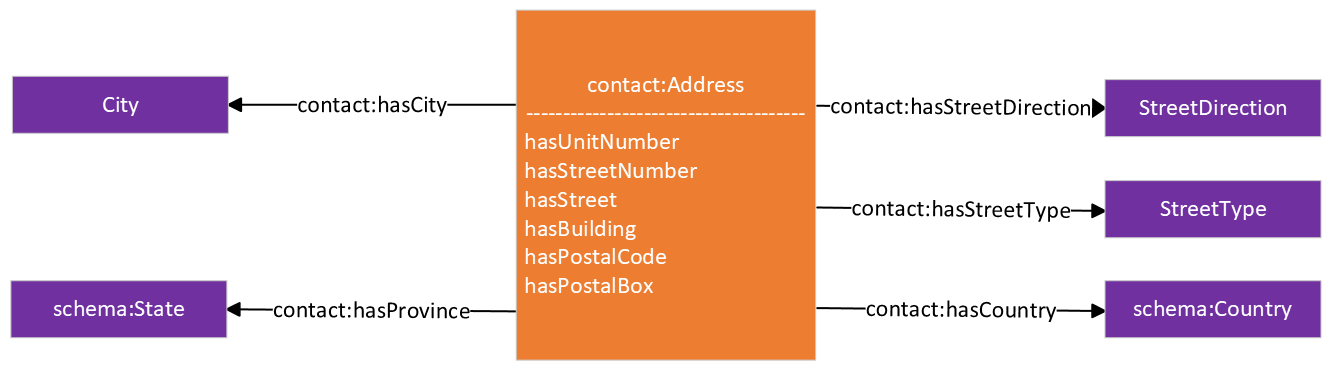
WHERE{

toronto:ward1 toronto:hasNeighborhood ?neighborhood

}

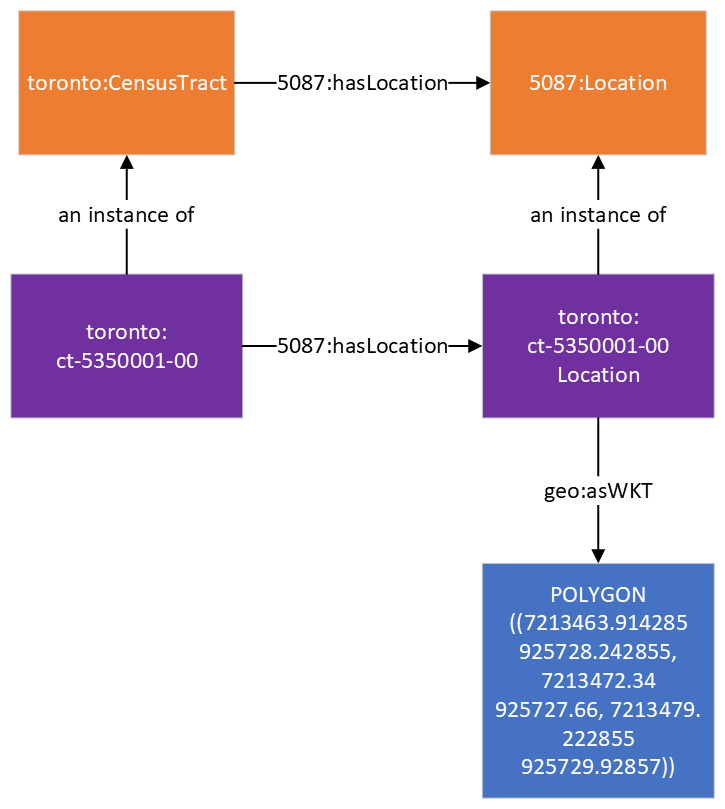
## Address

The address of an entity is represented using the Address representation from the Contact pattern from ISO 5087-2 where an entity is linked to an Address instance using the “hasAddress” property. This Address instance is then linked to individual components of an address using properties such as hasStreetNumber, hasStreet, hasPostalCode, etc.



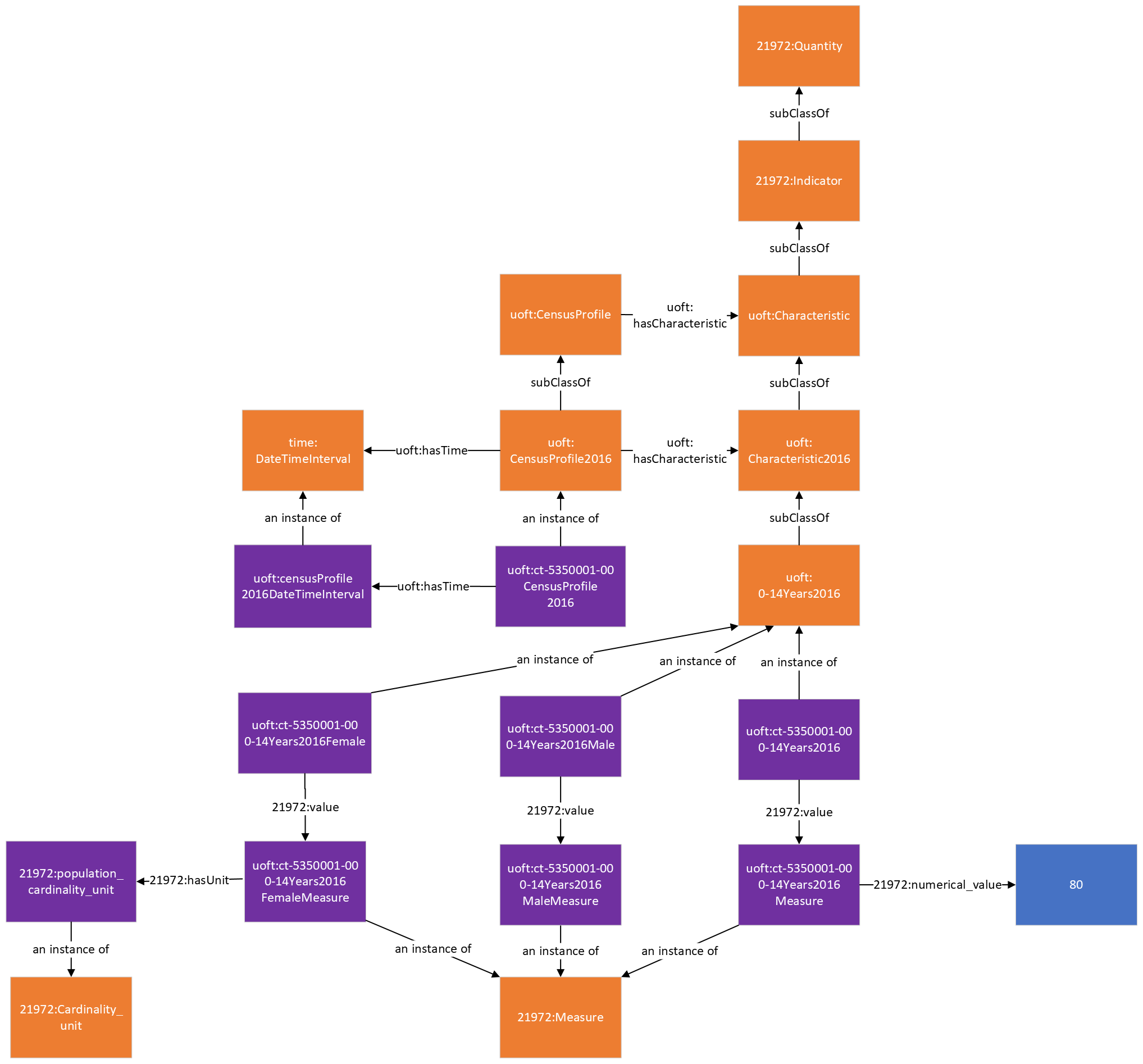
## Geospatial Location

The geospatial location of an entity is represented using the Location pattern from ISO 5087-1 where an entity is linked to a Location class instance using the “hasLocation” property. The location instance is then linked to the WKT polygon coordinates using the “asWKT” property.



## Census Profiles and Characteristics

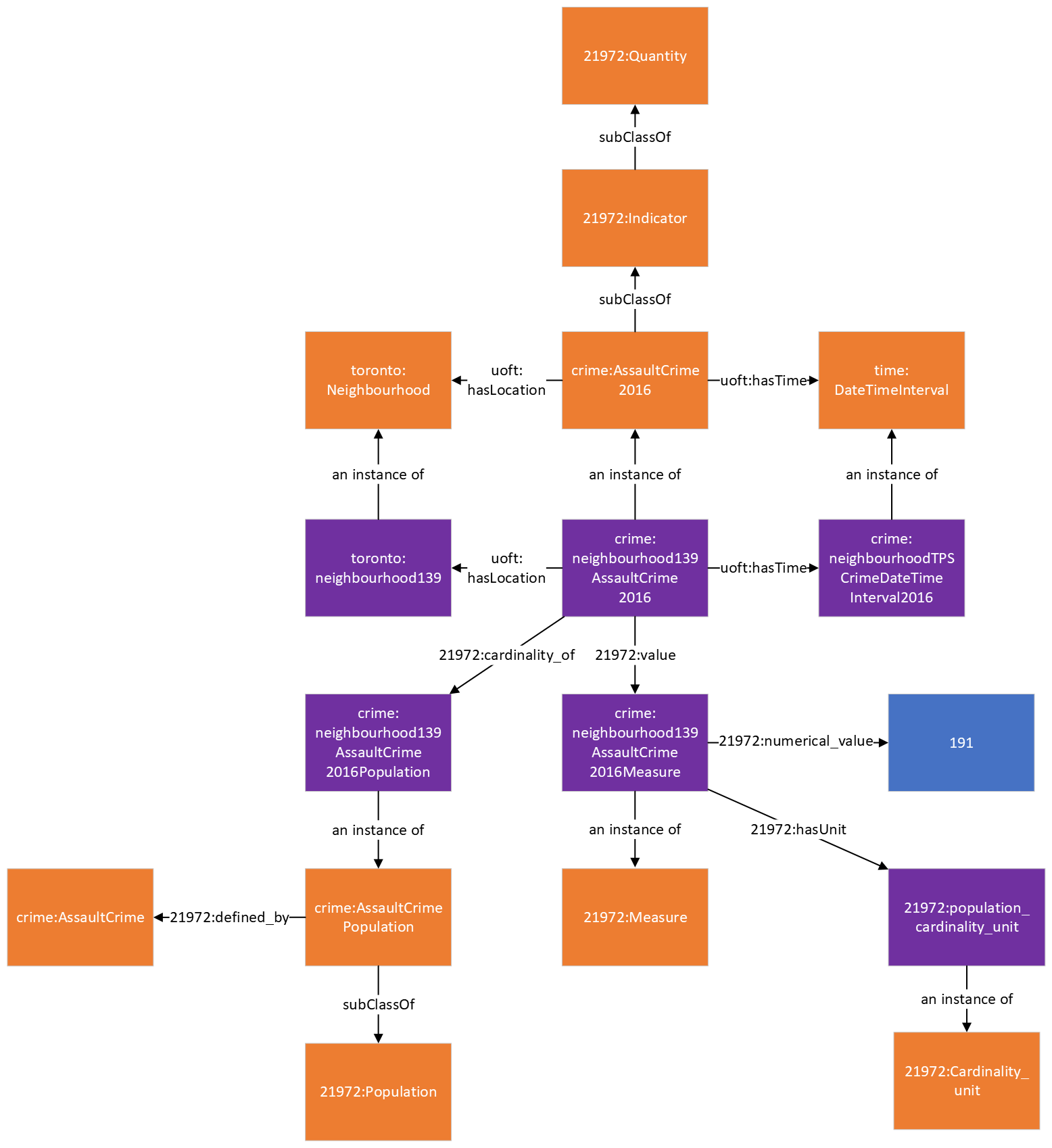
Census data for geographic areas are organized as datasets called census profiles. Census profiles contain a collection of individual census characteristics, which are a subclass of the Indicator class from ISO 21972 and linked using the hasCharacteristic property. Each characteristic also has a class that is used to represent it, and this class is also a subclass of the Characteristic class. For example, the 0-14Years2016 class in the diagram above represents the characteristic for the number of people who were between the ages 0 to 14 during the 2016 Canadian Census. An instance of this class represents the characteristic for a given geographic area. For example, the “ct-5350001-000-14Years2016” instance in the diagram below represents the characteristic for the number of people in the census tract 5350001.00 who were between the ages 0 to 14 in 2016. These characteristic instances are linked to an instance of the Measure class from ISO 21972 using the ”value” property and this measure instance is linked to the value of the characteristic using the ”numerical\_value” property.



A more in-depth explanation of the Canadian Census ontology can be found in our paper titled “Semantically Interoperable Census Data: Unlocking the Semantics of Census Data Using Ontologies and Linked Data” which can be found here: <https://ijpds.org/article/view/2378#:~:text=Using%20census%20data%20as%20an,data%20points%20from%20large%20datasets>.

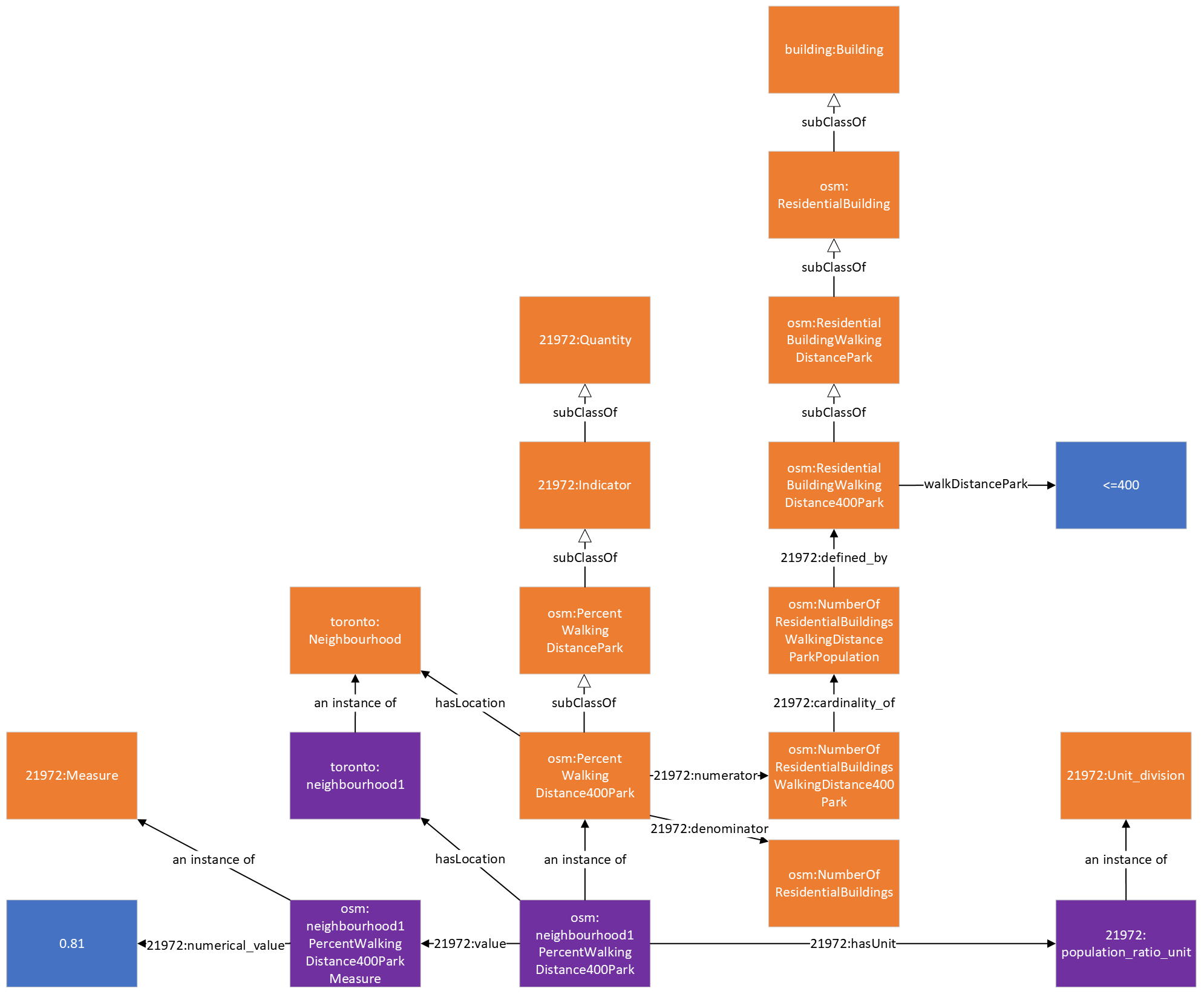
## Toronto Police Service Crime Data

This is a basic ontology for representing crime indicators using the Indicator pattern from ISO/IEC 21972. An instance of a crime indicator (e.g. AssualtCrime2016 which measures the number of assault crimes that occurred during the year 2016 in a given location) is defined as a subclass of the Indicator class from ISO/IEC 21972. Crime indicators can be linked to their location using the hasLocation property and its corresponding time period using the hasTime property. The value of a crime indicator is defined as an instance of the Measure class from ISO/IEC 21972 and can be linked to its numerical value using the numerical\_value property. Additionally, crime indicators can be linked to its population instance using the cardinality\_of property from ISO/IEC 21972.



## Complete Community Walking Distance Indicator

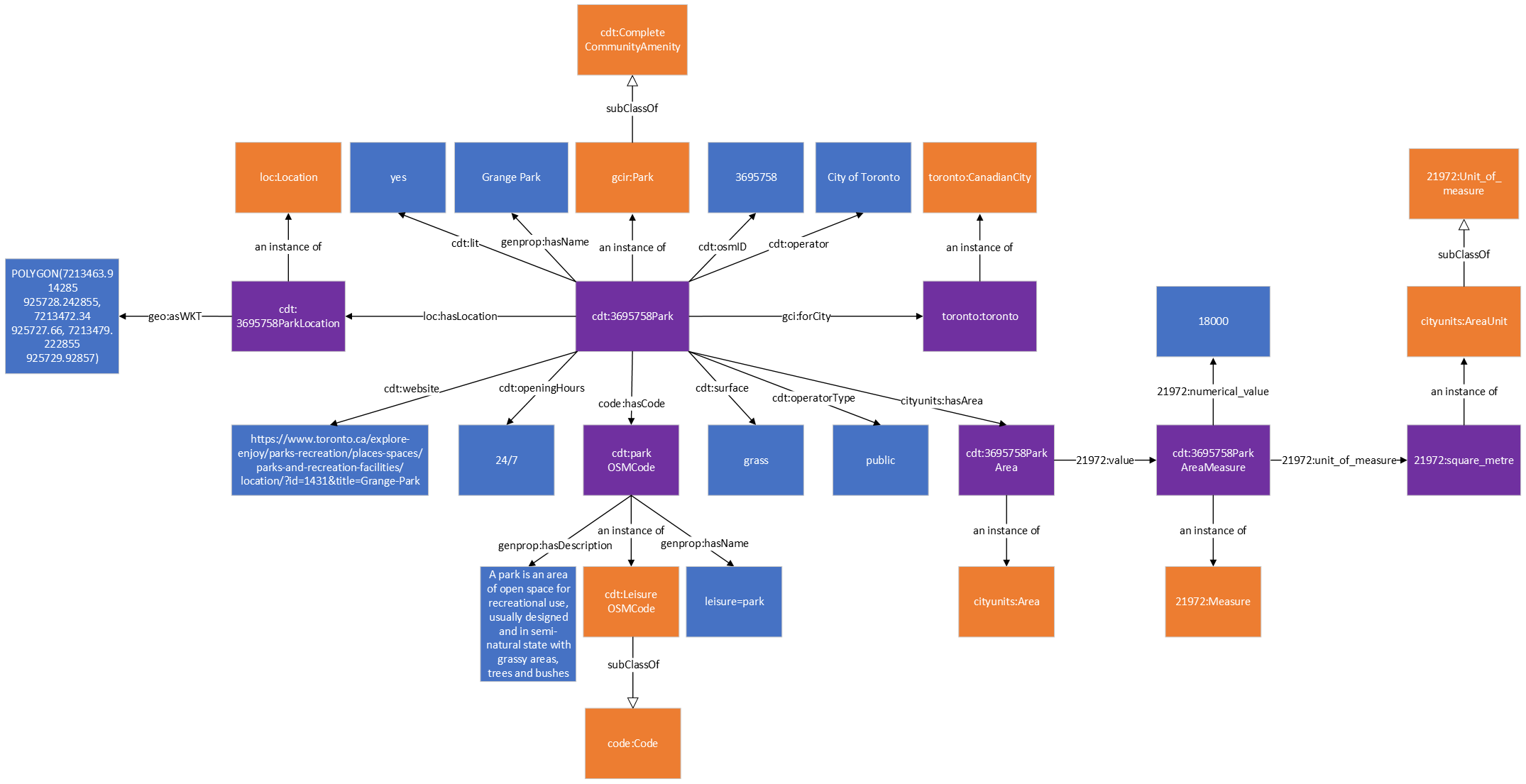
This is an ontological representation of a complete community walking distance indicator that shows the percentage of residential buildings in a geographic area that are within a specified distance of a complete community amenity. The diagram above shows the representation of a walking distance indicator for parks where individual instances of an indicator represent the walking distance indicator for a geographic area specified by the “hasLocation” property. This indicator is linked to a measure instance using the “value” property and the measure instance is linked to the value of the indicator using the “numerical\_value” property. In the diagram below, we can see that the numerical value for the “neighborhood1PercentWalkingDistance400Park” indicator is 0.81 which means that 81% of the residential buildings in neighborhood1 are within a 400m distance of a park. This indicator is calculated by dividing the number of residential buildings that are within a 400m distance of a park by the total number of residential buildings in the area and this relationship is represented using the “numerator” and “denominator” properties, respectively.



## Parks

This is an ontological representation of park data. Parks are represented as instances of the Park class from the Recreation Ontology for Global City Indicators. Parks can have a name which is linked via the hasName property, an operator which is linked via the operator property, an operator type (e.g. public, private) which is linked via the operatorType property, a value describing the physical surface of the park which is linked via the surface property, opening hours which is linked via the openingHours property, a website which is linked via the website property, an indicator for whether the park is lit or not via the lit property, a hasArea property that links the park to its area instance and area value using the numerical\_value property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property respectively, and a code instance that can be used to link the park to a specific set list of values (e.g., classification systems).

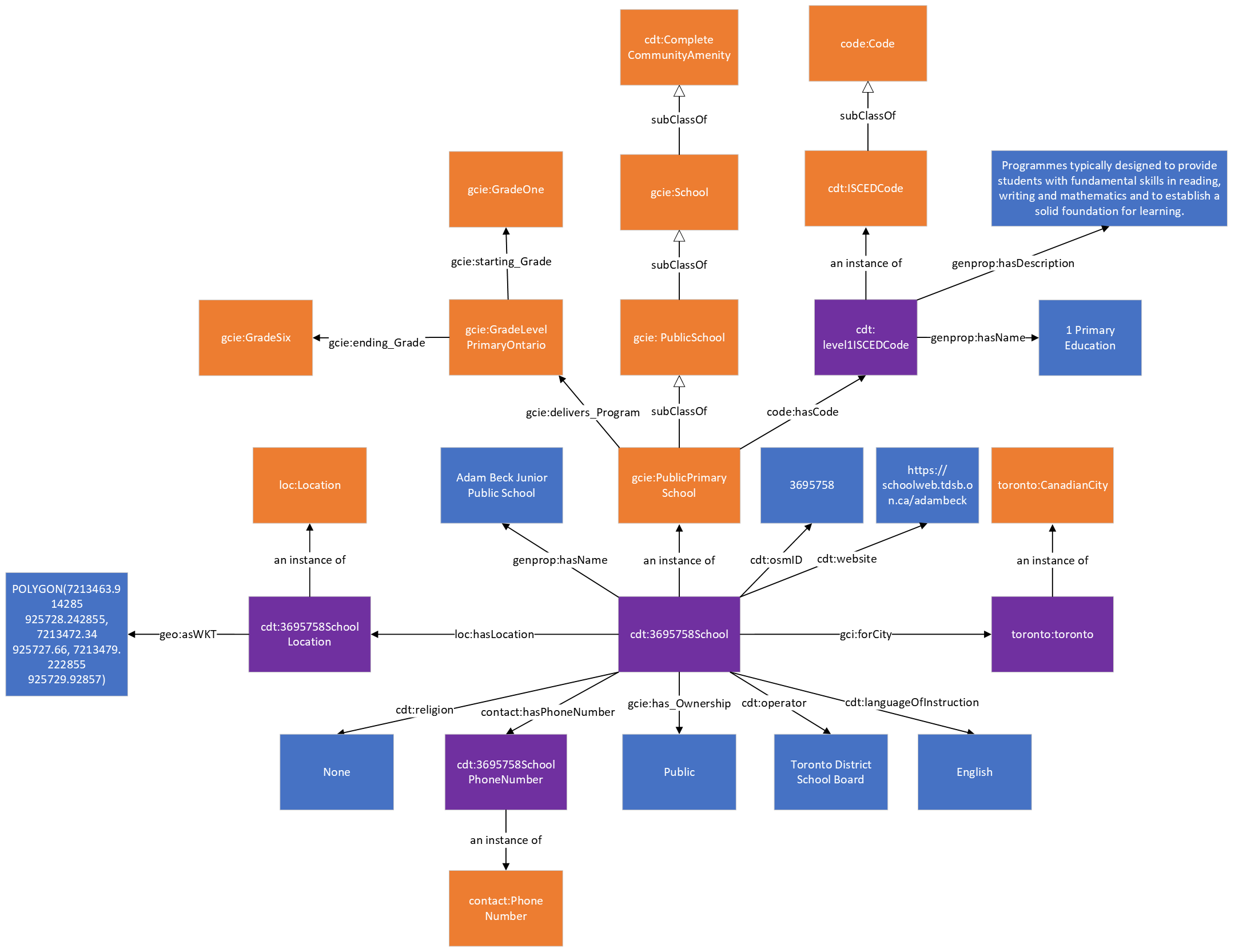
Additionally, for OpenStreetMap park data, the instance can also have an osmID property that links the park to its OSM ID (a unique identifier that is used in OpenStreetMap). It can also be linked to OpenStreetMap’s “leisure=park” tag that is used to identify parks in OpenStreetMap using the hasCode property and Code class.



## Schools

This is an ontological representation of school data. Schools are represented as instances of the School class from the GCI Education Ontology (GCIE). Schools can have a name which is linked via the hasName property, a website which is linked via the website property, a religion which is linked via the religion property, a phone number which is linked via the hasPhoneNumber property, an ownership type (e.g. public, private) which is linked via the has\_Ownership property, an operator which is linked via the operator property, a language of instruction (i.e. the language used to teach in a school) which is linked via the languageOfInstruction property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property respectively. School classes are linked to an ISCED code instance which describes the level of education that type of school provides. For example, in the diagram below, the PublicPrimarySchool class has an ISCED code of level 1 which represents Primary Education (i.e., programmes typically designed to provide students with fundamental skills in reading, writing and mathematics and to establish a solid foundation for learning.)

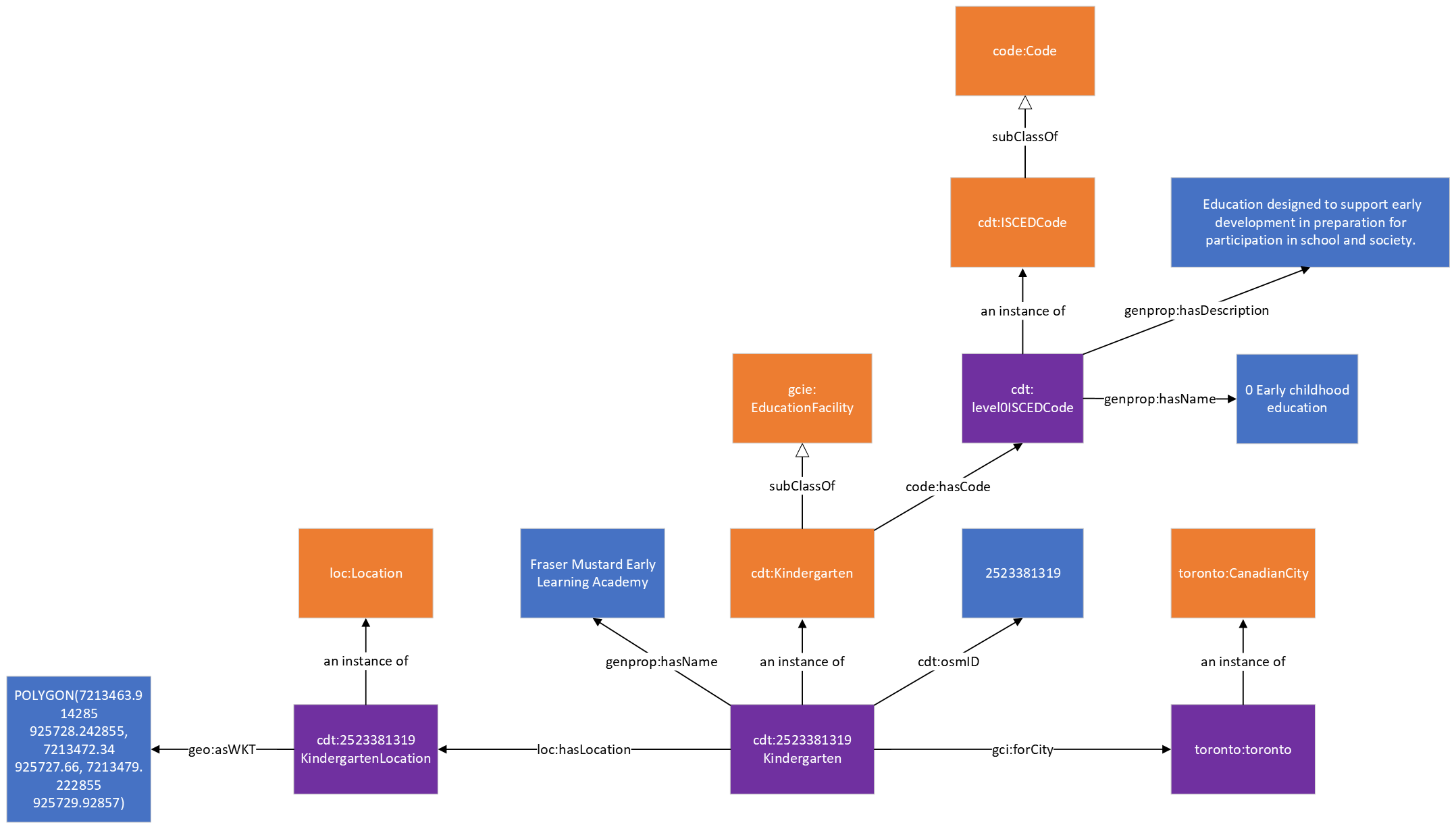
Additionally, for OpenStreetMap school data, the instance can also have an osmID property that links the school to its OSM ID (a unique identifier that is used in OpenStreetMap).



## Kindergartens

This is an ontological representation of kindergarten data. Kindergartens are represented as instances of the Kindergarten class which is a subclass of the EducationFacility class from the GCI Education Ontology. Kindergartens can have a name which is linked via the hasName property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property respectively. The Kindergarten class is linked to an ISCED code instance which describes the level of education that a kindergarten provides using the hasCode property. Kindergartens provide ISCED level 0 education (i.e., education designed to support early development in preparation for participation in school and society).

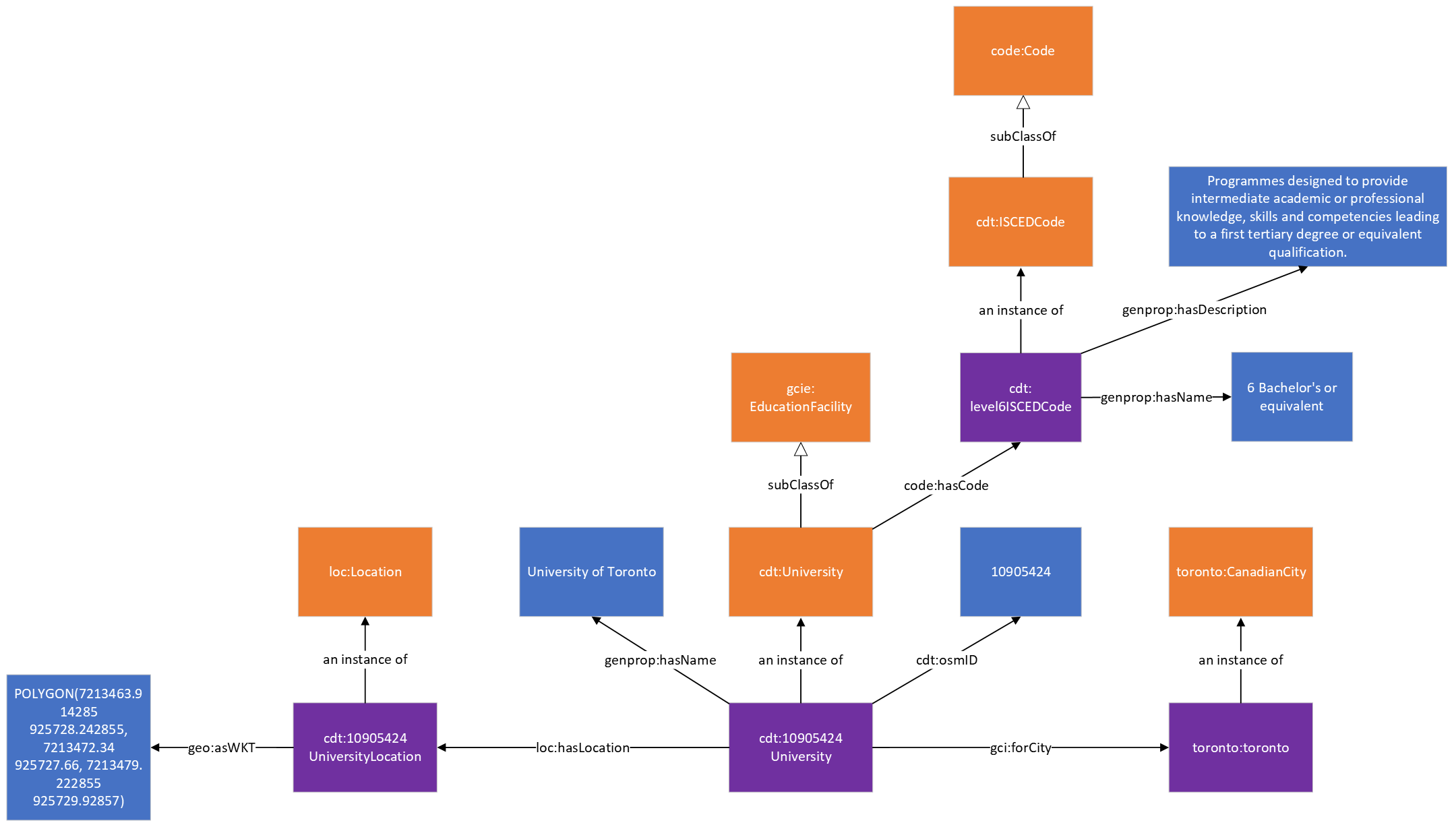
Additionally, for OpenStreetMap kindergarten data, the instance can also have an osmID property that links the kindergarten to its OSM ID (a unique identifier that is used in OpenStreetMap).



## Universities

This is an ontological representation of university data. Universities are represented as instances of the University class which is a subclass of the EducationFacility class from the GCI Education Ontology. Universities can have a name which is linked via the hasName property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property respectively. The University class is linked to an ISCED code instance which describes the level of education that a university provides using the hasCode property. Universities provide ISCED level 6 education (i.e., programmes designed to provide intermediate academic or professional knowledge, skills and competencies leading to a first tertiary degree or equivalent qualification) and higher.

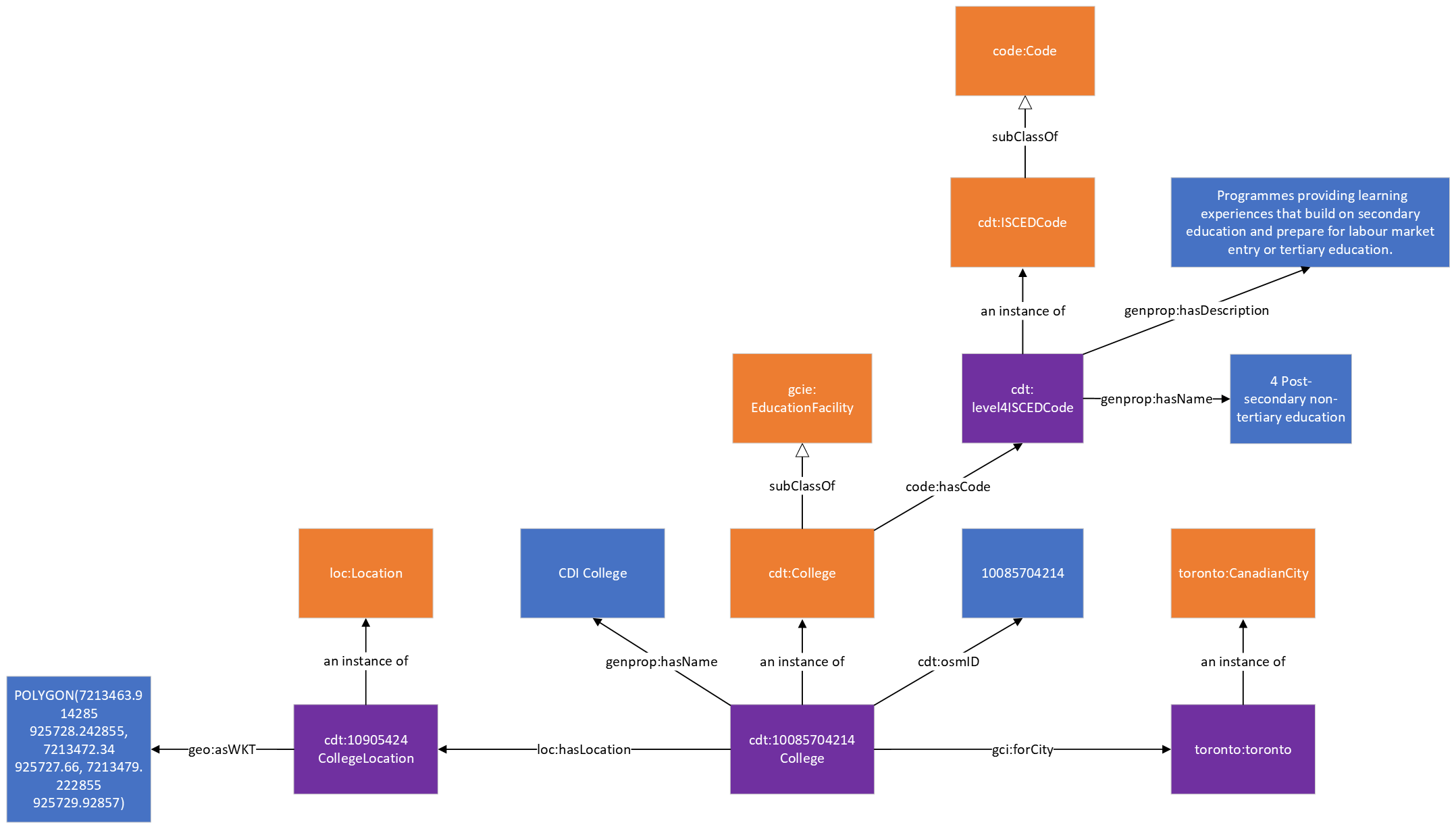
Additionally, for OpenStreetMap university data, the instance can also have an osmID property that links the university to its OSM ID (a unique identifier that is used in OpenStreetMap).



## Colleges

This is an ontological representation of college data. Colleges are represented as instances of the College class which is a subclass of the EducationFacility class from the GCI Education Ontology. Colleges can have a name which is linked via the hasName property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property respectively. The College class is linked to an ISCED code instance which describes the level of education that a college provides using the hasCode property. Colleges provide ISCED level 4 education (i.e., programmes providing learning experiences that build on secondary education and prepare for labour market entry or tertiary education.) and higher.

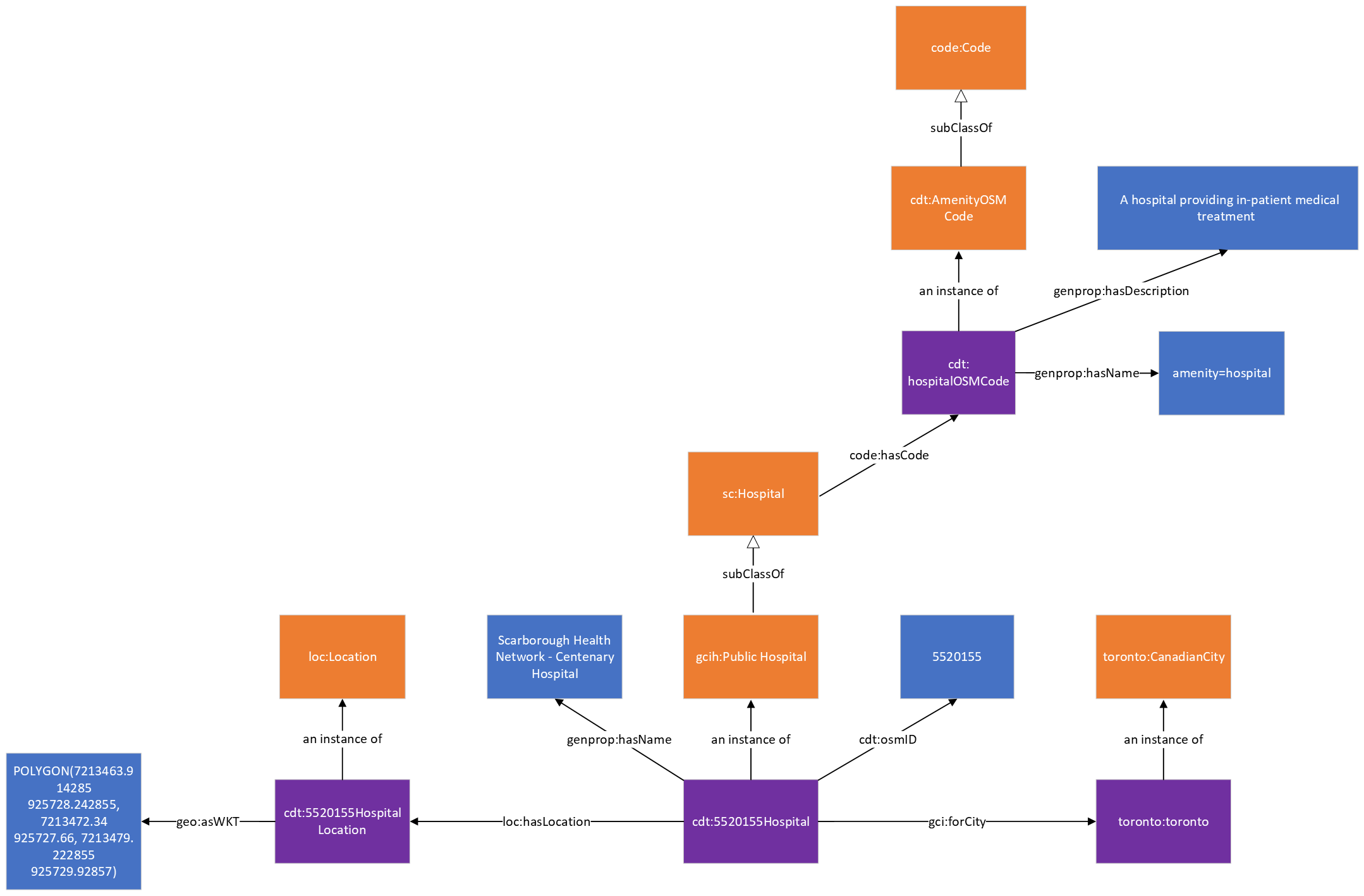
Additionally, for OpenStreetMap college data, the instance can also have an osmID property that links the college to its OSM ID (a unique identifier that is used in OpenStreetMap).



## Hospitals

This is an ontological representation of hospital data. Hospitals are represented as instances of the Hospital class from Schema.org which has the subclasses PublicHospital and PrivateHospital from the GCI Healthcare Ontology to indicate whether they are publicly or privately operated. Hospitals can have a name which is linked via the hasName property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property respectively. The Hospital class can also use the hasCode property in order to link it to a specific set list of values (e.g., classification systems).

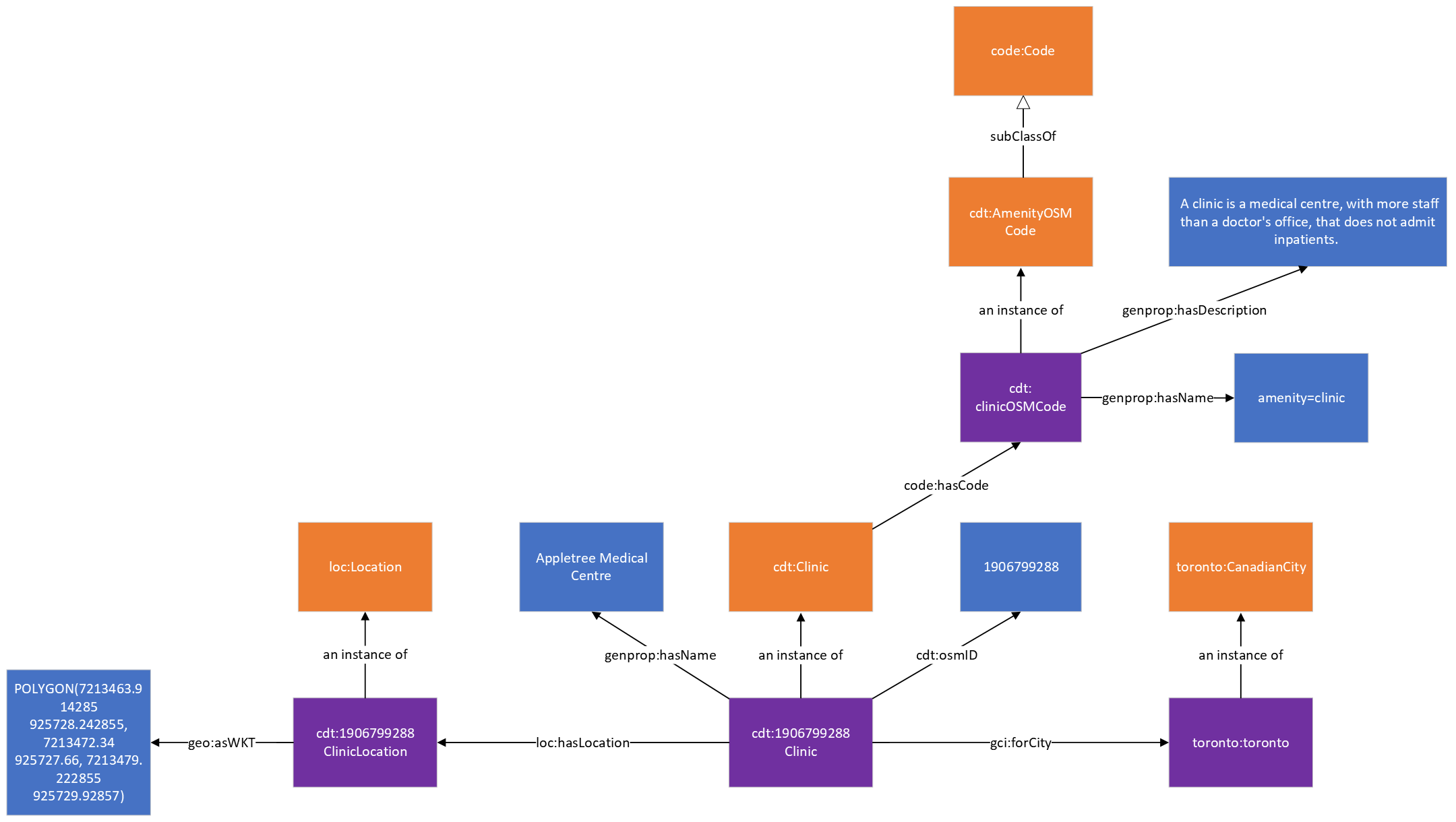
Additionally, for OpenStreetMap hospital data, the instance can also have an osmID property that links the hospital to its OSM ID (a unique identifier that is used in OpenStreetMap).



## Clinics

This is an ontological representation of clinic (i.e. medical centre) data. Clinics are represented as instances of the clinic class. Clinics can have a name which is linked via the hasName property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property respectively. The Clinic class can also use the hasCode property in order to link it to a specific set list of values (e.g., classification systems).

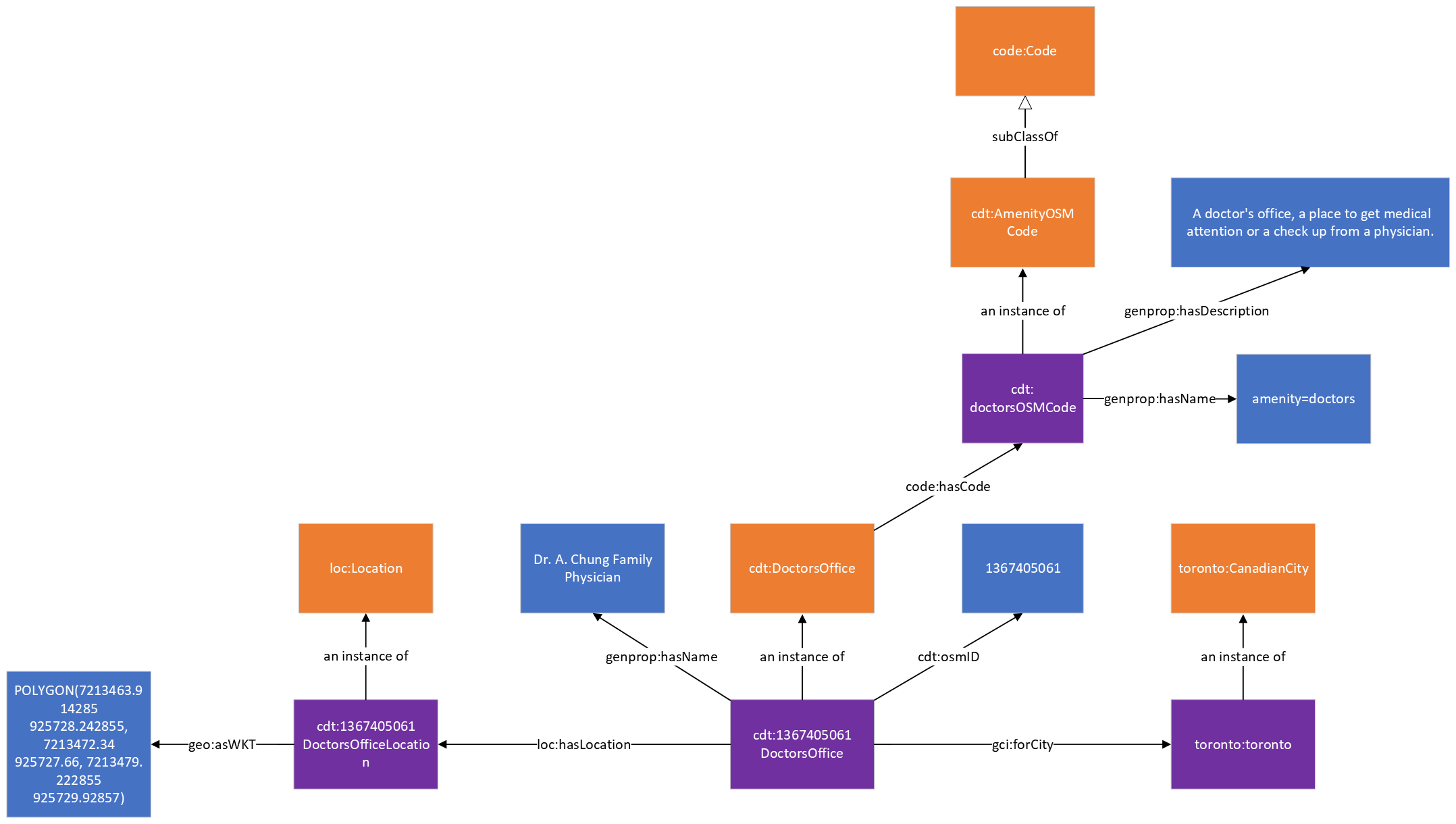
Additionally, for OpenStreetMap clinic data, the instance can also have an osmID property that links the clinic to its OSM ID (a unique identifier that is used in OpenStreetMap).



## Doctor’s Office

This is an ontological representation of doctor’s office data. Doctor’s offices are represented as instances of the DoctorsOffice class. Doctor’s offices can have a name which is linked via the hasName property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property, respectively. The DoctorsOffice class can also use the hasCode property in order to link it to a specific set list of values (e.g., classification systems).

Additionally, for OpenStreetMap doctor’s office data, the instance can also have an osmID property that links the doctor’s office to its OSM ID (a unique identifier that is used in OpenStreetMap).



## Pharmacy

This is an ontological representation of pharmacy data. Pharmacies are represented as instances of the Pharmacy class. Pharmacies can have a name which is linked via the hasName property, a Location instance and a set of geospatial coordinates linked using the hasLocation and asWKT property, respectively. The Pharmacy class can also use the hasCode property in order to link it to a specific set list of values (e.g., classification systems).

Additionally, for OpenStreetMap pharmacy data, the instance can also have an osmID property that links the pharmacy to its OSM ID (a unique identifier that is used in OpenStreetMap).

